Raport

Przedmiot: Analiza danych w biznesie

Temat: Preprocessing

Wykonał: Artur Kompała

1. Cel zadania oraz wybór zbioru danych

Celem zadania było przeprowadzenie preprocessingu na rzeczywistym zbiorze danych spełniającym kryteria: obecność braków danych, przynajmniej jeden atrybut ciągły i obecność obserwacji odstających.

Zbiór danych wykorzystany do projektu pochodzi z serwisu Kaggle. Zbiór danych zatytułowany Water Quality – Potability [1] zawiera informacje dotyczące jakości wody oraz jej przydatności do spożycia przez ludzi. Celem zbioru danych jest umożliwienie analizy parametrów jakości wody w kontekście oceny jej zdatności do picia. Każdy rekord w zbiorze reprezentuje pojedynczą próbkę wody, opisaną za pomocą dziewięciu atrybutów pomiarowych. Ostatnia kolumna Potability jest atrybutem decyzyjnym w tym zbiorze i wskazuje, czy dana próbka jest uznana za zdatną do spożycia (wartość 1), czy też nie (wartość 0) i to ten atrybut będzie klasyfikacyjnym.

Opis pozostałych atrybutów:

- pH – poziom pH wody; miernik kwasowości lub zasadowości cieczy.
- Hardness – twardość wody, czyli stężenie jonów wapnia i magnezu.
- Solids – całkowita ilość rozpuszczonych substancji stałych.
- Chloramines – stężenie chloramin w wodzie.
- Sulfate – stężenie siarczanów.
- Conductivity – przewodność elektryczna wody.
- Organic_carbon – zawartość węgla organicznego.
- Trihalomethanes – stężenie trójhalometanów.
- Turbidity – mętność wody.

2. Wstępna eksploracja danych

Zbiór danych składa się z 3276 obserwacji oraz 10 atrybutów, z których wszystkie poza kolumną docelową Potability mają charakter ciągły. Atrybuty opisujące próbki obejmują między innymi: ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes oraz Turbidity.

```
data.info()
✓ [3] 39ms

 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

```
data.describe()
✓ [4] 38ms
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

Znajdowanie brakujących wartości

```
missing_values = data.isnull().sum()
print("Missing values per column:\n", missing_values)
✓ [18] < 10 ms

Missing values per column:
 ph                 491
Hardness              0
Solids                0
Chloramines           0
Sulfate             781
Conductivity          0
Organic_carbon        0
Trihalomethanes     162
Turbidity             0
Potability            0
dtype: int64
```

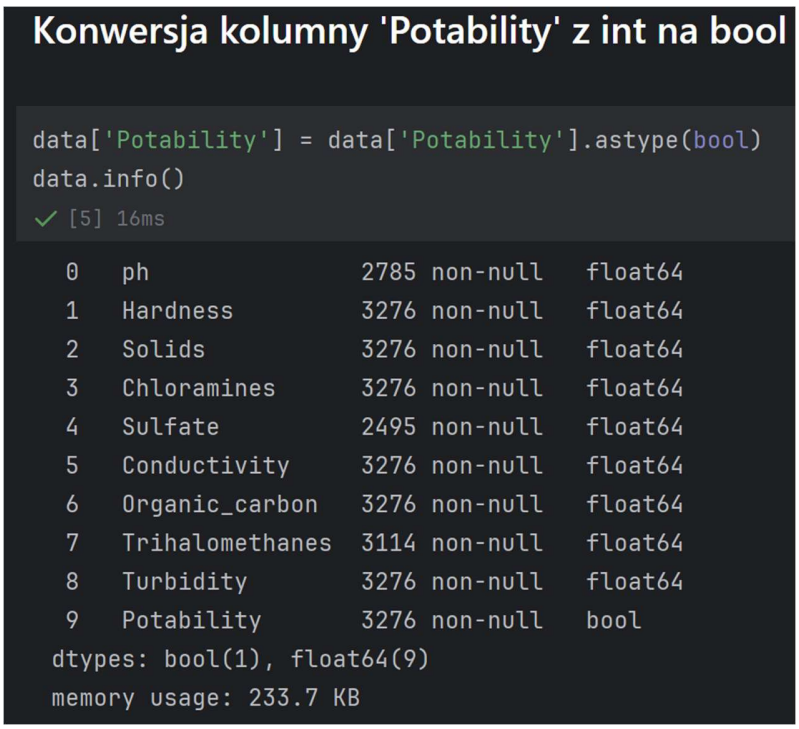
Wstępna analiza wykazała także braki danych w trzech kolumnach:

- ph – 491 brakujących wartości,
- Sulfate – 781 brakujących wartości,
- Trihalomethanes – 162 brakujących wartości.

Zaobserwowane braki danych mogą mieć wpływ na jakość, dlatego w kolejnych etapach konieczne było zastosowanie odpowiednich metod uzupełniania brakujących wartości.

3. Konwersja typu danych

Po załadowaniu zbioru danych dokonano także wstępnej analizy typów poszczególnych atrybutów. Stwierdzono, że atrybut decyzyjny Potability doskonale nadaje się do zadania klasyfikacji binarnej. Początkowo zmienna ta była zapisana jako liczba całkowita (0/1). W celu zapewnienia jednoznacznej interpretacji oraz lepszej zgodności z wymaganiami modeli klasyfikacyjnych, dokonano konwersji tej kolumny na typ logiczny (bool).



4. Postępowanie z brakującymi danymi

Postanowiono uzupełnić braki średnią z danej kolumny, ponieważ zmienne mają rozkład ciągły, a liczba braków nie jest ekstremalnie wysoka. To kompromis pomiędzy prostotą, a zachowaniem właściwości statystycznych oryginalnych danych.

Zastąpnienie brakujących wartości średnią kolumny

```
imputer = SimpleImputer(strategy='mean')
data['ph'] = imputer.fit_transform(data[['ph']])
data['Sulfate'] = imputer.fit_transform(data[['Sulfate']])
data['Trihalomethanes'] = imputer.fit_transform(data[['Trihalomethanes']])

✓ [7] 10ms
```

5. Usunięcie duplikatów

W celu poprawy jakości zbioru sprawdzono obecność zduplikowanych rekordów.

Wykrywanie duplikatów w danych

```
duplicated_data = data.duplicated().sum()
print(duplicated_data)
✓ [8] < 10 ms

0
```

Nie wykryto duplikatów, co nie wymagało dalszej akcji.

6. Detekcja i usuwanie obserwacji odstających (outlierów)

Przeprowadzono wykrywanie outlierów metodą rozstępu międzykwartylowego (IQR). Zidentyfikowano, ile obserwacji odstających znajduje się w każdej kolumnie ogółem kilkaset wartości odstających w całym zbiorze:

- ph – 142
- Hardness – 83
- Solids – 47
- Chloramines – 61
- Sulfate – 264
- Conductivity – 11
- Organic_carbon – 25
- Trihalomethanes – 54
- Turbidity – 19

Wykrywanie obserwacji odstających metodą rozstępu międzykwartylowego (IQR)

```python
def detect_outliers(df):
    outlier_counts = {}
    for col in df.select_dtypes(include=[float]).columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
        outlier_counts[col] = outliers.shape[0]
    print("Outlier counts per column:")
    print(outlier_counts)

detect_outliers(data)
```

✓ [33] 15ms

```
Outlier counts per column:
{'ph': 142, 'Hardness': 83, 'Solids': 47, 'Chloramines': 61, 'Sulfate': 264, 'Conductivity': 11, 'Organic_carbon': 25,
'Trihalomethanes': 54, 'Turbidity': 19}
```

Następnie usunięto obserwacje odstające końcowa liczność zbioru zmniejszyła się do 2657 rekordów.

```python
import pandas as pd

def remove_outliers(df):
    df_clean = df.copy()
    for col in df_clean.select_dtypes(include=[float]).columns:
        Q1 = df_clean[col].quantile(0.25)
        Q3 = df_clean[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df_clean = df_clean[(df_clean[col] >= lower_bound) & (df_clean[col] <= upper_bound)]
    return df_clean

data_no_outliers = remove_outliers(data)
print(data_no_outliers.count())
```

✓ [34] 16ms

```
ph                 2657
Hardness           2657
Solids             2657
Chloramines        2657
Sulfate            2657
Conductivity       2657
Organic_carbon     2657
Trihalomethanes    2657
Turbidity          2657
Potability         2657
dtype: int64
```

7. Skalowanie cech

W trakcie preprocessingu wypróbowano różne techniki skalowania danych. Analiza wyników modeli klasyfikacyjnych wykazała, że najlepszą skuteczność osiągnięto po zastosowaniu standaryzacji. W związku z tym standaryzacja została wybrana jako finalna metoda transformacji cech przed trenowaniem modeli.

```python
from sklearn.preprocessing import StandardScaler
print("Przed skalowaniem:")
print(data_no_outliers.select_dtypes(include=[float]).agg(['mean', 'std']))

data_std = data_no_outliers.copy()
scaler = StandardScaler()

for col in data_std.select_dtypes(include=[float]).columns:
    data_std[col] = scaler.fit_transform(data_std[[col]])

print("\nPo skalowaniu StandardScaler:")
print(data_std.select_dtypes(include=[float]).agg(['mean', 'std']))
```

✓ [11] 27ms

```
      Conductivity  Organic_carbon  Trihalomethanes  Turbidity
mean    425.738024       14.326481        66.476449   3.962308
std      79.907395        3.224768        14.976547   0.761498

Po skalowaniu StandardScaler:
                ph      Hardness        Solids   Chloramines       Sulfate  \
mean  1.176661e-16 -9.152550e-16  1.016207e-16 -1.149919e-16  1.353160e-15
std   1.000188e+00  1.000188e+00  1.000188e+00  1.000188e+00  1.000188e+00

      Conductivity  Organic_carbon  Trihalomethanes     Turbidity
mean  9.373174e-16    6.525120e-16    -5.321716e-16 -2.487033e-16
std   1.000188e+00    1.000188e+00     1.000188e+00  1.000188e+00
```

8. Porównanie wyników modeli klasyfikacyjnych

Przeprowadzono ocenę trzech modeli klasyfikacyjnych (Logistic Regression, KNN, SVM) na trzech wariantach danych:

- Dane surowe z zastąpionymi brakami
- Dane bez outlierów i zastąpionymi brakami
- Dane bez outlierów, po skalowaniu i zastąpionymi brakami

Uzyskane wyniki (dokładność):

Model	Dane surowe	Bez outlierów	Bez outlierów + skalowanie
Logistic Regression	62.8%	65.4%	65.4%
KNeighborsClassifier	54.1%	58.6%	62.6%
SVM	57.2%	60.9%	63.0%

```python
def evaluate_models(df):
    X = df.drop('Potability', axis=1)
    y = df['Potability']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    models = {
        "Logistic Regression": LogisticRegression(max_iter=1000),
        "KNeighborsClassifier": KNeighborsClassifier(),
        "SVM": SVC(class_weight='balanced')
    }
    results = {}
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        results[name] = accuracy_score(y_test, y_pred)
    return results

results_raw = evaluate_models(data)
results_no_outliers = evaluate_models(data_no_outliers)
results_no_outliers_and_std = evaluate_models(data_std)

comparison_df = pd.DataFrame({
    "Raw Data": results_raw,
    "No Outliers Data": results_no_outliers,
    "No Outliers & Std Data": results_no_outliers_and_std
})
print(comparison_df)
```

✓ [12] 1s 120ms

```
                     Raw Data  No Outliers Data  No Outliers & Std Data
Logistic Regression  0.628049          0.654135                0.654135
KNeighborsClassifier 0.541159          0.586466                0.625940
SVM                  0.571646          0.609023                0.629699
```

9. Wnioski

Preprocessing danych przyniósł zamierzony efekt. Dzięki usunięciu braków, oczyszczeniu danych z obserwacji odstających oraz odpowiedniemu skalowaniu cech udało się poprawić jakość danych wejściowych, co skutkowało wzrostem dokładności modeli klasyfikacyjnych. Eksperymenty wykazały, że wybór standaryzacji jako metody skalowania cech był optymalny w kontekście uzyskiwanych wyników. Podsumowując, wdrożone kroki preprocessingowe zadziałały zgodnie z oczekiwaniami, umożliwiając skuteczniejsze uczenie modeli i lepsze przewidywania na tym zbiorze danych.

10. Bibliografia

[1] Serwis internetowy Kaggle, Water Quality and Potability, https://www.kaggle.com/datasets/uom190346a/water-quality-and-potability