Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marysia Nazarczuk
Nr albumu: 417755

Statystyczna Analiza Danych

Projekt zaliczeniowy

Warszawa, Czerwiec 2024

Spis treści

1. Wstęp

Zbierane w tym badaniu dane pochodzą z analizy pojedynczych komórek szpiku kostnego ludzkich dawców. W szczególności, analizujemy różnorodność komórek układu odpornościowego. Naszym celem jest dokładne zidentyfikowanie komórek, takich jak limfocyty T, wykorzystując dwa rodzaje informacji: ekspresję genów i ilość białek na powierzchni komórek. Poprawne rozpoznanie tych komórek może mieć istotne znaczenie w rozwijaniu nowych terapii, takich jak terapia CAR-T, która wykorzystuje modyfikowane genetycznie limfocyty T do walki z nowotworami.

W tej pracy znajdują się problemy i rozwiązania związane z genem "CD36" zaproponowane w poleceniu do zadania [1].

Poniżej znajduje się nagłówek z niezbędnymi bibliotekami

```
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde
from sklearn.linear_model import ElasticNetCV
import seaborn as sns
from sklearn.model_selection import KFold
import itertools
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
import warnings
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import KFold
nput directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

2. Eksploracja

1. Sprawdź, ile obserwacji i zmiennych zawierają wczytane dane treningowe oraz testowe. Przyjrzyj się typom zmiennych i, jeśli uznasz to za słuszne, dokonaj odpowiedniej konwersji przed dalszą analizą. Upewnij się, czy dane są kompletne.
2. Zbadaj rozkład empiryczny zmiennej objaśnianej (przedstaw kilka podstawowych statystyk, do analizy dołącz histogram lub wykres estymatora gęstości).
3. Wybierz 250 zmiennych objaśniających najbardziej skorelowanych ze zmienną objaśnianą. Policz korelację dla każdej z par tych zmiennych. Zilustruj wynik za pomocą mapy ciepła (ang. *heatmap*).

2.1. Liczba obserwacji

Wczytujemy dane i sprawdzamy jaki mają typ i czy są kompletne

```
# Sprawdzanie, czy typ danych to liczby calkowite
for col in x_train.columns:
    if not x_train[col].dtype.kind in ['i', 'f']:
        print("Zmienna", col, "nie jest typu numeric")

for col in y_train.columns:
    if not y_train[col].dtype.kind in ['i', 'f']:
        print("Zmienna", col, "nie jest typu numeric")

for col in x_test.columns:
    if not x_test[col].dtype.kind in ['i', 'f']:
        print("Zmienna", col, "nie jest typu numeric")

# Sprawdzanie, czy dane sa kompletne
complete_x = x_train.dropna().shape[0] == x_train.shape[0]
complete_y = y_train.dropna().shape[0] == y_train.shape[0]
complete_test = x_test.dropna().shape[0] == x_test.shape[0]

print("Dane treningowe x_train sa kompletne:", complete_x)
print("Dane treningowe y_train sa kompletne:", complete_y)
print("Dane testowe sa kompletne:", complete_test)
```

Wszystkie zaimportowane są typu `float64` oraz nie ma `null` ani `NAN` czyli dane są kompletne i poprawne.

Aby znaleźć liczbę obserwacji i zmiennych, wystarczy znaleźć odpowiednio liczbę wierszy i kolumn

```
x_obs = x_train.shape[0]
x_vars = x_train.shape[1]
print("Liczba obserwacji w x_train:", x_obs)
print("Liczba zmiennych w x_train:", x_vars)

y_obs = y_train.shape[0]
y_vars = y_train.shape[1]
print("Liczba obserwacji w y_train:", y_obs)
print("Liczba zmiennych w y_train", y_vars)

test_obs = x_test.shape[0]
test_vars = x_test.shape[1]
print("Liczba obserwacji w x_test:", test_obs)
print("Liczba zmiennych w x_test:", test_vars)
```

otrzymujemy

- W `x_train` obserwacji jest 6800, zaś zmiennych jest 9000
- W `y_train` obserwacji jest 6800, zaś zmiennych jest 1
- W `x_test` obserwacji jest 1200, zaś zmiennych jest 9000

2.2. Rozkład empiryczny zmiennej objaśnialnej

Aby znaleźć rozkład empiryczny zmiennej objaśnianej, wystarczy znaleźć podstawowe statystyki, najważniejsze z nich to średnia i wariancja próbki, a dodatkowo mediana oraz kwartyle 25% i 75%.

```
# Oblicz podstawowe statystyki
meanvals = y_train.mean()
varvals = y_train.var()
medvals = y_train.median()
quantile25 = y_train.quantile(0.25)
quantile75 = y_train.quantile(0.75)

# Utworz DataFrame z wynikami
res = pd.DataFrame({
    'Mean': meanvals,
    'Variance': varvals,
    'Median': medvals,
    'Quantile_25': quantile25,
    'Quantile_75': quantile75
})

print(res)
```

otrzymujemy

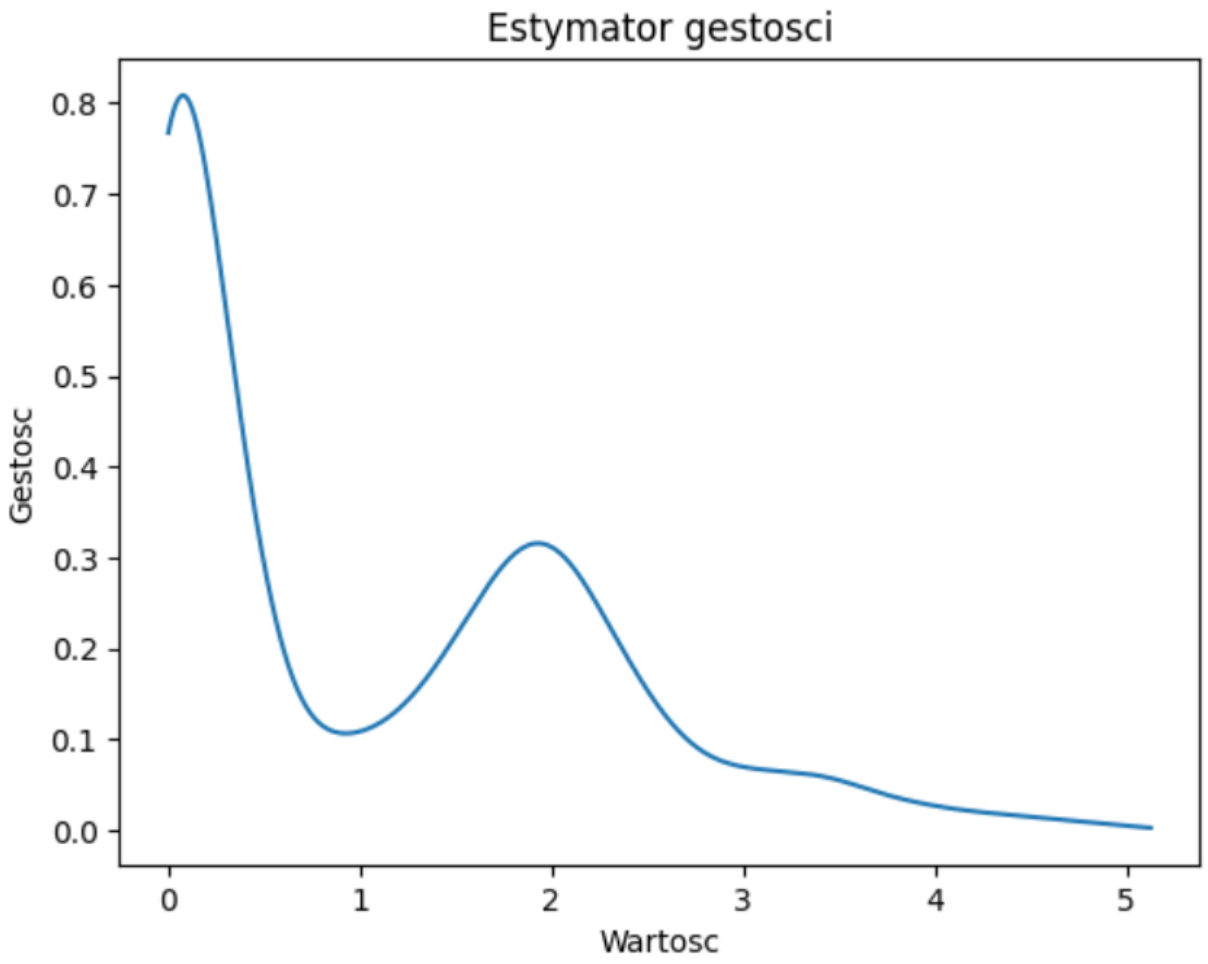
Tabela 2.1: Statystyki opisowe dla zmiennej CD36

	Mean	Variance	Median	Quantile_25	Quantile_75
CD36	1.086167	1.29217	0.543689	0.0	1.961154

Następnie generujemy wykres estymatora gęstości

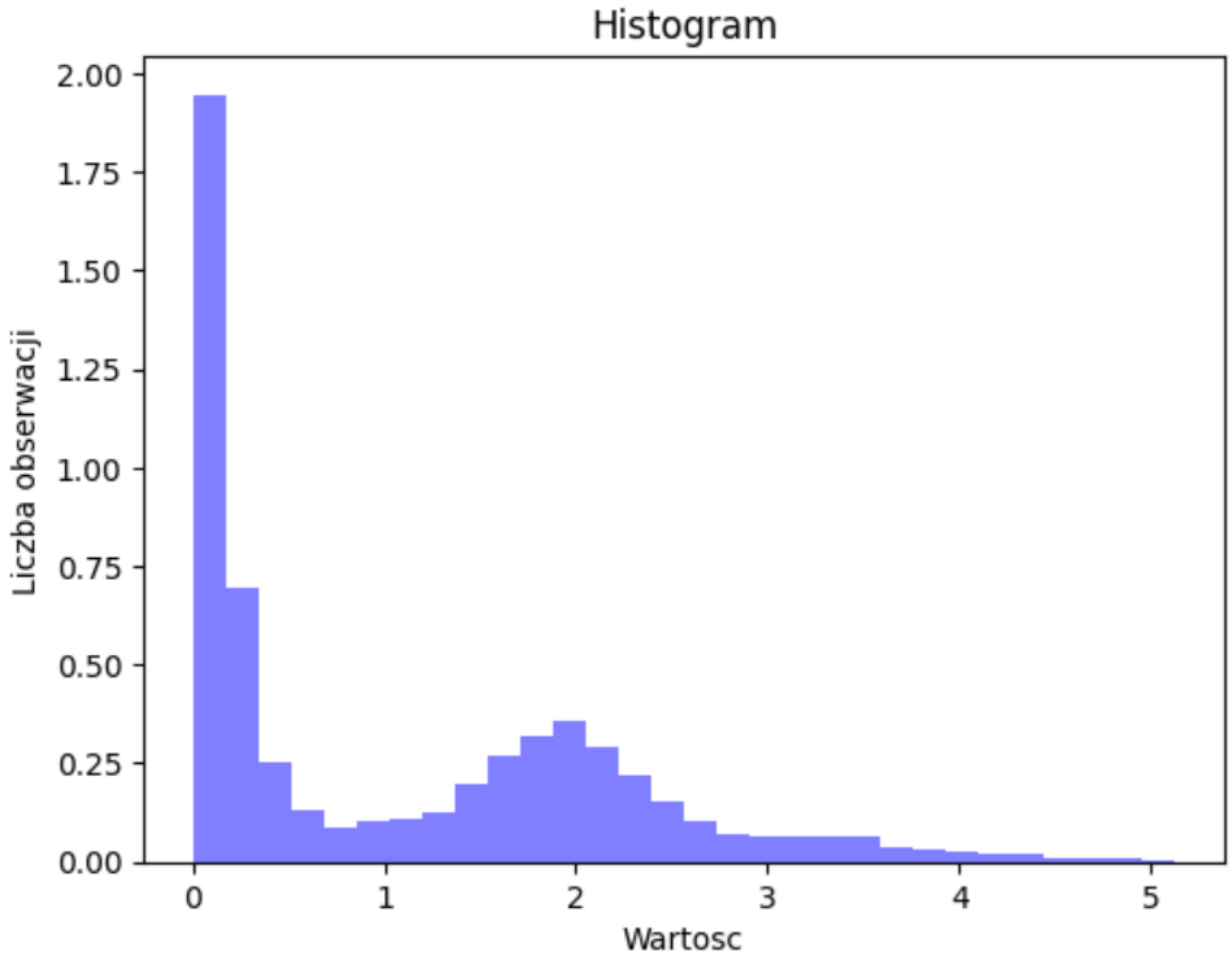
```
densityEstim = gaussian_kde(y_train["CD36"])
x = np.linspace(min(y_train["CD36"]), max(y_train["CD36"]), 1000)
plt.plot(x, densityEstim(x))
plt.title("Estymator gestosci")
plt.xlabel('Wartosc')
plt.ylabel('Gestosc')
plt.show()
```

otrzymujemy



oraz histogram

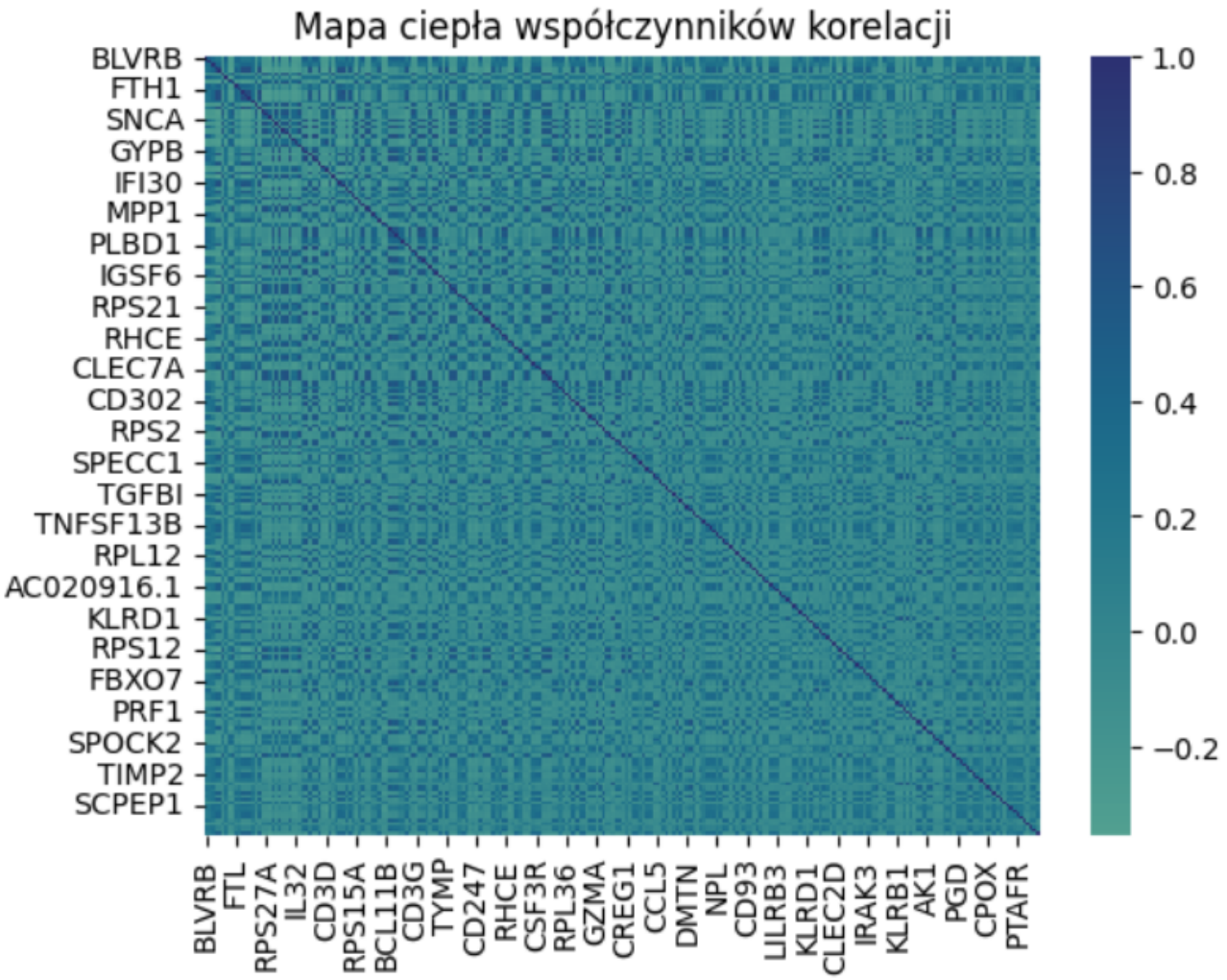
```
data = y_train["CD36"]
plt.hist(data, bins=30, density=True, alpha=0.5, color='b')
plt.title('Histogram')
plt.xlabel('Wartosc')
plt.ylabel('Liczba obserwacji')
plt.show()
```



2.3. Korelacja zmiennych

Wybierzmy teraz zestaw 250 zmiennych niezależnych, które są najbardziej skorelowane ze zmienną zależną, a następnie obliczmy współczynniki korelacji dla każdej pary zmiennych w tym zestawie. Zastosujemy współczynnik korelacji Kendalla, ponieważ nie mamy pewności, czy dane są rozkładane normalnie oraz czy zależności między zmiennymi są liniowe.

```
variable = y_train['CD36']
correlations = x_train.corrwith(variable, method='kendall')
abs_correlations = abs(correlations)
sorted_correlations = abs_correlations.sort_values(ascending=False)
top_250_vars = sorted_correlations.index[:250]
correlation_pairs = x_train[top_250_vars].corr(method='kendall')
sns.heatmap(correlation_pairs, center=0., cmap="crest")
plt.title("Mapa ciepla wspolczynnikow korelacji")
plt.show()
```


Mapa ciepła współczynników korelacji
BLVRB
FTH1
SNCA
GYPB
IFI30
MPP1
PLBD1
IGSF6
RPS21
RHCE
CLEC7A
CD302
RPS2
SPECC1
TGFBI
TNFSF13B
RPL12
AC020916.1
KLRD1
RPS12
FBXO7
PRF1
SPOCK2
TIMP2
SCPEP1
BLVRB
FTL
RPS27A
IL32
CD3D
RPS15A
BCL11B
CD3G
TYMP
CD247
RHCE
CSF3R
RPL36
GZMA
CREG1
CCL5
DMTN
NPL
CD93
LILRB3
KLRD1
CLEC2D
IRAK3
KLRB1
AK1
PGD
CPOX
PTAFR
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

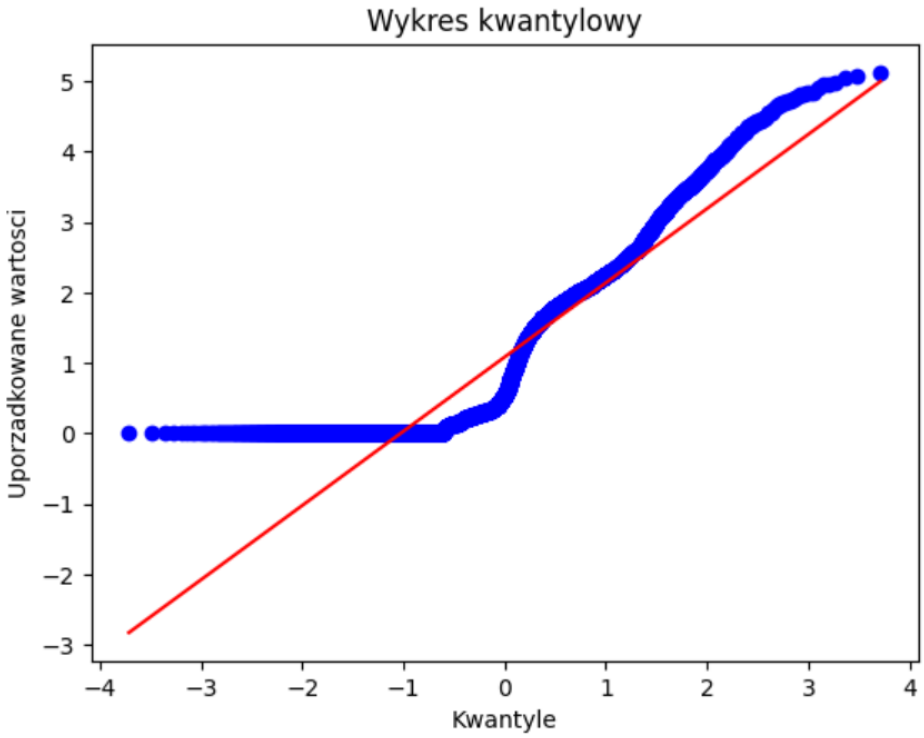
3. Testy statystyczne

1. Narysuj wykres kwantylowy porównujący zmienną objaśnianą z rozkładem normalnym. Na wykresie zaznacz prostą wyznaczoną przez kwantyle doświadczalne. Odpowiedz na pytanie, czy da się odczytać średnią i wariancję rozkładu doświadczalnego wprost z tego wykresu?
2. Przeprowadź test statystyczny hipotezy zgodności zmiennej objaśnianej z rozkładem normalnym.
3. (i) Wybierz zmienną objaśniającą najbardziej skorelowaną ze zmienną objaśnianą. Przeprowadź test statystyczny hipotezy zgodności wybranej zmiennej objaśniającej z wybranym (sensownym!) rozkładem (np. z rozkładem wykładniczym z parametrem 10, jeśli są ku temu przesłanki).
   (ii) Wybierz test statystyczny i sprawdź, czy wybrana zmienna objaśniająca ma podobny rozkład w zbiorze testowym i treningowym (w tym podpunkcie nie chodzi o test zgodności, ale o test sprawdzający lokalizację tych dwóch rozkładów - wzajemne położenie średnich lub median).

3.1. Wykres kwantylowy

Wygenerujmy wykres kwantylowy porównujący zmienną objaśnianą z rozkładem normalnym.

```
scipy.stats.probplot(x=y_train["CD36"], dist=scipy.stats.norm(),
    plot=plt)
plt.title('Wykres kwantylowy')
plt.xlabel('Kwantyle')
plt.ylabel('Uporzadkowane wartosci')
plt.show()
```



Z tego wykresu da się odczytać średnią (jest to wartość wykresu w zerze).

3.2. Test zgodności z rozkładem normalnym

W celu sprawdzenia, czy dana próbka pochodzi z rozkładu normalnego, zastosowano test normalności dostępny w bibliotece SciPy, `scipy.stats.normaltest` [2]. Test ten weryfikuje hipotezę zerową, że analizowana próbka jest zgodna z rozkładem normalnym. Metodologia Test normaltest jest oparty na metodzie D'Agostino i Pearsona, która łączy analizę skośności oraz kurtozy próbki, tworząc kompleksowy test normalności.

```
out = scipy.stats.normaltest(y_train)
```

Przeprowadzony test normalności dostarcza istotnych informacji na temat zgodności rozkładu badanej próbki z rozkładem normalnym. Otrzymujemy:

```
NormaltestResult(statistic=array([601.76868218]), pvalue=array([2.12613567e-131]))
```

3.3. Najbardziej skorelowana zmienna objaśniająca

3.3.1. Zmienna

Używając poniższej instrukcji dowiemy się jaka zmienna objaśniająca jest najbardziej skorelowana ze zmienną objaśnianą

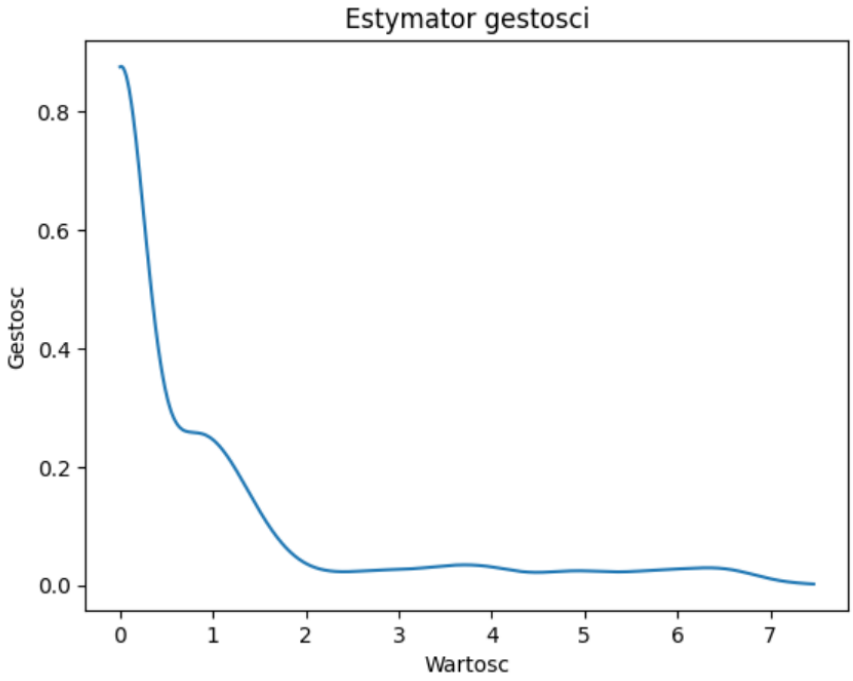
```
correlations.sort_values()
```

Otrzymujemy:

```
RPL10    -0.392748
ETS1     -0.383267
RPL3     -0.381298
RPS19    -0.372349
RPS27    -0.365489
             ...
FCN1      0.407361
VCAN      0.411220
CD36      0.449805
LGALS3    0.452651
BLVRB     0.494457
```

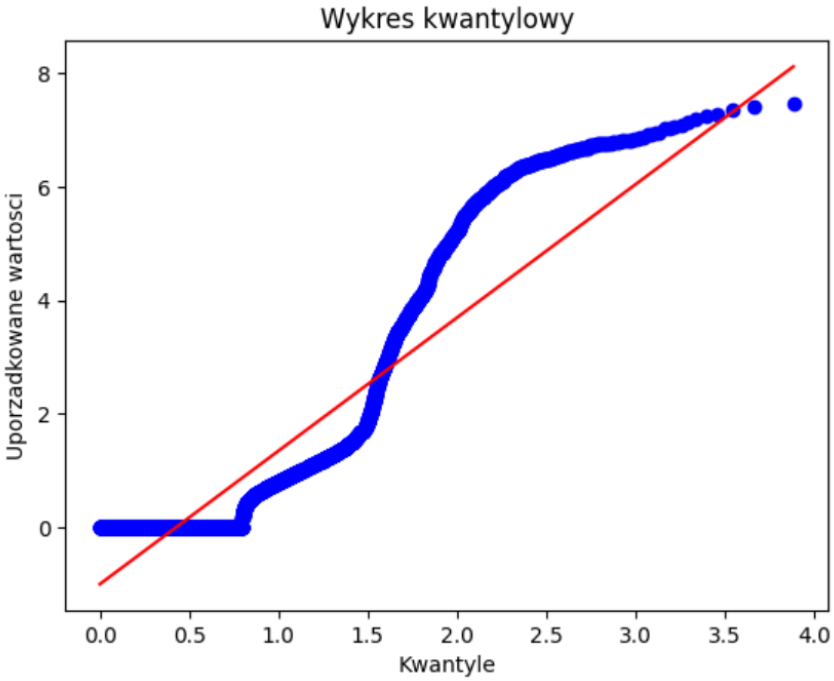
zatem szukaną zmienną jest `BLVRB`. Wygenerujmy więc wykład danej zmiennej objaśniającej

```
densityEstim = gaussian_kde(x_train["BLVRB"])
x = np.linspace(min(x_train["BLVRB"]), max(x_train["BLVRB"]), 1000)
plt.plot(x, densityEstim(x))
plt.title("Estymator gestosci")
plt.xlabel('Wartosc')
plt.ylabel('Gestosc')
plt.show()
```



oraz wykres kwantylowy wybranej zmiennej (kolor niebieski) oraz rozkładu `halfnorm` (kolor czerwony)

```
scipy.stats.probplot(x=x_train["BLVRB"], dist=scipy.stats.halfnorm
    (), plot=plt)
plt.title('Wykres kwantylowy')
plt.xlabel('Kwantyle')
plt.ylabel('Uporzadkowane wartosci')
plt.show()
```



sprawdzamy teraz czy dana zmienna ma rozkład halfnormalny. Tak będzie, jeśli po przemnożeniu jej wartości przez ± 1 otrzymamy rozkład normalny.

```
rand_signs = np.random.choice([-1,1], 6800)
z = x_train["BLVRB"] * rand_signs
out = scipy.stats.normaltest(z)
```

Otrzymujemy:

```
NormaltestResult(statistic=806.3135622776132, pvalue=8.151445130382803e-176)
```

Z otrzymanych danych nie ma podstaw do odrzucenia hipotezy zerowej.

3.3.2. Test

Użyjemy testu Kołmogorowa-Smirnowa [3], który może być również używany do sprawdzania, czy dwa jednowymiarowe rozkłady prawdopodobieństwa różnią się od siebie. W tym przypadku statystyka Kołmogorowa-Smirnowa jest określona jako

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|,$$

gdzie $F_{1,n}$ i $F_{2,m}$ są empirycznymi funkcjami rozkładu dla pierwszej i drugiej próby odpowiednio, a sup oznacza funkcję supremum.

Sprawdzamy, czy wybrana zmienna objaśniająca ma podobny rozkład w zbiorze testowym i treningowym

```
b_test = x_test["BLVRB"]
out = scipy.stats.ks_2samp(b_test, x_train["BLVRB"])
```

otrzymujemy

```
KstestResult(statistic=0.03259803921568628, pvalue=0.22370161255457072,
             statistic_location=0.2704502940177917, statistic_sign=1)
```

Wnioskujemy stąd, że należy odrzucić hipotezę zerową.

4. Elastic Net

Pierwszy model, który należy wytrenować, to ElasticNet. Podczas wykładu spotkaliśmy się z jego szczególnymi przypadkami: regresją grzbietową (ang. ridge regression) oraz lasso.

1. Wyszukaj i przedstaw w raporcie informacje o modelu ElasticNet. Opisz parametry, które są w nim estymowane, optymalizowaną funkcję oraz hiperparametry, od których ona zależy. Dla jakich wartości hiperparametrów otrzymujemy regresję grzbietową, a dla jakich lasso?

2. Zdefiniuj siatkę (ang. grid) hiperparametrów, opartą na co najmniej trzech wartościach każdego z hiperparametrów. Zadbaj o to, by w siatce znalazły się konfiguracje hiperparametrów odpowiadające regresji grzbietowej i lasso. Użyj walidacji krzyżowej do wybrania odpowiednich hiperparametrów (o liczbie podzbiorów użytych w walidacji krzyżowej należy zdecydować samodzielnie oraz uzasadnić swój wybór).

3. Narysuj wykres skrzypcowy (ang. violin plot) dla błędów średniokwadratowych, otrzymanych w poszczególnych foldach testowych walidacji krzyżowej (wartości na osi Y) dla danego zestawu wartości hiperparametrów (na osi X). Uzyskane wartości błędów średniokwadratowych w foldach powinny być również oznaczone na wykresie jako punkty. Dobrze przemyśl sposób prezentacji: użyj kolorów, umieść legendę lub wyczerpujący opis.

4. Podaj błąd treningowy i walidacyjny modelu dla wybranych wartości hiperparametrów (należy uśrednić wynik względem wszystkich podzbiorów testowych wyróżnionych w walidacji krzyżowej).

4.1. Podstawowe informacje

Model ElasticNet [4] to metoda regularyzacji w analizie regresji, która kombinuje cechy regresji grzbietowej oraz regresji Lasso. W ramach tej metody, estymowane są współczynniki regresji dla zmiennych niezależnych.

ElasticNet wprowadza dwa parametry, które są dopasowywane:

1. α (alpha) - parametr mieszania, który kontroluje proporcję między regularyzacją L1 (Lasso) a regularyzacją L2 (grzbietową). Wartości α zazwyczaj mieszczą się w przedziale od 0 do 1. Dla $\alpha = 0$, model ElasticNet sprowadza się do regresji grzbietowej, a dla $\alpha = 1$, do regresji Lasso. Wartości między 0 a 1 łączą obie formy regularyzacji.

2. λ (lambda) - parametr regularyzacji, który kontroluje siłę regularyzacji. Im większa wartość λ, tym silniejsza regularyzacja, co prowadzi do bardziej ograniczonych estymat współczynników regresji. Optymalizowana funkcja to funkcja straty, która uwzględnia zarówno regularyzację L1, jak i L2, zazwyczaj oparta na średnim błędzie kwadratowym.

Wartości hiperparametrów, które wpływają na funkcję optymalizowaną, to:

1. α (alpha) - parametr mieszania, kontrolujący proporcję między regularyzacją L1 a L2.

2. λ (lambda) - parametr regularyzacji, kontrolujący siłę regularyzacji.

Dla $\alpha = 0$ otrzymujemy regresję grzbietową, która wykorzystuje tylko regularyzację L2, a dla $\alpha = 1$ otrzymujemy regresję Lasso, korzystającą tylko z regularyzacji L1. Dla wartości α między 0 a 1, ElasticNet łączy obie formy regularyzacji, zapewniając kompromis między selekcją zmiennych (Lasso) a zachowaniem wszystkich zmiennych (grzbietowa).

4.2. Siatka hiperparametrów

Definiujemy siatkę hiperparametrów upewniając się, że do wartości parametru α należy 0 i 1.

```
# Definicja siatki hiperparametrow
alpha_values = [0., 0.25, 0.5, 0.75, 1]
l1_ratio_values = [0.1, 0.3, 0.5, 0.7, 0.9]
grid_elasticnet = itertools.product(alpha_values, l1_ratio_values)
```

oraz tworzymy foldy zbioru danych

```
# Definicja foldow
num_folds = 3
kf = KFold(n_splits=num_folds, shuffle=True)
results_elasticnet = []
```

Następnie tworzymy i trenujemy model Elasticnet

```
# Tworzenie i trenowanie modelu ElasticNet
for i, params in enumerate(grid_elasticnet):
    rmse = 0
    r2 = 0

    alpha, l1 = params
    model = ElasticNet(alpha=alpha, l1_ratio=l1)

    folds = kf.split(x_train)
    for j, (train_index, test_index) in enumerate(folds):

        # Podzial danych na zbiory treningowe i testowe
        train_fold_x, test_fold_x = x_train.iloc[train_index],
           x_train.iloc[test_index]
        train_fold_y, test_fold_y = y_train.iloc[train_index],
           y_train.iloc[test_index]

        model.fit(train_fold_x, train_fold_y)

        # Obliczanie predykcji
        train_pred = model.predict(train_fold_x)
        train_rmse = np.sqrt(mean_squared_error(train_fold_y,
           train_pred))
        train_r2 = r2 = r2_score(train_fold_y, train_pred)
        predictions = model.predict(test_fold_x)
        # Obliczanie RMSE oraz R^2
        rmse = np.sqrt(mean_squared_error(test_fold_y, predictions)
           )
        r2 = r2_score(test_fold_y, predictions)
        results_elasticnet.append([alpha, l1, j, train_rmse,
           train_r2, rmse, r2])
```

Po wytrenowaniu dla różnych hiperparametrów i użyciu poniższej operacji

```
result_df = pd.DataFrame(results_elasticnet, columns=["alpha", "l1"
   , "fold", "RMSE", "R2"])
```

otrzymujemy wyniki

	alpha	l1	fold	Train RMSE	Train R2	RMSE	R2
0	0.0	0.1	0	0.000127	1.000000	1.216151	-0.126853
1	0.0	0.1	1	0.000170	1.000000	1.190713	-0.090255
2	0.0	0.1	2	0.000189	1.000000	1.193470	-0.127919
3	0.0	0.3	0	0.000159	1.000000	1.170838	-0.049047
4	0.0	0.3	1	0.000214	1.000000	1.165899	-0.048229
...	...	...	...	...	...	...	...
70	1.0	0.7	1	0.861354	0.423328	0.872370	0.415845
71	1.0	0.7	2	0.866046	0.427808	0.847702	0.425974
72	1.0	0.9	0	0.937084	0.326993	0.927399	0.318393
73	1.0	0.9	1	0.931352	0.320080	0.947493	0.320414
74	1.0	0.9	2	0.941386	0.315172	0.938313	0.316296

4.3. Wykres skrzypcowy

Wygenerujmy teraz wykres skrzypcowy

```
sns.violinplot(x=result_df["fold"], y=result_df["RMSE"])
sns.swarmplot(x=result_df["fold"], y= result_df["R2"])
plt.title('Wykres skrzypcowy')
plt.ylim(-0.3, 1.5)
```

otrzymujemy

4.4. Błąd treningowy i walidacyjny

Dla poszczególnych parametrów podajemy błąd

```
df = result_df.groupby(["alpha", "l1"]).mean()
```

otrzymujemy wyniki

alpha	l1	fold	Train RMSE	Train R2	RMSE	R2
0.00	**0.1**	1.0	0.000162	1.000000	1.200111	-0.115009
	0.3	1.0	0.000200	1.000000	1.172824	-0.064970
	0.5	1.0	0.000172	1.000000	1.178466	-0.075886
	0.7	1.0	0.000183	1.000000	1.181939	-0.082118
	0.9	1.0	0.000233	1.000000	1.182892	-0.084770
0.25	**0.1**	1.0	0.514464	0.795078	0.526312	0.785356
	0.3	1.0	0.589163	0.731317	0.594795	0.726041
	0.5	1.0	0.631764	0.691018	0.634707	0.688034
	0.7	1.0	0.662792	0.659974	0.665267	0.657373
	0.9	1.0	0.685829	0.635882	0.687385	0.634095
0.50	**0.1**	1.0	0.565769	0.752199	0.572916	0.745812
	0.3	1.0	0.650770	0.672084	0.652946	0.669637
	0.5	1.0	0.698823	0.621952	0.699839	0.620752
	0.7	1.0	0.731312	0.585832	0.733388	0.582895
	0.9	1.0	0.763675	0.548561	0.764667	0.547329
0.75	**0.1**	1.0	0.598501	0.722674	0.602061	0.719180
	0.3	1.0	0.690593	0.630801	0.691615	0.629546
	0.5	1.0	0.743804	0.571616	0.745648	0.569122
	0.7	1.0	0.795023	0.510603	0.795741	0.509578
	0.9	1.0	0.843870	0.448651	0.845012	0.446825
1.00	**0.1**	1.0	0.623418	0.699144	0.625868	0.696744
	0.3	1.0	0.724768	0.593410	0.725927	0.592080
	0.5	1.0	0.792208	0.514199	0.792647	0.513538
	0.7	1.0	0.862982	0.423410	0.864284	0.421309
	0.9	1.0	0.936607	0.320748	0.937735	0.318368

Najmniejszy błąd otrzymaliśmy dla hiperparametrów $\alpha = 0.25$ oraz $l_1 = 0.1$ i wynosi on RMSE $= 0.525451$.

5. Lasy losowe

W tej części projektu należy wytrenować model lasów losowych.

1. Spośród wielu hiperparametrów charakteryzujących model lasów losowych wybierz trzy różne. Zdefiniuj trójwymiarową siatkę przeszukiwanych kombinacji hiperparametrów i za pomocą walidacji krzyżowej wybierz ich optymalne (w kontekście wykonywanej predykcji) wartości. Wykorzystany przy walidacji krzyżowej podział danych powinien być taki sam, jak w przypadku ElasticNet.

2. Narysuj wykres pudełkowy (ang. box plot) dla błędów średniokwadratowych, otrzymanych w poszczególnych foldach testowych walidacji krzyżowej (wartości na osi Y) dla danego zestawu wartości hiperparametrów (na osi X). Dobrze przemyśl sposób prezentacji: użyj kolorów, umieść legendę lub wyczerpujący opis.

3. Podaj błąd treningowy i walidacyjny modelu dla wybranych wartości hiperparametrów (należy uśrednić wynik względem wszystkich podzbiorów testowych wyróżnionych w walidacji krzyżowej).

5.1. Siatka kombinacji hiperparametrów

Wybierzmy następujące trzy różne hiperparametry: liczbę cech wykorzystywanych przy podziale (mtry), liczbę drzew w lesie (ntree) oraz maksymalna głębokość drzewa (max_depth). Zdefiniujmy siatkę hiperparametrów:

```
max_features = [80, 95, 105]  # odpowiednik ‘mtry‘ w R
n_estimators = [100, 200, 300]  # odpowiednik ‘ntree‘ w R
max_depth = [5, 10, 15]

grid_rf = itertools.product(max_features, n_estimators, max_depth)
```

oraz tworzymy foldy zbioru danych:

```
num_folds = 3
kf = KFold(n_splits=num_folds, shuffle=True)
results = []
```

Następnie tworzymy i trenujemy model RandomForestRegresor

```
for i, params in enumerate(grid_rf):
    rmse = 0
    r2 = 0

    max_features, n_estimators, max_depth = params
    model = RandomForestRegressor(max_features=max_features,
       n_estimators=n_estimators, max_depth=max_depth)
```

```
    folds = kf.split(x_train)
    for j, (train_index, test_index) in enumerate(folds):

        # Podzial danych na zbiory treningowe i testowe
        train_fold_x, test_fold_x = x_train.iloc[train_index],
           x_train.iloc[test_index]
        train_fold_y, test_fold_y = y_train.iloc[train_index],
           y_train.iloc[test_index]

        model.fit(train_fold_x, train_fold_y.values.ravel())

        # Obliczanie predykcji
        train_pred = model.predict(train_fold_x)
        train_rmse = np.sqrt(mean_squared_error(train_fold_y,
           train_pred))
        train_r2 = r2 = r2_score(train_fold_y, train_pred)
        predictions = model.predict(test_fold_x)
        # Obliczanie RMSE oraz R^2
        rmse = np.sqrt(mean_squared_error(test_fold_y, predictions)
           )
        r2 = r2_score(test_fold_y, predictions)

        results.append([max_features, n_estimators, max_death, j,
           train_rmse, train_r2, rmse, r2])
```

Po wytrenowaniu dla różnych hiperparametrów i posortowaniu

```
result_df = pd.DataFrame(results, columns=["max_features", "
   n_estimators", "max_depth", "fold", "Train RMSE", "Train R2", "
   RMSE", "R2"])
result_df.sort_values('RMSE')
```

otrzymujemy wyniki

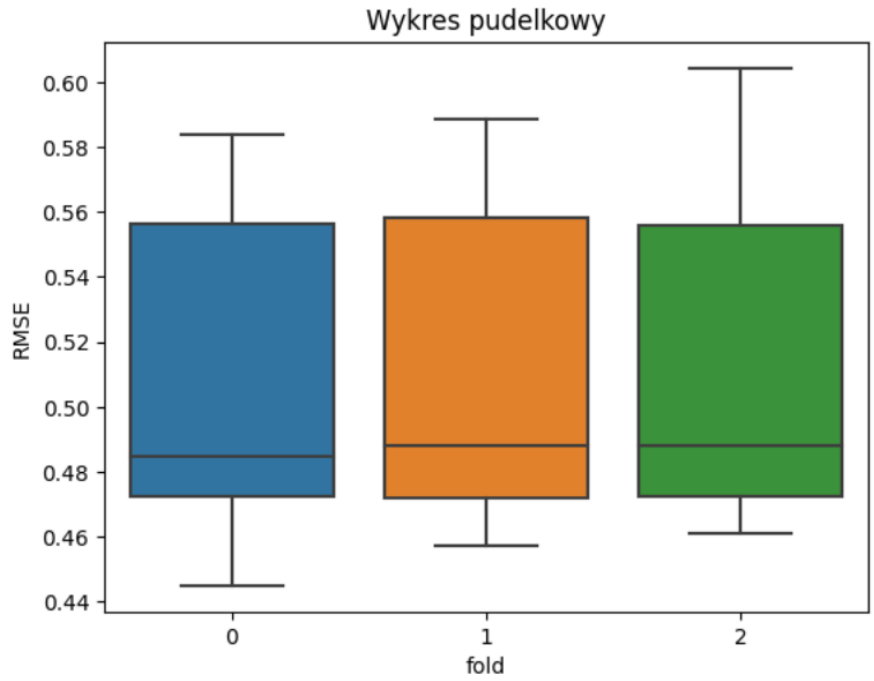
	max_features	n_estimators	max_depth	fold	Train RMSE	Train R2	RMSE	R2
69	105	200	5	0	0.249588	0.952870	0.444900	0.839206
52	95	300	5	1	0.248802	0.952927	0.457030	0.832337
15	80	200	5	0	0.255269	0.949853	0.459039	0.834992
60	105	100	5	0	0.242086	0.954944	0.459955	0.833995
70	105	200	5	1	0.246180	0.953061	0.460548	0.836045
...	...	...	...	...	...	...	...	...
37	95	200	5	1	0.517342	0.788438	0.576213	0.753018
29	95	100	5	2	0.530597	0.782485	0.582286	0.736598
18	80	300	5	0	0.537356	0.774809	0.583780	0.740096
10	80	200	5	1	0.534866	0.778371	0.588666	0.732107
2	80	100	5	2	0.533811	0.777210	0.604323	0.722849

otrzymujemy najlepsze wyniki dla max_features = 105, n_estimators = 200 oraz max_deaph = 5 i wynosi on RMSE = 0.444900.

5.2. Wykres pudełkowy

Generujemy wykres pudełkowy dla błędów średniokwadratowych

```
sns.boxplot(x=result_df["fold"], y=result_df["RMSE"])
plt.title('Wykres pudelkowy')
```



5.3. Błąd treningowy i walidacyjny

Dla poszczególnych parametrów podajemy błąd

```
df = result_df.groupby(["max_features", "n_estimators", "max_depth"
    ]).mean()
```

otrzymujemy wyniki

max_features	n_estimators	max_depth	fold	Train RMSE	Train R2	RMSE	R2
80	**100**	**5**	1.0	0.382489	0.875846	0.513330	0.794212
	200	**5**	1.0	0.381075	0.876470	0.513300	0.794375
	300	**5**	1.0	0.380213	0.876911	0.512841	0.794850
95	**100**	**5**	1.0	0.373807	0.880915	0.508731	0.798036
	200	**5**	1.0	0.370795	0.883192	0.505215	0.801078
	300	**5**	1.0	0.370225	0.883361	0.503978	0.801902
105	**100**	**5**	1.0	0.366817	0.885722	0.501673	0.803833
	200	**5**	1.0	0.366891	0.885426	0.501102	0.804227
	300	**5**	1.0	0.366889	0.885522	0.501993	0.803609

6. Podsumowanie wytrenowanych modeli

Zrób podsumowanie tabelaryczne wyników, jakie otrzymywały metody w walidacji krzyżowej w obu rozważanych powyżej modelach (tzn. ElasticNet oraz lasy losowe). Porównanie to jest powodem, dla którego zależy nam na zastosowaniu tych samych podziałów walidacji krzyżowej.

Do porównania dołącz trzeci model, referencyjny, również poddany walidacji krzyżowej, tzn. model, który dowolnym wartościom zmiennych objaśniających w foldzie testowym przypisuje średnią arytmetyczną zmiennej objaśnianej policzoną w foldach treningowych. Określ, który model wydaje Ci się najlepszy (uzasadnij swój wybór).

6.1. Model BaseLine

Dołączamy trzeci model, który dowolnym wartościom zmiennych objaśniających w foldzie testowym przypisuje średnią arytmetyczną zmiennej objaśnianej policzoną w foldach treningowych:

```
num_folds = 3
kf = KFold(n_splits=num_folds, shuffle=True)

benchmark_results = []
for train_index, test_index in kf.split(x_train):
    # Podzial danych na zbiory treningowe i testowe
    train_fold_x, test_fold_x = x_train.iloc[train_index], x_train.
        iloc[test_index]
    train_fold_y, test_fold_y = y_train.iloc[train_index], y_train.
        iloc[test_index]

    # Obliczanie sredniej wartosci zmiennej objasnianej w foldzie
        treningowym
    mean_train_y = train_fold_y.mean().values[0]

    # Przypisanie sredniej do wszystkich obserwacji w foldzie
        testowym
    train_pred = np.full_like(train_fold_y, fill_value=mean_train_y
        , dtype=np.float64)
    train_rmse = np.sqrt(mean_squared_error(train_fold_y,
        train_pred))
    train_r2 = r2 = r2_score(train_fold_y, train_pred)
    predictions = np.full_like(test_fold_y, fill_value=mean_train_y
        , dtype=np.float64)

    # Obliczanie RMSE oraz R^2
    rmse = np.sqrt(mean_squared_error(test_fold_y, predictions))
    r2 = r2_score(test_fold_y, predictions)
```

Po wytrenowaniu i posortowaniu otrzymujemy

```
    benchmark_results.append([len(benchmark_results), mean_train_y,
        rmse,r2, train_rmse, train_r2])
    benchmark_df = pd.DataFrame(benchmark_results, columns=[ "fold"
        , "mean_train_y", "RMSE", "R2", "Train RMSE", "Train R2"])
    benchmark_df.sort_values('RMSE', inplace=True)
```

```
fold            1.000000
mean_train_y    1.086168
RMSE            1.136959
R2             -0.001125
Train RMSE      1.136570
Train R2        0.000000
```

6.2. Porównanie modeli w walidacji krzyżowej

Poniższa tabela przedstawia wyniki walidacji krzyżowej dla trzech rozważanych modeli: ElasticNet, lasy losowe oraz model referencyjny. W każdym przypadku, dane zostały podzielone na $k = 3$ foldów, a wyniki dla każdego modelu były agregowane, aby uzyskać ogólną ocenę ich wydajności.

Model	RMSE	R^2	Train RMSE	Train R^2
ElasticNet	0.526312	0.785356	0.514464	0.795078
RandomForest	0.501102	0.804227	0.366891	0.885426
BaseLine	1.136856	-0.000864	1.136570	0.000000

6.3. Analiza wyników

Tabela powyżej pokazuje, że model lasów losowych osiągnął najwyższe wyniki we wszystkich foldach, a także najwyższą średnią wartość współczynnika determinacji R^2. Model ElasticNet również osiągnął dobre wyniki, ale nieco gorsze niż lasy losowe. Model referencyjny, który przypisuje średnią arytmetyczną zmiennej objaśnianej w foldach treningowych, osiągnął najniższe wyniki.

6.4. Wybór najlepszego modelu

Na podstawie przedstawionych wyników, model lasów losowych wydaje się najlepszy, ponieważ uzyskał najwyższą średnią wartość R^2 w walidacji krzyżowej. Wysokie wartości R^2 dla każdego z foldów wskazują na jego stabilność i zdolność do generalizacji. Model ElasticNet również jest dobrym wyborem, jednakże jego wyniki są nieco niższe w porównaniu do lasów losowych. Model referencyjny, choć prosty i intuicyjny, nie osiąga tak dobrych wyników jak pozostałe dwa modele.

7. Predykcja na zbiorze testowym

W oparciu o dane treningowe należy dopasować dowolnie wybrany model, a następnie zastosować go do przewidywania wartości zmiennej objaśnianej w zbiorze testowym.

Dla każdego wybranego modelu będę trenować model za pomocą walidacji krzyżowej. Następnie dla najlepszych hiperparametrów zastosujemy go do przewidywania wartości zmiennej objaśnianej w zbiorze testowym.

7.1. Lasy losowe

Spośród trzech modeli do tej pory rozpatrywanych najlepszy okazał się model RandomForest

```
model = RandomForestRegressor(max_features=105, n_estimators=200,
    max_depth=5)
model.fit(x_train, y_train.values.ravel())
y_test_restult = model.predict(x_test)
```

Wówczas najlepszym wynikiem jest

0.51315

7.2. Gradient Boosting Machine (GBM)

Wykorzystamy model GBM [5] Potrzebujemy nowej biblioteki

```
from sklearn.ensemble import GradientBoostingRegressor
```

7.2.1. Trenowanie modelu

Trenujemy model Gradient Boosting Machine

```
# Definiowanie hiperparametrow
learning_rate = [0.01, 0.1, 0.2]
n_estimators = [100, 200, 300]
max_depth = [3, 5, 7]

# Tworzenie siatki przeszukiwanych kombinacji hiperparametrow
grid_gbm = itertools.product(learning_rate, n_estimators, max_depth
    )

num_folds = 3
kf = KFold(n_splits=num_folds, shuffle=True)

results = []
```

```
# Resetowanie iteratora
grid_gbm = itertools.product(learning_rate, n_estimators, max_depth)

for i, params in enumerate(grid_gbm):
    lr, n_estimators, max_depth = params
    model = GradientBoostingRegressor(learning_rate=lr,
        n_estimators=n_estimators, max_depth=max_depth)

    rmse_list = []
    r2_list = []

    folds = kf.split(x_train)
    for j, (train_index, test_index) in enumerate(folds):
        # Podzial danych na zbiory treningowe i testowe
        train_fold_x, test_fold_x = x_train.iloc[train_index],
            x_train.iloc[test_index]
        train_fold_y, test_fold_y = y_train.iloc[train_index],
            y_train.iloc[test_index]

        model.fit(train_fold_x, train_fold_y.values.ravel())

        # Obliczanie predykcji
        train_pred = model.predict(train_fold_x)
        train_rmse = np.sqrt(mean_squared_error(train_fold_y,
            train_pred))
        train_r2 = r2_score(train_fold_y, train_pred)

        predictions = model.predict(test_fold_x)
        # Obliczanie RMSE oraz R^2
        rmse = np.sqrt(mean_squared_error(test_fold_y, predictions)
            )
        r2 = r2_score(test_fold_y, predictions)

        rmse_list.append(rmse)
        r2_list.append(r2)

        results.append([lr, n_estimators, max_depth, j, train_rmse,
            train_r2, rmse, r2])
```

Po obróbce

```
# Przeksztalcenie wynikow w DataFrame
results_df = pd.DataFrame(results, columns=['Learning Rate', 'N
    Estimators', 'Max Depth', 'Fold', 'Train RMSE', 'Train R2', '
    Test RMSE', 'Test R2'])

# Obliczanie srednich wynikow dla kazdej kombinacji hiperparametrow
mean_results = results_df.groupby(['Learning Rate', 'N Estimators',
    'Max Depth']).mean().reset_index()
```

i posortowaniu otrzymujemy

```
# Sortowanie wynikow wedlug Test RMSE
sorted_results = mean_results.sort_values(by='Test RMSE')
```

otrzymujemy

	Learning Rate	N Estimators	Max Depth	Fold	Train RMSE	Train R2	Test RMSE	Test R2
13	0.10	200	5	1.0	0.131590	0.986586	0.414829	0.866658
10	0.10	100	5	1.0	0.186795	0.972991	0.418157	0.864552
14	0.10	200	7	1.0	0.046520	0.998322	0.419052	0.863987
11	0.10	100	7	1.0	0.083841	0.994559	0.421447	0.862497
12	0.10	200	3	1.0	0.262002	0.946864	0.422291	0.861895
9	0.10	100	3	1.0	0.319744	0.920822	0.429341	0.857118
7	0.01	300	5	1.0	0.293366	0.933366	0.432340	0.855278
8	0.01	300	7	1.0	0.187931	0.972647	0.437593	0.851674
4	0.01	200	5	1.0	0.365852	0.896367	0.464609	0.832798
5	0.01	200	7	1.0	0.273885	0.941929	0.467428	0.830845
6	0.01	300	3	1.0	0.422561	0.861781	0.469210	0.829533
3	0.01	200	3	1.0	0.483478	0.819010	0.515954	0.793870
2	0.01	100	7	1.0	0.501397	0.805409	0.599670	0.721580
1	0.01	100	5	1.0	0.561965	0.755516	0.610152	0.711643
0	0.01	100	3	1.0	0.637830	0.685068	0.652633	0.670243

Rysunek 7.1: Przerwanie obliczenia dla lr = 0.1, n_estimaotrs = 300

Najlepszą wartość otrzymujemy dla hiperparametrów lr = 0.1, n_estimaotrs = 200 oraz , max_depth = 5 i wynosi ona

```
0.22442
```

7.3. Model XGBoost

Wykorzystamy model XGBoost [6] Potrzebujemy nowej biblioteki

```
from xgboost import XGBRegressor
```

7.3.1. Trenowanie modelu

```
# Definicja hiperparametrow
max_depth = [5, 10, 15]
n_estimators = [100, 200, 300]
learning_rate = [0.01]

# Generowanie siatki przeszukiwanych kombinacji hiperparametrow
grid_xgb = itertools.product(max_depth, n_estimators, learning_rate
    )
num_folds = 3
kf = KFold(n_splits=num_folds, shuffle=True)

results = []
```

```
for i, params in enumerate(grid_xgb):
    max_depth, n_estimators, learning_rate = params
    rmse = 0
    r2 = 0

    model = XGBRegressor(max_depth=max_depth, n_estimators=
       n_estimators, learning_rate=learning_rate, verbosity=0)
    rmse_list = []
    r2_list = []

    folds = kf.split(x_train)
    for j, (train_index, test_index) in enumerate(folds):
        # Podzial danych na zbiory treningowe i testowe
        train_fold_x, test_fold_x = x_train.iloc[train_index],
           x_train.iloc[test_index]
        train_fold_y, test_fold_y = y_train.iloc[train_index],
           y_train.iloc[test_index]

        model.fit(train_fold_x, train_fold_y.values.ravel())

        # Obliczanie predykcji
        train_pred = model.predict(train_fold_x)
        train_rmse = np.sqrt(mean_squared_error(train_fold_y,
           train_pred))
        train_r2 = r2_score(train_fold_y, train_pred)

        predictions = model.predict(test_fold_x)
        # Obliczanie RMSE oraz R^2
        rmse = np.sqrt(mean_squared_error(test_fold_y, predictions)
           )
        r2 = r2_score(test_fold_y, predictions)

        rmse_list.append(rmse)
        r2_list.append(r2)

        results.append([learning_rate, n_estimators, max_depth, j,
            rmse, r2])
```

Po obróbce

```
# Przeksztalcenie wynikow w DataFrame
results_df = pd.DataFrame(results, columns=['Learning Rate', 'N
   Estimators', 'Max Depth', 'Fold',  'Test RMSE', 'Test R2'])

# Obliczanie srednich wynikow dla kazdej kombinacji hiperparametrow
mean_results = results_df.groupby(['Learning Rate', 'N Estimators',
    'Max Depth']).mean().reset_index()
```

i posortowaniu otrzymujemy

```
# Sortowanie wynikow wedlug Test RMSE
sorted_results = mean_results.sort_values(by='Test RMSE')
```

otrzymujemy

	Learning Rate	N Estimators	Max Depth	Fold	Test RMSE	Test R2
4	0.01	300	5	1.0	0.434543	0.853660
2	0.01	200	5	1.0	0.471963	0.827550
3	0.01	200	10	1.0	0.484981	0.817938
1	0.01	100	10	1.0	0.615164	0.706990
0	0.01	100	5	1.0	0.616900	0.705288

Rysunek 7.2: Przerwanie obliczenia dla max_depth = 10, n_estimaotrs = 300

oraz

	Learning Rate	N Estimators	Max Depth	Fold	Test RMSE	Test R2
4	0.1	300	5	1.0	0.413040	0.867802
2	0.1	200	5	1.0	0.417745	0.864876
0	0.1	100	5	1.0	0.418291	0.864239
5	0.1	300	10	1.0	0.445086	0.846233
3	0.1	200	10	1.0	0.445117	0.846402
1	0.1	100	10	1.0	0.453087	0.840940

Rysunek 7.3: Przerwanie obliczenia dla max_depth = 15

Najlepszą wartość otrzymujemy dla hiperparametrów lr = 0.1, n_estimaotrs = 200 oraz , max_depth = 5 i wynosi ona

```
0.20560
```

Bibliografia

[1] Błażej Miasojedow. Polecenie do zadania: 'statystyczna analiza danych - projekt zaliczeniowy 2024'.

[2] SciPy Developers. scipy.stats.normaltest, 2024. Accessed: 2024-06-02.

[3] SciPy Developers. scipy.stats.ks_2samp, 2024. Accessed: 2024-06-02.

[4] Hui Zou and Trevor Hastie. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67(2):301–320, 03 2005.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree, 2017.

[6] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system, 2016.