Raport 2 - Eksploracja Danych

Tablety

Patryk Zając 297110

1 Przygotowanie danych

Najpierw zajmijmy się przygotowaniem danych. Poprawiamy dane z naszego pliku (przecinki zamieniamy na kropki i zamieniamy dane typu **object** na dane typu **float**):

```
import pandas as pd
tablety = pd.read_csv('Dane_tablety.csv', sep=';')
tablety = tablety.replace(',', '.', regex=True)
tablety = tablety.astype({'Waga':'float',
'Procesor_prędkość':'float', 'Żywotność_baterii':'float',
'Pamięć_wewnętrzna':'float', 'Ekran_rozmiar':'float',
'dim3m':'float', 'dim1m':'float'})
```

Teraz ustawiamy ziarno generatora:

```
import random
random.seed(300083)
```

Wyodrębniamy zmienną celu i dzielimy dane na zbiór uczący i testowy:

```
X = tablety.drop(columns=['Cena'])
y = tablety['Cena']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=300083)
```

2 Budowa modeli klasyfikujących

Budujemy dwa modele, jeden oparty na klasie **LinearRegression** z modułu *sklearn.linear_model* a drugi na klasie **SGDRegressor** z modułu *sklearn.linear_model*

2.1 LinearRegression

Do budowy modelu **LinearRegression** skorzystamy z modułu *sklearn.linear_model:*

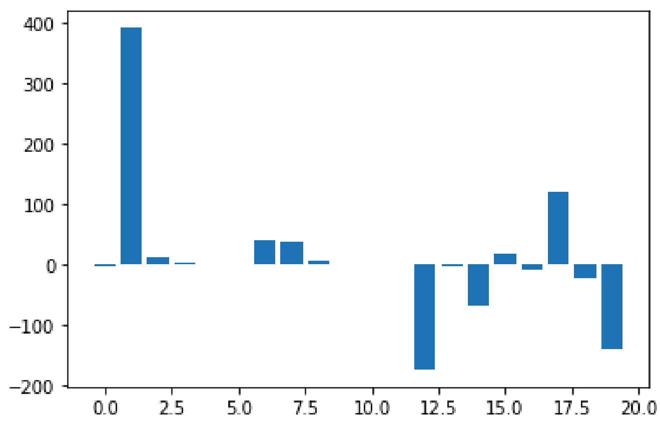
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

Zobaczymy jak wyglądają współczynniki zbudowanego modelu (ważność zmiennych):

```
pd.DataFrame(model.coef_, index = model.feature_names_in_,
columns = ['wspolczynnik modelu'])
```

	wspolczynnik modelu
marka	-3.002383
os_standardowy	390.626644
Waga	12.514153
Żywotność_baterii	3.272269
RAM	-0.007404
Pamięć_wewnętrzna	-1.232281
Procesor_prędkość	39.167345
Ekran_rozmiar	35.889119
N_MIES_WPROW	4.997914
n_px	-0.000069
Rozdzelczość_x	0.328538
Rozdzelczość_y	-0.188434
dim1m	-174.213124
dim3m	-1.722472
GPS	-69.722350
Procesor_n_rdzeni	16.559316
Konstrukcja	-8.079149
gsm	119.001202
marka_segment	-23.640629
inne_złącza	-141.033808

```
from matplotlib import pyplot
pyplot.bar([x for x in range(len(model.coef_))], model.coef_)
pyplot.show()
```



Widzimy że w szacowaniu ceny tabletu w naszym modelu największą rolę odgrywa system operacyjny – współczynnik równy **390.626644**. Najmniej znacząca natomiast jest szerokość w calach – współczynnik równy **-174.213124**.

Sprawdzamy teraz ile wynosi wyraz wolny w naszym równaniu:

```
print('Wyraz wolny=',model.intercept_)
```

```
Wyraz wolny= -30.728922246489788
```

Następnie wypisujemy jak wygląda dokładnie równanie szacujące cenę tabletu:

```
print('Cena = ', end='')
for b,n in zip(model.coef_, model.feature_names_in_):
  print(round(b,2), '*', n, '+ ',end='')
  print(round(model.intercept_,2))
```

```
Cena = -3.0 * marka + -30.73
390.63 * os_standardowy + -30.73
12.51 * Waga + -30.73
3.27 * Żywotność_baterii + -30.73
-0.01 * RAM + -30.73
-1.23 * Pamięć_wewnętrzna + -30.73
39.17 * Procesor_prędkość + -30.73
35.89 * Ekran_rozmiar + -30.73
5.0 * N_MIES_WPROW + -30.73
-0.0 * n_px + -30.73
0.33 * Rozdzelczość_x + -30.73
-0.19 * Rozdzelczość_y + -30.73
-174.21 * dim1m + -30.73
-1.72 * dim3m + -30.73
-69.72 * GPS + -30.73
16.56 * Procesor_n_rdzeni + -30.73
-8.08 * Konstrukcja + -30.73
119.0 * gsm + -30.73
-23.64 * marka_segment + -30.73
-141.03 * inne_złącza + -30.73
```

Wyznaczamy teraz współczynnik $\mathbf{R^2}$:

```
model.score(X_train,y_train)
```

```
0.8207607183659655
```

Widzimy, że jego wartość jest dość wysoka, jednak może być zbyt optymistyczna i dlatego wyznaczymy też **skorygowany współczynnik determinacji**, który określa się wzorem:

$$R^2 = 1 - (1 - R^2) \frac{n}{n-p}$$

```
def adjusted_R2(model, X, y):
  r2 = model.score(X,y)
  n, p = X.shape
  return 1 - (1-r2)*n/(n-p)
adjusted_R2(model, X_train, y_train)
```

```
0.7153258468165336
```

Wartość skorygowanego współczynnika jest nieco niższa.

Sprawdzimy teraz jakość predykcji naszego modelu:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
from math import sqrt
def ocen_model_regresji(y_true, y_pred, digits = 3):
  print('Średni błąd bezwzględny:',
round(mean_absolute_error(y_true,y_pred), digits))
  print('Błąd średniokwadratowy:', round(mean_squared_error(y_true,
y_pred),digits))
  print('Pierwiastek błędu
średniokwadratowego:',round(sqrt(mean_squared_error(y_true, y_pred)),
digits))
  print('Średni bezwzględny błąd
procentowy:',round(100*mean_absolute_percentage_error(y_true, y_pred),
digits),'%')
```

Dla zbioru uczącego mamy:

```
y_pred = model.predict(X_train)
ocen_model_regresji(y_train, y_pred)
```

```
Średni błąd bezwzględny: 63.975
Błąd średniokwadratowy: 5986.747
Pierwiastek błędu średniokwadratowego: 77.374
Średni bezwzględny błąd procentowy: 23.865 %
```

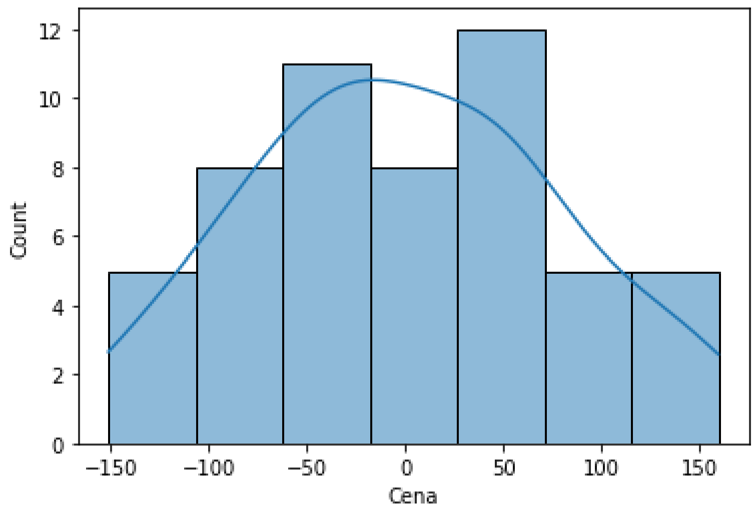
Natomiast dla zbioru testowego mamy:

```
y_pred = model.predict(X_test)
ocen_model_regresji(y_test, y_pred)
```

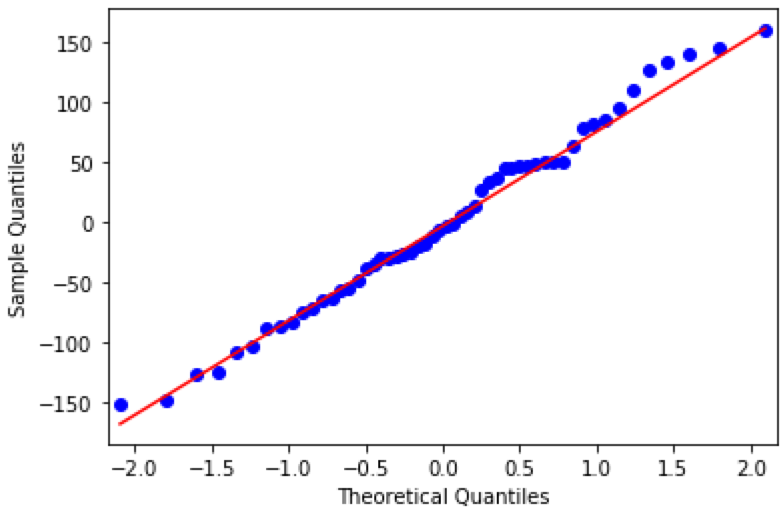
```
Średni błąd bezwzględny: 101.687
Błąd średniokwadratowy: 17863.636
Pierwiastek błędu średniokwadratowego: 133.655
Średni bezwzględny błąd procentowy: 29.938 %
```

Sprawdzimy teraz założenia modelu. Zaczniemy od sprawdzenia najpierw czy reszty (błędy modelu) mają rozkład normalny ze średnią 0 i stałą wariancją. Narysujemy **histogram**, **wykres kwantyl-kwantyl** i wykonamy **test Shapiro-Wilka**.

```
import seaborn as sns
y_pred = model.predict(X_train)
reszty = y_train - y_pred
sns.histplot(x=reszty, kde=True)
```




```
import statsmodels.api as sm
sm.qqplot(data = reszty, line='q');
```




```
from scipy.stats import shapiro
shapiro(reszty)
```

```
ShapiroResult(statistic=0.9825579524040222, pvalue=0.6156514883041382)
```

Histogram jest zbliżony do histogramu rozkładu normalnego. Jeżeli chodzi o wykres Q-Q to niebieskie punkty układają się mniej więcej wzdłuż linii czerwonej co sugeruje nam, że rozkład reszt jest normalny. Hipotezą zerową testu była normalność rozkładu, jak widzimy, p-wartość testu jest znacznie większa od 0.05 (czyli standardowego poziomu istotności). Nie mamy zatem podstaw aby odrzucić hipotezę zerową, więc **rozkład reszt jest normalny**.

Zbadamy teraz niezależność reszt za pomocą **testu Durbina-Watsona**:

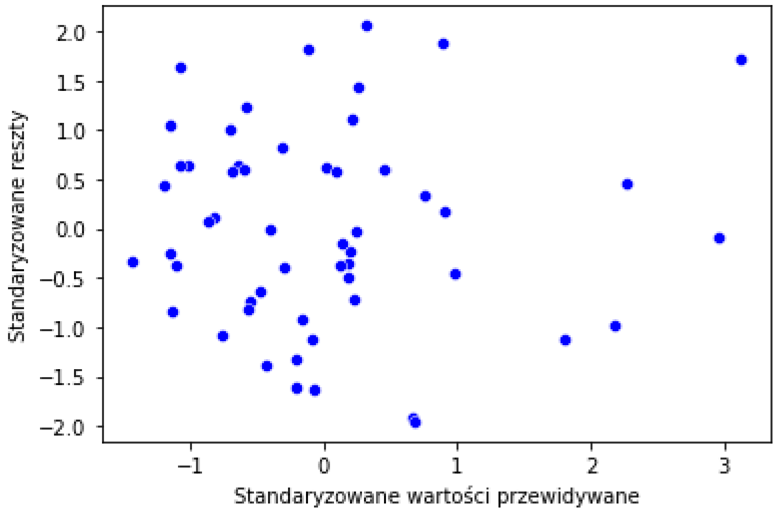
```
from statsmodels.stats.stattools import durbin_watson
durbin_watson(reszty)
```

```
1.8831610765551825
```

Zastosujemy tzw. **regułę kciuka**, która mówi, że wartość statystyki testowej powinna należeć do przedziału [1.5, 2.5]. Nasz wynik nie wykracza poza ten przedział, co świadczy o tym, że **reszty nie są ze sobą skorelowane**.

Teraz zajmiemy się homoskedastycznością reszt, czyli równością ich wariancji. Sprawdzamy ją rysując **wykres rozrzutu standaryzowanych reszt względem standaryzowanych wartości przewidywanych**. Jeżeli wariancje są równe na wykresie nie powinny być widoczne żadne wyraźne wzorce.

```
import matplotlib.pyplot as plt
from scipy.stats import zscore
sns.scatterplot(x = zscore(y_pred), y=zscore(reszty), color =
'blue')
plt.xlabel('Standaryzowane wartości przewidywane')
plt.ylabel('Standaryzowane reszty')
```



Nie są widoczne żadne wzorce, **zachowana jest homoskedastyczność**.

Sprawdźmy jeszcze czy w zbiorze występują obserwacje odstające:

```
abs(zscore(reszty)) > 3
```

Widzimy, że **obserwacji odstających nie ma**.

2.2 SGDRegressor

Do budowy modelu **SGDRegressor** skorzystamy również z modułu *sklearn.linear_model*:

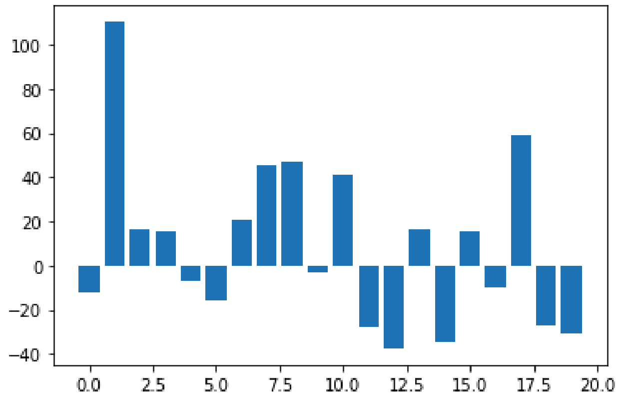
```
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
# Skaluje dane od -1 do 1
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
sgdr = SGDRegressor(random_state=300083)
sgdr.fit(X_train, y_train)
```

Zobaczymy jak wyglądają współczynniki zbudowanego modelu (ważność zmiennych):

```
pd.DataFrame(sgdr.coef_, index = model.feature_names_in_, columns
= ['wspolczynnik modelu'])
```

	wspolczynnik modelu
marka	-12.370726
os_standardowy	110.423895
Waga	16.537129
Żywotność_baterii	15.452278
RAM	-6.699605
Pamięć_wewnętrzna	-15.885177
Procesor_prędkość	21.196502
Ekran_rozmiar	45.489556
N_MIES_WPROW	47.170760
n_px	-3.149867
Rozdzelczość_x	41.207192
Rozdzelczość_y	-27.600739
dim1m	-37.463690
dim3m	16.153119
GPS	-34.573273
Procesor_n_rdzeni	15.773204
Konstrukcja	-9.634332
gsm	59.116221
marka_segment	-26.978393
inne_złącza	-31.049059

```
from matplotlib import pyplot
pyplot.bar([x for x in range(len(sgdr.coef_))], sgdr.coef_)
pyplot.show()
```



Widzimy że w szacowaniu ceny tabletu w naszym modelu największą rolę odgrywa system operacyjny -- współczynnik równy **110.423895**. Najmniej znacząca natomiast jest szerokość w calach – współczynnik równy **-37.463690**.

Sprawdzamy teraz ile wynosi wyraz wolny w naszym równaniu:

```
print('Wyraz wolny=',sgdr.intercept_)
```

```
Wyraz wolny= [337.25102827]
```

Następnie wypisujemy jak wygląda dokładnie równanie szacujące cenę tabletu:

```
print('Cena = ', end='')
for b,n in zip(sgdr.coef_, model.feature_names_in_):
  print(round(b,2), '*', n, '+ ',end='')
  print(sgdr.intercept_)
```

```
Cena = -12.37 * marka + [337.25102827]
110.42 * os_standardowy + [337.25102827]
16.54 * Waga + [337.25102827]
15.45 * Żywotność_baterii + [337.25102827]
-6.7 * RAM + [337.25102827]
-15.89 * Pamięć_wewnętrzna + [337.25102827]
21.2 * Procesor_prędkość + [337.25102827]
45.49 * Ekran_rozmiar + [337.25102827]
47.17 * N_MIES_WPROW + [337.25102827]
-3.15 * n_px + [337.25102827]
41.21 * Rozdzelczość_x + [337.25102827]
-27.6 * Rozdzelczość_y + [337.25102827]
-37.46 * dim1m + [337.25102827]
16.15 * dim3m + [337.25102827]
-34.57 * GPS + [337.25102827]
15.77 * Procesor_n_rdzeni + [337.25102827]
-9.63 * Konstrukcja + [337.25102827]
59.12 * gsm + [337.25102827]
-26.98 * marka_segment + [337.25102827]
-31.05 * inne_złącza + [337.25102827]
```

Wyznaczamy teraz współczynnik $\mathbf{R^2}$:

```
sgdr.score(X_train,y_train)
```

```
0.8151347203416093
```

Widzimy, że jego wartość jest dość wysoka, jednak może być zbyt optymistyczna i dlatego wyznaczymy też skorygowany współczynnik determinacji:

```
def adjusted_R2(model, X, y):
  r2 = model.score(X,y)
  n, p = X.shape
  return 1 - (1-r2)*n/(n-p)
adjusted_R2(sgdr, X_train, y_train)
```

```
0.7063904381896147
```

Wartość skorygowanego współczynnika jest nieco niższa.

Sprawdzimy teraz jakość predykcji naszego modelu:

```
from sklearn.metrics import mean_absolute_error,
mean_squared_error, mean_absolute_percentage_error
from math import sqrt
def ocen_model_regresji(y_true, y_pred, digits = 3):
  print('Średni błąd bezwzględny:',
round(mean_absolute_error(y_true,y_pred), digits))
  print('Błąd średniokwadratowy:',
round(mean_squared_error(y_true, y_pred),digits))
  print('Pierwiastek błędu
średniokwadratowego:',round(sqrt(mean_squared_error(y_true,
y_pred)), digits))
  print('Średni bezwzględny błąd
procentowy:',round(100*mean_absolute_percentage_error(y_true,
y_pred), digits),'%')
```

Dla zbioru uczącego mamy:

```
y_pred = sgdr.predict(X_train)
ocen_model_regresji(y_train, y_pred)
```

```
Średni błąd bezwzględny: 63.938
Błąd średniokwadratowy: 6174.66
Pierwiastek błędu średniokwadratowego: 78.579
Średni bezwzględny błąd procentowy: 23.647 %
```

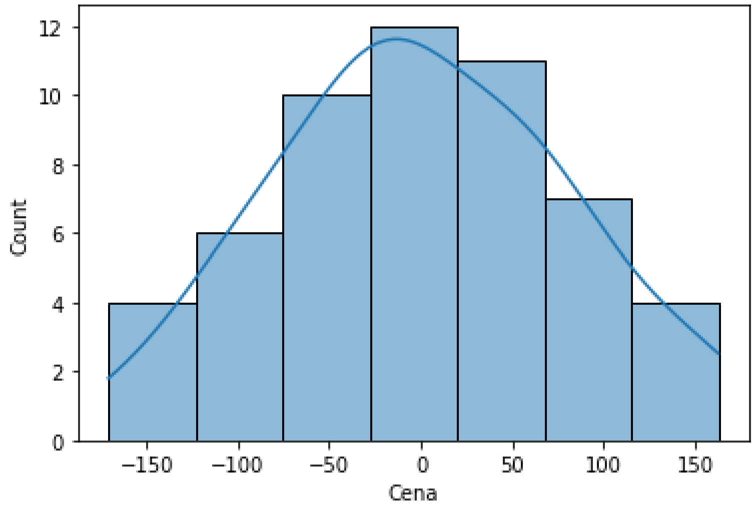
Natomiast dla zbioru testowego mamy:

```
y_pred = sgdr.predict(X_test)
ocen_model_regresji(y_test, y_pred)
```

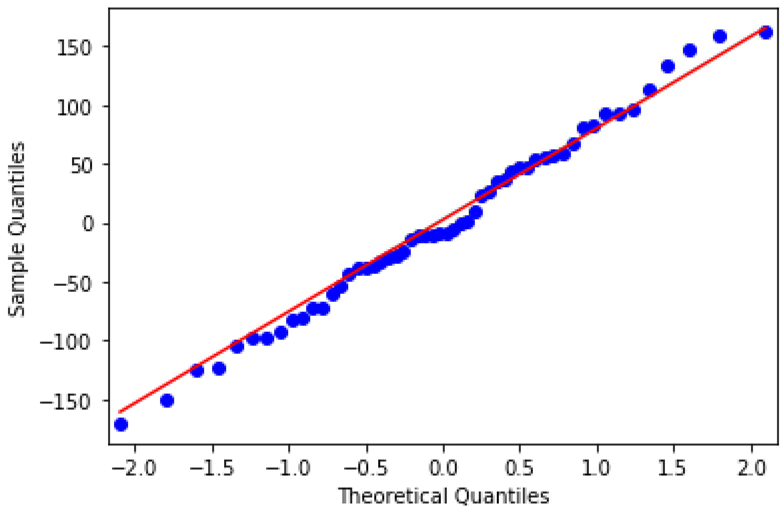
```
Średni błąd bezwzględny: 102.694
Błąd średniokwadratowy: 19090.893
Pierwiastek błędu średniokwadratowego: 138.17
Średni bezwzględny błąd procentowy: 29.989 %
```

Sprawdzimy teraz założenia modelu. Zaczniemy od sprawdzenia najpierw czy reszty (błędy modelu) mają rozkład normalny ze średnią 0 i stałą wariancją. Narysujemy histogram, wykres kwantyl-kwantyl i wykonamy test Shapiro-Wilka.

```
import seaborn as sns
y_pred = sgdr.predict(X_train)
reszty = y_train - y_pred
sns.histplot(x=reszty, kde=True)
```




```
import statsmodels.api as sm
sm.qqplot(data = reszty, line='q');
```



```
from scipy.stats import shapiro
shapiro(reszty)
```

```
ShapiroResult(statistic=0.988083004951477, pvalue=0.8665271401405334)
```

Histogram jest zbliżony do histogramu rozkładu normalnego. Jeżeli chodzi o wykres Q-Q to niebieskie punkty układają się mniej więcej wzdłuż linii czerwonej co sugeruje nam że rozkład reszt jest normalny. Hipotezą zerową testu była normalność rozkładu, jak widzimy, p-wartość testu jest znacznie większa od 0.05 (czyli standardowego poziomu istotności). Nie mamy zatem podstaw aby odrzucić hipotezę zerową więc **rozkład reszt jest normalny**.

Zbadamy teraz niezależność reszt za pomocą testu Durbina-Watsona:

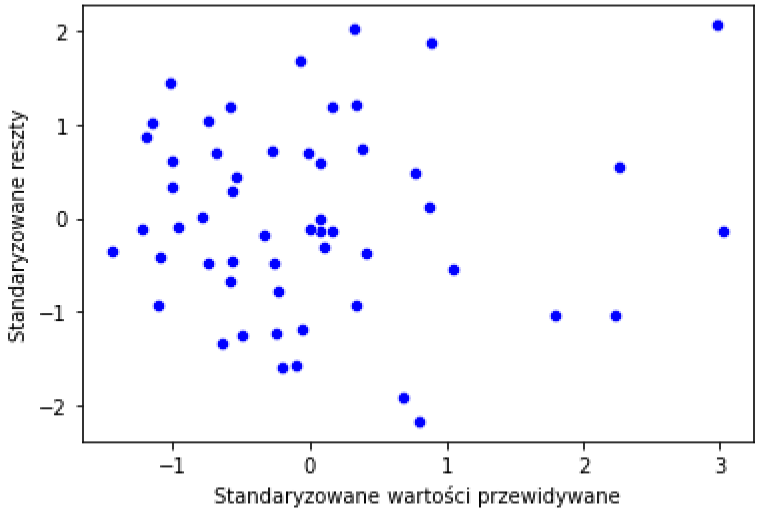
```
from statsmodels.stats.stattools import durbin_watson
durbin_watson(reszty)
```

```
1.828757805450216
```

Nasz wynik nie wykracza poza przedział [1.5, 2.5], co świadczy o tym, że **reszty nie są ze sobą skorelowane**.

Teraz zajmiemy się homoskedastycznością reszt. Sprawdzamy ją rysując wykres rozrzutu standaryzowanych reszt względem standaryzowanych wartości przewidywanych. Jeżeli wariancje są równe na wykresie nie powinny być widoczne żadne wyraźne wzorce.

```
import matplotlib.pyplot as plt
from scipy.stats import zscore
sns.scatterplot(x = zscore(y_pred), y=zscore(reszty), color =
'blue')
plt.xlabel('Standaryzowane wartości przewidywane')
plt.ylabel('Standaryzowane reszty')
```



Nie są widoczne żadne wzorce, **zachowana jest homoskedastyczność**.

Sprawdźmy jeszcze czy w zbiorze występują obserwacje odstające:

```
abs(zscore(reszty)) > 3
```

Widzimy, że **obserwacji odstających nie ma**.

3 Porównanie modeli

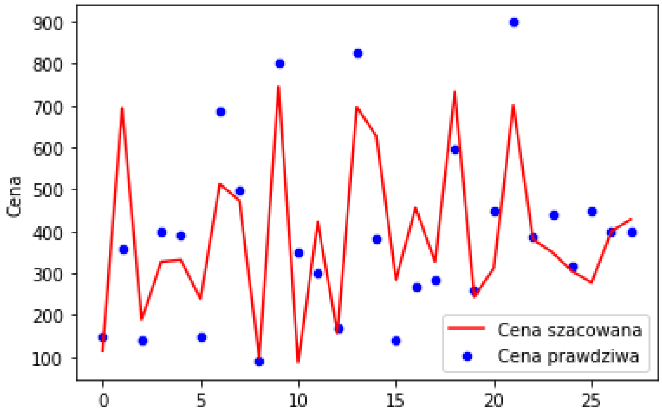
Tak wyglądają współczynniki dla modeli:

- **LinearRegression**
  - Zbiór uczący
    - Średni błąd bezwzględny: **63.975**
    - Błąd średniokwadratowy: **5986.747**
    - Pierwiastek błędu średniokwadratowego: **77.374**
    - Średni bezwzględny błąd procentowy: **23.865%**
  - Zbiór testowy
    - Średni błąd bezwzględny: **101.687**
    - Błąd średniokwadratowy: **17863.636**
    - Pierwiastek błędu średniokwadratowego: **133.655**
    - Średni bezwzględny błąd procentowy: **29.938%**
- **SGDRegressor**
  - Zbiór uczący
    - Średni błąd bezwzględny: **63.938**
    - Błąd średniokwadratowy: **6174.66**
    - Pierwiastek błędu średniokwadratowego: **78.579**
    - Średni bezwzględny błąd procentowy: **23.647%**
  - Zbiór testowy
    - Średni błąd bezwzględny: **102.694**
    - Błąd średniokwadratowy: **19090.893**
    - Pierwiastek błędu średniokwadratowego: **138.17**
    - Średni bezwzględny błąd procentowy: **29.989%**

Natomiast tak wyglądają wykresy wartości przewidywanych względem obserwowanych na zbiorach testowych:

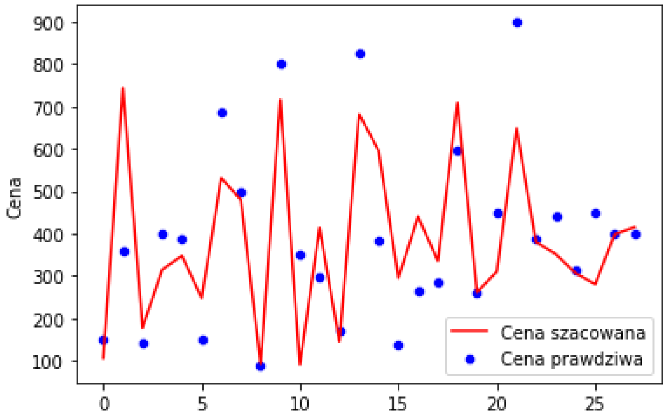
- **LinearRegression**

```
import seaborn as sns
cena_pred = model.predict(X_test)
cena = y_test.to_numpy()
sns.scatterplot(data = cena, color = 'blue')
sns.lineplot(data = cena_pred, color = 'red')
plt.ylabel('Cena')
plt.legend(['Cena szacowana','Cena prawdziwa'])
```



- **SGDRegressor**

```
import seaborn as sns
cena_pred = sgdr.predict(X_test)
cena = y_test.to_numpy()
sns.scatterplot(data = cena, color = 'blue')
sns.lineplot(data = cena_pred, color = 'red')
plt.ylabel('Cena')
plt.legend(['Cena szacowana','Cena prawdziwa'])
```



Model stworzony za pomocą **LinearRegression** radzi sobie trochę gorzej jeśli chodzi o zbiór testowy w porównaniu do zbioru treningowego. Nie są to jednak bardzo duże różnice (różnica w średnim bezwzględnym błędzie procentowym wynosi tylko **6%**, więc nie mamy doczynienia z przeuczeniem modelu). Podobnie jest dla modelu zbudowanego za pomocą **SGDRegressor**.

Zarówno model stworzony za pomocą **LinearRegression** jak i **SGDRegressor** dają bardzo zbliżone do siebie wyniki, różnice są minimalne (można to także zauważyć na powyższych wykresach, są one prawie identyczne). Dla zbioru testowego minimalnie lepiej radzi sobie model **LinearRegression**.

Modele radzą sobie całkiem dobrze, średni bezwzględny błąd procentowy na poziomie **23-29%** to nie najgorszy wynik (na wykresach możemy też zauważyć, że w niektórych momentach model trafia prawie idealnie w obserwowaną cenę tabletu). Jednak można by te modele poprawić, aby uzyskać lepsze wyniki. Moglibyśmy np. przeanalizować dane w celu znalezienia czy jakieś predyktory nie są ze sobą silnie skorelowane, co może wpływać negatywnie na jakość modelu.