

# UMB W03: dostęp do danych

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

import umb_tools as umb
```

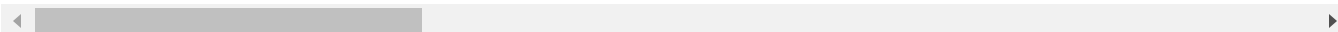
```
# konfiguracja
plt.rcParams["figure.figsize"] = [5, 4]
pd.set_option("display.float_format", lambda x: "%.4f" % x)
```

## 1. Wczytywanie zbioru danych

```
# odczyt pliku TSV (zwracane są: zbiór danych w postaci DataFrame biblioteki Pandas oraz lista
(df, c_names) = umb.read_data("data/BreastCancer.txt")
df
```

#BreastCancer	labels	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cor
842302	1	17.9900	10.3800	122.8000	1001.0000	0.1184	
874858	1	14.2200	23.1200	94.3700	609.9000	0.1075	
875263	1	12.3400	26.8600	81.1500	477.4000	0.1034	
87556202	1	14.8600	23.2100	100.4000	671.4000	0.1044	
875938	1	13.7700	22.2900	90.6300	588.9000	0.1200	
...	...	...	...	...	...	...	...
8910720	2	10.7100	20.3900	69.5000	344.9000	0.1082	
8910506	2	12.8700	16.2100	82.3800	512.2000	0.0943	
8910499	2	13.5900	21.8400	87.1600	561.0000	0.0796	
8912055	2	11.7400	14.0200	74.2400	427.3000	0.0781	
92751	2	7.7600	24.5400	47.9200	181.0000	0.0526	

569 rows × 31 columns



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 569 entries, 842302 to 92751
Data columns (total 31 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   labels                                569 non-null    int32
1   radius_mean                          569 non-null    float64
2   texture_mean                         569 non-null    float64
3   perimeter_mean                      569 non-null    float64
4   area_mean                          569 non-null    float64
5   smoothness_mean                    569 non-null    float64
6   compactness_mean                   569 non-null    float64
7   concavity_mean                    569 non-null    float64
8   concave points_mean               569 non-null    float64
9   symmetry_mean                     569 non-null    float64
10  fractal_dimension_mean            569 non-null    float64
11  radius_se                         569 non-null    float64
12  texture_se                        569 non-null    float64
13  perimeter_se                     569 non-null    float64
14  area_se                          569 non-null    float64
15  smoothness_se                   569 non-null    float64
16  compactness_se                  569 non-null    float64
17  concavity_se                    569 non-null    float64
18  concave points_se               569 non-null    float64
19  symmetry_se                     569 non-null    float64
20  fractal_dimension_se            569 non-null    float64
21  radius_worst                    569 non-null    float64
22  texture_worst                   569 non-null    float64
23  perimeter_worst                 569 non-null    float64
24  area_worst                      569 non-null    float64
25  smoothness_worst                569 non-null    float64
26  compactness_worst               569 non-null    float64
27  concavity_worst                 569 non-null    float64
28  concave points_worst            569 non-null    float64
29  symmetry_worst                  569 non-null    float64
30  fractal_dimension_worst         569 non-null    float64
dtypes: float64(30), int32(1)
memory usage: 140.0+ KB
```

```
# pobranie etykiet klas
labels = df.iloc[:, 0].to_numpy()

# pobranie macierzy danych
data = df.iloc[:, 1:].to_numpy()

# pobieranie nazw próbek
samples = df.index

# wymiary macierzy danych
print(f"\nSamples: {data.shape[0]} \nFeatures: {data.shape[1]}")
```

Samples: 569  
Features: 30

```
# pobranie indeksów próbek z klas
(c_labels, c_index) = umb.class_info(labels)

# liczba klas
c_n = len(c_labels)

# informacje o klasach
print(f"\nClasses: {c_n}")
for i in range(c_n):
    print(f"  name = {c_names[i]}, label = {c_labels[i]}, samples = {len(c_index[i])}")
```

```
Classes: 2
  name = Malignant, label = 1, samples = 212
  name = Benign, label = 2, samples = 357
```

---

## 2. Podstawowe właściwości atrybutów

```
# numer atrybutu
feature_nr = 1

# wartości atrybutu
f_data = data[:, feature_nr]
```

```
# podstawowe właściwości statystyczne we wszystkich próbkach
print(f"\nmin = {np.min(f_data):.4f}\nmax = {np.max(f_data):.4f}")
print(f"mean = {np.mean(f_data):.4f}\nstd. dev. = {np.std(f_data):.4f}")

# podstawowe właściwości statystyczne w ramach klas
for i in range(c_n):
    print(f"\n{c_names[i]}:")

    print(f"  min = {np.min(f_data[c_index[i]]):.4f}\n  max = {np.max(f_data[c_index[i]]):.4f}")
    print(f"  mean = {np.mean(f_data[c_index[i]]):.4f}\n  std. dev. = {np.std(f_data[c_index[i]]):.4f}")
```

```
min = 9.7100
max = 39.2800
mean = 19.2896
std. dev. = 4.2973
```

```
Malignant:
  min = 10.3800
  max = 39.2800
  mean = 21.6049
  std. dev. = 3.7705
```

```
Benign:
  min = 9.7100
  max = 33.8100
  mean = 17.9148
  std. dev. = 3.9895
```

---

### 3. Wizualizacja pojedynczych atrybutów

---

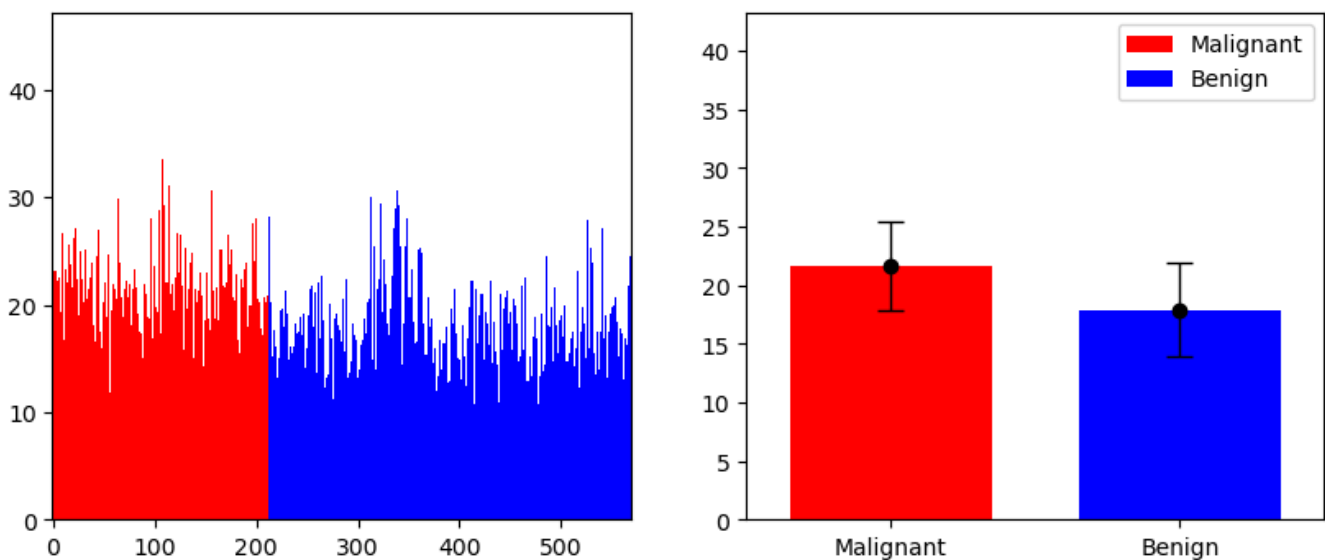
#### Wykresy słupkowe wartości i średnich w klasach

```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

# wartości poszczególnych próbek
umb.bar_plot(f_data, labels, c_names, ax[0])

# wartości średnie w klasach
umb.bar_plot_mean(f_data, labels, c_names, ax[1])

plt.show()
```



---

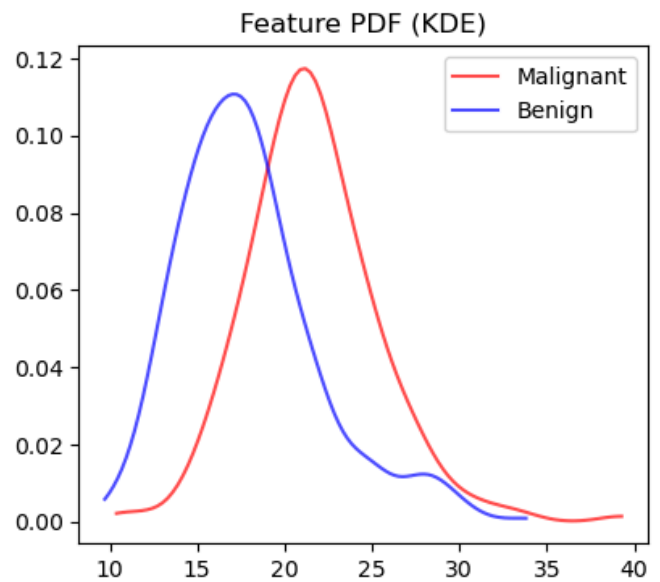
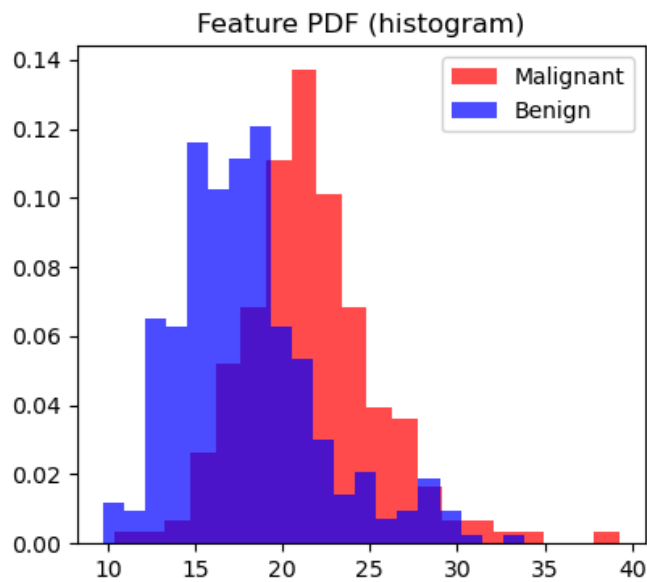
#### Estymowane rozkłady prawdopodobieństwa w klasach

```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

# funkcja gęstości prawdopodobieństwa estymowana za pomocą histogramu
umb.histogram(f_data, labels, c_names, ax[0])
ax[0].set_title("Feature PDF (histogram)")

# funkcja gęstości prawdopodobieństwa estymowana za pomocą KDE
umb.kde_plot(f_data, labels, c_names, ax[1])
ax[1].set_title("Feature PDF (KDE)")

plt.show()
```



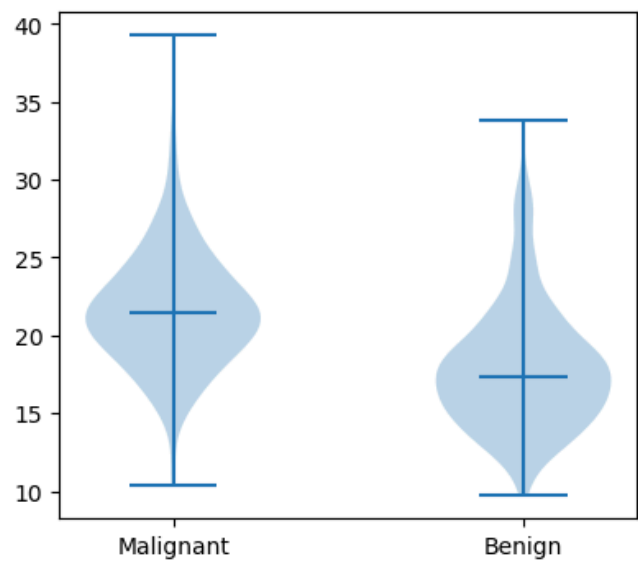
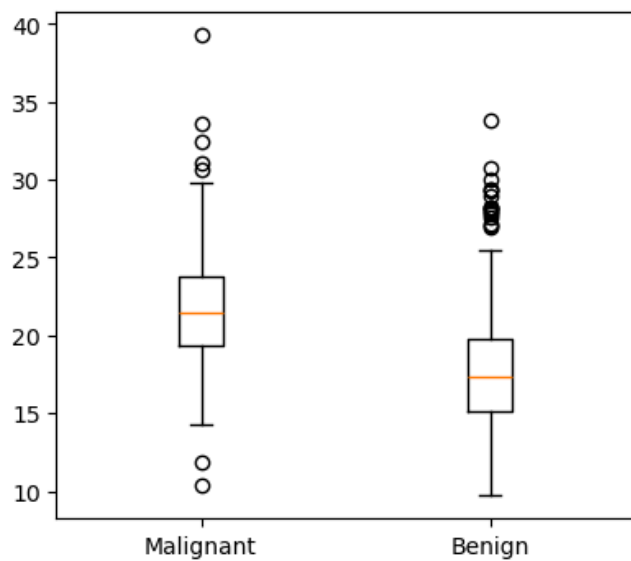
## Rozrzuty wartości w klasach

```
fig, ax = plt.subplots(1, 2, figsize=(10, 4))

# box plot
umb.box_plot(f_data, labels, c_names, ax[0])

# violon plot
umb.violin_plot(f_data, labels, c_names, ax[1])

plt.show()
```



---

## 4. Wizualizacja wielu atrybutów

---

### Wykresy punktowe par atrybutów

```
# numery atrybutów
feature_nr = [1, 2, 3]

# pobranie wartości atrybutów
f_data = data[:, feature_nr]

print(data.shape)
```

(569, 30)

```
n = len(feature_nr)

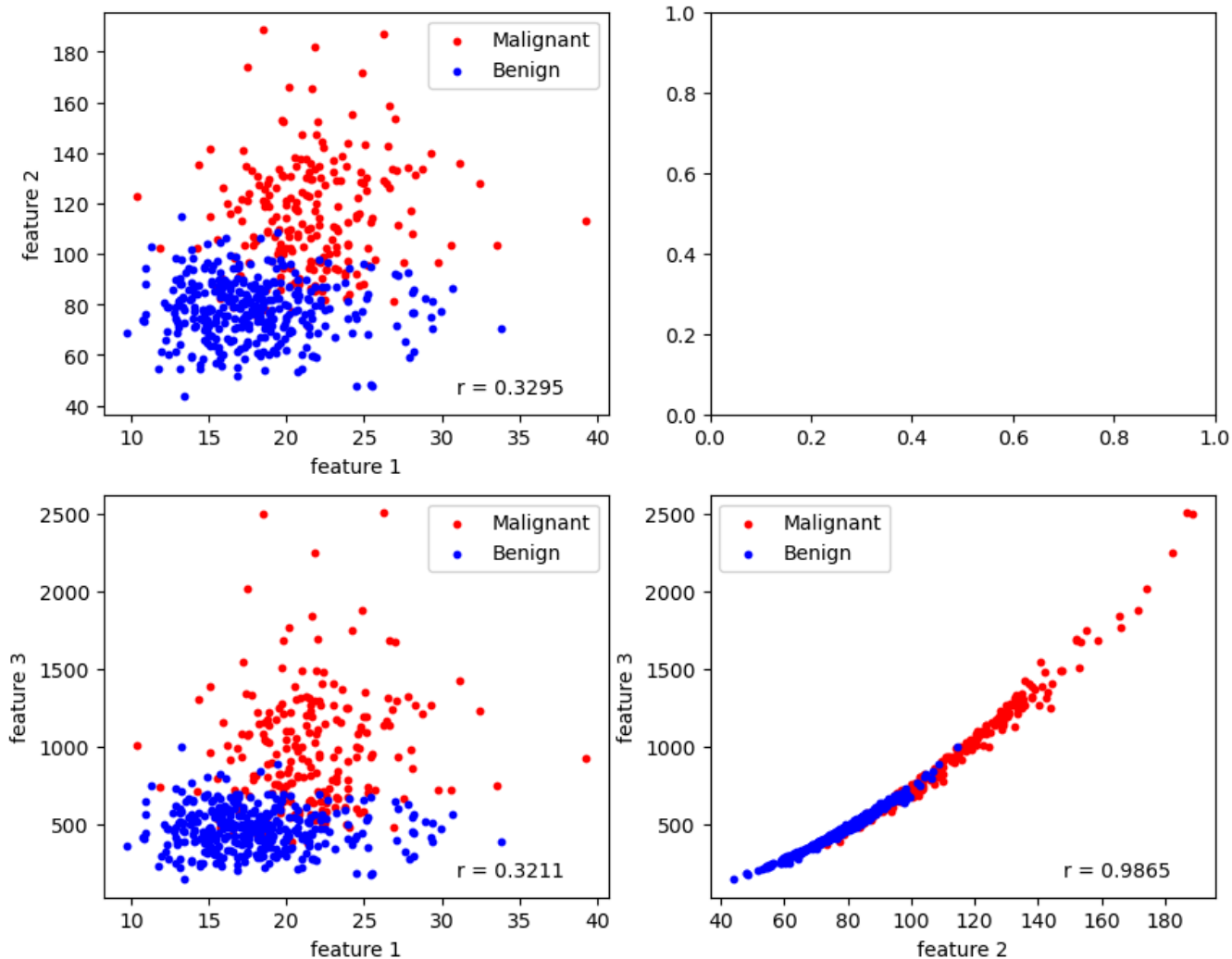
fig, ax = plt.subplots(n-1, n-1, figsize=(10, 8))

for k in range(0, n):
    for l in range(k+1, n):
        x = f_data[:, k]
        y = f_data[:, l]

        umb.scatter_plot(x, y, labels, c_names, ax[l-1][k])

        ax[l-1][k].set_xlabel(f"feature {feature_nr[k]}")
        ax[l-1][k].set_ylabel(f"feature {feature_nr[l]}")

plt.show()
```



## Heat map

```
from matplotlib.colors import LinearSegmentedColormap
```

```
# standaryzacja danych
m = np.mean(data, axis=0)
s = np.std(data, axis=0)
map = (data - m) / s

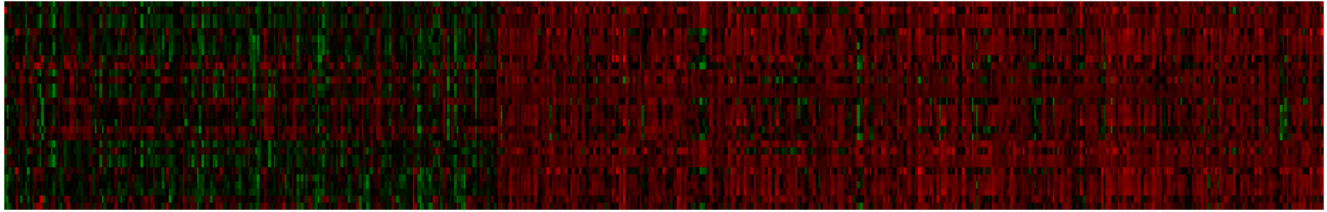
# ograniczenie zakresu zmienności danych
map[map>4] = 4
map[map<-4] = -4

# zdefiniowanie mapy kolorów
cmap = LinearSegmentedColormap.from_list("RedBlackGreen", ["r", "k", "g"], N=256)
#cmap = "seismic"

# rysowanie
fig, ax = plt.subplots(figsize=(12, 4))
```



```
plt.imshow(map.T, interpolation="none", cmap=cmap, aspect=3)
plt.axis("off");
```



---

## Analiza składowych głównych (PCA, *Principal Component Analysis*)

```
fig = plt.figure(figsize=(10, 4))

# wykres 2D
ax = fig.add_subplot(1, 2, 1)
umb.pca_plot(data, labels, c_names, ax)

# wykres 3D
ax = fig.add_subplot(1, 2, 2, projection="3d")
umb.pca_plot_3d(data, labels, c_names, ax)

plt.show()
```

