

```
In [1]: import pandas as pd
        from scipy.io import arff

        # Wczytanie pliku ARFF (upewnij się, że plik jest w tym samym katalogu co notebo
        data, meta = arff.loadarff("seismic-bumps.arff")

        # Konwersja do Pandas DataFrame
        df = pd.DataFrame(data)

        # Dekodowanie wartości kategoriycznych (jeśli istnieją)
        for col in df.select_dtypes([object]):
            df[col] = df[col].str.decode('utf-8')

        # Podział na macierz cech (X) i wektor etykiet (y)
        X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej jako cechy
        y = df.iloc[:, -1]  # Ostatnia kolumna jako etykiety

        # Wyświetlenie pierwszych 5 wierszy macierzy cech i etykiet
        print("Macierz cech (X):")
        display(X.head()) # Jeśli używasz Jupyter Notebook

        print("\nWektor etykiet (y):")
        display(y.head()) # Jeśli używasz Jupyter Notebook
```

Macierz cech (X):

	seismic	seismoacoustic	shift	genergy	gpuls	gdenergy	gdpuls	ghazard	nbumps
0	a	a	N	15180.0	48.0	-72.0	-72.0	a	0.0
1	a	a	N	14720.0	33.0	-70.0	-79.0	a	1.0
2	a	a	N	8050.0	30.0	-81.0	-78.0	a	0.0
3	a	a	N	28820.0	171.0	-23.0	40.0	a	1.0
4	a	a	N	12640.0	57.0	-63.0	-52.0	a	0.0

Wektor etykiet (y):

```
0    0
1    0
2    0
3    0
4    0
```

Name: class, dtype: object

```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt
        from scipy.io import arff

        # Wczytanie pliku ARFF
        data, meta = arff.loadarff("seismic-bumps.arff") # Upewnij się, że plik jest w

        # Konwersja do Pandas DataFrame
        df = pd.DataFrame(data)

        # Dekodowanie wartości kategoriycznych (jeśli istnieją)
        for col in df.select_dtypes([object]):
            df[col] = df[col].str.decode('utf-8')
```

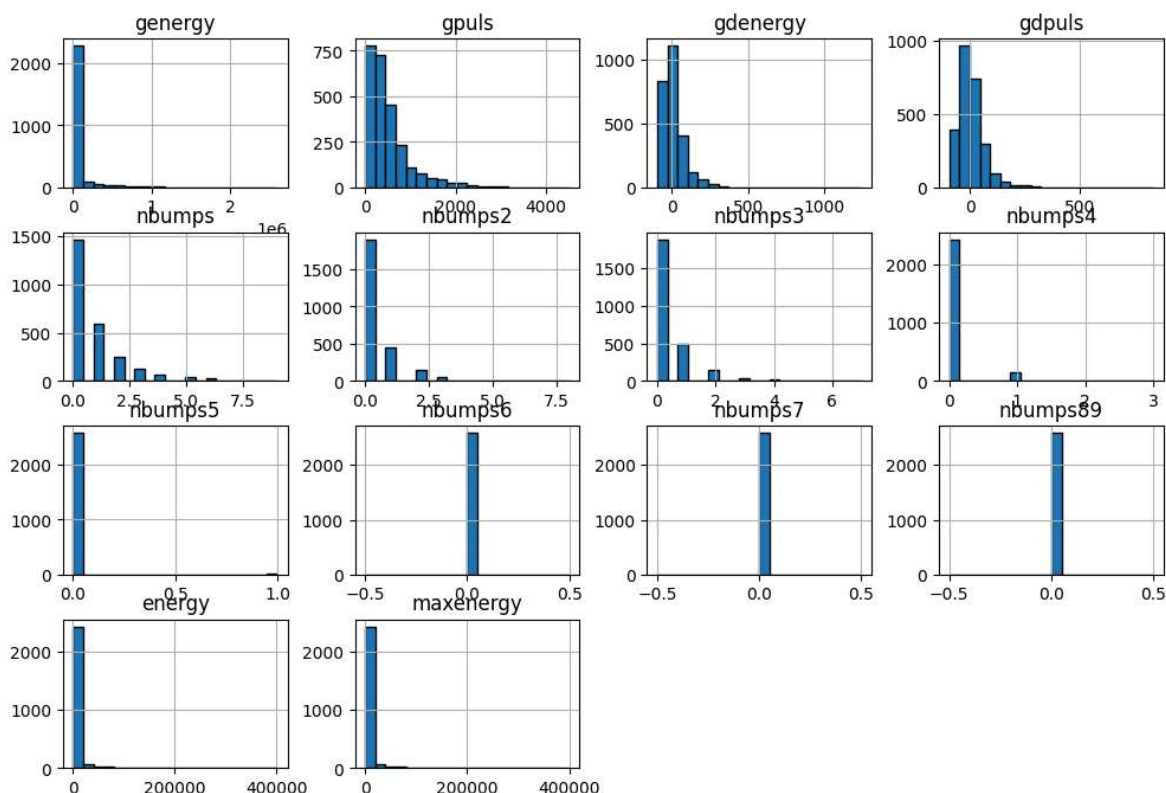
```
# Podział na macierz cech (X) i wektor etykiet (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Tworzenie histogramów dla wszystkich cech numerycznych
X.hist(figsize=(12, 8), bins=20, edgecolor='black')
plt.suptitle("Histogramy cech numerycznych", fontsize=14)
plt.show()
```

C:\Users\krzys\AppData\Local\Programs\Python\Python310\lib\site-packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) DejaVu Sans.

```
fig.canvas.print_figure(bytes_io, **kw)
```

□ Histogramy cech numerycznych



```
In [4]: import pandas as pd
from scipy.io import arff
from sklearn.model_selection import train_test_split

# Wczytanie pliku ARFF
data, meta = arff.loadarff("seismic-bumps.arff") # Upewnij się, że plik jest w

# Konwersja do Pandas DataFrame
df = pd.DataFrame(data)

# Dekodowanie wartości kategorycznych (jeśli istnieją)
for col in df.select_dtypes([object]):
    df[col] = df[col].str.decode('utf-8')

# Podział na cechy (X) i etykiety (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Podział na zbiór treningowy (80%) i testowy (20%)
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Połączenie cech i etykiet w jeden DataFrame
train_df = pd.concat([X_train, y_train], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)

# Zapisanie danych do plików CSV
train_df.to_csv("train_data.csv", index=False)
test_df.to_csv("test_data.csv", index=False)

print(" Dane zapisane do plików: train_data.csv i test_data.csv")

```

Dane zapisane do plików: train_data.csv i test_data.csv

```

In [6]: import pandas as pd
        from scipy.io import arff
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler, OneHotEncoder
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score, classification_report

# Wczytanie pliku ARFF
data, meta = arff.loadarff("seismic-bumps.arff")

# Konwersja do Pandas DataFrame
df = pd.DataFrame(data)

# Dekodowanie wartości kategoriycznych (jeśli istnieją)
for col in df.select_dtypes([object]):
    df[col] = df[col].str.decode('utf-8')

# Podział na cechy (X) i etykiety (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Sprawdzenie, które kolumny są tekstowe (kategoriyczne)
categorical_cols = X.select_dtypes(include=['object']).columns

# Konwersja zmiennych kategoriycznych na liczbowe (One-Hot Encoding)
X = pd.get_dummies(X, columns=categorical_cols)

# Podział na zbiór treningowy (80%) i testowy (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Standaryzacja cech (teraz już tylko numeryczne)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Trenowanie modelu regresji logistycznej
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)

# Predykcja na zbiorze testowym
y_pred = log_reg.predict(X_test_scaled)

# Ocena modelu
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

```

```
# Wyświetlenie wyników
print(f" Dokładność modelu: {accuracy:.4f}")
print("\n Raport klasyfikacji:")
print(classification_rep)
```

Dokładność modelu: 0.9284

Raport klasyfikacji:

	precision	recall	f1-score	support
0	0.94	0.99	0.96	483
1	0.20	0.03	0.05	34
accuracy			0.93	517
macro avg	0.57	0.51	0.51	517
weighted avg	0.89	0.93	0.90	517

```
In [8]: import pandas as pd
from scipy.io import arff
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Wczytanie pliku ARFF
data, meta = arff.loadarff("seismic-bumps.arff")

# Konwersja do Pandas DataFrame
df = pd.DataFrame(data)

# Dekodowanie wartości kategorycznych (jeśli istnieją)
for col in df.select_dtypes([object]):
    df[col] = df[col].str.decode('utf-8')

# Podział na cechy (X) i etykiety (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Sprawdzenie, które kolumny są tekstowe (kategoryczne)
categorical_cols = X.select_dtypes(include=['object']).columns

# Konwersja zmiennych kategorycznych na liczbowe (One-Hot Encoding)
X = pd.get_dummies(X, columns=categorical_cols)

# Podział na zbiór treningowy (80%) i testowy (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Standaryzacja cech
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Trenowanie modelu SVM (Maszyna Wektorów Nośnych) z class_weight="balanced"
svm_model = SVC(kernel="linear", C=1.0, class_weight="balanced", random_state=42)
svm_model.fit(X_train_scaled, y_train)

# Predykcja na zbiorze testowym
y_pred_svm = svm_model.predict(X_test_scaled)
```

```
# Ocena modelu (dodanie zero_division=1, aby uniknąć ostrzeżeń)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
classification_rep_svm = classification_report(y_test, y_pred_svm, zero_division=1)

# Wyświetlenie wyników
print(f" Dokładność modelu SVM: {accuracy_svm:.4f}")
print("\n Raport klasyfikacji SVM:")
print(classification_rep_svm)
```

Dokładność modelu SVM: 0.7988

Raport klasyfikacji SVM:

	precision	recall	f1-score	support
0	0.97	0.81	0.88	483
1	0.18	0.59	0.28	34
accuracy			0.80	517
macro avg	0.57	0.70	0.58	517
weighted avg	0.91	0.80	0.84	517

```
In [9]: import pandas as pd
from scipy.io import arff
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Wczytanie pliku ARFF
data, meta = arff.loadarff("seismic-bumps.arff")

# Konwersja do Pandas DataFrame
df = pd.DataFrame(data)

# Dekodowanie wartości katagorycznych (jeśli istnieją)
for col in df.select_dtypes([object]):
    df[col] = df[col].str.decode('utf-8')

# Podział na cechy (X) i etykiety (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Sprawdzenie, które kolumny są tekstowe (katagoryczne)
categorical_cols = X.select_dtypes(include=[object]).columns

# Konwersja zmiennych katagorycznych na liczbowe (One-Hot Encoding)
X = pd.get_dummies(X, columns=categorical_cols)

# Podział na zbiór treningowy (80%) i testowy (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Standaryzacja cech
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Trenowanie modelu k-Nearest Neighbors (kNN)
knn_model = KNeighborsClassifier(n_neighbors=5) # Można dostosować liczbę sąsia
knn_model.fit(X_train_scaled, y_train)
```

```

# Predykcja na zbiorze testowym
y_pred_knn = knn_model.predict(X_test_scaled)

# Ocena modelu
accuracy_knn = accuracy_score(y_test, y_pred_knn)
classification_rep_knn = classification_report(y_test, y_pred_knn, zero_division=0)

# Wyświetlenie wyników
print(f" Dokładność modelu kNN: {accuracy_knn:.4f}")
print("\n Raport klasyfikacji kNN:")
print(classification_rep_knn)

```

Dokładność modelu kNN: 0.9342

Raport klasyfikacji kNN:

	precision	recall	f1-score	support
0	0.94	0.99	0.97	483
1	0.50	0.09	0.15	34
accuracy			0.93	517
macro avg	0.72	0.54	0.56	517
weighted avg	0.91	0.93	0.91	517

```

In [11]: import pandas as pd
from scipy.io import arff
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Wczytanie pliku ARFF
data, meta = arff.loadarff("seismic-bumps.arff")

# Konwersja do Pandas DataFrame
df = pd.DataFrame(data)

# Dekodowanie wartości katagorycznych (jeśli istnieją)
for col in df.select_dtypes([object]):
    df[col] = df[col].str.decode('utf-8')

# Podział na cechy (X) i etykiety (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Sprawdzenie, które kolumny są tekstowe (katagoryczne)
categorical_cols = X.select_dtypes(include=['object']).columns

# Konwersja zmiennych katagorycznych na liczbowe (One-Hot Encoding)
X = pd.get_dummies(X, columns=categorical_cols)

# Podział na zbiór treningowy (80%) i testowy (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Standaryzacja cech
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```

```

# Definicja przestrzeni hiperparametrów dla kNN
knn_params = {
    'n_neighbors': range(1, 21), # Liczba sąsiadów od 1 do 20
    'weights': ['uniform', 'distance'], # Sposób ważenia sąsiadów
    'metric': ['euclidean', 'manhattan', 'minkowski'] # Metryka odległości
}

# Wyszukiwanie najlepszych hiperparametrów za pomocą Grid Search
grid_search_knn = GridSearchCV(KNeighborsClassifier(), knn_params, cv=5, scoring='accuracy')
grid_search_knn.fit(X_train_scaled, y_train)

# Najlepsze parametry i dokładność
best_knn_params = grid_search_knn.best_params_
best_knn_score = grid_search_knn.best_score_

# Wyświetlenie wyników Grid Search
print("✅ Najlepsze hiperparametry dla kNN:", best_knn_params)
print(f"📊 Najlepsza dokładność na zbiorze treningowym: {best_knn_score:.4f}")

# Trening najlepszego modelu kNN
best_knn = KNeighborsClassifier(**best_knn_params)
best_knn.fit(X_train_scaled, y_train)

# Predykcja na zbiorze testowym
y_pred_knn_best = best_knn.predict(X_test_scaled)

# Ocena najlepszego modelu
accuracy_knn_best = accuracy_score(y_test, y_pred_knn_best)
classification_rep_knn_best = classification_report(y_test, y_pred_knn_best, zero_division=0)

# Wyświetlenie wyników najlepszego modelu kNN
print(f"\n Dokładność najlepszego modelu kNN na zbiorze testowym: {accuracy_knn_best:.4f}")
print("\n Raport klasyfikacji dla najlepszego kNN:")
print(classification_rep_knn_best)

```

✅ Najlepsze hiperparametry dla kNN: {'metric': 'euclidean', 'n_neighbors': 20, 'weights': 'uniform'}

📊 Najlepsza dokładność na zbiorze treningowym: 0.9342

Dokładność najlepszego modelu kNN na zbiorze testowym: 0.9342

Raport klasyfikacji dla najlepszego kNN:

	precision	recall	f1-score	support
0	0.93	1.00	0.97	483
1	1.00	0.00	0.00	34
accuracy			0.93	517
macro avg	0.97	0.50	0.48	517
weighted avg	0.94	0.93	0.90	517

```

In [13]: import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import arff
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.manifold import TSNE

```



```
# Wczytanie pliku ARFF
data, meta = arff.loadarff("seismic-bumps.arff")

# Konwersja do Pandas DataFrame
df = pd.DataFrame(data)

# Dekodowanie wartości kategoriycznych (jeśli istnieją)
for col in df.select_dtypes([object]):
    df[col] = df[col].str.decode('utf-8')

# Podział na cechy (X) i etykiety (y)
X = df.iloc[:, :-1] # Wszystkie kolumny oprócz ostatniej
y = df.iloc[:, -1]  # Ostatnia kolumna jako etykieta

# Sprawdzenie, które kolumny są tekstowe (kategoriyczne)
categorical_cols = X.select_dtypes(include=['object']).columns

# Konwersja zmiennych kategoriycznych na liczbowe (One-Hot Encoding)
X = pd.get_dummies(X, columns=categorical_cols)

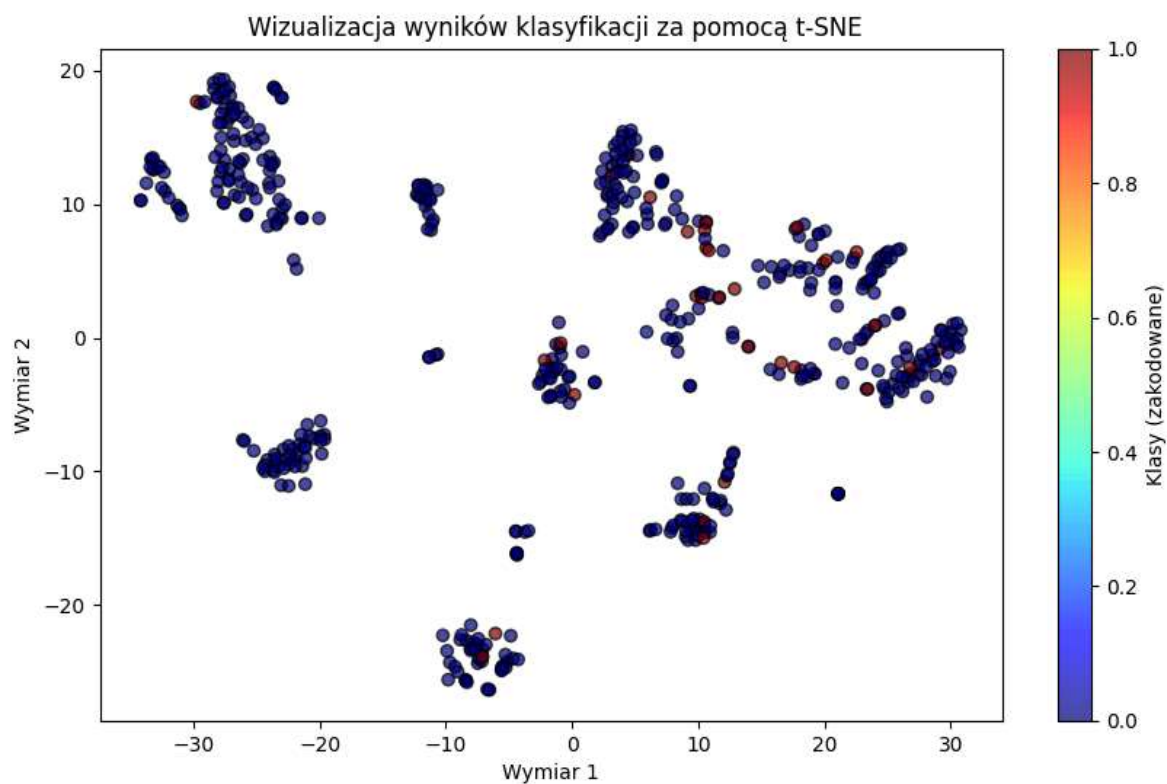
# Konwersja etykiet na liczby (jeśli są kategoriczne)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y) # Zamienia klasy na liczby

# Podział na zbiór treningowy (80%) i testowy (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,

# Standaryzacja cech
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Redukcja wymiarowości za pomocą t-SNE (2D)
tsne = TSNE(n_components=2, random_state=42, perplexity=30, learning_rate=200)
X_tsne = tsne.fit_transform(X_test_scaled)

# Tworzenie wykresu t-SNE
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_test, cmap="jet", alpha=0.
plt.colorbar(scatter, label="Klasy (zakodowane)")
plt.title("Wizualizacja wyników klasyfikacji za pomocą t-SNE")
plt.xlabel("Wymiar 1")
plt.ylabel("Wymiar 2")
plt.show()
```

In []: