Politechnika Rzeszowska
Katedra Informatyki i Automatyki

Sprawozdanie z przedmiotu
Analiza danych w językach R i Python

Prowadzący:
dr inż. Bartosz Jędrzejec

Krzysztof Kołodziej, L01,
169562@stud.prz.edu.pl
Jakub Duda, L01,
169532@stud.prz.edu.pl

Spis treści

1. Opis problematyki klasyfikacji danych

1.1 Wstęp

Klasyfikacja danych jest kluczowym elementem eksploracji danych, czyli procesu automatycznego odkrywania nieznanych wcześniej zależności i wzorców w zbiorach danych. Proces ten pozwala na przekształcenie surowych danych w wartościowe informacje, które mogą wspierać podejmowanie decyzji w różnych dziedzinach. Informacje uzyskane dzięki klasyfikacji mogą być wykorzystywane m.in. do przewidywania zachowań czy generowania raportów. [2]

Klasyfikację można podzielić na: [2]

- **Klasyfikacja dwuklasowa**: Przypisuje obiekt do jednej z dwóch dostępnych klas (np. "ssak" lub "nie ssak").
- **Klasyfikacja wieloklasowa**: Przyporządkowuje obiekt do jednej z wielu klas (np. rozpoznanie gatunku zwierzęcia).

1.2 Etapy procesu klasyfikacji

Proces klasyfikacji można podzielić na dwa główne etapy. Pierwszy to **etap uczenia się**, w którym algorytm analizuje dane treningowe, aby nauczyć się wzorców i reguł, które później będą wykorzystywane do przypisywania nowych danych do odpowiednich klas. W tym etapie model buduje swoją wiedzę na podstawie dostarczonych informacji. Drugi to **etap predykcji**, w którym model, opierając się na wyuczonych wzorcach, dokonuje klasyfikacji nowych danych, przypisując je do odpowiednich kategorii. [2]

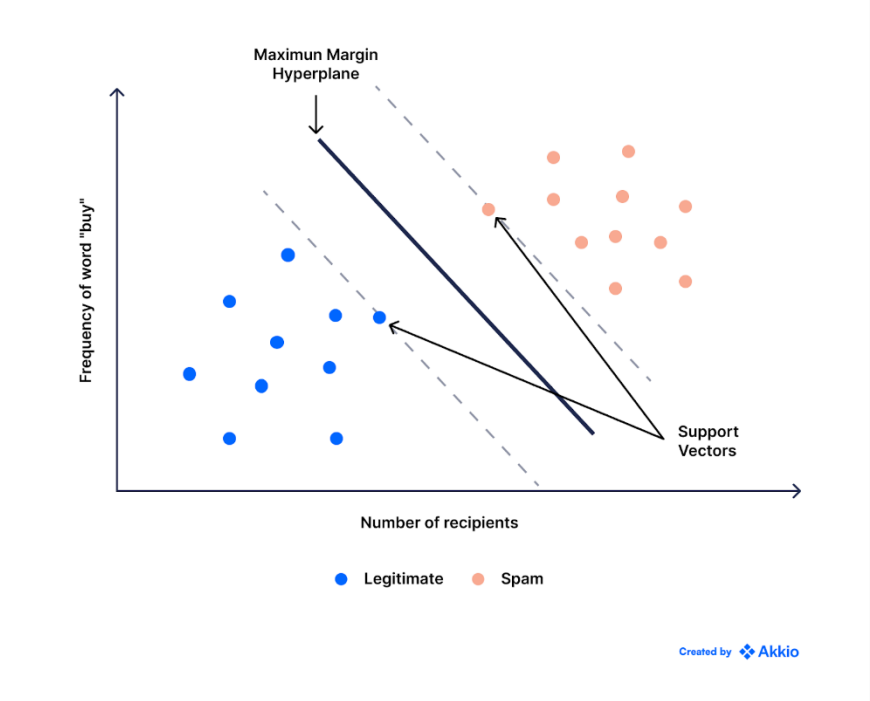
Klasyfikację można zaliczyć do metod **nadzorowanych**, ponieważ algorytm uczy się na wcześniej oznaczonych danych. Oznacza to, że każdy obiekt w zbiorze treningowym ma już przypisaną klasę, a zadaniem modelu jest nauczyć się, jak na podstawie cech obiektu przypisać go do odpowiedniej kategorii. W fazie uczenia algorytm otrzymuje dane wejściowe wraz z prawidłowymi wynikami (czyli z etykietami klas), zakładając, że są one prawidłowe, co pozwala mu wyciągać wnioski dotyczące przyszłych klasyfikacji. [2]

Przykładami algorytmów stosowanych w klasyfikacji nadzorowanej są:

- **Drzewa decyzyjne**: Bardzo powszechny sposób przedstawiania i wizualizacji możliwych wyników decyzji lub działania na podstawie prawdopodobieństw. Każde drzewo jest hierarchiczną reprezentacją przestrzeni wyników, w której każdy węzeł reprezentuje działanie lub wybór, a liście reprezentują stany wyniku. Są łatwe w zaimplementowanie i szczególnie przydatne w sytuacjach, gdy istotna jest przejrzystość procesu decyzyjnego oraz łatwość interpretacji wyników [1]

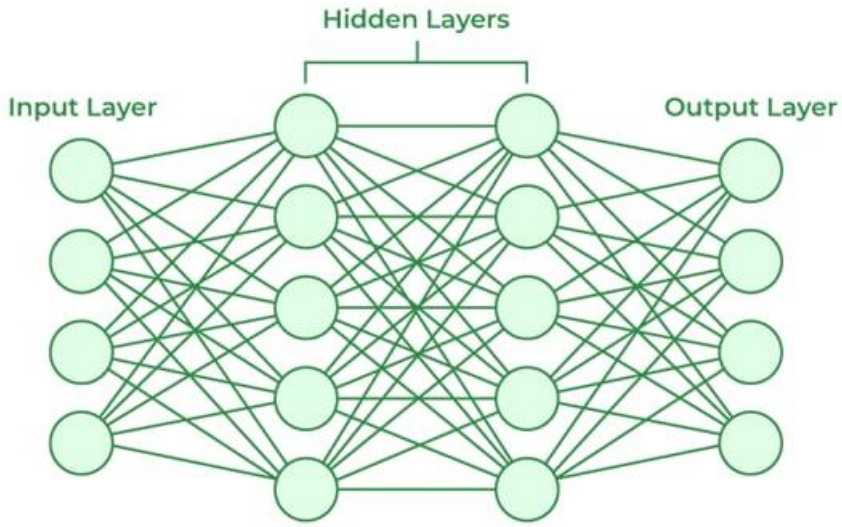
Rysunek 1 - algorytm drzewa decyzyjnego [1]

- **Maszyna wektorów nośnych (SVM)**: Algorytm, który tworzy granicę decyzyjną między klasami, tak aby maksymalizować odległość między najbliższymi przykładami różnych klas. Jest szczególnie dobry w oddzielaniu podobnych danych. Jest ceniony za skuteczność w sytuacjach i stosowany w sytuacjach, gdy dane są trudne do rozdzielenia, ponieważ tworzy hiperplan maksymalnie oddzielający różne klasy, co zmniejsza ryzyko błędnej klasyfikacji. [1]



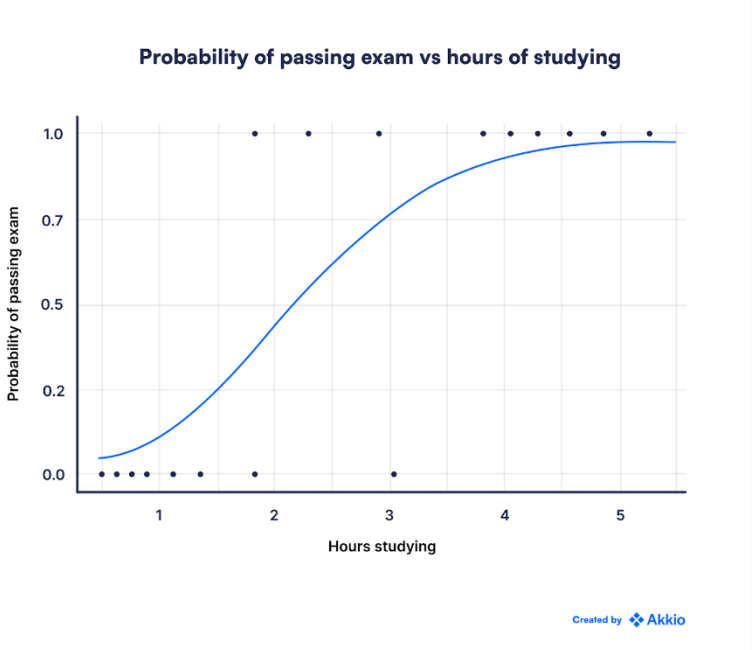
Rysunek 1 - algorytm SVM [1]

- **Sztuczne sieci neuronowe**: sama nazwa wskazuje, emulują biologiczne sieci neuronowe w komputerach. Sieci neuronowe działają poprzez trenowanie zestawu parametrów na danych. Parametry te są następnie używane do określania wyników modelu. Sieci mogą być jednokierunkowe lub rekurencyjne w których występuje sprzężenie zwrotne. Sztuczne sieci neuronowe mają szerokie zastosowanie w wielu dziedzinach, zwłaszcza tam, gdzie występuje duża złożoność danych lub trudność w wykrywaniu wzorców. Dzięki swojej zdolności do przetwarzania dużych ilości danych i "uczenia się" z nich, sieci neuronowe znajdują zastosowanie w takich obszarach, jak: przetwarzanie języka naturalnego, rozpoznawanie obrazów i wideo itp. [1]



Rysunek 2 - sztuczne sieci neuronowe [3]

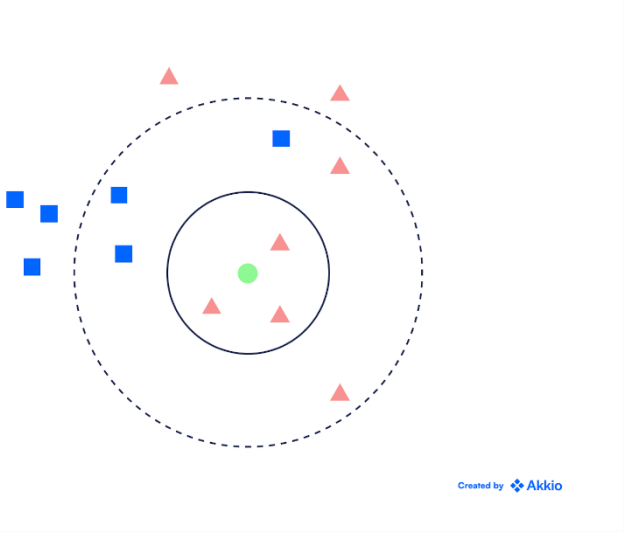
- **Regresja logistyczna**: W odróżnieniu od regresji liniowej, która prognozuje wartości ciągłe, regresja logistyczna stosuje funkcję sigmoidalną (logistyczną), aby wyniki mieściły się w zakresie od 0 do 1, czyli odzwierciedlały prawdopodobieństwo wystąpienia danej klasy. Jest często stosowana w klasyfikacji binarnej, np. do diagnozy chorób (zdrowy vs chory), klasyfikacji wiadomości (spam vs nie-spam) czy przewidywania wyników (sukces vs porażka). [1]



Rysunek 3 - regresja logistyczna [1]

- **K-Najbliższych sąsiadów (KNN)**: Jego zasada działania opiera się na założeniu, że obiekty o podobnych cechach będą należeć do podobnych klas lub mieć zbliżone wartości. W klasyfikacji KNN, aby przewidzieć klasę nowego punktu, algorytm wybiera k najbliższych sąsiadów tego punktu (na podstawie metryki odległości, np. euklidesowej) i przypisuje mu klasę, która występuje najczęściej w tej grupie sąsiadów. Algorytm KNN znajduje szerokie zastosowanie w różnych dziedzinach, szczególnie tam, gdzie ważne jest

dopasowanie nowych danych do wzorców na podstawie istniejących danych historycznych np. systemy rekomendacji, rozpoznawanie obrazów i wideo itd. [1]



Rysunek 4 - algorytm K-Najbliższych sąsiadów [1]

- **Naiwny klasyfikator Bayesa**: Algorytm probabilistyczny, który zakłada niezależność cech. Klasyfikatory Naiwnego Bayesa są często używane w klasyfikacji tekstów, ponieważ łatwo jest obliczyć prawdopodobieństwo na podstawie częstości, a tekst zwykle ma dużą liczbę cech (np. pojedyncze tokeny w słowach). [1]

Metody te wykorzystują wiedzę z fazy uczenia, aby skutecznie dokonywać predykcji w fazie wyznaczania wyniku, przypisując nowe dane do właściwych klas.

1.3 Metody stosowane do poprawności wiarygodności i dokładności budowanych modeli

Istnieje kilka kluczowych metod stosowanych w uczeniu maszynowym do poprawy wiarygodności i dokładności modeli. Te techniki pozwalają na ocenę skuteczności modelu i minimalizację błędów związanych z przeuczeniem lub niedouczeniem. Oto kilka z nich:

- **Walidacja krzyżowa z pozostawieniem jednej wartości (LOOCV)**: W tej metodzie przeprowadzamy trening na całym zestawie danych, ale pozostawiamy tylko jeden punkt danych z dostępnego zestawu danych, a następnie iterujemy dla każdego punktu danych. W LOOCV model jest trenowany na $N-1$ próbki i testowane na jednej pominiętej próbce, powtarzając ten proces dla każdego punktu danych w zestawie danych. Wykorzystujemy wszystkie punkty danych, co sprawia, że jest ona mało obciążona błędem. Testujemy jednocześnie tylko jeden punkt danych. Jeśli punkt danych jest wartością odstającą, może to prowadzić do większej zmienności. Metoda ta zajmuje również dużo czasu wykonania, ponieważ iteruje się przez „liczbę punktów danych” razy. [4]
- **Walidacja krzyżowa K-Fold:** Polega na dzieleniu zbioru danych na k podzbiorów (znanych jako folds), a następnie przeprowadzeniu treningu na wszystkich podzbiorach, pozostawiając jeden (k-1) podzbiór do oceny wytrenowanego modelu. Jej zaletą jest możliwość uzyskania bardziej wiarygodnych i stabilnych wyników oceny modelu, zwłaszcza na ograniczonym zbiorze danych [4]
- **Stratyfikowana walidacja krzyżowa:** Jest to technika stosowana w uczeniu maszynowym, aby zapewnić, że każdy etap procesu walidacji krzyżowej zachowuje ten sam rozkład klas, co cały zestaw danych. Jest to szczególnie ważne w przypadku niezrównoważonych zestawów danych, w których pewne klasy mogą być niedoreprezentowane. W tej metodzie:
    1. Zbiór danych podzielony jest na k części, przy zachowaniu proporcji klas w każdej części.
    2. Podczas każdej iteracji jeden z elementów jest używany do testowania, a pozostałe do trenowania.

3. Proces powtarza się k razy, przy czym każdy podzbiór służy jako zbiór testowy dokładnie raz.

Stratyfikowana walidacja krzyżowa jest niezbędna w przypadku problemów klasyfikacji, w których zachowanie równowagi rozkładu klas ma kluczowe znaczenie dla prawidłowego uogólniania modelu na niewidziane wcześniej dane. [4]

- **Walidacja holdout**: W tej metodzie przeprowadzamy trening na 50% danego zestawu danych, a pozostałe 50% jest używane do celów testowych. To prosty i szybki sposób na ocenę modelu. Główną wadą tej metody jest to, że przeprowadzamy trening na 50% zestawu danych, może się zdarzyć, że pozostałe 50% danych zawiera pewne ważne informacje, które pomijamy podczas trenowania naszego modelu. [4]

2. Charakterystyka danych

2.1 Źródło i typ danych

Zbiór danych **Red Wine Quality Dataset** (Cortez et al., 2009) pochodzi z UCI Machine Learning Repository i jest używany do analiz w kontekście oceny jakości win czerwonych. Zawiera dane na temat fizykochemicznych właściwości win oraz ich ocenę jakościową [5].

Kolumny w zbiorze danych [5]:

1. **Kwasowość stała** (pH) w winie (liczba zmiennoprzecinkowa, np. 7.4)
2. **Kwasowość lotna** (pH) w winie (liczba zmiennoprzecinkowa, np. 0.7)
3. **Kwas cytrynowy** (w gramach na litr) (liczba zmiennoprzecinkowa, np. 0.0)
4. **Cukier resztkowy** (w gramach na litr) (liczba zmiennoprzecinkowa, np. 1.9)
5. **Zawartość chlorków** (w gramach na litr) (liczba zmiennoprzecinkowa, np. 0.076)
6. **Wolny dwutlenek siarki** (w miligramach na litr) (liczba zmiennoprzecinkowa, np. 11)
7. **Całkowity dwutlenek siarki** (w miligramach na litr) (liczba zmiennoprzecinkowa, np. 34)
8. **Gęstość** wina (liczba zmiennoprzecinkowa, np. 0.9978)
9. **pH** wina (liczba zmiennoprzecinkowa, np. 3.51)
10. **Zawartość siarczanów** (w gramach na litr) (liczba zmiennoprzecinkowa, np. 0.56)
11. **Zawartość alkoholu** (w procentach) (liczba zmiennoprzecinkowa, np. 9.4)
12. **Quality** – **ocena jakości wina** (skala od 0 do 10, liczba całkowita)

2.2 Informacje charakteryzujące dane

Ten zbiór danych zawiera 1599 rekordów oraz 12 kolumn. Wszystkie kolumny poza kolumną Quality zawierają informacje typu zmiennoprzecinkowego. Kolumna Quality zawiera dane typu całkowitego.

Za pomocą następującego programu wyznaczamy wartości minimalne, maksymalne, średnią, odchylenie standardowe oraz kwartyle [6].

```
import pandas as pd

#załadowanie danych
df = pd.read_csv('winequality-red.csv')
#generowanie podstawowych statystyk opisowych
statistics = df.describe()
# Zaokrąglenie statystyk do trzech miejsc po przecinku
statistics = statistics.round(3)
# Wyświetlenie wyników
print(statistics)
```

Index	Kwasowość stała	Kwasowość lotna	Kwas cytrynowy	Cukier resztkowy	Zawartość chlorków	Wolny dwutlenek siarki
Liczba	1599	1599	1599	1599	1599	1599
Max	15.9	1.58	1	15.5	0.611	72
75%	9.2	0.64	0.42	2.6	0.09	21
Średnia	8.32	0.528	0.271	2.539	0.087	15.875
50%	7.9	0.52	0.26	2.2	0.079	14
25%	7.1	0.39	0.09	1.9	0.07	7
Min	4.6	0.12	0	0.9	0.012	1
std	1.741	0.179	0.195	1.41	0.047	10.46

Rysunek 5 Informacje charakteryzujące dane

Index	Całkowity dwutlenek siarki	Gęstość	pH	Siarczany	Alkohol	Ocena Jakości
Liczba	1599	1599	1599	1599	1599	1599
max	289	1.004	4.01	2	14.9	8
75%	62	0.998	3.4	0.73	11.1	6
Średnia	46.468	0.997	3.311	0.658	10.423	5.636
50%	38	0.997	3.31	0.62	10.2	6
std	32.895	0.002	0.154	0.17	1.066	0.808
25%	22	0.996	3.21	0.55	9.5	5
min	6	0.99	2.74	0.33	8.4	3

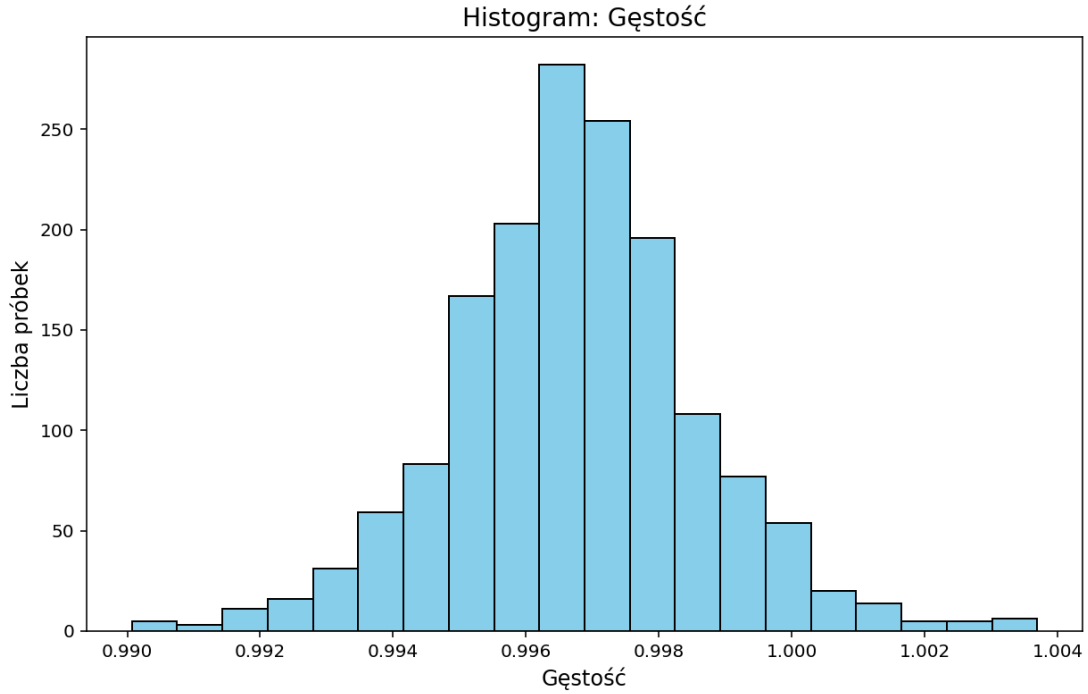
Rysunek 6 Informacje charakteryzujące dane cd.

Za pomocą następującego programu wyznaczymy rozkład wartości każdej z danych [6,7].

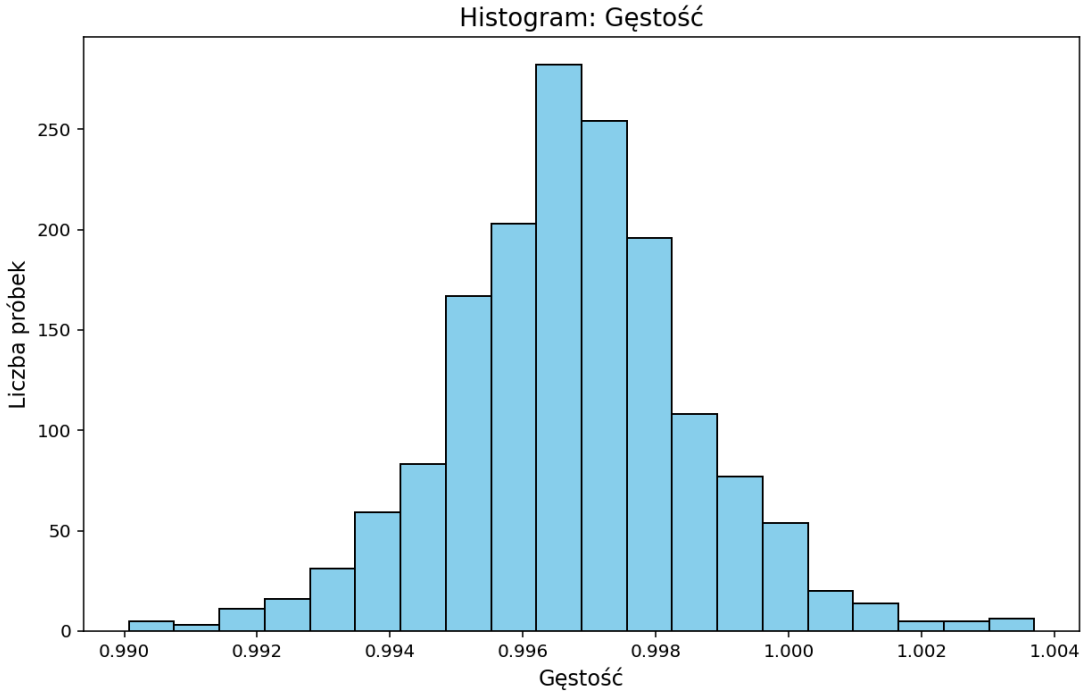
```
import pandas as pd
import matplotlib.pyplot as plt

# Załadowanie danych
df = pd.read_csv('winequality-red.csv')
# Mapowanie nazw kolumn na polski
column_names_pl = {
    'fixed acidity': 'Kwasowość stała',
    'volatile acidity': 'Kwasowość lotna',
    'citric acid': 'Kwas cytrynowy',
    'residual sugar': 'Cukier resztkowy',
    'chlorides': 'Chlorki',
    'free sulfur dioxide': 'Wolny dwutlenek siarki',
    'total sulfur dioxide': 'Całkowity dwutlenek siarki',
```

```
    'density': 'Gęstość',
    'pH': 'pH',
    'sulphates': 'Siarka',
    'alcohol': 'Alkohol',
    'quality': 'Jakość'
}

# Generowanie histogramów dla każdej kolumny w zbiorze danych
for column in df.columns:
    # Utworzenie wykresu
    plt.figure(figsize=(10, 6))
    plt.hist(df[column], bins=20, color='skyblue', edgecolor='black')
    # Ustawienie tytułu wykresu i etykiet
    # Użycie polskiego tłumaczenia lub oryginalnej nazwy
    column_name_pl = column_names_pl.get(column, column)
    plt.title(f'Histogram: {column_name_pl}', fontsize=14)
    plt.xlabel(column_name_pl, fontsize=12)
    plt.ylabel('Liczba próbek', fontsize=12)
    # Zapisanie wykresu do pliku
    filename = f'{column}_histogram.png'
    plt.savefig(filename)  # Zapisanie wykresu w formacie PNG
    plt.close()  # Zamknięcie wykresu po zapisaniu, aby przygotować nowy

print("Wszystkie wykresy zostały zapisane.")
```

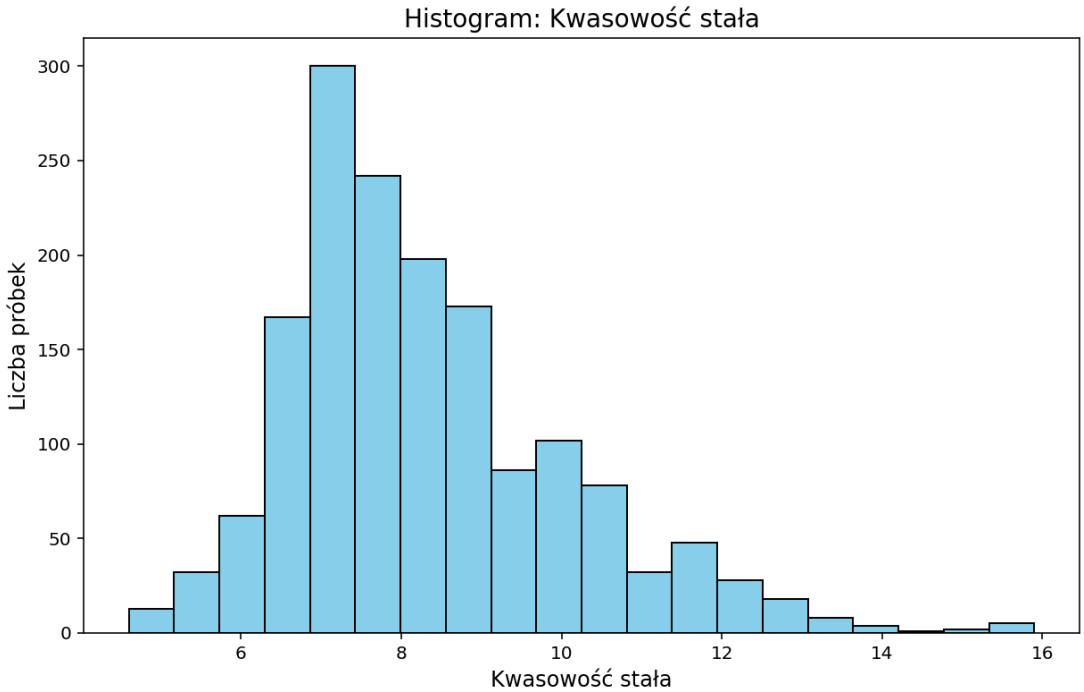
Na następnych zdjęciach zostaną przedstawione histogramy rozkładu wartości poszczególnych cech danych.



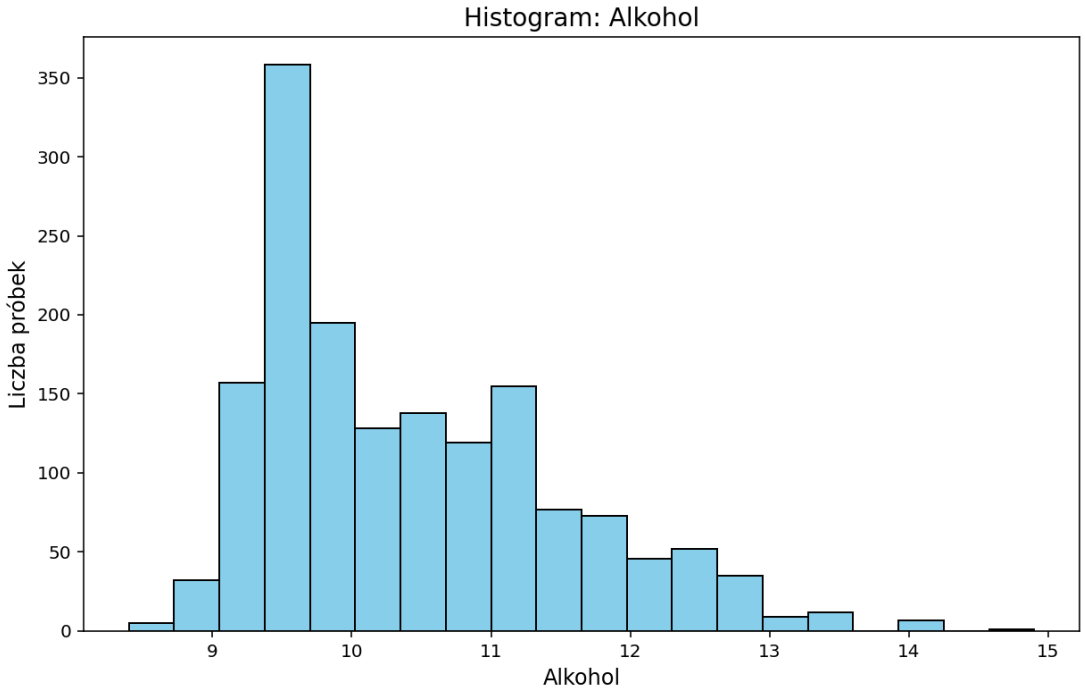
Rysunek 7 Histogram rozkładu danych gęstości win



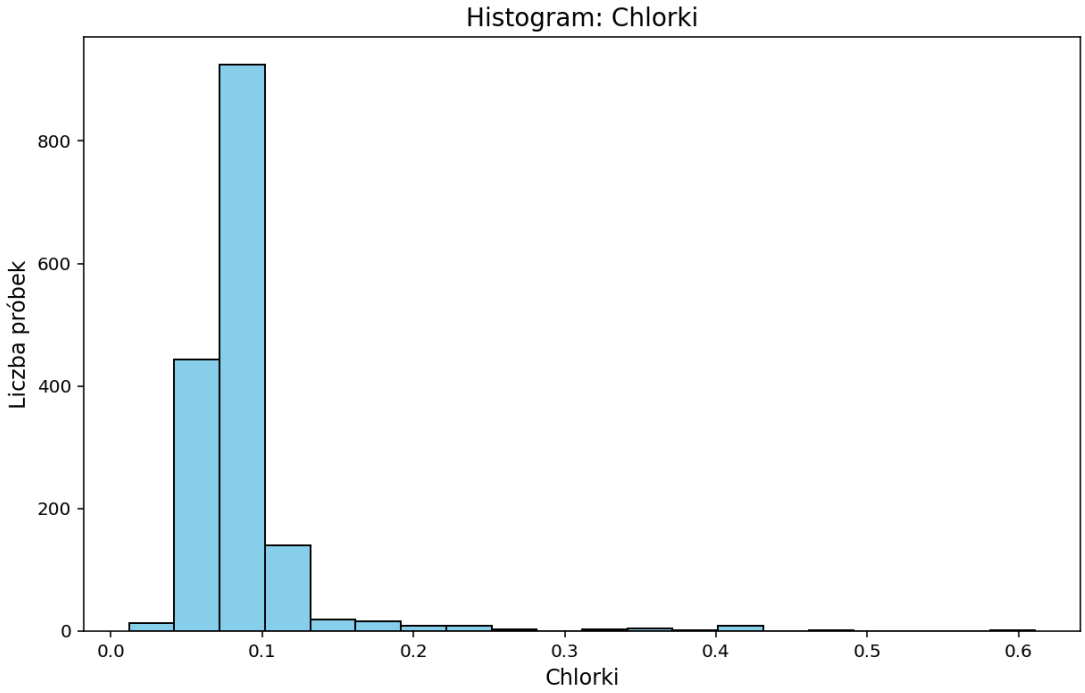
Rysunek 8 Histogram rozkładu danych kwasowości ulotnej



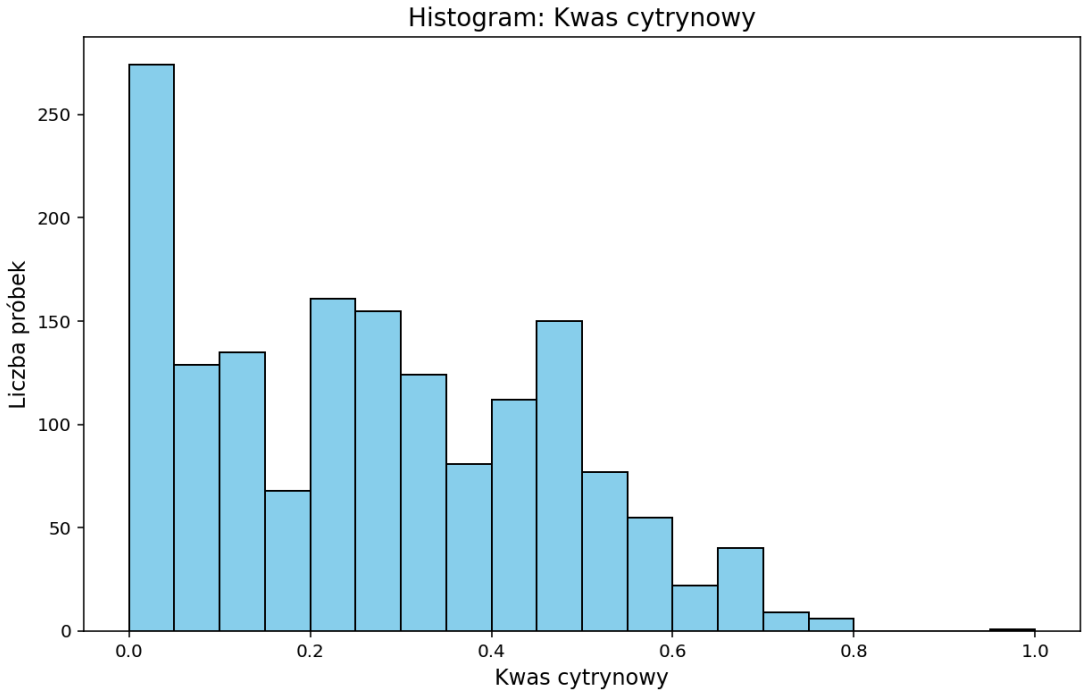
Rysunek 9 Histogram rozkładu danych kwasowości stałej



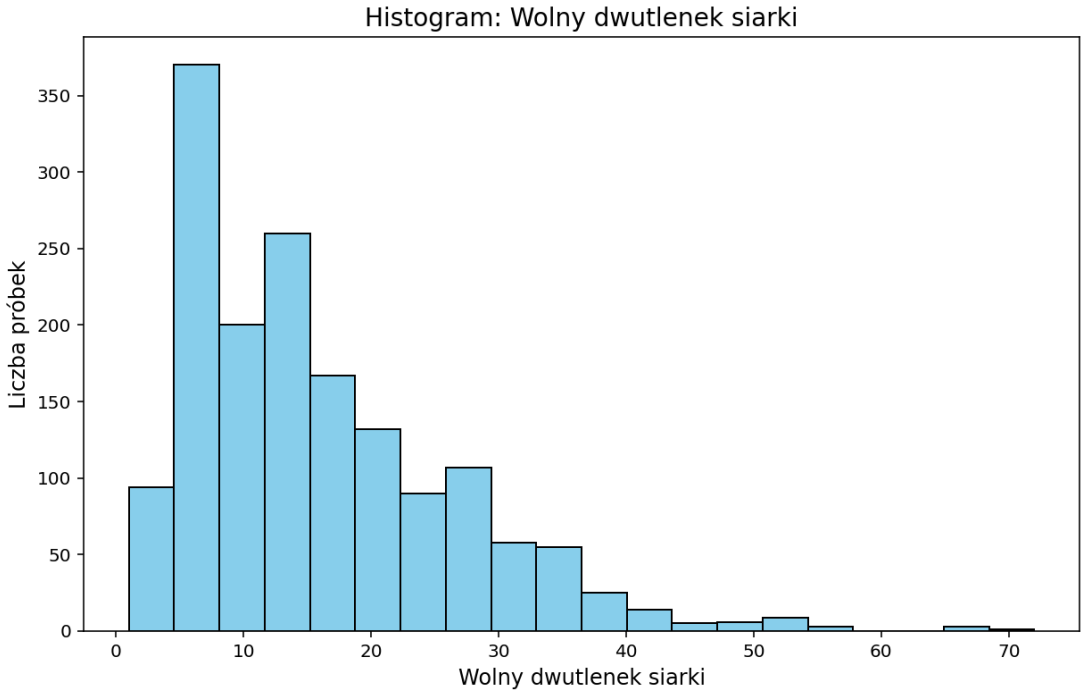
Rysunek 10 Histogram rozkładu zawartości alkoholu w winie



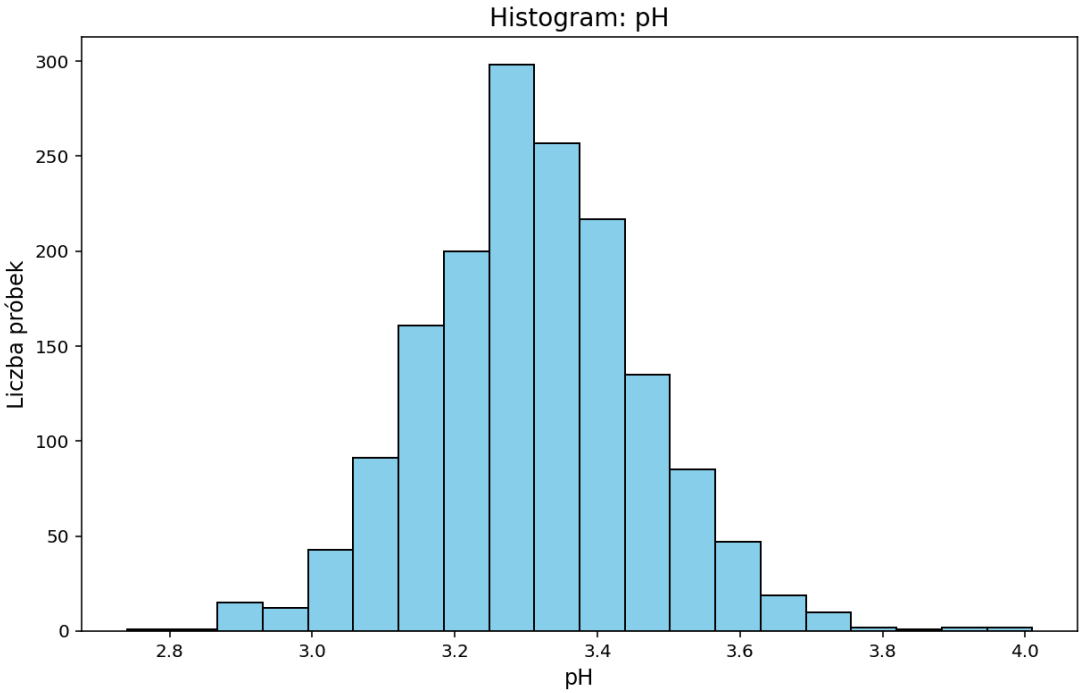
Rysunek 11 Histogram rozkładu zawartości chlorków



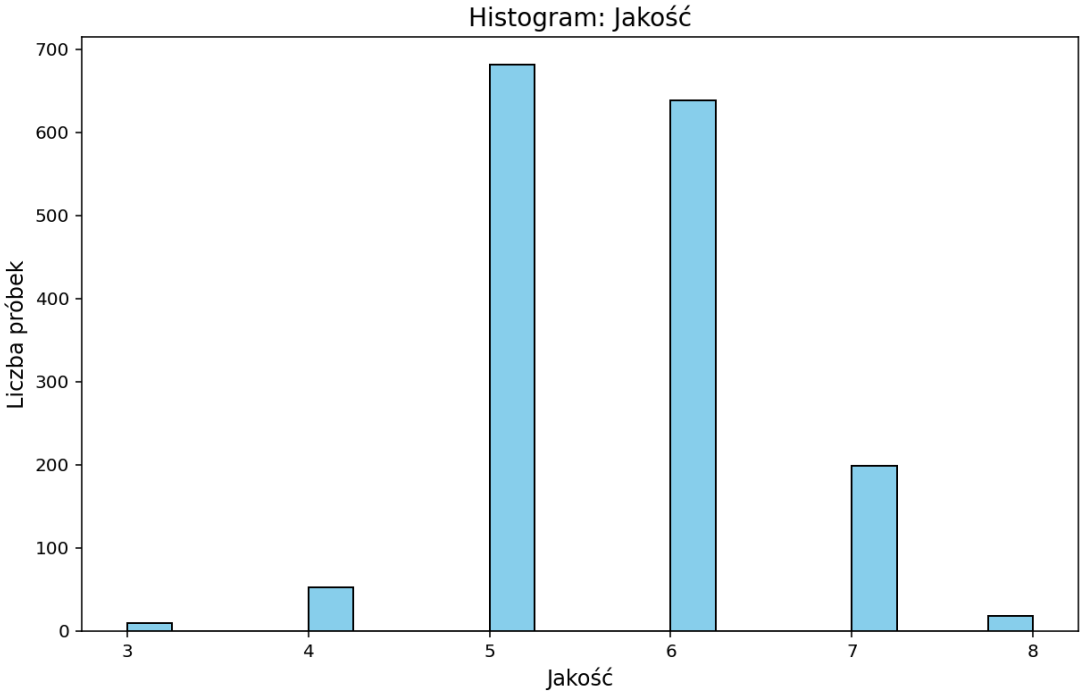
Rysunek 12 Histogram rozkładu zawartości kwasu cytrynowego



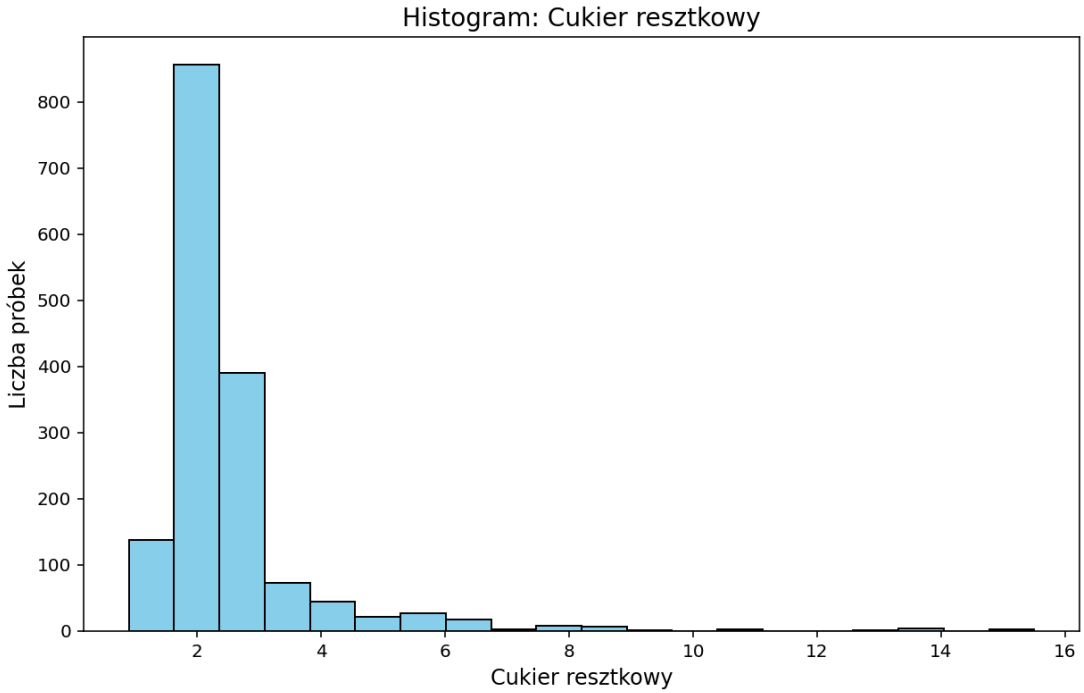
Rysunek 13 Histogram rozkładu zawartości wolnego dwutlenku siarki



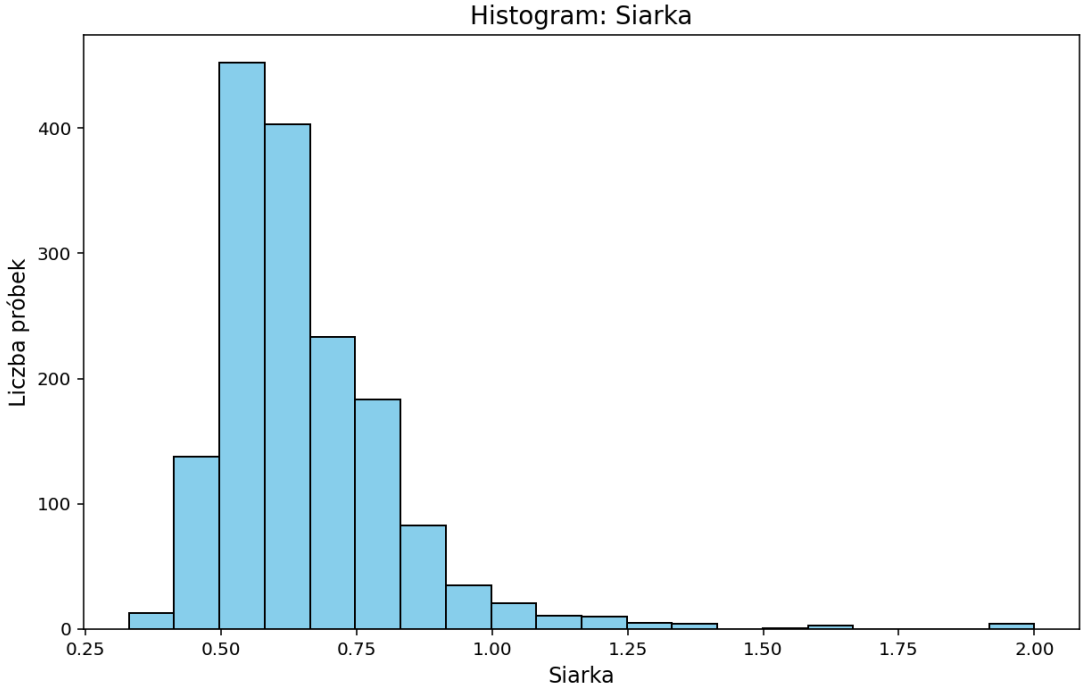
Rysunek 14 Histogram poziomu pH w winie



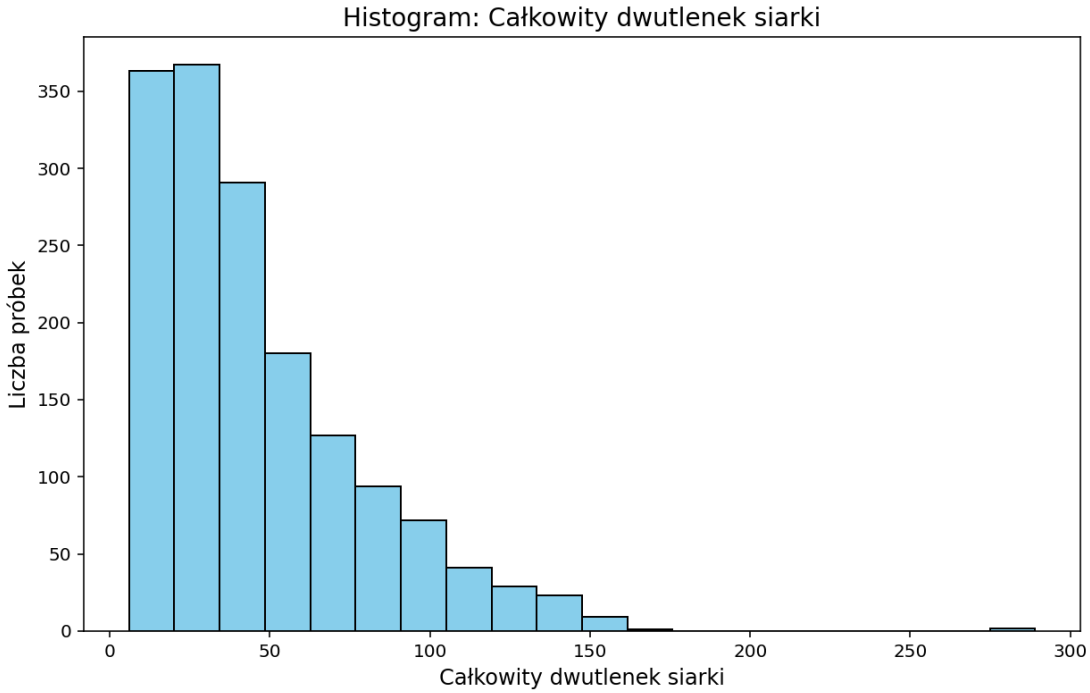
Rysunek 15 Histogram jakości wina



Rysunek 16 Histogram zawartości cukru resztkowego



Rysunek 17 Histogram rozkładu zawartości siarczanów w winie



Rysunek 18 Histogram rozkładu całkowitej zawartości dwutlenku siarki w winie

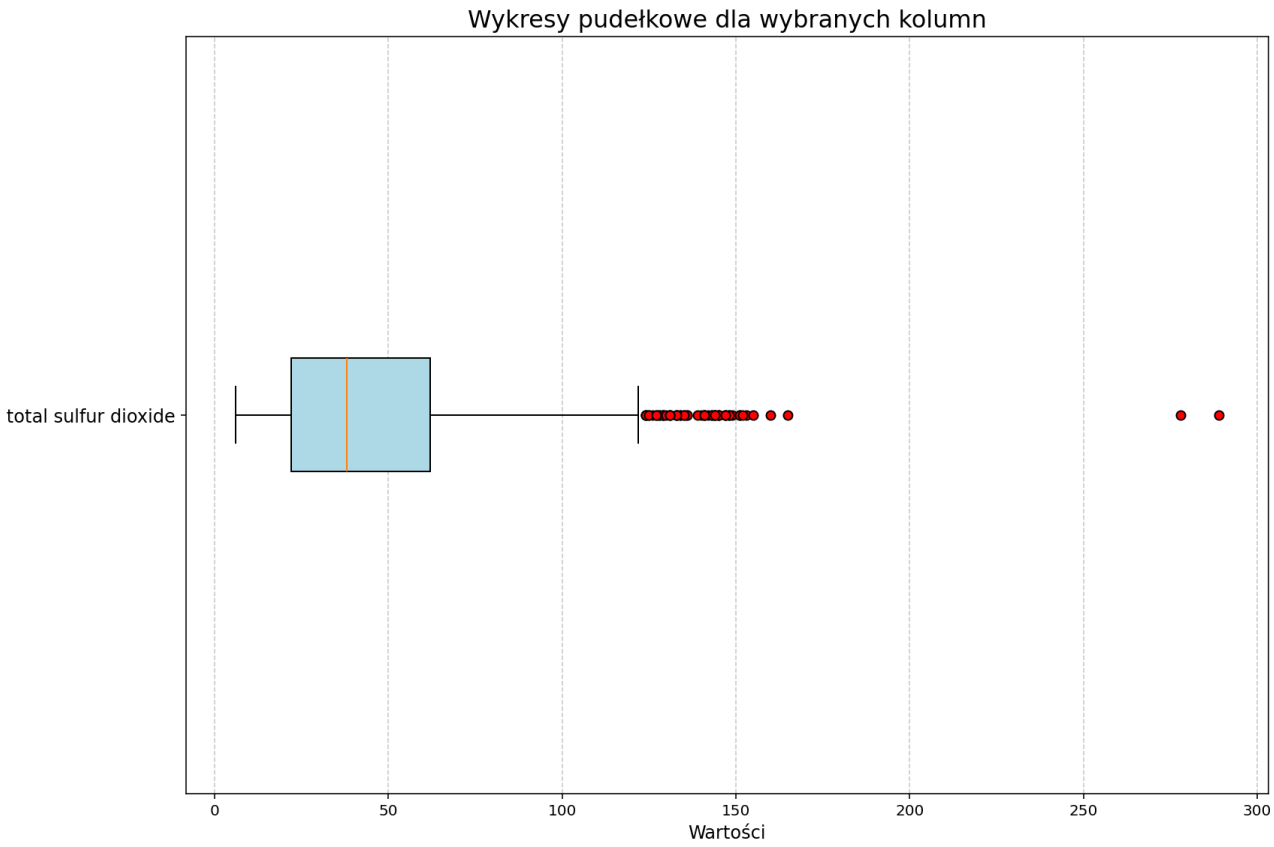
- **Kwasowość stała**: Średnia wynosi 8.32, przy czym rozkład jest zróżnicowany (odchylenie standardowe 1.741). Wartości wahają się od 4.6 do 15.9, co sugeruje szeroką zmienność tej cechy.
- **Kwasowość lotna**: Średnia to 0.528, z mniejszą zmiennością (std: 0.179). Maksymalne wartości sięgają 1.58, co może być uważane za odstające w stosunku do typowych wartości (między 0.39 a 0.64 w kwartylach).
- **Kwas cytrynowy**: Wartość średnia to 0.271, a odchylenie standardowe 0.195. Warto zauważyć, że minimalna wartość wynosi 0, co może oznaczać próbki wina, w których kwas cytrynowy nie występuje.
- **Cukier resztkowy**: Średnia wynosi 2.539, jednak maksymalna wartość aż 15.5 wskazuje na istnienie próbek o bardzo słodkich winach. Odchylenie standardowe 1.41 sugeruje znaczną różnorodność.
- **Chlorki**: Średnia wynosi 0.087, z niewielką zmiennością (std: 0.047). Maksymalna wartość (0.611) może być odstająca.
- **Wolny dwutlenek siarki**: Średnia to 15.875, przy dużym rozrzucie (std: 10.46). Wartości maksymalne (72.0) mogą być odstające w stosunku do typowych wartości w kwartylach (7.0–21.0).
- **Całkowity dwutlenek siarki**: Średnia wynosi 46.468, ale odchylenie standardowe aż 32.895 sugeruje, że niektóre próbki mogą zawierać bardzo wysokie ilości tej substancji (maksymalnie 289.0).
- **Gęstość**: Średnia wynosi 0.997, a zmienność jest niewielka (std: 0.002). Większość próbek mieści się w zakresie typowym dla win (0.996–0.998 w kwartylach).
- **pH**: Średnia wartość 3.311 wskazuje na lekko kwaśne pH, co jest typowe dla win. Rozkład jest stosunkowo wąski (std: 0.154).
- **Siarka**: Średnia wynosi 0.658, z maksymalnymi wartościami sięgającymi 2.0. Wyższe wartości mogą być odstającymi.
- **Alkohol**: Średnia to 10.423%, przy czym rozkład jest umiarkowany (std: 1.066). Minimalne i maksymalne

wartości (8.4–14.9) pokazują różnorodność w zawartości alkoholu.

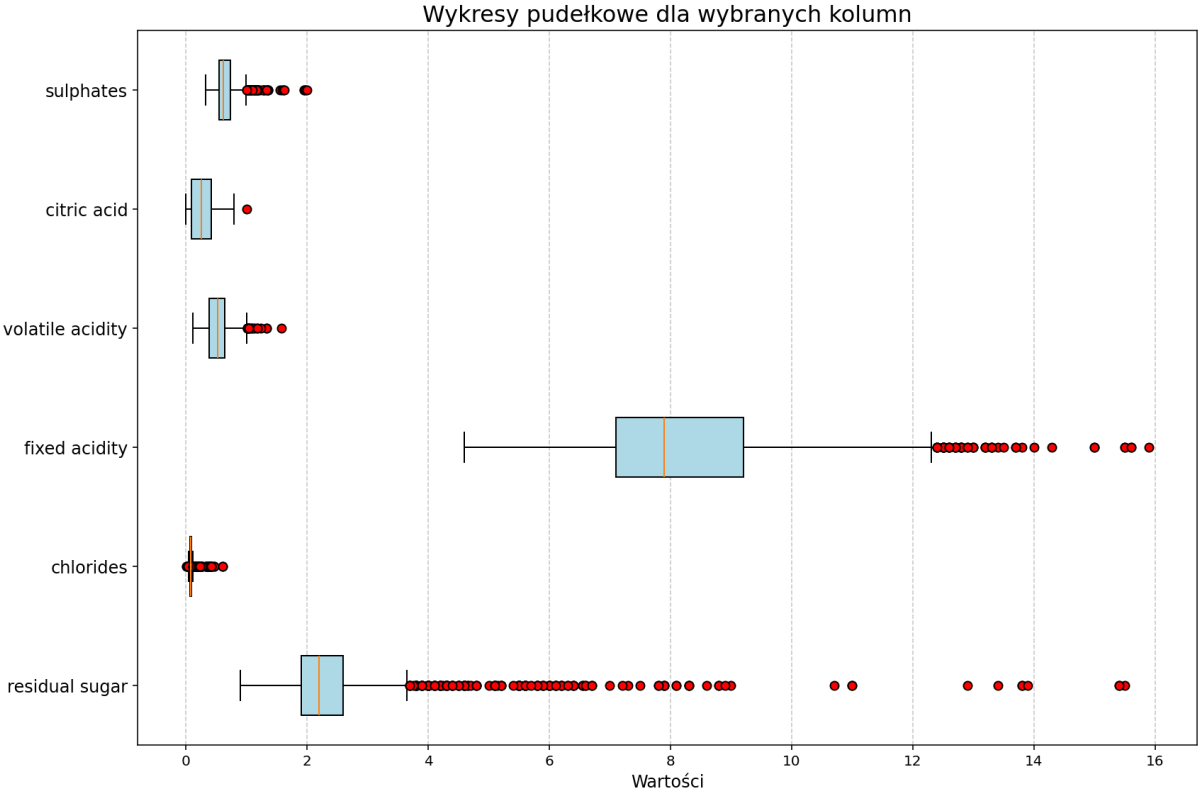
- **Jakość**: Średnia wynosi 5.636, z odchyleniem standardowym 0.808. Wartości wahają się od 3 do 8, co sugeruje skupienie wokół średniej z niewielką liczbą bardzo niskich i bardzo wysokich ocen.

Dodatkowo dla następujących kolumn: **total sulfur dioxide** oraz osobno: **residual sugar, chlorides, fixed acidity, volatile acidity, citric acid, sulphates** wykonamy wykres boxowy aby lepiej zobrazować te dane. Wykorzystamy do tego następujący program[6,7]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Załadowanie danych
df = pd.read_csv('winequality-red.csv')

# Wybrane kolumny do analizy
columns_to_plot = [
   'total sulfur dioxide'
   'residual sugar',
   'chlorides',
   'fixed acidity',
   'volatile acidity',
   'citric acid',
   'sulphates'
]

# Tworzenie jednego wykresu boxplot
plt.figure(figsize=(12, 8))
plt.boxplot([df[column] for column in columns_to_plot],
        vert=False,  # Poziome wykresy pudełkowe
        patch_artist=True,
        boxprops=dict(facecolor='lightblue', color='black'),
        whiskerprops=dict(color='black'),
        capprops=dict(color='black'),
        flierprops=dict(markerfacecolor='red', marker='o'))
plt.yticks(range(1, len(columns_to_plot) + 1), columns_to_plot, fontsize=12)  # Etykiety osi Y
plt.title('Wykresy pudełkowe dla wybranych kolumn', fontsize=16)
plt.xlabel('Wartości', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()

# Zapis wykresu do pliku
plt.savefig('boxplot_all_columns.png')
plt.show()
```



Rysunek 19 Wykres boxowy dla całkowitego dwutlenku siarki



Rysunek 20 Wykresy boxowe dla reszty wybranych danych

2.3 Przygotowanie danych do budowy modelu klasyfikacji

Aby móc skutecznie zbudować model klasyfikacji, dane należy wyczyścić, znormalizować i usunąć wartości puste.

Dzięki użyciu następującego kodu w języku Python, ładujemy zbiór danych a **następnie sprawdzamy, czy występują wartości puste** [6].

```
import pandas as pd

df = pd.read_csv('winequality-red.csv')

print(df.isnull().values.any())
```

Otrzymaliśmy *False*, więc ten zbiór danych nie zawiera wartości pustych. Następnym krokiem będzie **usunięcie duplikatów**. Możemy to wykonać następującym programem [6].

```
import pandas as pd

# Załadowanie danych
df = pd.read_csv('winequality-red.csv')

# Sprawdzanie, czy dane zawierają duplikaty
if df.duplicated().any():
    print("Znaleziono duplikaty w danych.")

    # Wyświetlenie liczby duplikatów
    print(f"Liczba duplikatów: {df.duplicated().sum()}")

    # Usunięcie duplikatów
    df = df.drop_duplicates()
    print("Duplikaty zostały usunięte.")
else:
    print("Brak duplikatów w danych.")

# Zapisanie oczyszczonego zbioru danych do nowego pliku
df.to_csv('cleaned_winequality-red.csv', index=False)
print("Oczyszczone dane zapisano w pliku 'cleaned_winequality-red.csv'.")
```

Po uruchomieniu programu, okazało się, że znalezione zostały 240 duplikaty. Zostały one usunięte a oczyszczony zbiór danych został zapisany do nowego pliku.

Kolejnym krokiem będzie usunięcie danych odstających. Najwięcej danych odstających zauważyliśmy w kolumnach: **total sulfur dioxide, residual sugar, chlorides, fixed acidity, volatile acidity, citric acid, sulphates**. Użyjemy do tego rozstępu międzykwartylowego (IQR). Wykorzystamy do tego następujący program [6, 8].

```
import pandas as pd

# Załadowanie danych
df = pd.read_csv('cleaned_winequality-red.csv')

# Funkcja do usuwania odstających wartości na podstawie IQR
def remove_outliers_iqr(df, column):
    q1 = df[column].quantile(0.25)  # 1. kwartyl
    q3 = df[column].quantile(0.75)  # 3. kwartyl
    iqr = q3 - q1  # rozstęp międzykwartylowy
    lower_bound = q1 - 1.5 * iqr  # dolna granica
    upper_bound = q3 + 1.5 * iqr  # górna granica
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Usuwanie wartości odstających dla wybranych kolumn
columns_to_check = [
    'total sulfur dioxide',
    'residual sugar',
    'chlorides',
    'fixed acidity',
    'volatile acidity',
    'citric acid',
    'sulphates'
]

for column in columns_to_check:
    df = remove_outliers_iqr(df, column)

# Sprawdzenie efektów
print(df.describe())
# Zapisanie przetworzonych danych do nowego pliku CSV
df.to_csv('cleaned_data.csv', index=False)
```

2.4 Normalizacja danych

Użyjemy normalizacji Min-Max, która zmienia zakres wartości od 0 do 1. W przypadku tych danych jest to o tyle przydatne, że mamy różne jednostki danych, a potrzebujemy je znormalizować w celu pozytywnej dalszej analizy i budowy modelów, poza kolumną quality. Program normalizacji metodą Min-Max [6].

```
import pandas as pd

# Załadowanie danych
df = pd.read_csv('cleaned_data.csv')

# Wybór tylko kolumn liczbowych
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
numeric_columns = [col for col in numeric_columns if col != 'quality']

# Normalizacja Min-Max za pomocą Pandas
df[numeric_columns] = df[numeric_columns].apply(lambda x: (x - x.min()) / (x.max() -
x.min()))

# Sprawdzenie wyników
print(df.describe())

# Zapisanie znormalizowanych danych do nowego pliku CSV
df.to_csv('normalized_cleaned_data.csv', index=False)
```

3. Opis eksperymentów i opracowanie uzyskanych wyników

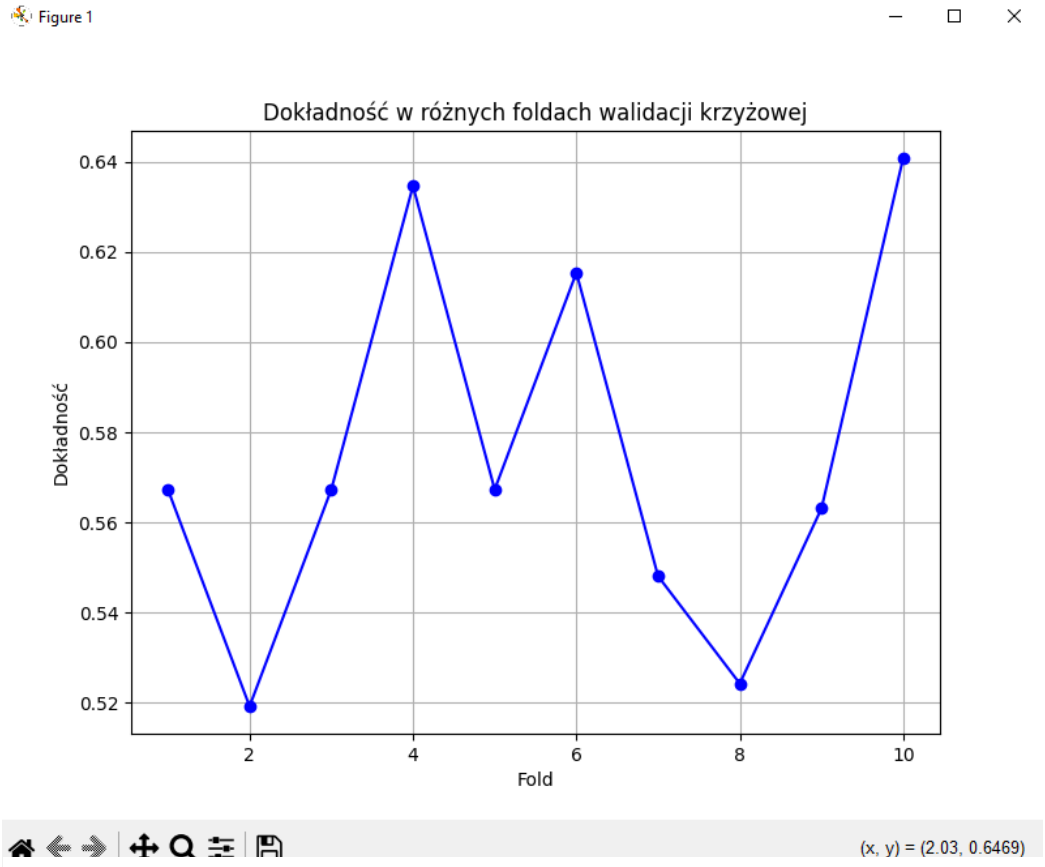
3.1 Klasyfikacja danych za pomocą drzewa decyzyjnego z użyciem stratyfikowanej walidacji krzyżowej

W tym kodzie wykorzystujemy walidację krzyżową do oceny wydajności modelu drzewa decyzyjnego na zbiorze danych. Proces walidacji polega na podzieleniu danych na 10 "foldów", w których model jest trenowany na 9 foldach, a oceniany na pozostałym foldzie. Średnia dokładność na danych walidacyjnych jest obliczana na podstawie wyników z wszystkich foldów, co daje bardziej miarodajną ocenę zdolności modelu do generalizacji. Dodatkowo, wyniki są wizualizowane na wykresie, pokazując dokładność w różnych foldach [9,10].

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import time

# Załadowanie danych
df = pd.read_csv('normalized_cleaned_data.csv')

# Wybór cech (wszystko oprócz 'quality')
X = df.drop('quality', axis=1)
y = df['quality']

# Stratyfikowana walidacja krzyżowa (zachowanie proporcji klas)
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Model drzewa decyzyjnego z dodatkowymi parametrami
model = DecisionTreeClassifier(
    random_state=42,
    max_depth=3,
    min_samples_split=2,
    min_samples_leaf=1,
    criterion='entropy'
)

# Mierzenie czasu rozpoczęcia walidacji
start_time = time.time()
# Przeprowadzenie walidacji krzyżowej
cv_results = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
# Mierzenie czasu zakończenia walidacji
end_time = time.time()
# Obliczanie czasu trwania walidacji
elapsed_time = end_time - start_time
# Wyniki walidacji krzyżowej
accuracy_validation = cv_results.mean()
```

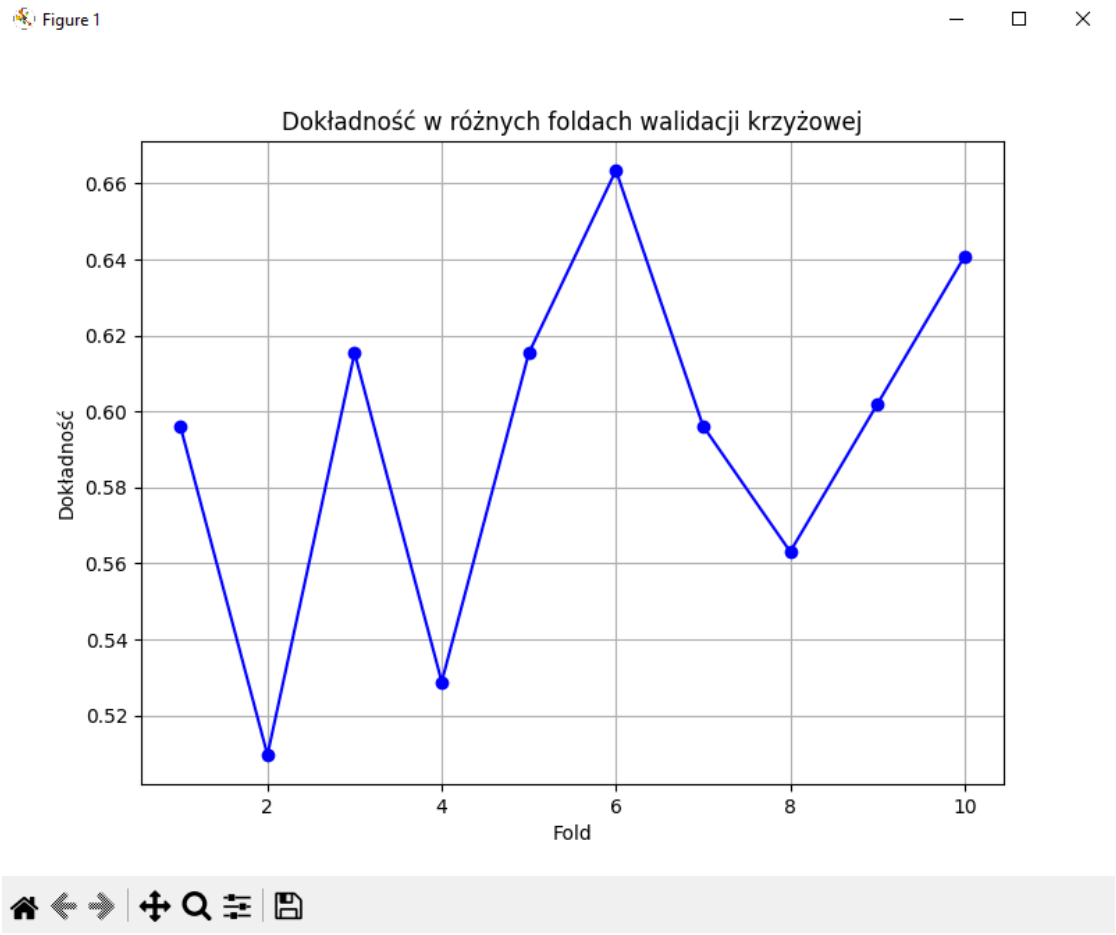
```
# Wyniki
print(f"Średnia dokładność na danych walidacyjnych: {accuracy_validation * 100:.2f}%")
print(f"Czas trwania walidacji krzyżowej: {elapsed_time:.2f} sekund")

# Aby pokazać wyniki graficznie, np. wykres zależności dokładności w różnych foldach
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cv_results) + 1), cv_results, marker='o', linestyle='-', color='b')
plt.title("Dokładność w różnych foldach walidacji krzyżowej")
plt.xlabel('Fold')
plt.ylabel('Dokładność')
plt.grid(True)
plt.show()
```

Uzyskanym wynikiem dla parametrów losowych średniej dokładności na danych walidacyjnych jest **57.47%** (Rys.21), a czas trwania walidacji krzyżowej wyniósł **0.11 sekundy**.



Rysunek 21 Dokładność w różnych foldach walidacji krzyżowej

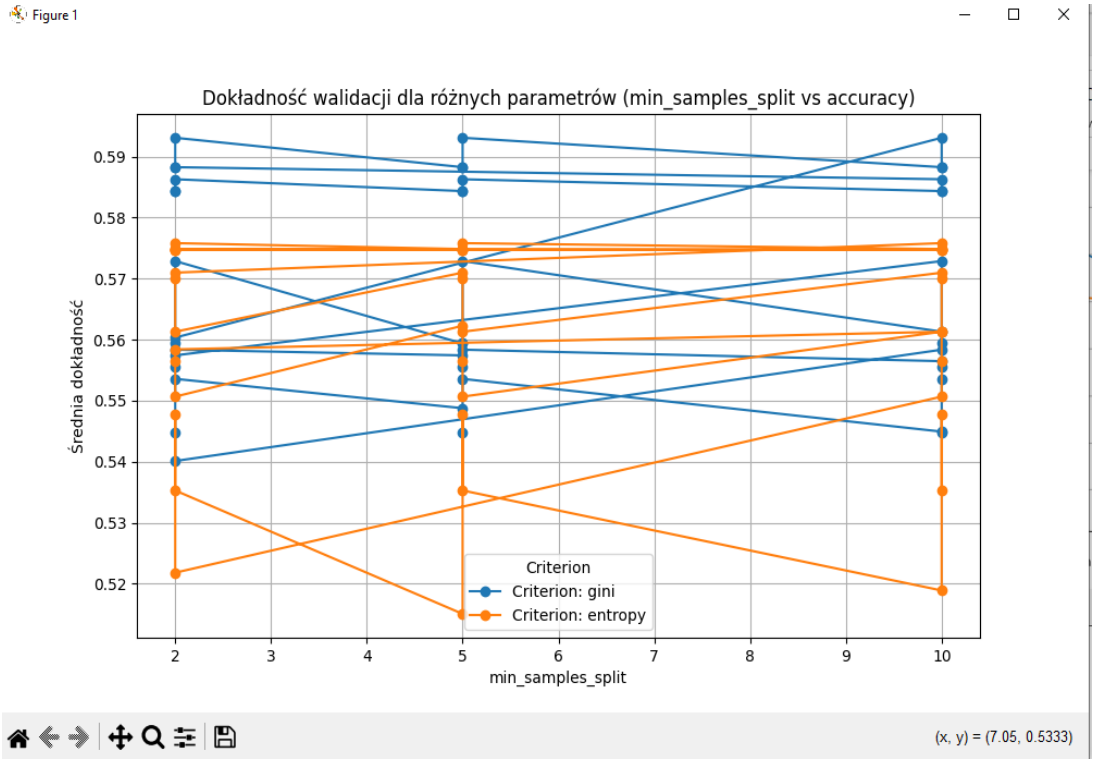
Przy zmienionych parametrach drzewa decyzyjnego na następujące: min_samples_leaf=10, criterion='gini', max_depth=4, uzyskane wyniki się różnią: średnia dokładność na danych walidacyjnych: **59.31%**, czas trwania walidacji krzyżowej: **0.10 sekundy** (Rys.22). I są to najlepsze możliwe parametry.



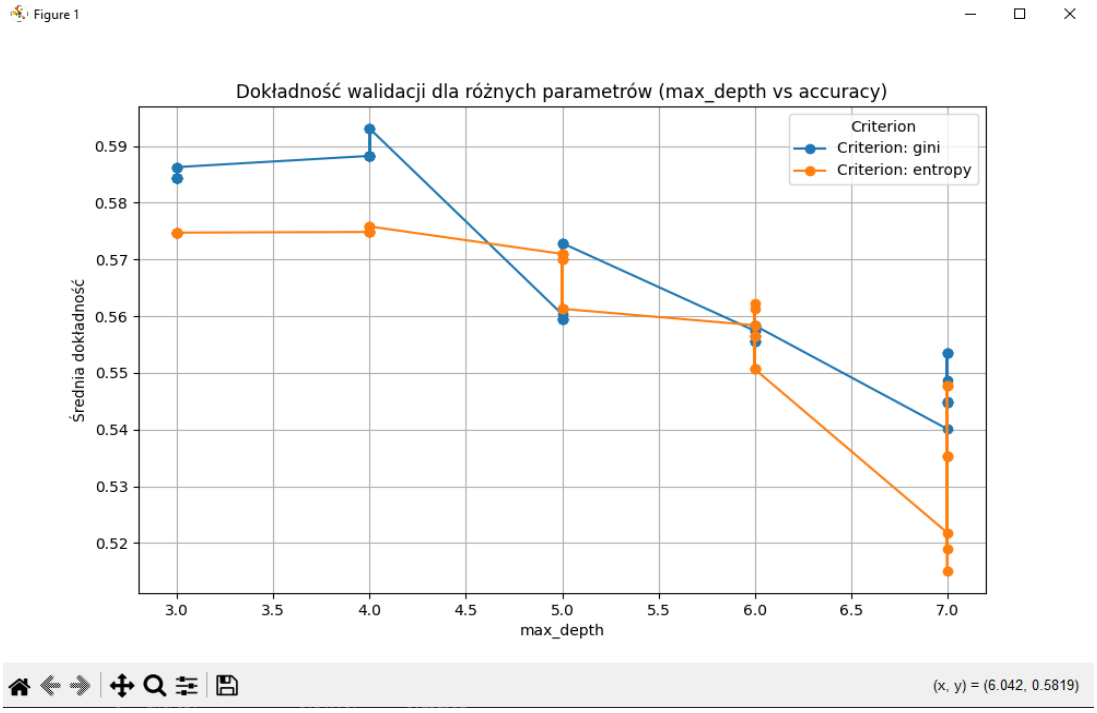
Rysunek 22 Dokładność w różnych foldach walidacji krzyżowej przy zmienionych parametrach

W celu porównania działania programu dla różnych wartości parametrów, użyjemy następujących zakresów:

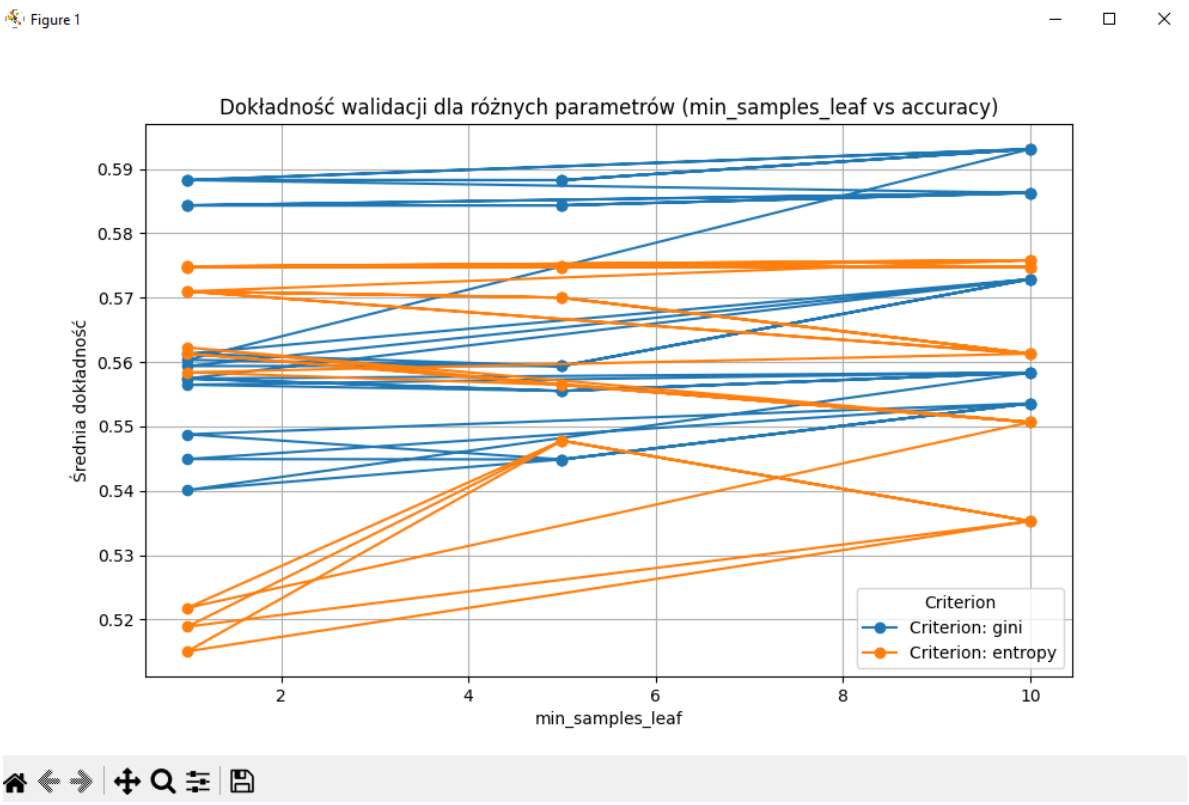
```
max_depth_values = [3, 4, 5, 6, 7]
min_samples_split_values = [2, 5, 10]
min_samples_leaf_values = [1, 5, 10]
criterion_values = ['gini', 'entropy']
```



Rysunek 23 - Zależność średniej dokładności od min_samples_split oraz criterion



Rysunek 24- Zależność średniej dokładności od criterion oraz max_depth



Rysunek 25 - Zależność średniej dokładności od min_samples_leaf i criterion

max_depth	min_samples_split	min_samples_leaf	criterion	accuracy_validation	elapsed_time
3	2	10	gini	0.59	0.08
3	10	10	gini	0.59	0.08
3	5	5	gini	0.58	0.08
3	10	5	gini	0.58	0.08
3	5	10	gini	0.59	0.09
4	2	1	gini	0.59	0.09
4	2	5	gini	0.59	0.09
4	2	10	gini	0.59	0.09
4	5	1	gini	0.59	0.09
4	5	5	gini	0.59	0.09
4	5	10	gini	0.59	0.09
4	10	1	gini	0.59	0.09
4	10	5	gini	0.59	0.09
4	10	10	gini	0.59	0.09
3	2	1	gini	0.58	0.09
3	2	5	gini	0.58	0.09
3	5	1	gini	0.58	0.09
3	10	1	gini	0.58	0.09
3	2	1	entropy	0.57	0.09
3	2	5	entropy	0.57	0.09
3	2	10	entropy	0.57	0.09

3	5	1	entropy	0.57	0.09
3	5	5	entropy	0.57	0.09
3	5	10	entropy	0.57	0.09
3	10	1	entropy	0.57	0.09
3	10	5	entropy	0.57	0.09
3	10	10	entropy	0.57	0.09
4	2	10	entropy	0.58	0.1
4	5	10	entropy	0.58	0.1
4	10	10	entropy	0.58	0.1
4	2	1	entropy	0.57	0.1
4	2	5	entropy	0.57	0.1
4	5	1	entropy	0.57	0.1
4	5	5	entropy	0.57	0.1
4	10	1	entropy	0.57	0.1
5	2	10	gini	0.57	0.1
5	5	10	gini	0.57	0.1
5	10	10	gini	0.57	0.1
5	2	1	gini	0.56	0.1
5	2	5	gini	0.56	0.1
5	5	1	gini	0.56	0.1
5	5	5	gini	0.56	0.1
5	10	1	gini	0.56	0.1
5	10	5	gini	0.56	0.1
6	2	1	gini	0.56	0.1
6	2	5	gini	0.56	0.1
6	2	10	gini	0.56	0.1
6	5	5	gini	0.56	0.1
6	5	10	gini	0.56	0.1
6	10	5	gini	0.56	0.1
6	10	10	gini	0.56	0.1
7	2	10	gini	0.55	0.1
7	5	10	gini	0.55	0.1
7	10	10	gini	0.55	0.1
4	10	5	entropy	0.57	0.11
5	2	1	entropy	0.57	0.11
5	2	5	entropy	0.57	0.11
5	5	1	entropy	0.57	0.11
5	5	5	entropy	0.57	0.11
5	10	1	entropy	0.57	0.11
5	10	5	entropy	0.57	0.11
5	2	10	entropy	0.56	0.11
5	5	10	entropy	0.56	0.11
5	10	10	entropy	0.56	0.11
6	5	1	gini	0.56	0.11

6	10	1	gini	0.56	0.11
6	2	10	entropy	0.55	0.11
6	10	10	entropy	0.55	0.11
7	2	1	gini	0.54	0.11
7	2	5	gini	0.54	0.11
7	5	5	gini	0.54	0.11
7	10	5	gini	0.54	0.11
6	2	1	entropy	0.56	0.12
6	2	5	entropy	0.56	0.12
6	5	5	entropy	0.56	0.12
6	10	1	entropy	0.56	0.12
6	5	10	entropy	0.55	0.12
7	2	5	entropy	0.55	0.12
7	5	1	gini	0.55	0.12
7	2	10	entropy	0.54	0.12
7	5	10	entropy	0.54	0.12
7	10	10	entropy	0.54	0.12
6	5	1	entropy	0.56	0.13
6	10	5	entropy	0.56	0.13
7	5	5	entropy	0.55	0.13
7	10	5	entropy	0.55	0.13
7	10	1	gini	0.54	0.13
7	2	1	entropy	0.52	0.13
7	10	1	entropy	0.52	0.13
7	5	1	entropy	0.52	0.15

W tabeli powyżej możemy zauważyć wszystkie kombinacje wybranych parametrów i wyniki szybkości działania i dokładności działania drzewa decyzyjnego. Optymalne parametry drzewa decyzyjnego to głębokość drzewa (max_depth) równa 4, minimalna liczba próbek potrzebna do podziału (min_samples_split) wynosząca 2, 5 lub 10, minimalna liczba próbek w liściu (min_samples_leaf) równa 1, 5 lub 10, oraz kryterium podziału (criterion) ustawione na "gini". W takim przypadku model osiąga najwyższą dokładność na poziomie 0.59 przy czasie obliczeń wynoszącym 0.09 sekundy.

Dla szybkiego działania przy umiarkowanej dokładności można wybrać drzewo o głębokości równej 3. Jednak dla uzyskania najlepszej dokładności bez znaczącego wzrostu czasu wykonania optymalna jest głębokość drzewa wynosząca 4. Z kolei zbyt duża głębokość drzewa, na przykład 7, prowadzi do spadku dokładności i potencjalnego przeuczenia modelu.

Porównując dwa kryteria podziału, "gini" i "entropy", lepsze wyniki daje kryterium "gini", osiągając wyższą dokładność (0.59 w porównaniu do 0.58 dla "entropy") przy podobnym czasie obliczeń. Dlatego też kryterium "gini" jest bardziej rekomendowane.

Drzewa decyzyjne nie wymagają, aby dane były znormalizowane, normalizacja danych została zastosowana dla kolejnych używanych metod, lecz po sprawdzeniu działania drzewa na danych znormalizowanych otrzymujemy zbliżone o ile nie taką samą dokładność.

3.2 Klasyfikacja danych za pomocą maszyny wektorów nośnych (SVM) z użyciem stratyfikowanej walidacji krzyżowej

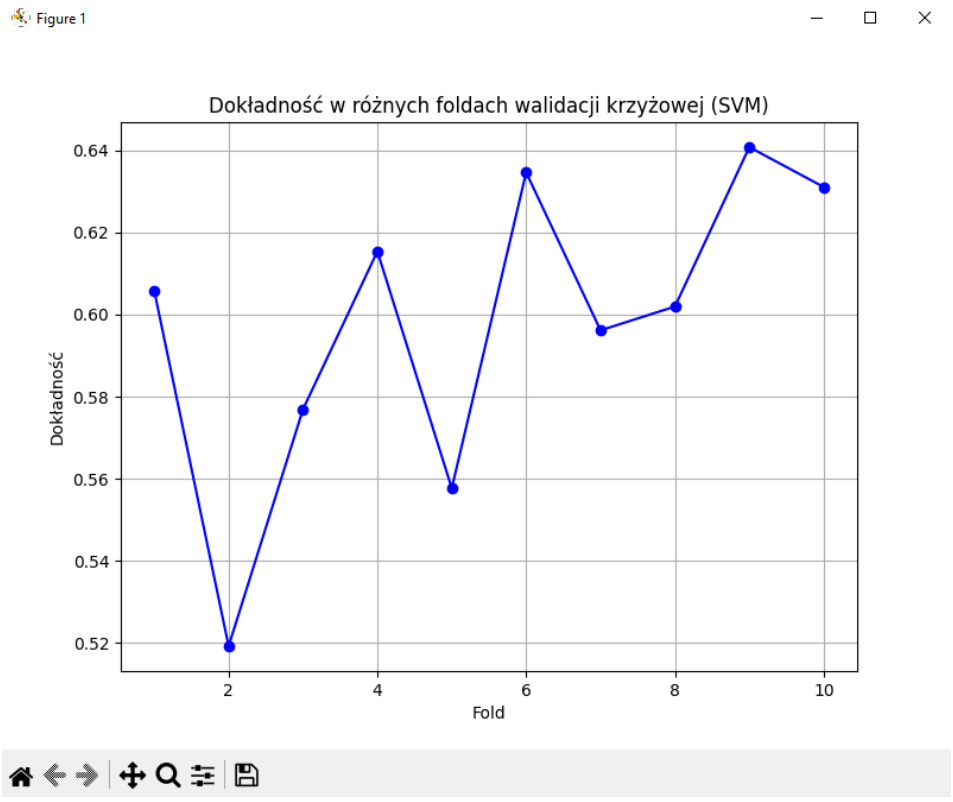
W przypadku maszyny wektorów nośnych, również zastosujemy stratyfikowaną walidację krzyżową, z 10 foldami, aby uzyskane wyniki były porównywalne z resztą metod klasyfikacji danych.

Kod programu [11]:

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import time
import matplotlib.pyplot as plt

# Załadowanie danych
df = pd.read_csv('normalized_cleaned_data.csv')

# Wybór cech (wszystko oprócz 'quality')
X = df.drop('quality', axis=1)
y = df['quality']

# Stratyfikowana walidacja krzyżowa
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Parametry SVM
model = SVC(
    kernel='rbf',  # Można zmienić na 'linear', 'poly', 'sigmoid'
    C=1.0,         # Parametr regularyzacji
    gamma='scale', # Funkcja jądra
    random_state=42
)

# Mierzenie czasu rozpoczęcia walidacji
start_time = time.time()

# Przeprowadzenie walidacji krzyżowej
cv_results = cross_val_score(model, X, y, cv=kf, scoring='accuracy')

# Mierzenie czasu zakończenia walidacji
end_time = time.time()

# Obliczanie czasu trwania walidacji
elapsed_time = end_time - start_time

# Wyniki walidacji krzyżowej
accuracy_validation = cv_results.mean()
accuracy_std = cv_results.std()
```

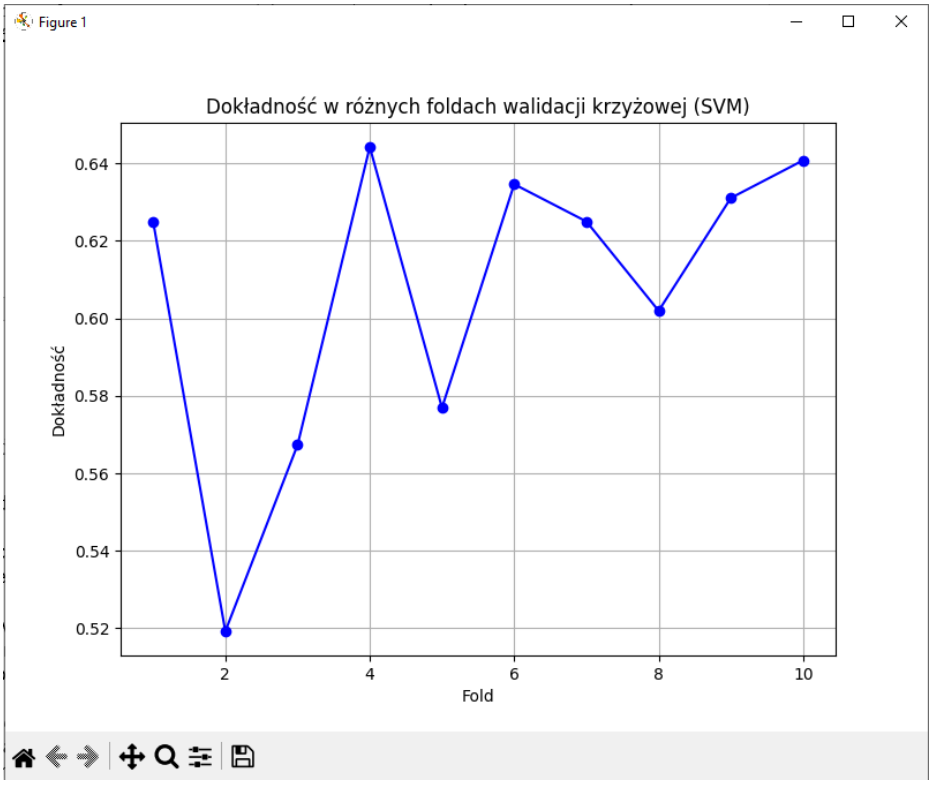
```
# Wyniki
print(f"Średnia dokładność na danych walidacyjnych: {accuracy_validation * 100:.2f}%")
print(f"Odchylenie standardowe dokładności: {accuracy_std * 100:.2f}%")
print(f"Czas trwania walidacji krzyżowej: {elapsed_time:.2f} sekund")

# Wykres dokładności w różnych foldach
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cv_results) + 1), cv_results, marker='o', linestyle='-', color='b')
plt.title("Dokładność w różnych foldach walidacji krzyżowej (SVM)")
plt.xlabel('Fold')
plt.ylabel('Dokładność')
plt.grid(True)
plt.show()
```

Wybór parametrów jest tutaj przypadkowy, w dalszych rozważaniach zostanie znaleziony i pokazany wynik działania programu dla najlepszych parametrów. Średnia uzyskana dokładność wyniosła **59.80%**, a czas walidacji krzyżowej **0.45 sekundy**.



Rysunek 26 Dokładność działania programu dla poszczególnych foldów

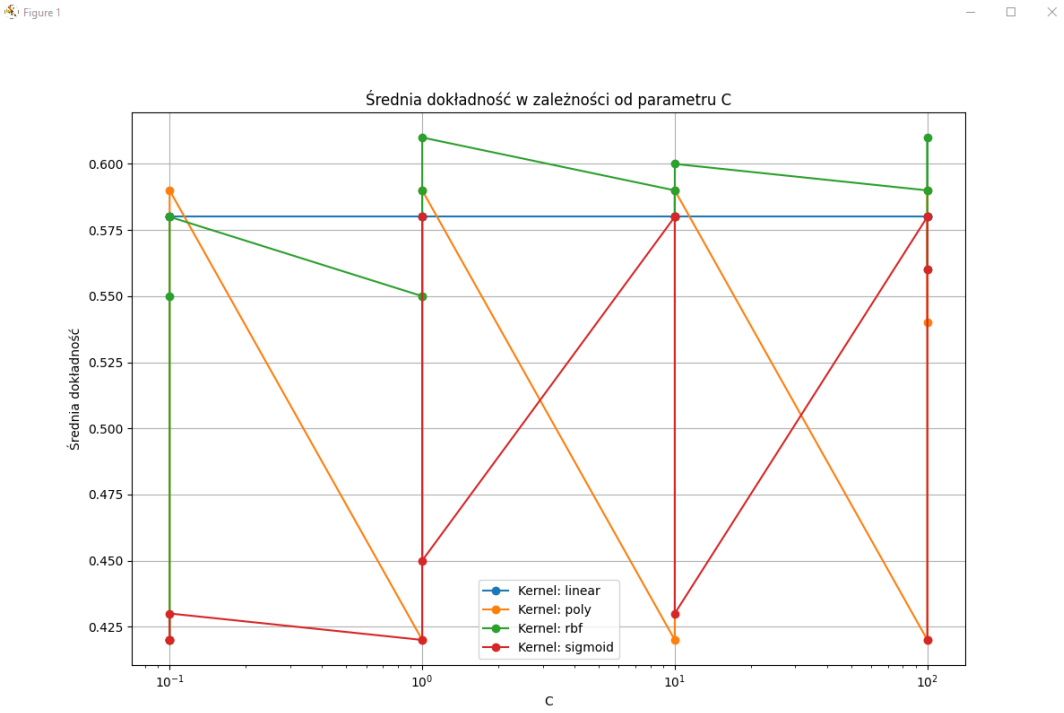
Gdy gamma została zmieniona na 1, przy reszcie parametrów pozostawionych jak w programie wyżej, została osiągnięta wyższa dokładność równa **60.66%**, przy czasie wykonywania **0,43** sekundy.



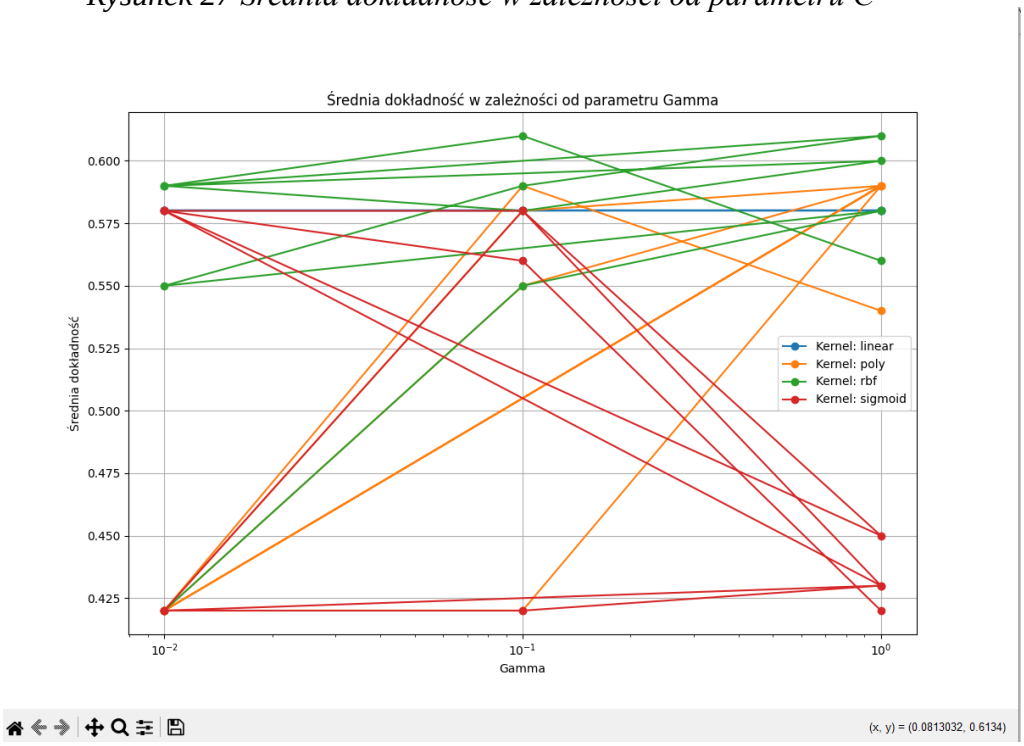
Rysunek 26 Dokładność na różnych foldach dla zmienionego parametru gamma

W celu porównania działania programu dla różnych podanych parametrów, użyjemy różnych kombinacji następujących parametrów:

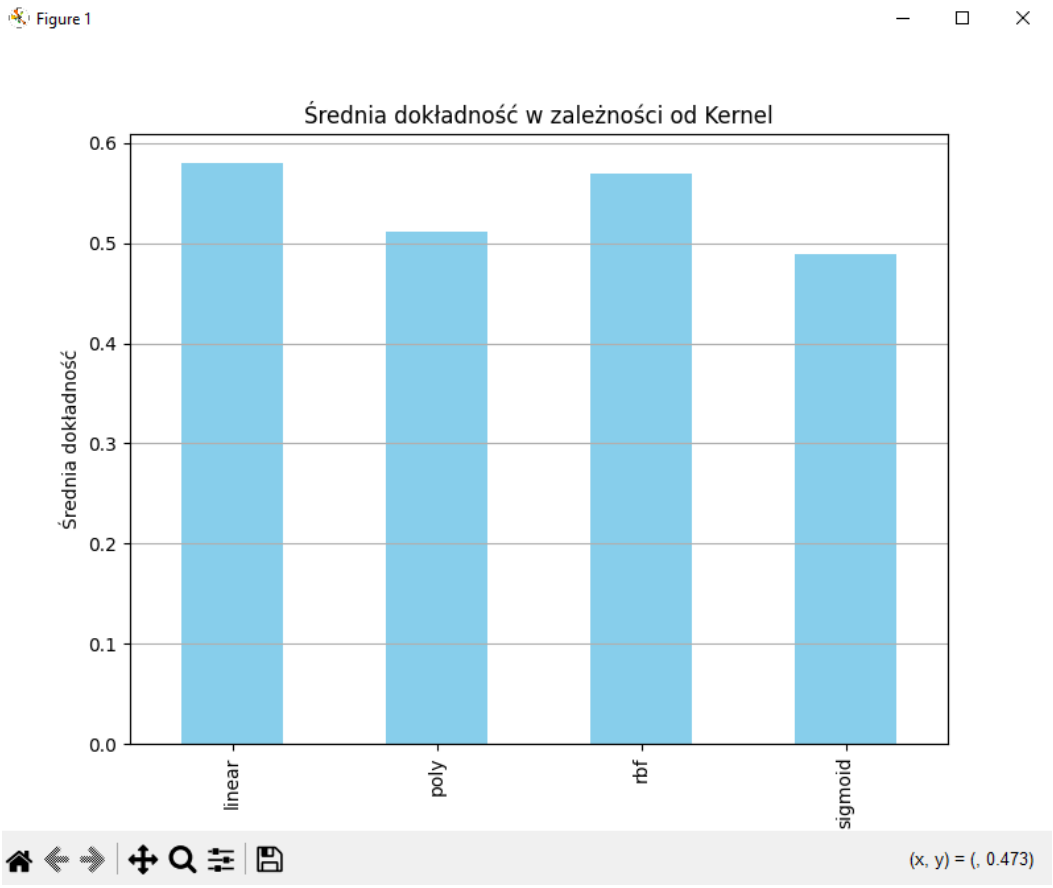
```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': [0.01, 0.1, 1]
}
```



Rysunek 27 Średnia dokładność w zależności od parametru C



Rysunek 28 Średnia dokładność w zależności od parametru Gamma



Rysunek 29 Średnia dokładność w zależności od parametru Kernel

C	Kernel	Gamma	Mean Accuracy	Execution Time
100.0	rbf	0.1	0.61	0.56
1.0	rbf	1.0	0.61	0.46
10.0	rbf	1.0	0.6	0.49
100.0	rbf	0.01	0.59	0.53
1.0	rbf	0.1	0.59	0.48
1.0	poly	1.0	0.59	0.46
10.0	rbf	0.01	0.59	0.44
0.1	poly	1.0	0.59	0.34
100.0	poly	0.1	0.59	0.32
100.0	linear	0.01	0.58	0.72
100.0	linear	0.1	0.58	0.69
100.0	linear	1.0	0.58	0.69
10.0	rbf	0.1	0.58	0.46
0.1	rbf	1.0	0.58	0.44
10.0	sigmoid	0.1	0.58	0.42
1.0	sigmoid	0.1	0.58	0.41

10.0	sigmoid	0.01	0.58	0.36
100.0	sigmoid	0.01	0.58	0.35
10.0	linear	0.01	0.58	0.33
10.0	linear	0.1	0.58	0.32
10.0	linear	1.0	0.58	0.32
10.0	poly	0.1	0.58	0.3
1.0	linear	0.1	0.58	0.28
1.0	linear	1.0	0.58	0.28
0.1	linear	0.01	0.58	0.27
1.0	linear	0.01	0.58	0.27
0.1	linear	0.1	0.58	0.25
0.1	linear	1.0	0.58	0.25
100.0	rbf	1.0	0.56	0.94
100.0	sigmoid	0.1	0.56	0.44
1.0	rbf	0.01	0.55	0.52
0.1	rbf	0.1	0.55	0.46
1.0	poly	0.1	0.55	0.31
1.0	sigmoid	1.0	0.45	0.47
0.1	sigmoid	1.0	0.43	0.54
10.0	sigmoid	1.0	0.43	0.36
1.0	sigmoid	0.01	0.42	0.48
0.1	rbf	0.01	0.42	0.47
0.1	sigmoid	0.1	0.42	0.37
0.1	sigmoid	0.01	0.42	0.36
100.0	sigmoid	1.0	0.42	0.33
0.1	poly	0.1	0.42	0.29
1.0	poly	0.01	0.42	0.29
10.0	poly	0.01	0.42	0.28
100.0	poly	0.01	0.42	0.28
0.1	poly	0.01	0.42	0.27

Najlepsze wyniki średniej dokładności (0.61) zostały osiągnięte przy następujących ustawieniach: C = 1.0, Kernel = rbf, Gamma = 1.0 (czas: 0.46 sekundy) oraz C = 100.0, Kernel = rbf, Gamma = 0.1 (czas: 0.56 sekundy). Kernel rbf okazuje się szczególnie skuteczny dla tego zestawu danych, zwłaszcza przy wyższych wartościach parametru C.

Modele wykorzystujące Kernel = linear charakteryzują się krótszym czasem obliczeń w porównaniu do bardziej złożonych funkcji jądra, takich jak rbf, poly czy sigmoid. Na przykład, dla C = 0.1, Kernel = linear, Gamma = 0.1, model osiąga dokładność 0.58 w czasie 0.25 sekundy.

Niższe wartości Gamma często prowadzą do spadku dokładności. Na przykład, ustawienia C = 1.0, Kernel = rbf, Gamma = 0.01 dają dokładność 0.55 przy czasie 0.52 sekundy. Z kolei wyższe wartości Gamma, takie jak Gamma = 1.0, zazwyczaj poprawiają dokładność.

Funkcja jądra sigmoid zwykle uzyskuje gorsze wyniki pod względem średniej dokładności w porównaniu do innych funkcji jądra. Przykładowo, C = 0.1, Kernel = sigmoid, Gamma = 0.01 osiąga tylko 0.42 dokładności.

Kernel poly charakteryzuje się zmienną wydajnością. Przy umiarkowanych wartościach parametrów C i Gamma może osiągnąć dobrą dokładność w stosunkowo krótkim czasie. Na przykład, dla C = 10.0, Kernel = poly,

Gamma = 0.1, dokładność wynosi 0.58, a czas obliczeń to 0.3 sekundy.

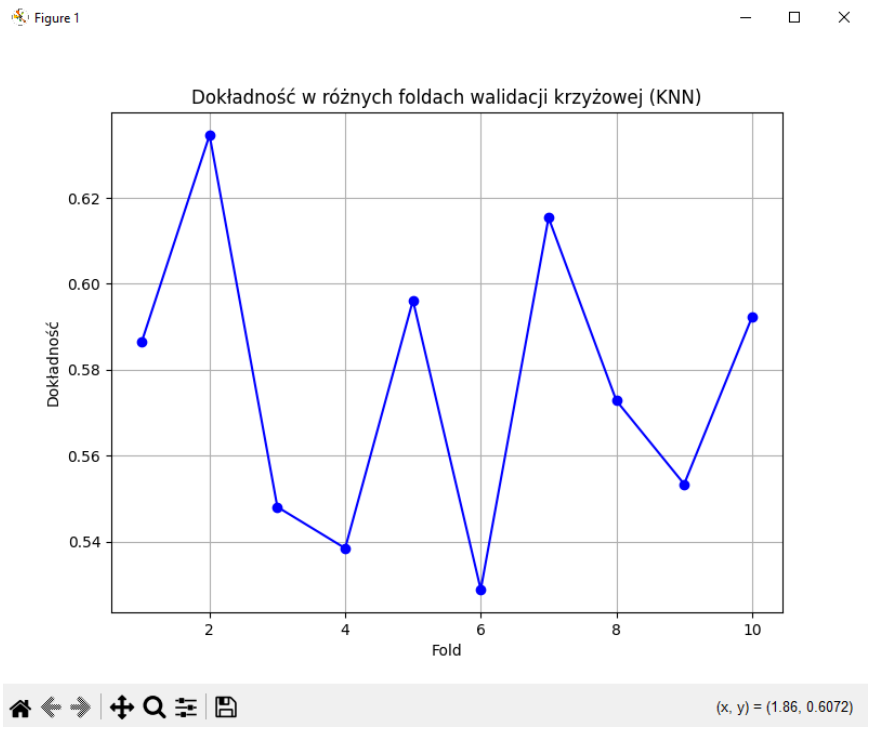
3.3 Klasyfikacja danych za pomocą K-Najbliższych Sąsiadów (KNN) z użyciem stratyfikowanej walidacji krzyżowej

Ostatnią klasyfikacji danych będzie metoda K-najbliższych sąsiadów. Program reprezentujący działanie tego algorytmu został przedstawiony poniżej [12].

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import time

# Załadowanie danych
df = pd.read_csv('normalized_cleaned_data.csv')

# Wybór cech (wszystko oprócz 'quality')
X = df.drop('quality', axis=1)
y = df['quality']

# Stratyfikowana walidacja krzyżowa (zachowanie proporcji klas)
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Ustawienie K-Najbliższych Sąsiadów (KNN) z wybraną wartością K
k_value = 5  # Wybrana liczba sąsiadów (K)
model = KNeighborsClassifier(n_neighbors=k_value)

# Mierzenie czasu rozpoczęcia walidacji
start_time = time.time()

# Przeprowadzenie walidacji krzyżowej
cv_results = cross_val_score(model, X, y, cv=kf, scoring='accuracy')

# Mierzenie czasu zakończenia walidacji
end_time = time.time()

# Obliczanie czasu trwania walidacji
elapsed_time = end_time - start_time

# Wyniki walidacji krzyżowej
accuracy_validation = cv_results.mean()
```

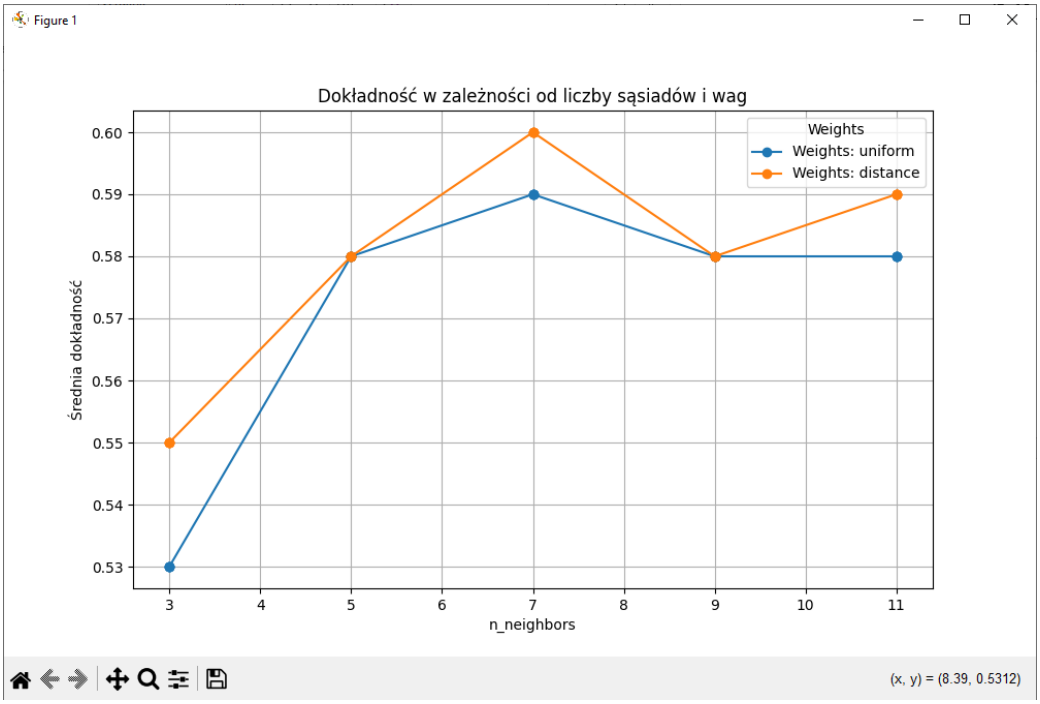
```
# Wyświetlenie wyników
print(f"Średnia dokładność na danych walidacyjnych: {accuracy_validation * 100:.2f}%")
print(f"Czas trwania walidacji krzyżowej: {elapsed_time:.2f} sekund")

# Aby pokazać wyniki graficznie, np. wykres zależności dokładności w różnych foldach
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(cv_results) + 1), cv_results, marker='o', linestyle='-', color='b')
plt.title("Dokładność w różnych foldach walidacji krzyżowej (KNN)")
plt.xlabel('Fold')
plt.ylabel('Dokładność')
plt.grid(True)
plt.show()
```

Uzyskana średnia dokładność wyniosła 58%, przy czasie 0,11 sekundy. Taki wynik został uzyskany dla losowych paramaterów.

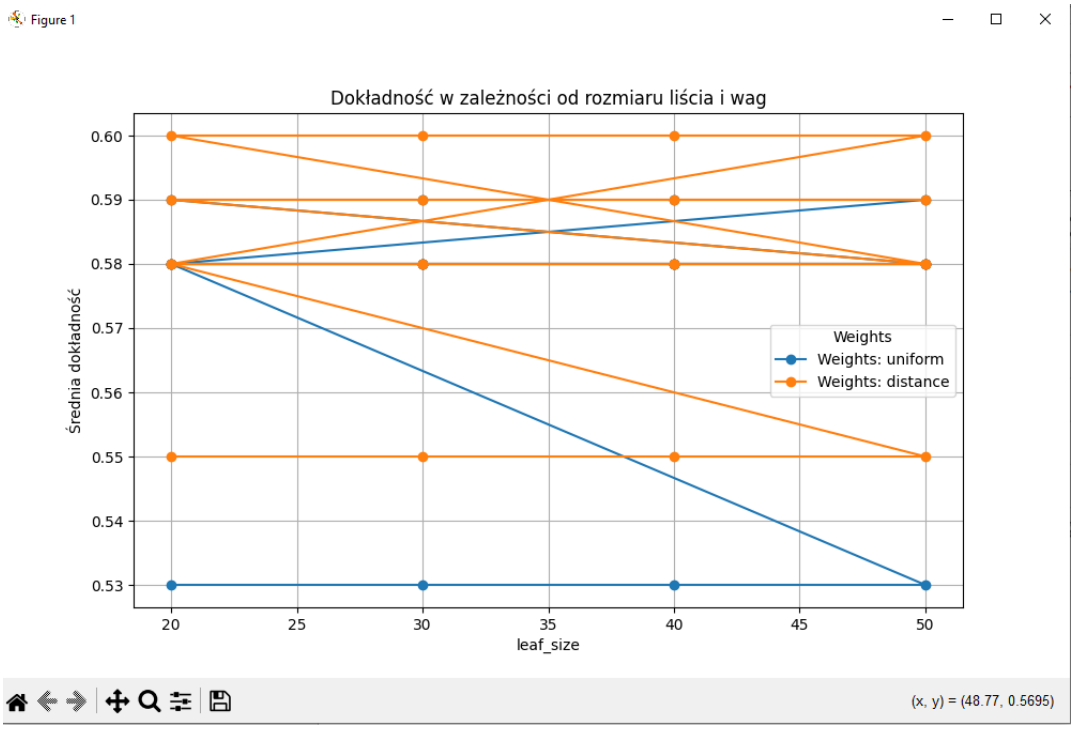


Rysunek 30 Dokładność w różnych foldach walidacji krzyżowej

Teraz sprawdzimy działanie algorytmu dla różnych parametrów liczby sąsiadów, wag, rozmiaru liścia.



Rysunek 31 Dokładność w zależności od liczby sąsiadów i wag



Rysunek 32 Dokładność w zależności od rozmiaru liścia i wag

n_neighbors	weights	leaf_size	accuracy_validation	elapsed_time
7	distance	20	0.6	0.07
7	distance	30	0.6	0.08
7	distance	40	0.6	0.07
7	distance	50	0.6	0.07
7	uniform	20	0.59	0.13
7	uniform	30	0.59	0.12
7	uniform	40	0.59	0.11
7	uniform	50	0.59	0.11
11	distance	20	0.59	0.08
11	distance	30	0.59	0.07
11	distance	40	0.59	0.07
11	distance	50	0.59	0.07
5	uniform	20	0.58	0.12
5	uniform	30	0.58	0.11
5	uniform	40	0.58	0.1
5	uniform	50	0.58	0.12
5	distance	20	0.58	0.09
5	distance	30	0.58	0.08
5	distance	40	0.58	0.06
5	distance	50	0.58	0.07
9	uniform	20	0.58	0.14
9	uniform	30	0.58	0.12
9	uniform	40	0.58	0.12
9	uniform	50	0.58	0.12
9	distance	20	0.58	0.08
9	distance	30	0.58	0.08
9	distance	40	0.58	0.08
9	distance	50	0.58	0.07
11	uniform	20	0.58	0.12
11	uniform	30	0.58	0.12
11	uniform	40	0.58	0.12
11	uniform	50	0.58	0.11
3	distance	20	0.55	0.08
3	distance	30	0.55	0.07
3	distance	40	0.55	0.07
3	distance	50	0.55	0.07
3	uniform	20	0.53	0.16
3	uniform	30	0.53	0.12
3	uniform	40	0.53	0.12
3	uniform	50	0.53	0.13

Wartości parametru n_neighbors mają wpływ na stabilność i precyzję modelu KNN. W tym przypadku testowano kilka wartości tego parametru, zaczynając od 3, a kończąc na 11. Widać, że mniejsze wartości n_neighbors (np. 3) prowadzą do mniejszej dokładności, co może być związane z większą wrażliwością modelu na szumy w danych. Im wyższa liczba sąsiadów, tym bardziej zrównoważona jest decyzja modelu, ale z drugiej strony może to prowadzić do pewnego wygładzenia klasyfikacji, zmniejszając jej precyzję w przypadku trudnych, nieregularnych danych. Najwyższą dokładność uzyskano przy n_neighbors = 7, co może sugerować, że przy tej liczbie sąsiadów model znajduje dobry balans pomiędzy generalizowaniem danych a ich dostosowywaniem do poszczególnych przypadków.

Jeśli chodzi o weights, to parametr ten również odgrywa kluczową rolę w dokładności modelu. Wartość distance (gdzie bliżsi sąsiedzi mają wyższą wagę) okazała się być bardziej efektywna niż uniform, w którym każdy sąsiad ma równą wagę. W większości przypadków, gdy zastosowano wagę distance, model osiągał wyższe wyniki dokładności, co jest zgodne z intuicją – bliżsi sąsiedzi są bardziej reprezentatywni i mogą bardziej precyzyjnie wpływać na decyzję modelu. Z kolei wagi uniform mogą powodować, że dalsi sąsiedzi mają równie duży wpływ, co może obniżyć precyzję, zwłaszcza w przypadku złożonych danych.

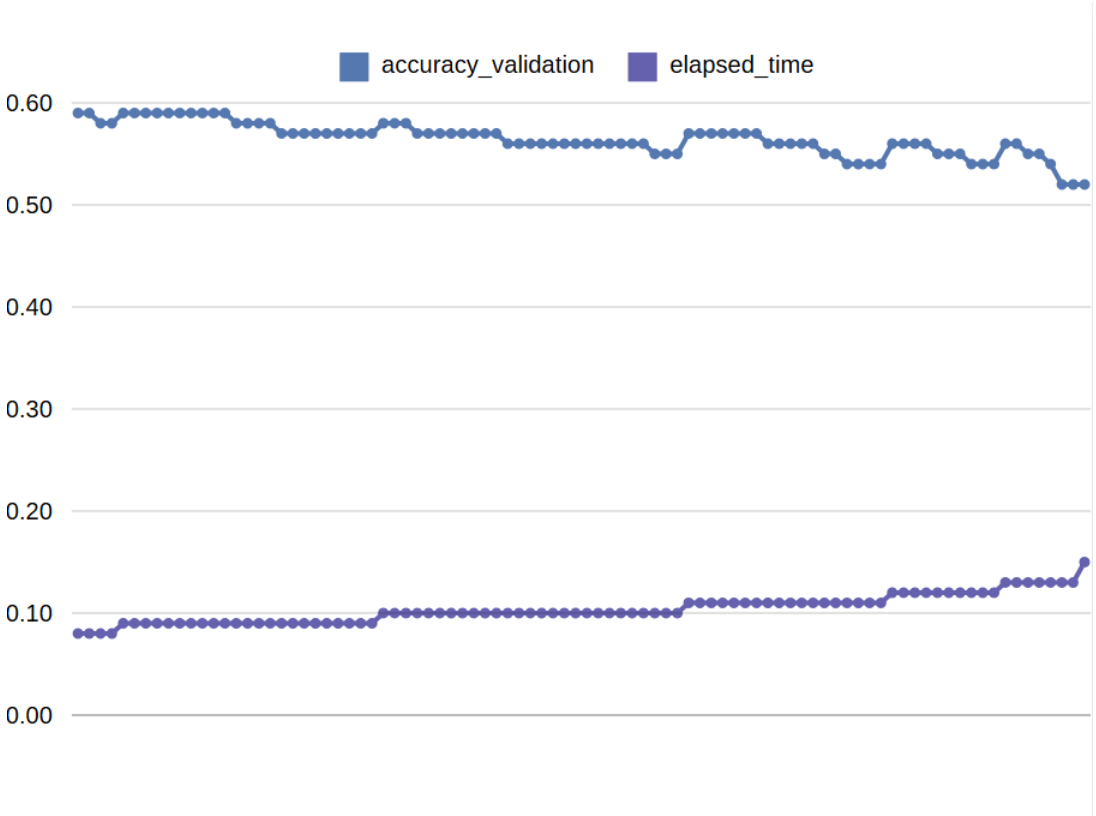
Parametr leaf_size wpływa na wydajność algorytmu KNN, a jego odpowiednia wartość jest istotna dla optymalizacji procesu wyszukiwania sąsiadów. Mniejsze wartości leaf_size mogą spowodować, że algorytm będzie bardziej dokładny, ale kosztem czasu wykonania, ponieważ wymaga to większej ilości operacji w celu znalezienia najbliższych sąsiadów. Z kolei większe leaf_size prowadzi do szybszego wyszukiwania, ale może nieco pogorszyć dokładność, gdyż proces znajdowania sąsiadów nie jest tak dokładny. Z analizy wyników wynika, że dla wartości leaf_size = 20 osiągnięto najlepsze wyniki pod względem dokładności (0.6), co sugeruje, że przy tej wielkości liściach algorytm jest wystarczająco szybki, a jednocześnie nie traci na precyzji klasyfikacji.

W ogólnej analizie wyników dla różnych kombinacji parametrów, najkorzystniejsze wyniki uzyskano dla konfiguracji z n_neighbors = 7, weights = distance oraz leaf_size = 20. Ta kombinacja dała najwyższą średnią dokładność (0.6), co sugeruje, że dla testowanego zbioru danych te parametry stanowią optymalną konfigurację. Wartości te balansują wydajność i precyzję, zapewniając zarówno dobrą jakość klasyfikacji, jak i akceptowalny czas wykonania. Dla innych konfiguracji parametrów, jak np. leaf_size = 30, 40, 50, dokładność pozostaje na poziomie 0.59, co wskazuje na niewielki spadek precyzji w porównaniu do wartości 20.

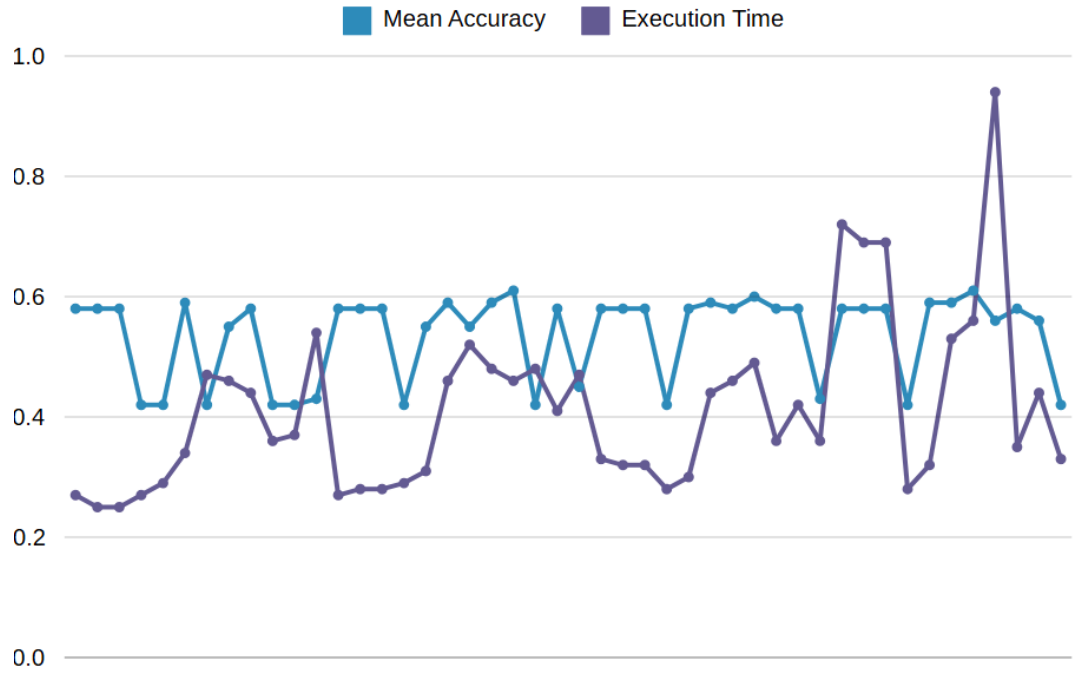
3.4 Porównanie otrzymanych wyników

Analizując uzyskane wyniki, możemy dojść do wniosku, iż największą średnią dokładność uzyskaliśmy korzystając z klasyfikatora SVN równą **61%**. Drugą wartością jest **60%** przy zastosowaniu KNN, a najmniejszą średnią dokładność uzyskaliśmy przy użyciu drzewa decyzyjnego równą **59%**. Natomiast pod względem czasu działania, przy określonych wyżej wynikach był najdłuższy w SVM, a najkrótszy w KNN.

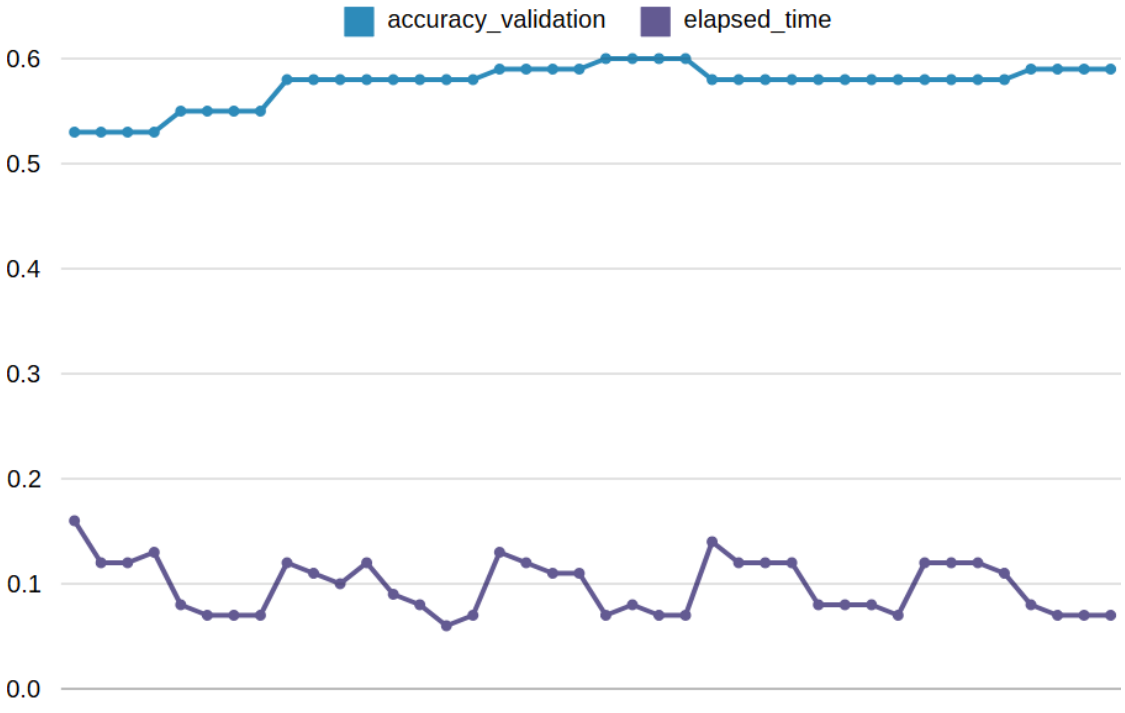
KNN	SVM	Drzewa decyzyjne
60%	61%	59%
0,07s	0,46s	0,09s



Rysunek 32 Poszczególne wyniki przy różnych parametrach dla drzewa decyzyjnego



Rysunek 34 Poszczególne wyniki przy różnych parametrach dla SVM



Rysunek 35 Poszczególne wyniki przy różnych parametrach dla KNN

Podsumowanie

Podczas realizacji eksperymentów zdobyliśmy cenne doświadczenie związane z analizą danych, optymalizacją parametrów oraz porównywaniem popularnych algorytmów klasyfikacyjnych. Proces przygotowania danych miał kluczowe znaczenie dla jakości modeli, co nauczyło nas, jak istotna jest jakość danych wejściowych. Zrozumieliśmy także, jak dobór odpowiednich parametrów wpływa na wyniki algorytmów. Zdobyliśmy umiejętności w interpretacji wyników oraz analizie wpływu danych na efektywność klasyfikacji, co pozwoliło nam lepiej zrozumieć przyczyny uzyskanych rezultatów

W ramach przeprowadzonych eksperymentów skupiliśmy się na analizie trzech algorytmów klasyfikacyjnych: Drzewa Decyzyjnego, Maszyny Wektorów Nośnych (SVM) oraz K-Najbliższych Sąsiadów (KNN). Celem było zbadanie ich wydajności, dokładności oraz stabilności przy różnych konfiguracjach parametrów, a także określenie, który z nich najlepiej nadaje się do klasyfikacji danych dotyczących jakości win.

SVM ma najgorszą wydajność w porównaniu do pozostałych algorytmów, szczególnie przy jądrze "sigmoid". Jednak przy odpowiednich parametrach (np. jądro "rbf", C = 1, gamma = 1.0) może osiągnąć dobre wyniki.

KNN okazała się najbardziej stabilnym algorytmem, osiągającym stosunkowo wysokie wyniki (średnia dokładność 0.59-0.6) przy różnych konfiguracjach, a także najkrótszy czas wykonania, co czyni go dobrym wyborem, gdy zależy nam na szybkości.

Drzewo Decyzyjne oferuje wyniki na poziomie 0.54-0.59. Wydaje się mniej czułe na zmiany parametrów w porównaniu do innych algorytmów. Ponadto, ma stabilny czas wykonania, ale nie osiąga najlepszych wyników, zwłaszcza w porównaniu z KNN.

Średni wynik działania algorytmów klasyfikujących, może wynikać z zbyt małej ilości rekordów, ponieważ duplikaty zostały usunięte, wartości odstające również, tak więc algorytmy działały na tylko 1038 rekordach. Zmienna docelowa (ocena jakości wina) może być subiektywna i zależeć od ludzkiej percepcji. Jeśli ocena była wykonywana przez degustatorów, istnieje ryzyko wprowadzenia szumu wynikającego z różnic w gustach lub innych niekontrolowanych czynników.

Bibliografia

[1] Akkio.com. (2024, 1 8). Pobrano z lokalizacji https://www.akkio.com/post/5-types-of-machine-learning-classification-algorithms
[2] Fatyga Piotr, P. R. (2023, 11 14). Klasyfikacja danych - przegląd wybranych metod, .
[3] geekforgeeks.org. (2024, 8 7). Pobrano z lokalizacji https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
[4] geeksforgeeks.org. (2024, 8 7). Pobrano z lokalizacji https://www.geeksforgeeks.org/cross-validation-machine-learning/
[5] https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009
[6] https://pandas.pydata.org/docs/user_guide/index.html#user-guide
[7] https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html
[8] https://stackoverflow.com/questions/34782063/how-to-use-pandas-filter-with-iqr
[9] https://www.datacamp.com/tutorial/decision-tree-classification-python?dc_referrer=https%3A%2F%2Fwww.google.com%2F
[10] https://towardsdatascience.com/what-is-stratified-cross-validation-in-machine-learning-8844f3e7ae8e
[11] https://scikit-learn.org/1.5/modules/svm.html
[12] https://www.w3schools.com/python/python_ml_knn.asp