

Lista 1

Wczytanie zbioru IRIS, Wine, GLASS

Zainportowanie potrzebnych bibliotek...

```
In [ ]: from ucimlrepo import fetch_ucirepo
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import decomposition
import numpy as np
```

Wczytanie zbioru IRIS będący zbiorem danych dotyczącym 3 gatunków irysów: setosa, versicolor, virginica. Atrybutami są:

- długość kielicha
- szerokość kielicha
- długość płatka
- szerokość płatka

Wypisanie pierwszych 20 wierszy zbioru danych.

```
In [ ]: iris = fetch_ucirepo(id=53)

iris_X = iris.data.features
iris_y = iris.data.targets

iris_df = pd.concat([iris_X, iris_y], axis=1)

iris_df.head(20)
```

```
Out[ ]:
```

	sepal length	sepal width	petal length	petal width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

Wczytanie zbioru GLASS, który zawiera dane dotyczące sześciu klas szkła. Atrybutami są:

- RI: współczynnik załamania światła
- Na: zawartość sodu
- Mg: zawartość magnezu
- Al: zawartość glinu
- Si: zawartość krzemiu
- K: zawartość potasu
- Ca: zawartość wapnia
- Ba: zawartość baru
- Fe: zawartość żelaza

Wypisanie pierwszych 20 wierszy zbioru danych.

```
In [ ]: glass_identification = fetch_ucirepo(id=42)

glass_X = glass_identification.data.features
glass_y = glass_identification.data.targets

glass_df = pd.concat([glass_X, glass_y], axis=1)

glass_df.head(20)
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.00	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.00	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.00	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.00	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.00	1
5	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	1
6	1.51743	13.30	3.60	1.14	73.09	0.58	8.17	0.0	0.00	1
7	1.51756	13.15	3.61	1.05	73.24	0.57	8.24	0.0	0.00	1
8	1.51918	14.04	3.58	1.37	72.08	0.56	8.30	0.0	0.00	1
9	1.51755	13.00	3.60	1.36	72.99	0.57	8.40	0.0	0.11	1
10	1.51571	12.72	3.46	1.56	73.20	0.67	8.09	0.0	0.24	1
11	1.51763	12.80	3.66	1.27	73.01	0.60	8.56	0.0	0.00	1
12	1.51589	12.88	3.43	1.40	73.28	0.69	8.05	0.0	0.24	1
13	1.51748	12.86	3.56	1.27	73.21	0.54	8.38	0.0	0.17	1
14	1.51763	12.61	3.59	1.31	73.29	0.58	8.50	0.0	0.00	1
15	1.51761	12.81	3.54	1.23	73.24	0.58	8.39	0.0	0.00	1
16	1.51784	12.68	3.67	1.16	73.11	0.61	8.70	0.0	0.00	1
17	1.52196	14.36	3.85	0.89	71.36	0.15	9.15	0.0	0.00	1
18	1.51911	13.90	3.73	1.18	72.12	0.06	8.89	0.0	0.00	1
19	1.51735	13.02	3.54	1.69	72.73	0.54	8.44	0.0	0.07	1

Wczytanie zbioru Wine zawierającego dane dotyczące 3 klas wina. Atrybutami są:

- Alcohol - zawartość alkoholu
- Malic acid - kwas jabłkowy
- Ash - popiół
- Alcalinity of ash - zasadowość popiołu
- Magnesium - magnez
- Total phenols - całkowita zawartość fenoli
- Flavonoids - flawonoidy
- Nonflavanoid phenols - nieflawonoidowe fenole
- Proanthocyanins - proantocyjanidyny
- Color intensity - intensywność koloru
- Hue - barwa
- OD280/OD315 of diluted wines - stosunek absorbancji
- Proline - prolin

Wypisanie pierwszych 20 wierszy zbioru danych.

```
In [ ]: wine = fetch_uci_repo(id=109)

wine_X = wine.data.features
wine_y = wine.data.targets

wine_df = pd.concat([wine_X, wine_y], axis=1)

wine_df.head(20)
```

	Alcohol	Malicacid	Ash	Alkalinity_of_ash	Magnesium	Total_phenols	Flavonoids	Nonflavanoid_phenols	Proanthocyanins	Color_intensity	Hue	OD280_OD315_of_diluted_wines	Proline	
0	14.23	1.71	2.43		15.6	127	2.80	3.06	0.28	2.29		5.64	1.04	3.92
1	13.20	1.78	2.14		11.2	100	2.65	2.76	0.26	1.28		4.38	1.05	3.40
2	13.16	2.36	2.67		18.6	101	2.80	3.24	0.30	2.81		5.68	1.03	3.17
3	14.37	1.95	2.50		16.8	113	3.85	3.49	0.24	2.18		7.80	0.86	3.45
4	13.24	2.59	2.87		21.0	118	2.80	2.69	0.39	1.82		4.32	1.04	2.93
5	14.20	1.76	2.45		15.2	112	3.27	3.39	0.34	1.97		6.75	1.05	2.85
6	14.39	1.87	2.45		14.6	96	2.50	2.52	0.30	1.98		5.25	1.02	3.58
7	14.06	2.15	2.61		17.6	121	2.60	2.51	0.31	1.25		5.05	1.06	3.58
8	14.83	1.64	2.17		14.0	97	2.80	2.98	0.29	1.98		5.20	1.08	2.85
9	13.86	1.35	2.27		16.0	98	2.98	3.15	0.22	1.85		7.22	1.01	3.55
10	14.10	2.16	2.30		18.0	105	2.95	3.32	0.22	2.38		5.75	1.25	3.17
11	14.12	1.48	2.32		16.8	95	2.20	2.43	0.26	1.57		5.00	1.17	2.82
12	13.75	1.73	2.41		16.0	89	2.60	2.76	0.29	1.81		5.60	1.15	2.90
13	14.75	1.73	2.39		11.4	91	3.10	3.69	0.43	2.81		5.40	1.25	2.73
14	14.38	1.87	2.38		12.0	102	3.30	3.64	0.29	2.96		7.50	1.20	3.00
15	13.63	1.81	2.70		17.2	112	2.85	2.91	0.30	1.46		7.30	1.28	2.88
16	14.30	1.92	2.72		20.0	120	2.80	3.14	0.33	1.97		6.20	1.07	2.65
17	13.83	1.57	2.62		20.0	115	2.95	3.40	0.40	1.72		6.60	1.13	2.57
18	14.19	1.59	2.48		16.5	108	3.30	3.93	0.32	1.86		8.70	1.23	2.82
19	13.64	3.10	2.56		15.2	116	2.70	3.03	0.17	1.66		5.10	0.96	3.36

Analiza zbiorów IRIS, Wine, GLASS: klasy (liczba, interpretacja), instancje, atrybuty, dystrybucja klas w zbiorze.

IRIS

Wypisanie podstawowych informacji o danych ze zbioru IRIS.

```
In [ ]: iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   sepal length    150 non-null   float64 
 1   sepal width     150 non-null   float64 
 2   petal length    150 non-null   float64 
 3   petal width     150 non-null   float64 
 4   class          150 non-null   object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Zbiór posiada 150 instancji, 4 atrybuty jedną etykietę. Typami atrybutów są liczby zmiennoprzecinkowe, zaś etykiety są typem `object`. Zbiór nie posiada brakujących danych. Jeżeli posiadałyby musielibyśmy podjąć decyzję co z nimi zrobić. Czy usunąć, czy uzupełnić.

Przeanalizujmy czym są etykiety, jakie klasy występują w zbiorze oraz jakie jest ich rozkład.

```
In [ ]: iris_df['class']
```

```
Out[ ]: 0      Iris-setosa
 1      Iris-setosa
 2      Iris-setosa
 3      Iris-setosa
 4      Iris-setosa
 ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: class, Length: 150, dtype: object
```

```
In [ ]: print(iris_df['class'].nunique())
```

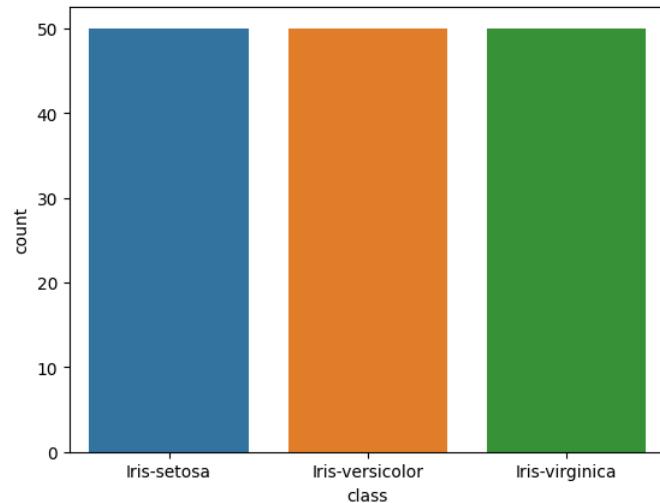
```
3
```

```
In [ ]: print(iris_df['class'].value_counts())
```

```
class
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
Name: count, dtype: int64
```

Zbiór Iris zawiera 150 instancji, 4 atrybuty i 3 klasy. Klasy to: Iris-setosa, Iris-versicolor, Iris-virginica. Klasy są równomiernie rozłożone w zbiorze, po 50 instancjach każdej. Dystrybucja klas w zbiorze jest równomierna i wygląda następująco:

```
In [ ]: sns.countplot(x='class', data=iris_df, hue='class')
plt.show()
```



Następnie możemy przyjrzeć się bliżej wartościom atrybutów. W tym celu wyświetlimy podstawowe statystyki dotyczące wartości atrybutów.

```
In [ ]: iris_df.describe()
```

Out[]:

	sepal length	sepal width	petal length	petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

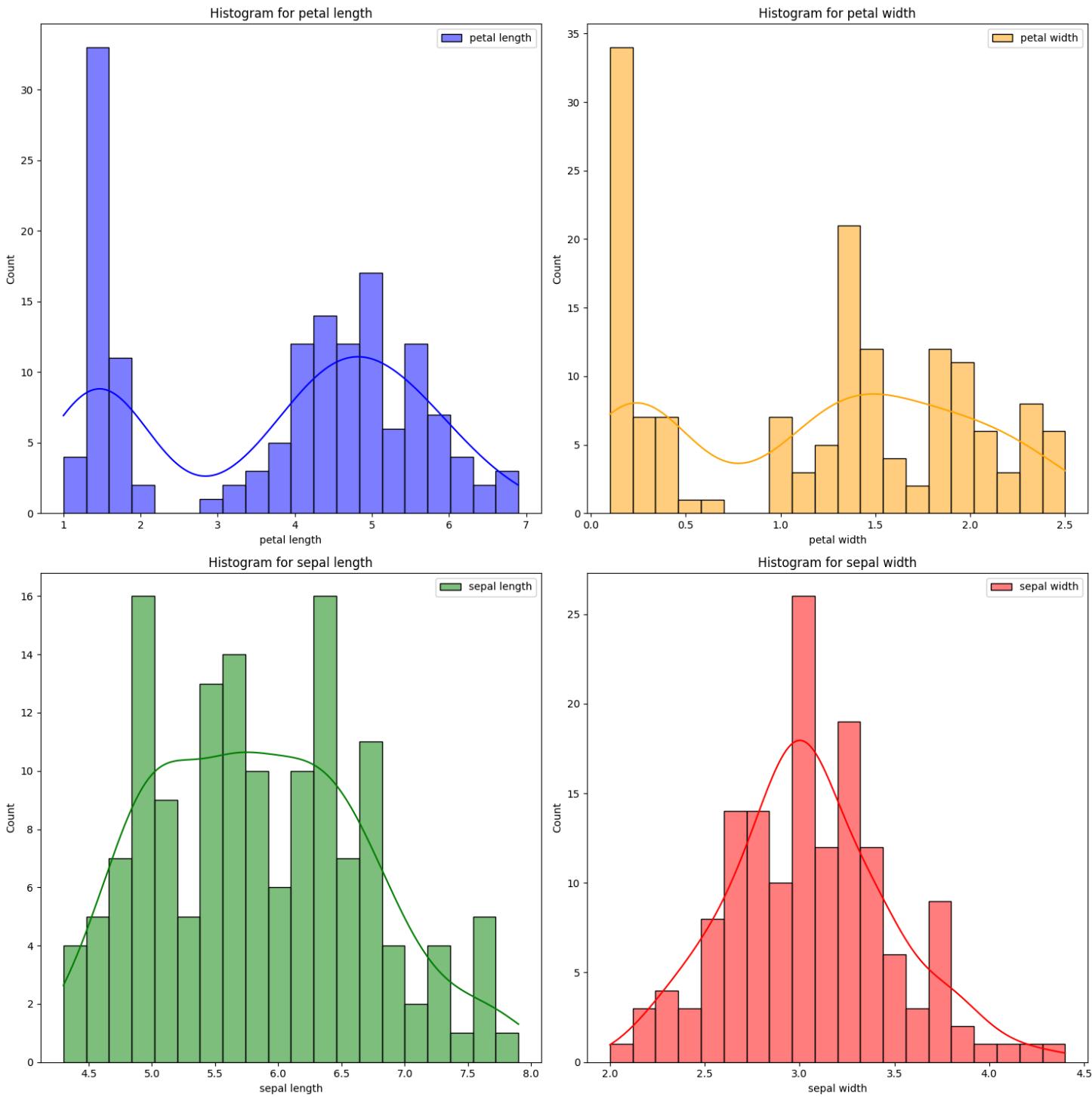
Wartości długości kielicha mieścią się w przedziale od 4.3 do 7.9, wartości szerokości kielicha mieścią się w przedziale od 2.0 do 4.4, wartości długości płatka mieścią się w przedziale od 1.0 do 6.9, wartości szerokości płatka mieścią się w przedziale od 0.1 do 2.5.

Średnia wartość długości kielicha wynosi 5.84, szerokości kielicha wynosi 3.05, długości płatka wynosi 3.76, szerokości płatka wynosi 1.20.

Odchylenie standardowe dla szerokości płatka jest największe i wynosi 1.76, co może świadczyć o tym, że wartości długości płatka są bardziej rozproszone niż wartości pozostałych atrybutów.

Również możemy zwizualizować rozkład atrbutów w zbiorze danych.

```
In [ ]:  
columns_to_plot = ['petal length', 'petal width', 'sepal length', 'sepal width']  
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 15))  
  
axes = axes.flatten()  
  
colors = ['blue', 'orange', 'green', 'red']  
  
for i, (column, color) in enumerate(zip(columns_to_plot, colors)):  
    sns.histplot(data=iris_df, x=column, kde=True, label=column, color=color, ax=axes[i], bins=20 )  
    axes[i].set_title(f'Histogram for {column}')  
    axes[i].legend()  
  
for j in range(len(columns_to_plot), len(axes)):  
    fig.delaxes(axes[j])  
  
plt.tight_layout()  
plt.show()
```



Teraz zaznaczmy na każdym rozkładzie atrybutów klasy, do której należy dana próbka.

```
In [ ]:
columns_to_plot = ['petal length', 'petal width', 'sepal length', 'sepal width']
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 15))

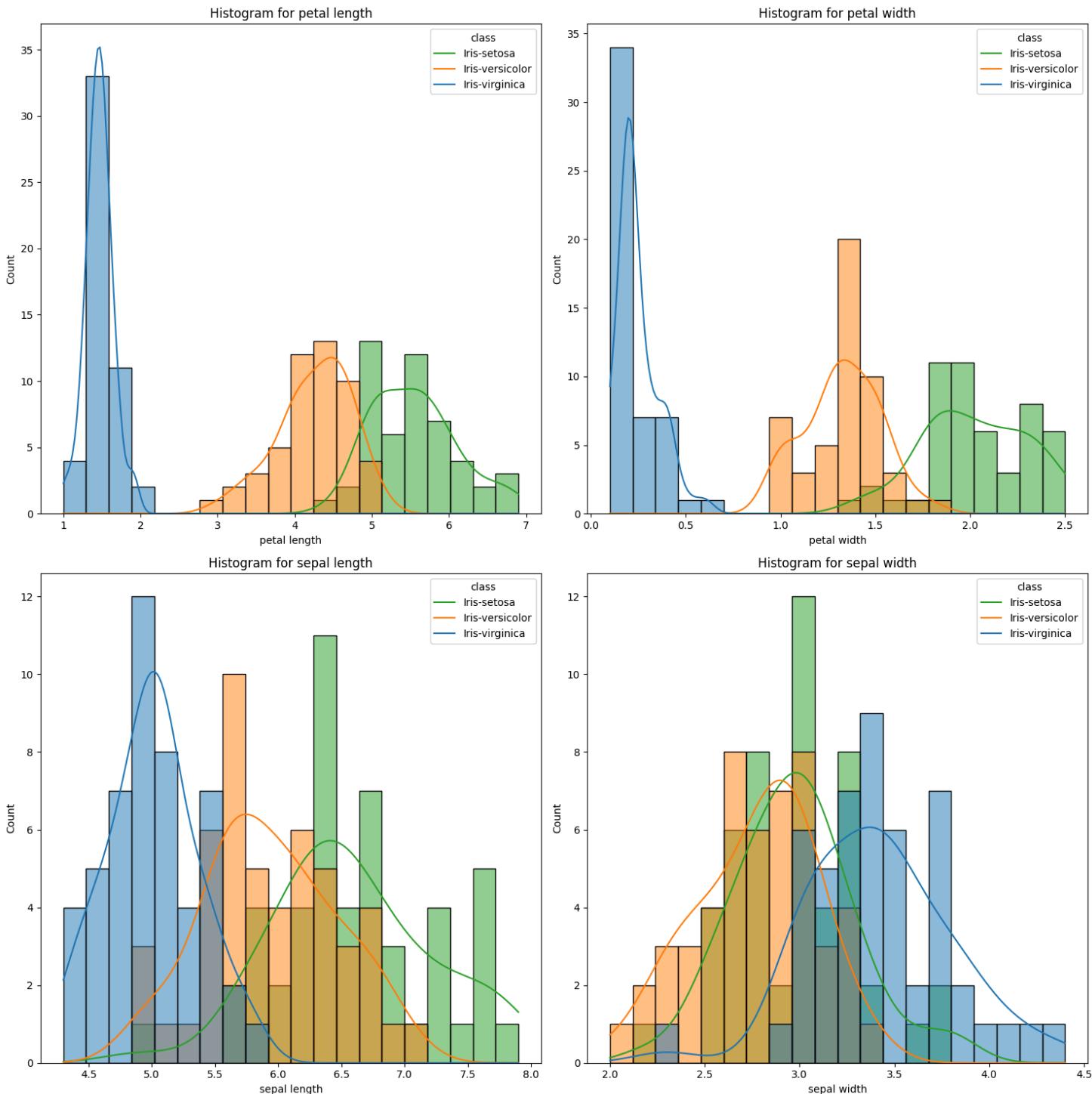
axes = axes.flatten()

colors = ['blue', 'orange', 'green', 'red']

for i, (column, color) in enumerate(zip(columns_to_plot, colors)):
    sns.histplot(data=iris_df, x=column, kde=True, hue='class', color=color, ax=axes[i], bins=20)
    axes[i].set_title(f'Histogram for {column}')
    axes[i].legend( loc='upper right', title='class', labels=iris_df['class'].unique())

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

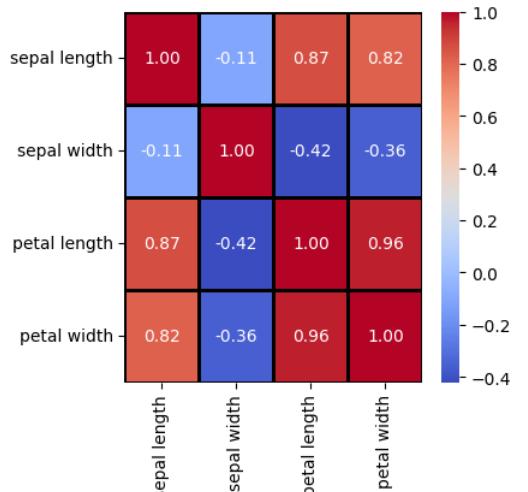
plt.tight_layout()
plt.show()
```



Możemy zwizualizować macierz korelacji. Celem użycia macierzy korelacji podczas analizy danych jest zidentyfikowanie zależności pomiędzy atrybutami. Dzięki tej analize można również eliminować zbędne zmienne, które nie wniosą dodatkowej informacji do modelu.

```
In [ ]: corr_matrix = iris_X.corr()
plt.figure(figsize=(4, 4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=2, linecolor='black')
```

```
Out[ ]: <Axes: >
```

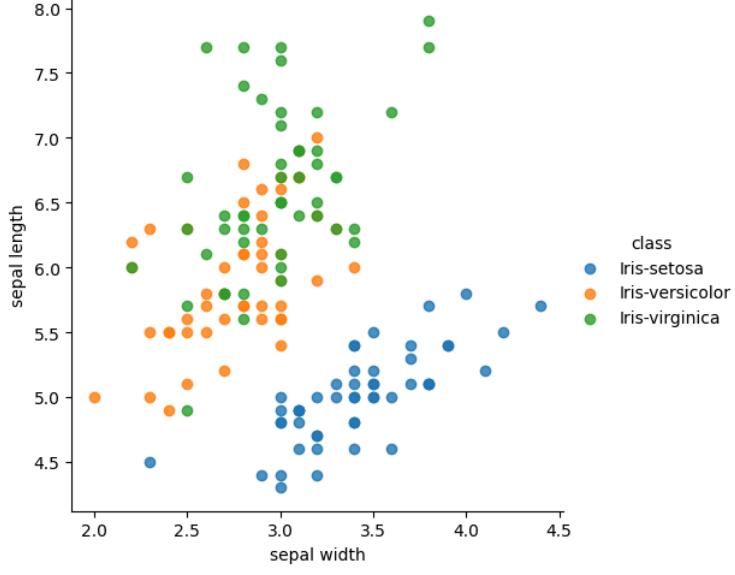


Rozkład rozmiarów kielichów jest zbliżony do rozkładu normalnego. Wartości długości kielicha mieszczą się w przedziale od 4 do 8 cm, a szerokości od 2 do 4.5 cm. Natomiast rozkład długości i szerokości płatków jest bardziej zróżnicowany. Długość płatków mieści się w przedziale od 1 do 7 cm, a szerokość od 0.1 do 2.5 cm.

Wyrysowanie wykresu zależności długości/szerokości płatków IRIS a klasą (z kolorem)

```
In [ ]: sns.lmplot(data=iris_df, x='sepal width', y='sepal length', hue='class', fit_reg=False)
```

```
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x1743b2e65d0>
```

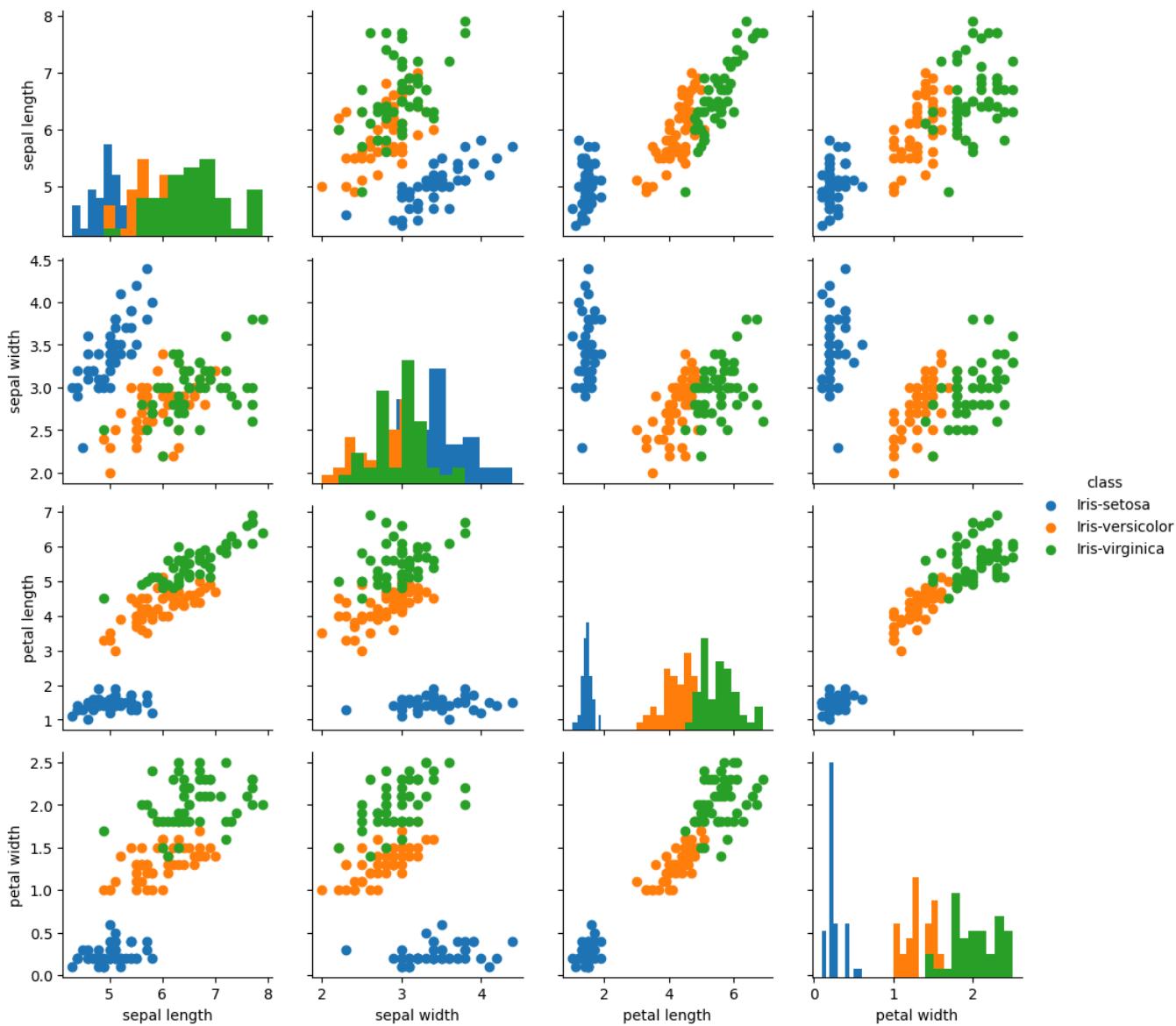


Po tym wykresie można zauważać, że klasa Iris-setosa jest najbardziej odseparowana od pozostałych klas. Iris-versicolor i Iris-virginica są bardziej zróżnicowane, ale nadal można zauważać pewne różnice między nimi. Iris-setosa charakteryzuje się krótszymi ale szerszymi kielichami. Iris-versicolor i Iris-virginica są bardziej podobne do siebie, ale można zauważać, że Iris-versicolor ma krótsze kielichy.

```
In [ ]: plt.figure(figsize=(10, 10))
g = sns.PairGrid(iris_df, hue='class')
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
g.add_legend()
```

```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x174619e78d0>
```

```
<Figure size 1000x1000 with 0 Axes>
```



Po wykreśnięciu zbiorowego wykresu, można zauważać, rozmiary płatków wśród klas lepiej rozdzielają dane na grupy niż rozmiary kielichów. Można gołym okiem zauważać, że Iris-setosa jest najbardziej odseparowana od pozostałych klas pod względem rozmiarów płatków. Iris-versicolor i Iris-virginica są również możliwa rozdzielić, ale nie tak wyraźnie jak Iris-setosa.

Użycie PCA i narysowanie wyniku działania PCA

```
In [ ]: from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
iris_X_scaled = scalar.fit_transform(iris_X)

pca = decomposition.PCA(n_components=2)
iris_X_pca = pca.fit_transform(iris_X_scaled)

iris_X_pca_df = pd.DataFrame(data=iris_X_pca, columns=['PC1', 'PC2'])
iris_X_pca_df['class'] = iris_y
iris_X_pca_df.head()
```

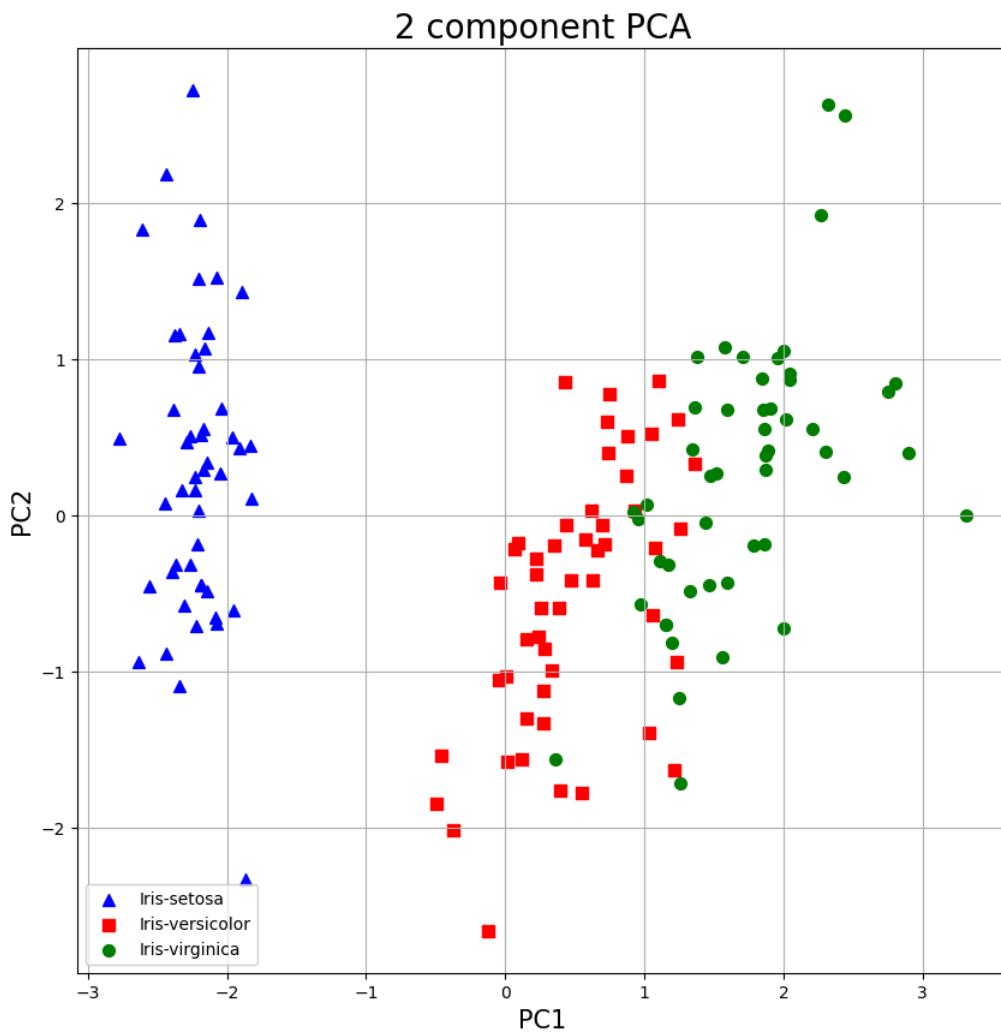
```
Out[ ]:
```

	PC1	PC2	class
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa

```
In [ ]: fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111)
ax.set_xlabel('PC1', fontsize=15)
ax.set_ylabel('PC2', fontsize=15)
ax.set_title('2 component PCA', fontsize=20)
colors = ("blue", "red", "green")
markers = ("^", "s", "o")

targets = iris_X_pca_df['class'].unique()
for target, color, marker in zip(targets, colors, markers):
    indicesToKeep = iris_X_pca_df['class'] == target
    ax.scatter(
        iris_X_pca_df.loc[indicesToKeep, 'PC1'],
        iris_X_pca_df.loc[indicesToKeep, 'PC2'],
        s=50,
        c=color,
        marker=marker,
    )

ax.legend(targets)
ax.grid()
```



WINE

Analiza zbioru Wine. Wypisanie najważniejszych informacji o danych.

In []: `wine_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Alcohol          178 non-null    float64
 1   Malicacid        178 non-null    float64
 2   Ash               178 non-null    float64
 3   Alcalinity_of_ash 178 non-null    float64
 4   Magnesium         178 non-null    int64  
 5   Total_phenols     178 non-null    float64
 6   Flavanoids        178 non-null    float64
 7   Nonflavanoid_phenols 178 non-null    float64
 8   Proanthocyanins  178 non-null    float64
 9   Color_intensity   178 non-null    float64
 10  Hue               178 non-null    float64
 11  00280_0D315_of_diluted_wines 178 non-null    float64
 12  Proline           178 non-null    int64  
 13  class              178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

Zbior wine posiada 178 instancji, 13 atrybutów i 3 klasy. Klasa to: 1, 2, 3. Atrybuty są liczbami zmiennoprzecinkowymi z wyjątkiem zawartości magnezu oraz Prolinu, które są liczbami całkowitymi. Również w zbiorze nie ma braków danych.

In []: `wine_df['class']`

```
Out[ ]: 0      1
 1      1
 2      1
 3      1
 4      1
 ..
173     3
174     3
175     3
176     3
177     3
Name: class, Length: 178, dtype: int64
```

In []: `wine_df['class'].unique()`

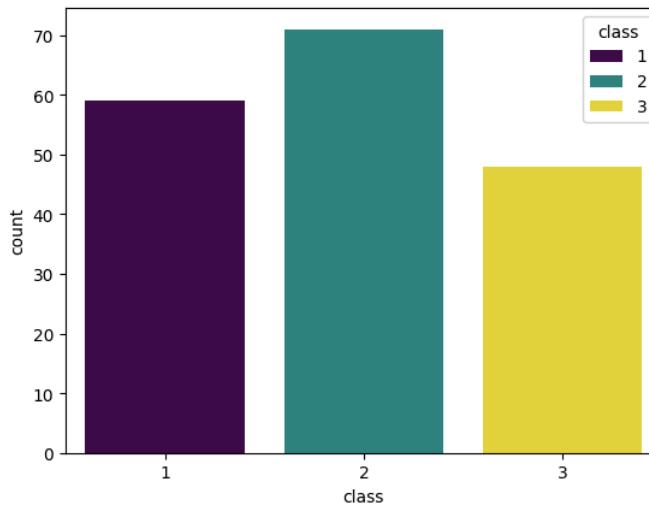
```
Out[ ]: 3
```

```
In [ ]: wine_df['class'].value_counts()
```

```
Out[ ]: class
2    71
1    59
3    48
Name: count, dtype: int64
```

Zbiór posiada 3 klasy win, są one nierównomiernie rozłożone w zbiorze. Najwięcej instancji posiada klasa 2, a najmniej klasa 1. Dystrybucja klas w zbiorze wygląda następująco:

```
In [ ]: sns.countplot(x='class', data=wine_df, hue='class', palette='viridis')
plt.show()
```



Wskaźniki dotyczące atrybutów w zbiorze danych:

```
In [ ]: wine_X.describe()
```

```
Out[ ]:   Alcohol  Malicacid      Ash  Alcalinity_of_ash  Magnesium  Total_phenols  Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity  Hue  OD280_OD315_
count    178.000000  178.000000  178.000000  178.000000  178.000000  178.000000  178.000000  178.000000  178.000000  178.000000  178.000000
mean     13.000618  2.336348  2.366517  19.494944  99.741573  2.295112  2.029270  0.361854  1.590899  5.058090  0.957449
std      0.811827  1.117146  0.274344  3.339564  14.282484  0.625851  0.998859  0.124453  0.572359  2.318286  0.228572
min     11.030000  0.740000  1.360000  10.600000  70.000000  0.980000  0.340000  0.130000  0.410000  1.280000  0.480000
25%    12.362500  1.602500  2.210000  17.200000  88.000000  1.742500  1.205000  0.270000  1.250000  3.220000  0.782500
50%    13.050000  1.865000  2.360000  19.500000  98.000000  2.355000  2.135000  0.340000  1.555000  4.690000  0.965000
75%    13.677500  3.082500  2.557500  21.500000  107.000000  2.800000  2.875000  0.437500  1.950000  6.200000  1.120000
max     14.830000  5.800000  3.230000  30.000000  162.000000  3.880000  5.080000  0.660000  3.580000  13.000000  1.710000
```

Dość duże odchylenie standardowe posiada Magnez, oraz Prolin.

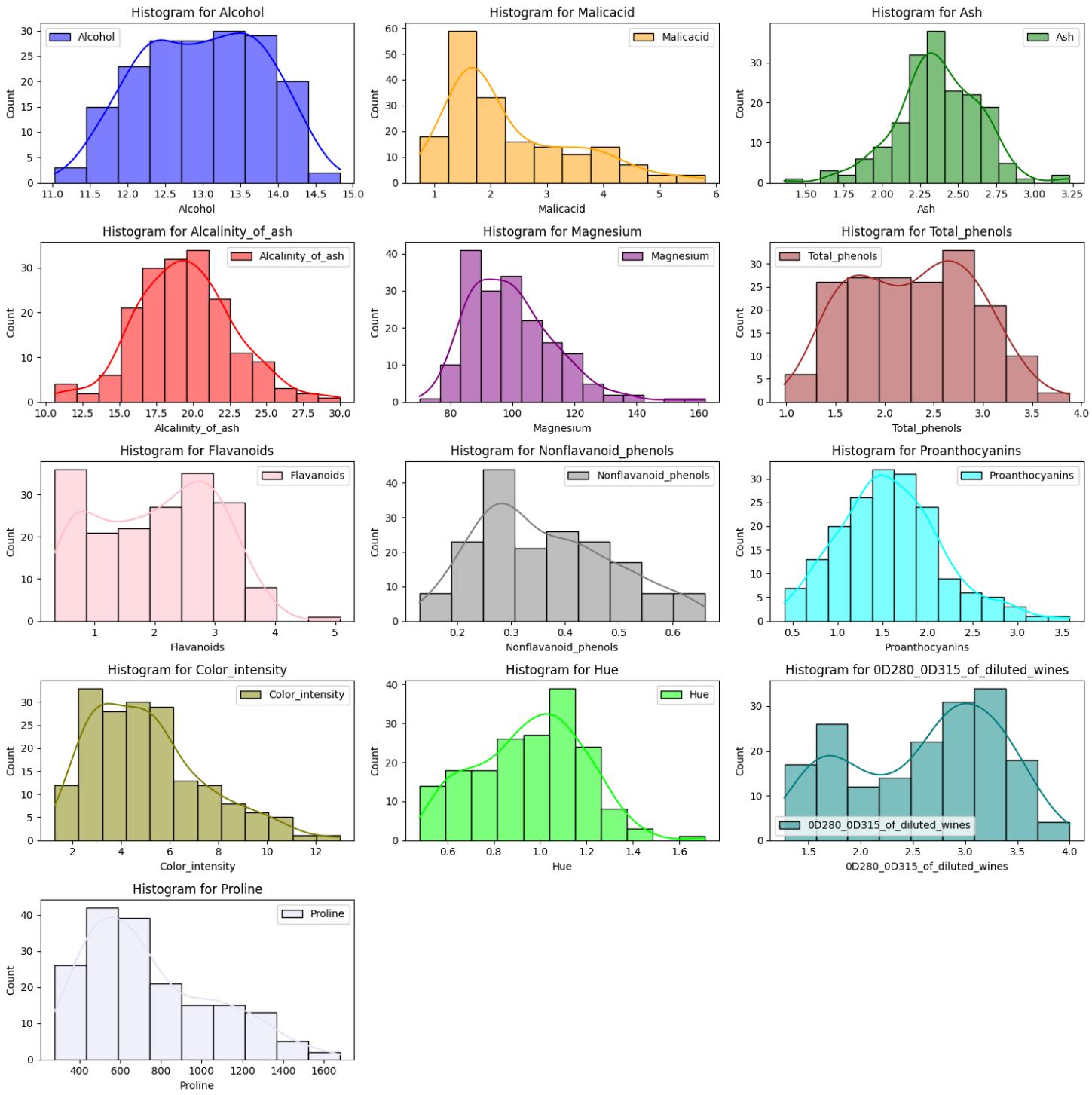
Wizualizacja rozkładów wartości poszczególnych atrybutów opisujących wino.

```
In [ ]: columns_to_plot = ['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols', 'Proanthocyanins', 'Color_intensity', 'Hue', 'OD280_OD315']
fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(15, 15))
axes = axes.flatten()
colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink', 'gray', 'cyan', 'olive', 'lime', 'teal', 'lavender']

for i, (column, color) in enumerate(zip(columns_to_plot, colors)):
    sns.histplot(data=wine_df, x=column, kde=True, label=column, color=color, ax=axes[i])
    axes[i].set_title(f'Histogram for {column}')
    axes[i].legend()

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



Teraz możemy wyświetlić rozkłady wartości atrybutów dodatkowo zaznaczając kolorami klasy win

```
In [ ]:
columns_to_plot = ['Alcohol', 'Malicacid', 'Ash', 'Alkalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols', 'Proanthocyanins', 'Color_intensity', 'Hue', 'Proline', 'OD280_OD315_of_diluted_wines']

fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(15, 15))

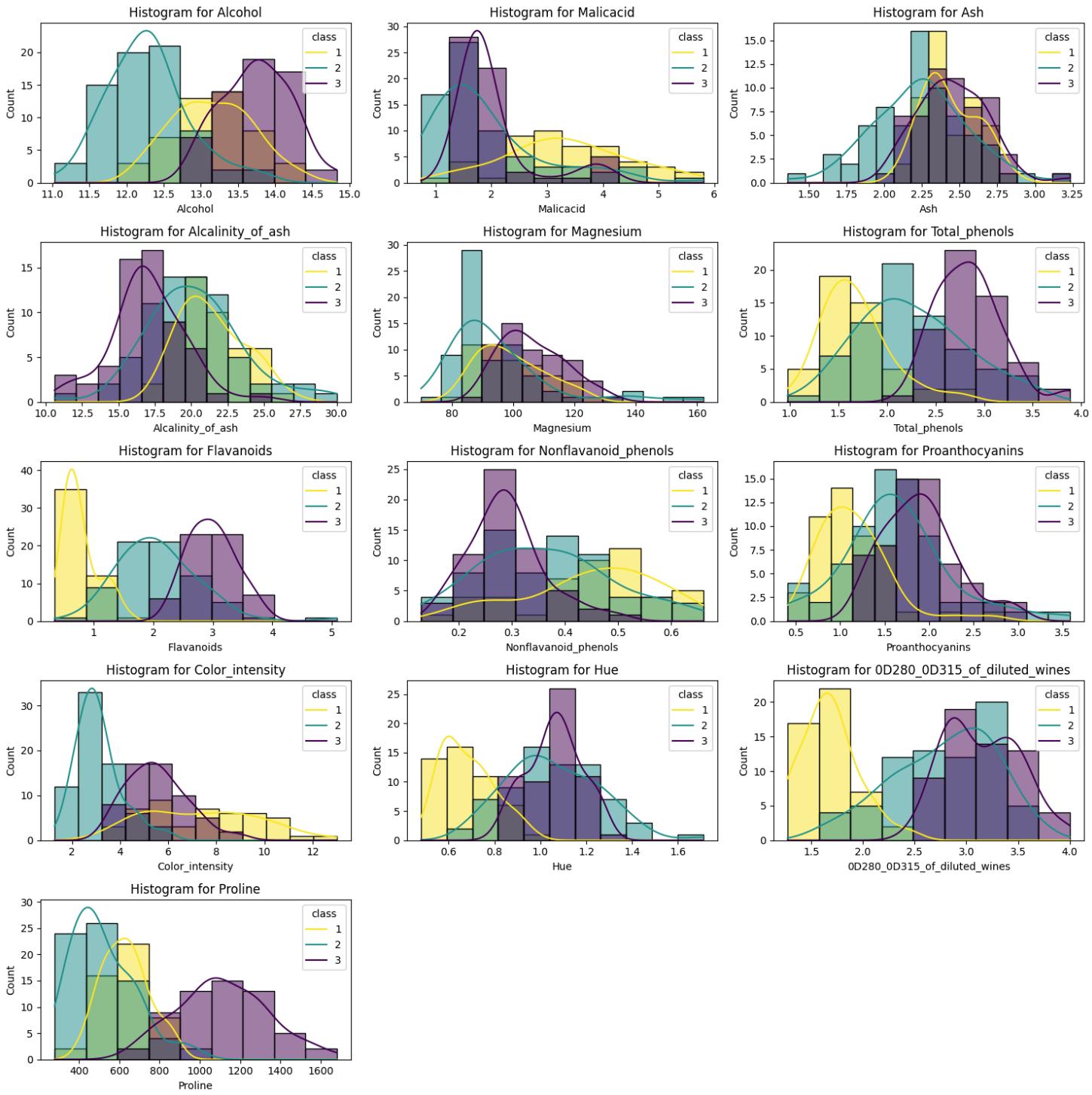
axes = axes.flatten()

colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink', 'gray', 'cyan', 'olive', 'lime', 'teal', 'lavender']

for i, (column, color) in enumerate(zip(columns_to_plot, colors)):
    sns.histplot(data=wine_df, x=column, kde=True, hue='class', color=color, ax=axes[i], palette='viridis')
    axes[i].set_title(f'Histogram for {column}')
    axes[i].legend( loc='upper right', title='class', labels=wine_df['class'].unique())

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

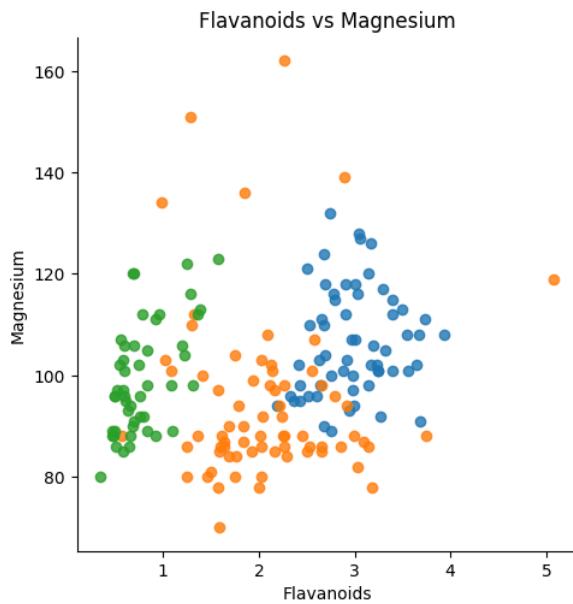
plt.tight_layout()
plt.show()
```



Teraz możemy zauważyc, że rozkłady atrybutów, które nie przypominają rozkładu normalnego, zawierają separowalne podzbioru rozkładów danych klas. Widać to szczególnie dla: Alkoholu, Phenoli, Flawonoidów, Białka (OD280/OD315), Intensywności koloru, Proantocjanów.

Wykaśmy zależność pomiędzy Flawonoidami a Magnezem, zaznaczając kolorem klasy win.

```
In [ ]: sns.lmplot(data=wine_df, x='Flavanoids', y='Magnesium', hue='class', fit_reg=False)
plt.xlabel('Flavanoids')
plt.ylabel('Magnesium')
plt.title('Flavanoids vs Magnesium')
plt.show()
```

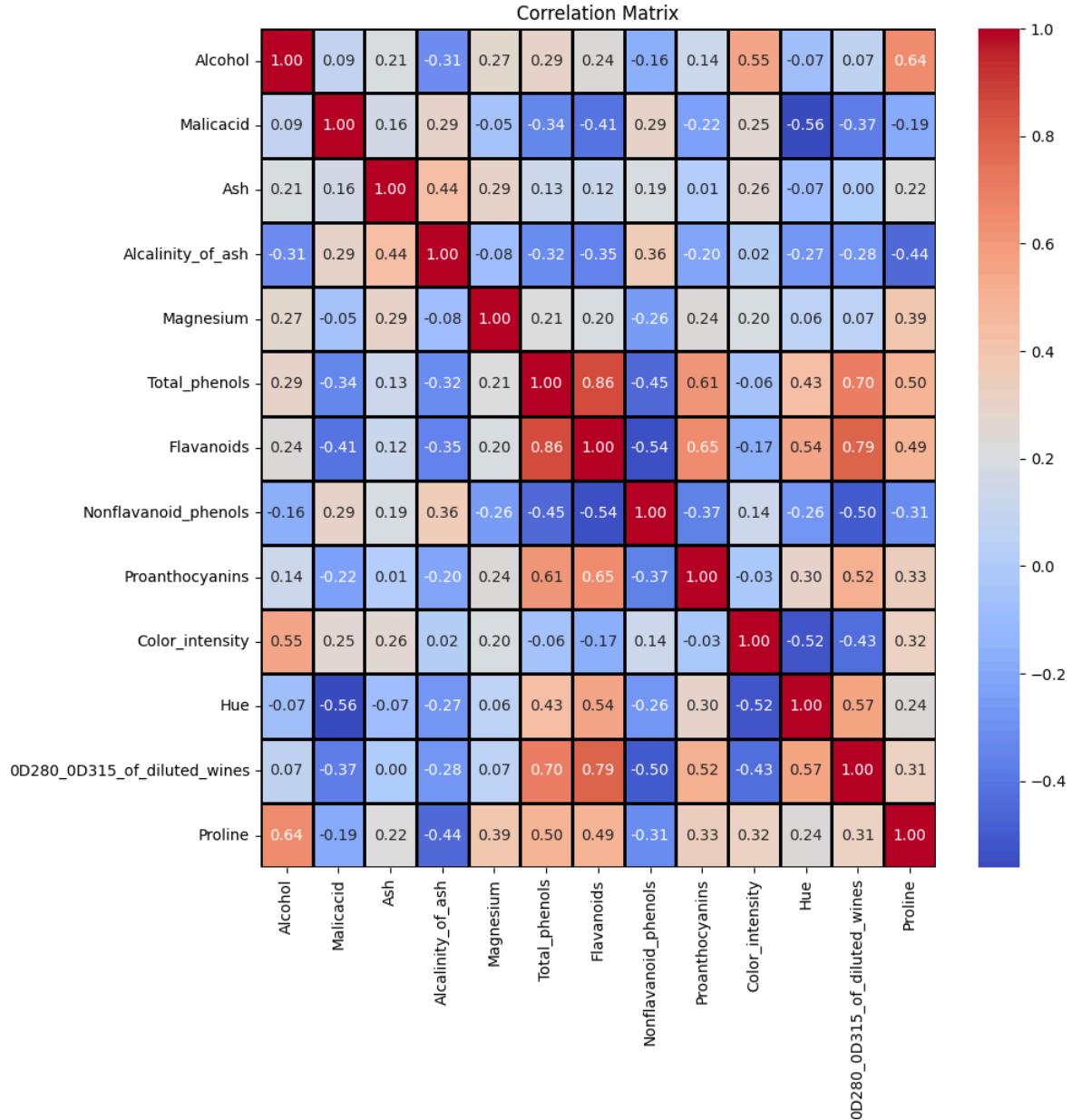


Wyświetlenie macierzy korelacji atrybutów.

```
In [ ]: corr_matrix = wine_X.corr()

plt.figure(figsize=(10, 10))
plt.title('Correlation Matrix')
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=2, linecolor='black')
```

```
Out[ ]: <Axes: title={'center': 'Correlation Matrix'}>
```

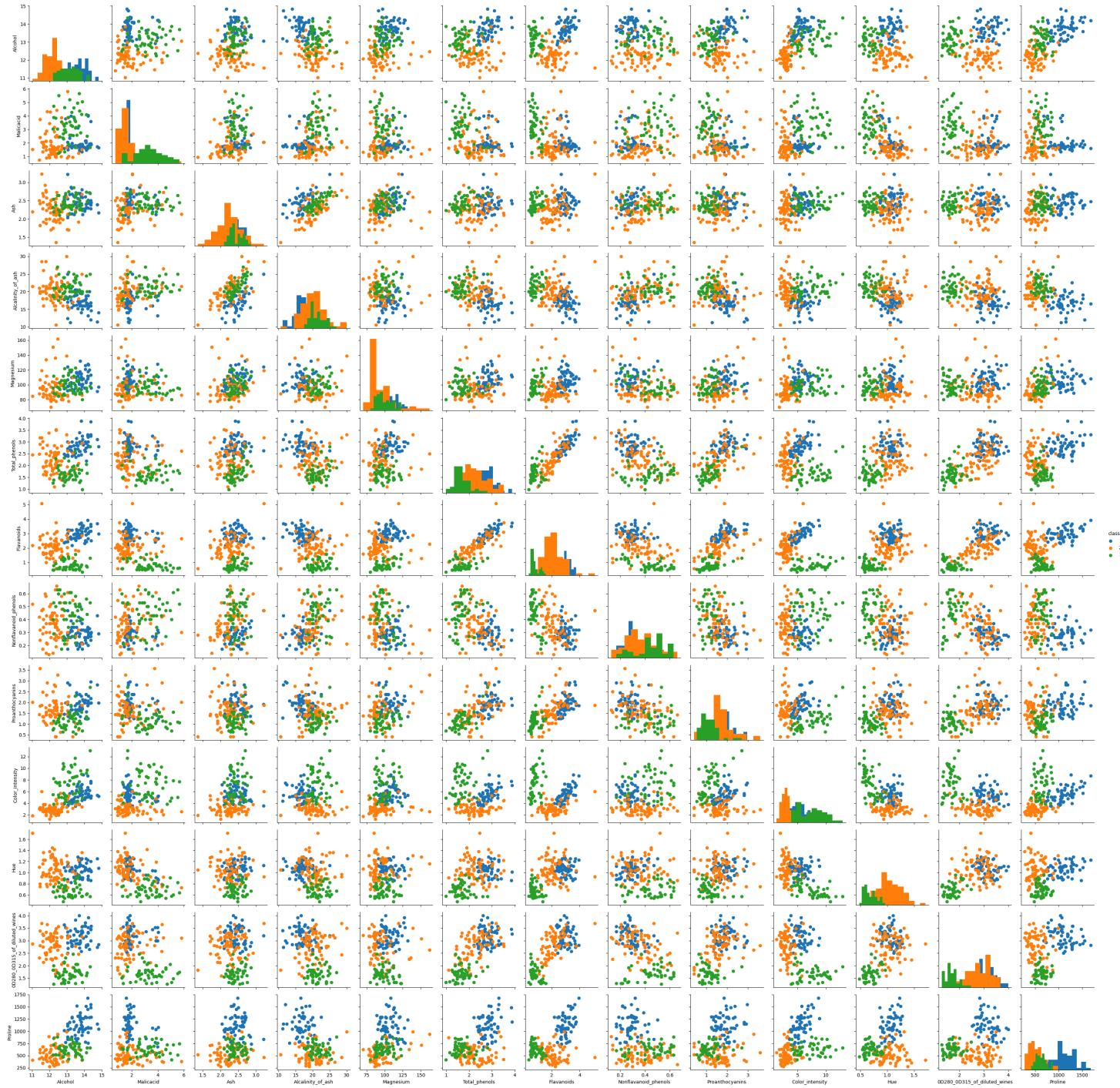


Wyświetlenie wykresu zależności pomiędzy atrybutami wszystkimi atrybutami zaznaczając kolorem klasy win.

```
In [ ]: g2 = sns.PairGrid(wine_df, hue='class')
g2.map_diag(plt.hist)
```

```
g2.map_offdiag(plt.scatter)
g2.add_legend()
```

Out[]: <seaborn.axisgrid.PairGrid at 0x1746294bdd0>



Ustandaryzujemy dane.

```
In [ ]:
scalar = StandardScaler()
wine_X_scaled = scalar.fit_transform(wine_X)

pca = decomposition.PCA(n_components=2)
wine_X_pca = pca.fit_transform(wine_X_scaled)

wine_X_pca_df = pd.DataFrame(data=wine_X_pca, columns=['PC1', 'PC2'])
wine_X_pca_df['class'] = wine_y
wine_X_pca_df.head()
```

Out[]:

	PC1	PC2	class
0	3.316751	-1.443463	1
1	2.209465	0.333393	1
2	2.516740	-1.031151	1
3	3.757066	-2.756372	1
4	1.008908	-0.869831	1

```
In [ ]:
# plot
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111)
ax.set_xlabel('PC1', fontsize=15)
ax.set_ylabel('PC1', fontsize=15)
ax.set_title('2 component PCA', fontsize=20)
markers = ("^", "s", "o")

targets = wine_X_pca_df['class'].unique()
for target, marker in zip(targets, markers):
    indicesToKeep = wine_df['class'] == target
```

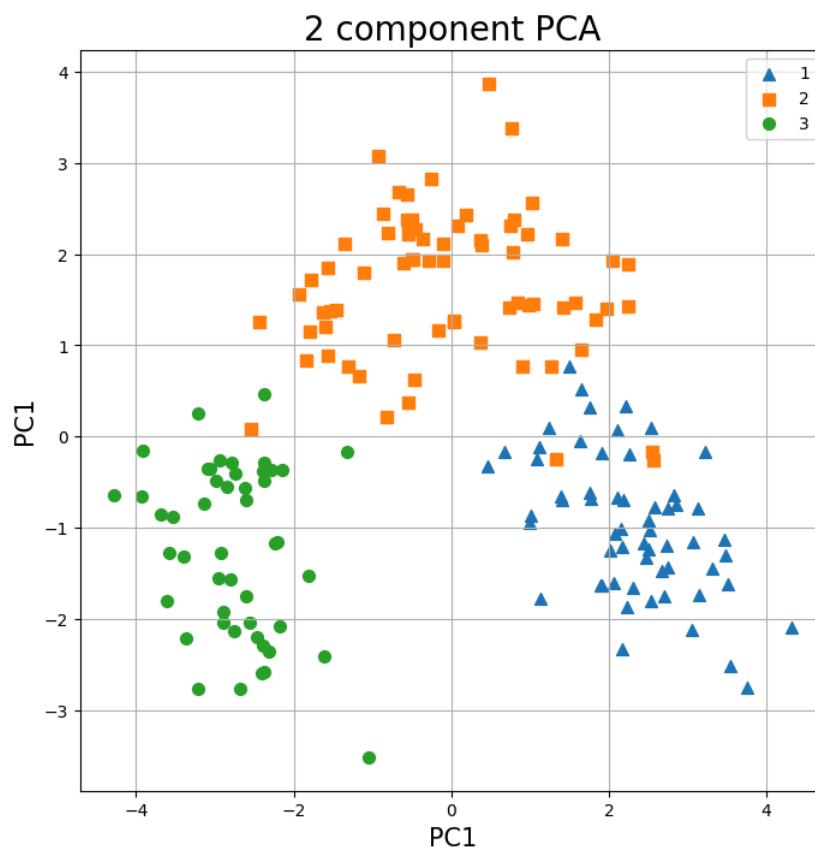
```

        ax.scatter(
            wine_X_pca_df.loc[indicesToKeep, 'PC1'],
            wine_X_pca_df.loc[indicesToKeep, 'PC2'],
            s=50,
            marker=marker,
        )

ax.legend(targets)
ax.grid()

plt.show()

```



Wykonanie ustandaryzowania danych pozwoliło algorytmowi PCA wyodrębnić najważniejsze składowe w taki sposób aby można łatwo odseparować klasy.

GLASS

Wypisanie podstawowych informacji

```
In [ ]: glass_df.info()
```

#	Column	Non-Null Count	Dtype
0	RI	214	float64
1	Na	214	float64
2	Mg	214	float64
3	Al	214	float64
4	Si	214	float64
5	K	214	float64
6	Ca	214	float64
7	Ba	214	float64
8	Fe	214	float64
9	Type_of_glass	214	int64

dtypes: float64(9), int64(1)
memory usage: 16.8 KB

W zbiorze danych Glass zawiera się 214 instancji. Zbiór posiada 9 atrybutów oraz 1 etykię. Atrybuty są liczbami zmiennoprzecinkowymi, zaś etykiety są typem `int64`. Zbiór nie posiada braków danych.

```
In [ ]: print(glass_df['Type_of_glass'].value_counts())
```

Type_of_glass	count
2	76
1	70
7	29
3	17
5	13
6	9

Name: count, dtype: int64

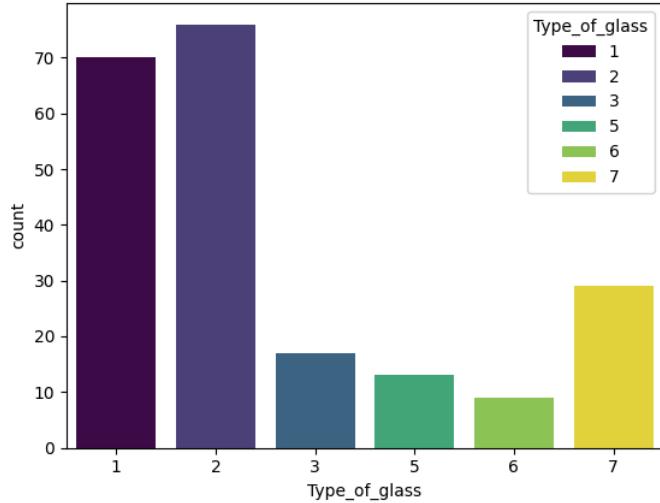
Można zauważyc, że klasy są liczbami całkowitymi. Zamienimy je na etykiety, które będą bardziej czytelne.

1. building_windows_float_processed
2. building_windows_non_float_processed
3. vehicle_windows_float_processed
4. vehicle_windows_non_float_processed (brak w zbiorze)
5. containers
6. tableware
7. headlamps

W tym zborze znajduje się 6 klas szkła. Klasa nie są równomiernie rozłożone w zbiorze.

```
In [ ]: sns.countplot(data=glass_df, x='Type_of_glass', hue='Type_of_glass', palette='viridis')
```

```
Out[ ]: <Axes: xlabel='Type_of_glass', ylabel='count'>
```



Tutaj można zaobserwować, że rozkład nie przypomina rozkładu normalnego. Dominacja klas 2 oraz 1. Przyszły model może nauczyć się dobrze klasyfikować klasę 2 oraz 1, a gorzej klasę 3, 4, 5, 6.

```
In [ ]: glass_X.describe()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009
std	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439
min	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000
25%	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000
50%	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000
75%	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000
max	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000

Bardzo małe odchylenie standardowe w przypadku RI świadczą o tym, że wartości współczynnika załamania światła są bardzo zbliżone do siebie. Natomiast w przypadku zawartości sodu, magnezu, glinu, krzemu, potasu, wapnia, baru oraz żelaza odchylenie standardowe jest znacznie większe, co oznacza, że wartości tych atrybutów są bardziej rozproszone.

```
In [ ]: columns_to_plot = ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(10,10))

axes = axes.flatten()

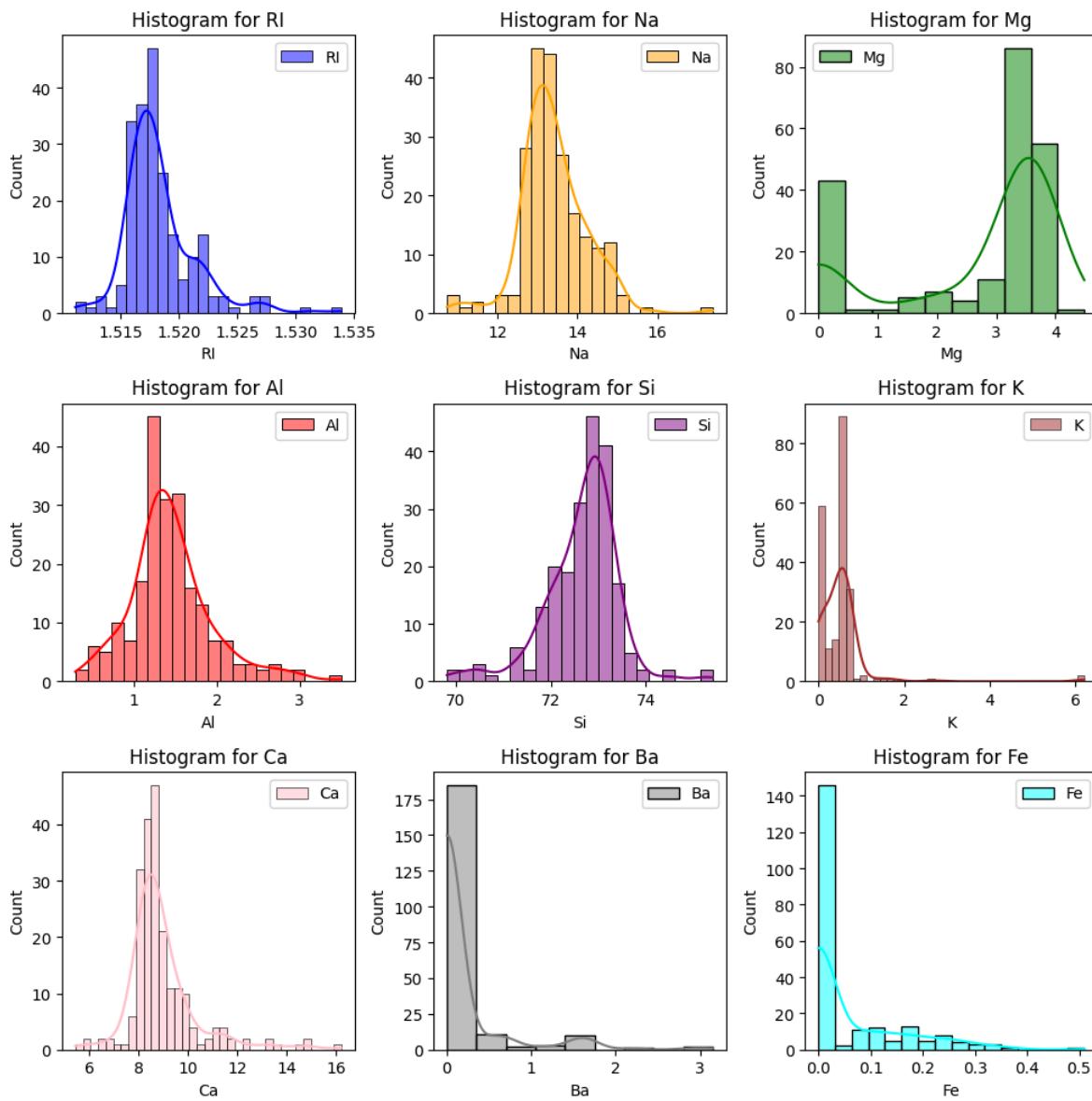
colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink', 'gray', 'cyan']

for i, (column, color) in enumerate(zip(columns_to_plot, colors)):
    sns.histplot(data=glass_df, x=column, kde=True, label=column, color=color, ax=axes[i])
    axes[i].set_title(f'Histogram for {column}')
    axes[i].legend()

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()

plt.show()
```



Od razu widać wartości w rozkładach które dominują. Można zauważyć, że niektóre atrybuty posiadają podobny rozkład do rozkładu normalnego, ale niektóre atrybuty posiadają rozkład bardziej skupiony wokół jednej wartości.

Powtórzmy wykreślenie rozkładów wartości atrybutów ale zaznaczając kolorem klasy szkła.

Można zauważyc, że rozkłady Fe, Ba oraz Mg różnią się od takiego typowego rozkładu normalnego.

```
In [ ]:
columns_to_plot = ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']

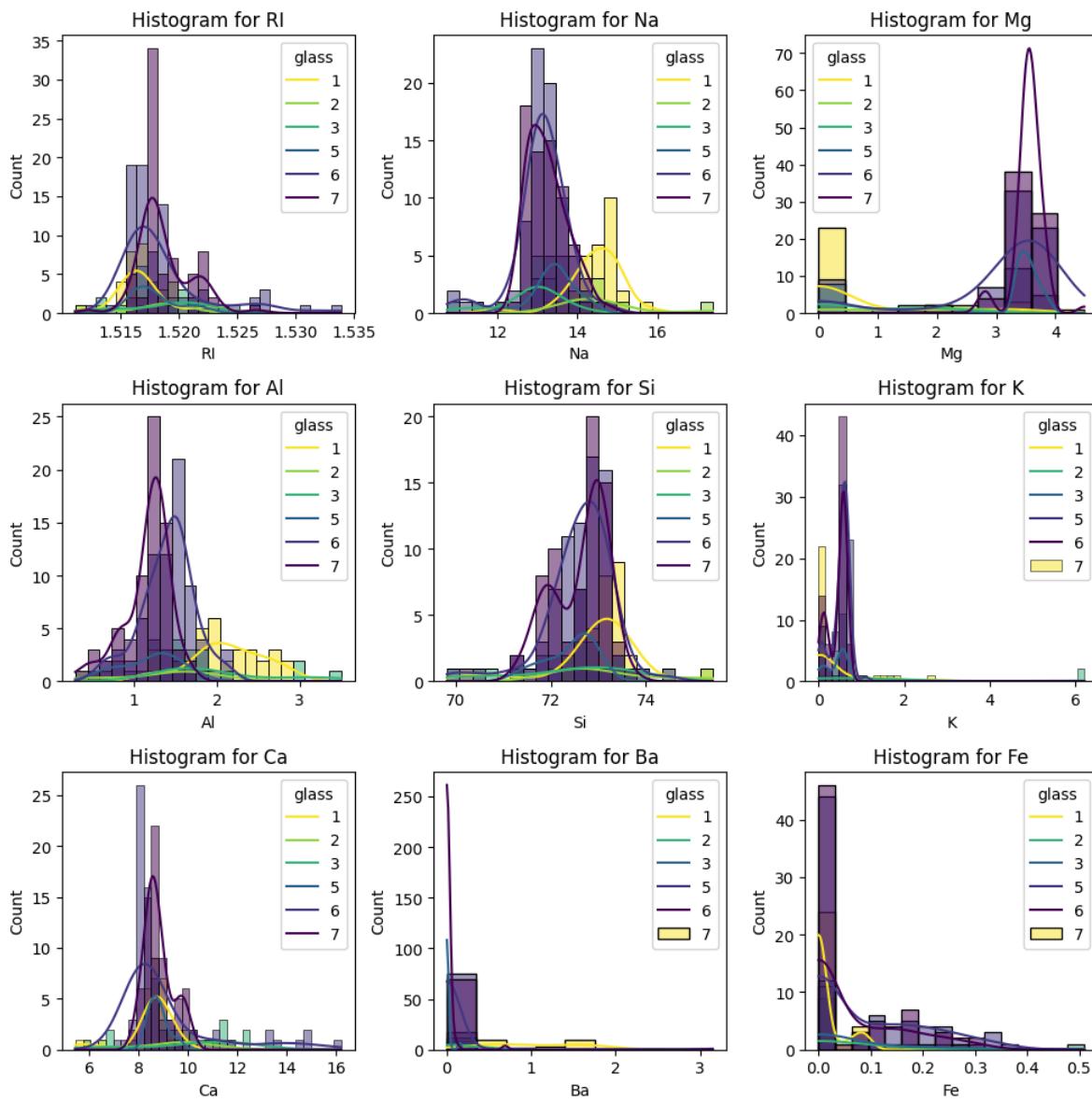
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(10,10))

axes = axes.flatten()

for i, (column, color) in enumerate(zip(columns_to_plot, colors)):
    sns.histplot(data=glass_df, x=column, kde=True, hue='Type_of_glass', palette='viridis', ax=axes[i])
    axes[i].set_title(f'Histogram for {column}')
    axes[i].legend(title='glass', labels=glass_df['Type_of_glass'].unique())

for j in range(len(columns_to_plot), len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

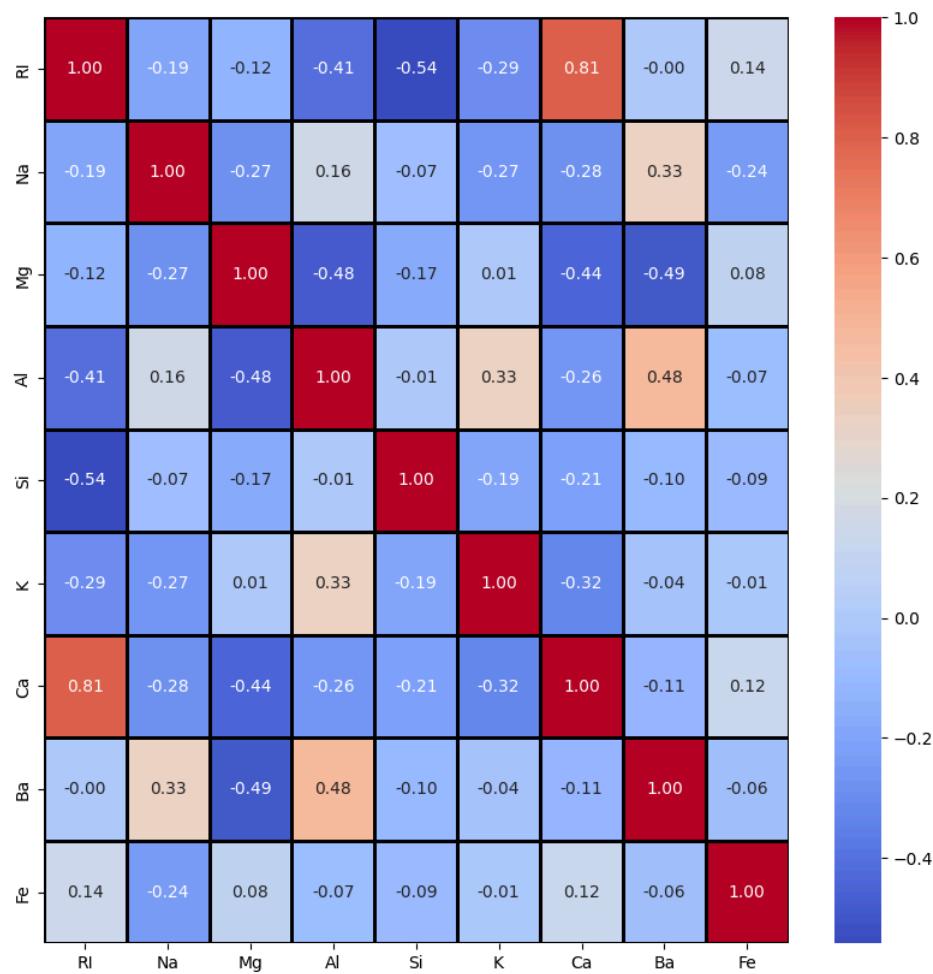


Widac zdominowanie dwóch klas w rozkładach.

Obliczymy macierz korelacji atrybutów.

```
In [ ]: corr_matrix = glass_X.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=2, linecolor='black')

Out[ ]: <Axes: >
```

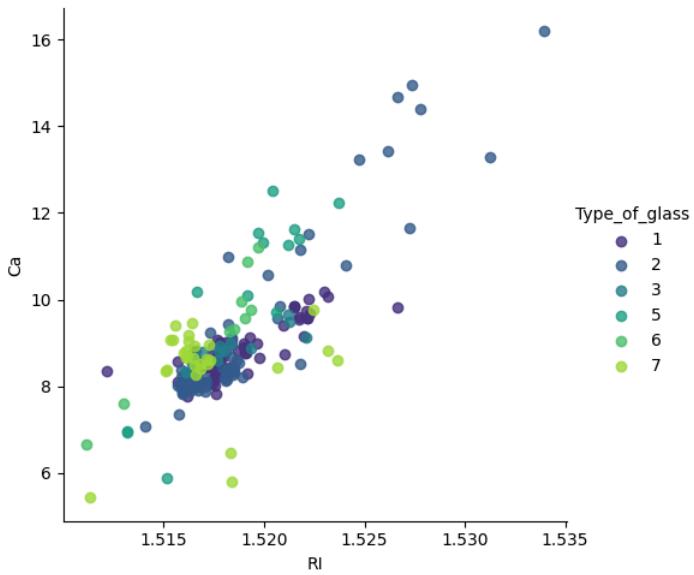


Dość silnie skorelowane są: (+) Ca - RI, oraz (+) Al - Ba. Natomiast (-) Ca - Mg, (-) Ba - Mg, (-) Si - RI.

Wybierzmy sobie dwa atrybuty i narysujmy wykres zależności między nimi. Wybieramy atrybuty RI oraz Na, ponieważ ich rozkłady są najbardziej zbliżone do rozkładu normalnego.

```
In [ ]: sns.lmplot(data=glass_df, x='RI', y='Ca', hue='Type_of_glass', palette='viridis', fit_reg=False)
```

```
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x1746f7e53d0>
```

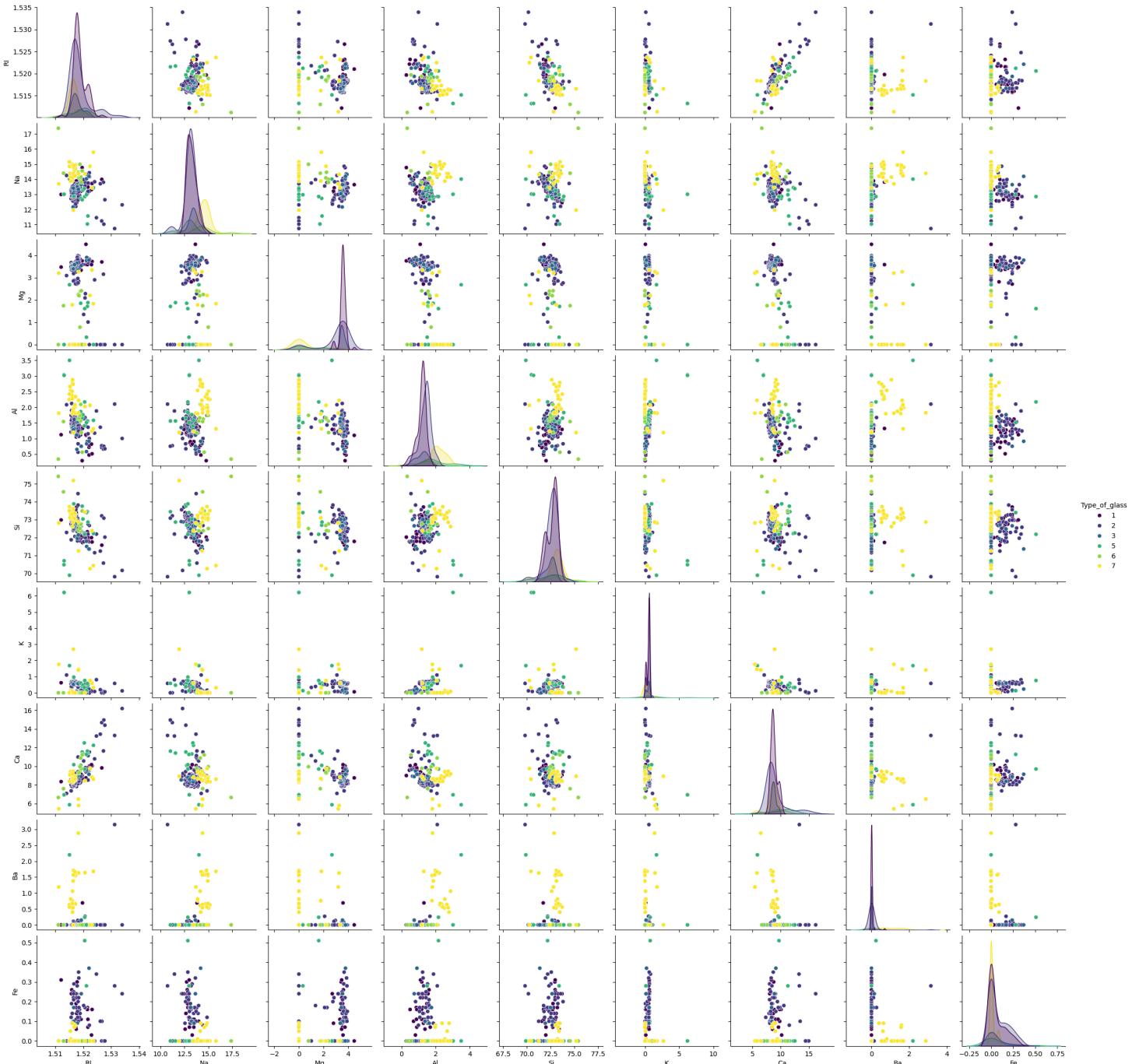


Można zauważyć, że ciężko jest odseparować klasy na podstawie tych dwóch atrybutów.

Wykreślmy zależności pomiędzy atrybutami, zaznaczając kolorem klasę szkła.

```
In [ ]: sns.pairplot(glass_df, hue='Type_of_glass', palette='viridis')
```

```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x174624ba850>
```



Wykonajmy standaryzację danych przed użyciem PCA.

```
In [ ]:
scaler = StandardScaler()
glass_X_scaled = scaler.fit_transform(glass_X)

pca = decomposition.PCA(n_components=2)
glass_X_pca = pca.fit_transform(glass_X_scaled)

glass_X_pca_df = pd.DataFrame(data=glass_X_pca, columns=['PC1', 'PC2'])
glass_X_pca_df['Type_of_glass'] = glass_y
glass_X_pca_df.head()
```

	PC1	PC2	Type_of_glass
0	1.151140	-0.529488	1
1	-0.574137	-0.759788	1
2	-0.940160	-0.929836	1
3	-0.142083	-0.961677	1
4	-0.351092	-1.091249	1

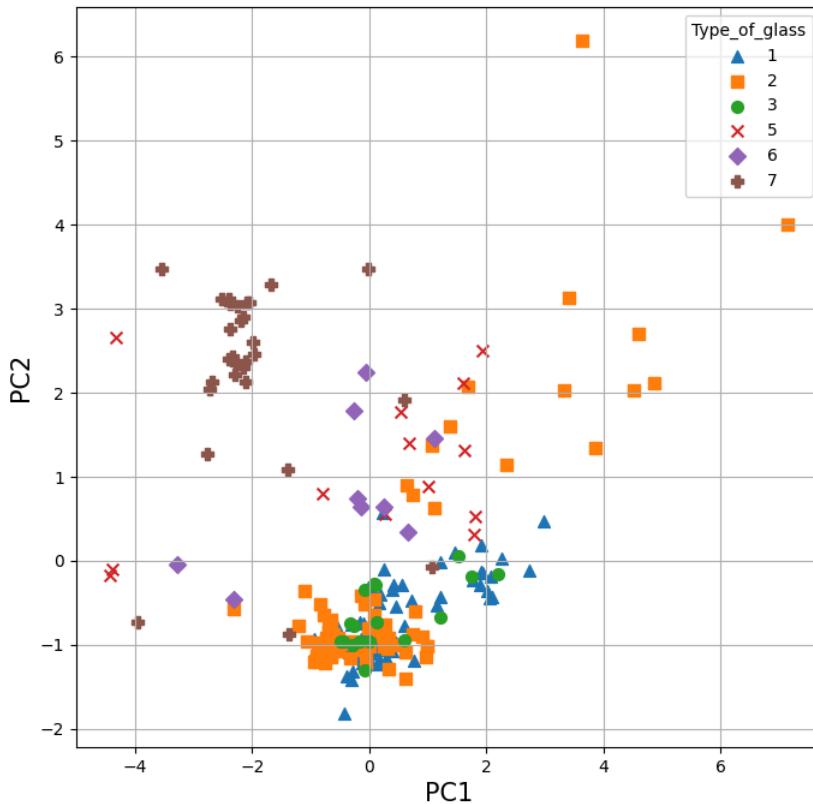
```
In [ ]:
# plot
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111)
ax.set_xlabel('PC1', fontsize=15)
ax.set_ylabel('PC2', fontsize=15)
markers = ("^", "s", "o", "x", "D", "P", "H")

targets = glass_X_pca_df['Type_of_glass'].unique()
for target, marker in zip(targets, markers):
    indicesToKeep = glass_X_pca_df['Type_of_glass'] == target
    ax.scatter(
        glass_X_pca_df.loc[indicesToKeep, 'PC1'],
        glass_X_pca_df.loc[indicesToKeep, 'PC2'],
        s=50,
        marker=marker)
```

```

ax.legend(targets, title='Type_of_glass', loc='upper right')
ax.grid()
plt.show()

```



Nawet po użyciu PCA klasy są bardzo trudne do odseparowania. Wniosek jest taki, że na postawie zbioru danych Glass, trudno jest odseparować klasy na podstawie atrybutów, które posiadamy.

Pytania pomocnicze

- Czym się różnią zbiory danych analizowane w treści zadania? Na czym może polegać „trudność” analizy. Który z nich wydaje się być łatwiejszy/trudniejszy?
 - Odp: Zbiory różnią się ilością atrybutów opisujących dane, ilością klas oraz rozkładem klas w zbiorze. Zbiór Glass jest najtrudniejszy do analizy, ponieważ posiada najwięcej atrybutów, najwięcej klas oraz nierównomierny rozkład klas w zbiorze. Zbiór Iris jest najłatwiejszy do analizy, ponieważ posiada najmniej atrybutów, najmniej klas oraz równomierny rozkład klas w zbiorze.
 - Odp: Ogólnie rzecz biorąc to ilość wymiarów ma wpływ na czytelność danych, możliwość ich analizy dla człowieka. Im więcej wymiarów tym trudniej jest zwizualizować dane. W przypadku zbioru Glass, który ma 9 atrybutów, jest to trudniejsze niż w przypadku zbioru Iris, który ma 4 atrybuty. W przypadku zbioru Wine, który ma 13 atrybutów, jest to jeszcze trudniejsze niż w przypadku zbioru Glass.
- Czy nierównomierny rozkład klas w zbiorze może stanowić problem dla analizy i dalszej budowy modelu danych?
 - Odp: Tak, nierównomierny rozkład klas w zbiorze może stanowić problem dla analizy i dalszej budowy modelu danych. W przypadku nierównomiernego rozkładu klas, model może nauczyć się dobrze klasyfikować klasy, które są liczniejsze, a gorzej klasy, które są mniej liczne. W takim przypadku model może być mniej dokładny w klasyfikacji klas, które są mniej liczne. Przykładem jest zbiór Glass, gdzie klasy 2 oraz 1 są liczniejsze niż pozostałe klasy. Dominujące klasy mogą być dobrze klasyfikowane, a mniej liczne klasy mogą być gorzej klasyfikowane.
- Jak działa PCA i kiedy warto go stosować
 - Odp: PCA (Principal Component Analysis) to metoda redukcji wymiarów. PCA wybiera składowe, które najlepiej opisują zróżnicowanie klas. PCA warto stosować wtedy, kiedy chcemy zredukować wymiarowość danych, a jednocześnie zachować jak najwięcej informacji. Jest to forma przetwarzania danych, która pozwala na zredukowanie ilości wymiarów, co pozwala na zmniejszenie złożoności modelu, a jednocześnie zachowanie jak najwięcej informacji. Dobrym nawykiem jest ustandaryzowanie danych przed użyciem PCA.