COMARCH
Szkolenia

# Python w analizie danych

## Wstęp do Data Science

**Paweł Goleń**

Trener

# Paweł Goleń

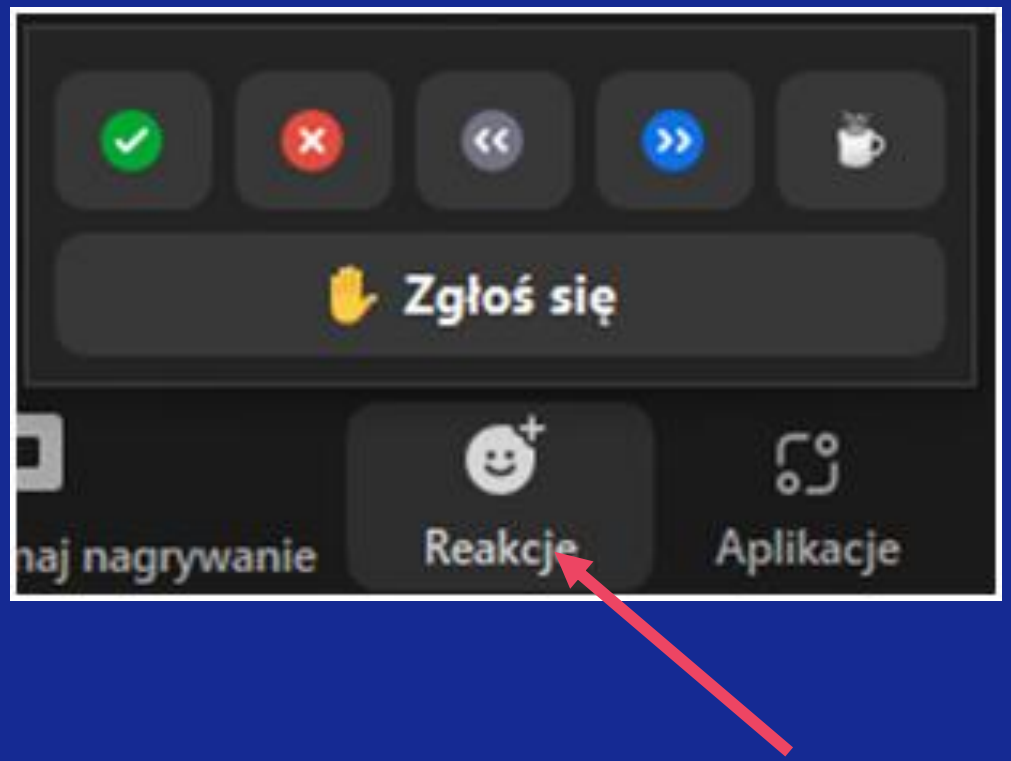
AI Enabled Automation Developer

COMARCH
Szkolenia

# Agenda

1. Podstawy statystyki
2. Python – wybrane elementy
3. Analiza z NumPy i Pandas
4. Wizualizacja w Matplotlib Seaborn

# Szkolenia zdalne - Reakcje




COMARCH

# Wprowadzenie

Omówienie celów i zakresu szkolenia:

- Celem tego szkolenia jest dostarczenie solidnych podstaw w zakresie analizy danych przy użyciu języka Python. Po zakończeniu szkolenia uczestnik szkolenia będzie w stanie samodzielnie przeprowadzić analizę danych, korzystając z bibliotek takich jak Pandas, Numpy, Matplotlib i Seaborn.

- Zakres szkolenia obejmuje tematykę analizy danych, począwszy od podstaw statystyki, aż po zaawansowane techniki analizy danych. Praca będzie polegać na obcowaniu z rzeczywistymi danymi, tak aby zastosować zdobytą wiedzę w praktyce.

# Wprowadzenie

Na czym polega analiza danych i dlaczego jest istotna:

- Analiza danych to kluczowy element podejmowania decyzji biznesowych. Chodzi o wydobywanie cennych informacji z danych, które pomagają zrozumieć trendy, prognozować wyniki i podejmować świadome decyzje. Bez analizy danych trudno jest efektywnie działać w dzisiejszym złożonym środowisku biznesowym.

Rola Pythona w analizie danych:

- Python stał się liderem w dziedzinie analizy danych, a to z kilku powodów. Jego czytelna składnia, ogromna społeczność, oraz bogactwo bibliotek, takich jak Pandas czy Numpy, sprawiają, że jest doskonałym narzędziem dla analityków danych. Podczas szkolenia zgłębimy, dlaczego Python jest tak popularny w tej dziedzinie.

# Wprowadzenie

Omówienie środowiska pracy - efektywne korzystanie z Jupyter Notebook:

- Jupyter Notebook to potężne narzędzie, które ułatwia interaktywną pracę z danymi i kodem. Dzięki niemu możliwe jest eksperymentowanie, wizualizacja i dokumentacja pracy. Podczas szkolenia przekazana zostanie wiedza jak korzystać z różnych funkcji Jupyter Notebook, aby móc skutecznie przeprowadzać analizę danych.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Instalacja Anacondy:**

- Krok 1: Pobierz Anacondę

  Przejdź na oficjalną stronę Anacondy: https://www.anaconda.com/products/distribution

- Krok 2: Wybierz wersję

  Pobierz odpowiednią wersję Anacondy dla swojego systemu operacyjnego (Windows, macOS, Linux). Zazwyczaj poleca się pobranie najnowszej stabilnej wersji.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Instalacja Anacondy:**

Krok 3: Uruchom instalator

- Uruchom pobrany plik instalacyjny Anacondy i postępuj zgodnie z instrukcjami na ekranie.

Krok 4: Wybierz opcje instalacji

- Podczas instalacji możesz zostawić domyślne opcje, chyba że masz konkretne preferencje dotyczące instalacji.

Krok 5: Zakończ instalację

- Po zakończeniu instalacji Anacondy powinna być gotowa do użycia.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Uruchomienie Jupyter Notebook:**

Krok 1: Uruchom Anacondę Navigator

- Po zainstalowaniu Anacondy, otwórz Anaconda Navigator. Możesz znaleźć go w menu Start (Windows) lub w terminalu (Linux/macOS) wpisując anaconda-navigator i naciskając Enter.

Krok 2: Uruchom Jupyter Notebook

- W Anaconda Navigatorze, znajdź i uruchom Jupyter Notebook. Kliknij na ikonę "Launch" obok Jupyter Notebook.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Uruchomienie Jupyter Notebook:**

Krok 3: Przeglądarka Jupyter Notebook

- Po chwili powinna otworzyć się przeglądarka internetowa z interfejsem Jupyter Notebook. Możesz przeglądać swoje pliki, tworzyć nowe notatniki i uruchamiać kod.

Krok 4: Utwórz nowy notatnik

- Kliknij na przycisk "New" i wybierz "Python 3". Otworzy się nowy notatnik, gdzie możesz wprowadzać kod.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:**

1. Komórki (Cells):
   - Jupyter Notebook jest podzielony na komórki, które można traktować jako jednostki kodu lub tekstu. Komórki kodu są przeznaczone do wprowadzania i wykonywania kodu, natomiast komórki tekstu mogą zawierać opisy, instrukcje czy też równania matematyczne.

2. Typy Komórek:
   - Code (Kod): Komórki, w których wprowadza się i wykonuje kod Pythona.
   - Markdown: Komórki, które zawierają tekst sformatowany w języku Markdown. Wykorzystywane do dodawania komentarzy, opisów czy instrukcji.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:**

3. Uruchamianie Komórek:
   - Aby uruchomić kod w komórce, możesz użyć przycisku "Run" lub skrótu klawiszowego Shift + Enter.
   - Wynik działania kodu będzie wyświetlany poniżej komórki.

4. Interaktywność:
   - Jupyter Notebook pozwala na interaktywną pracę z danymi. Możesz eksperymentować z kodem, zmieniać wartości i natychmiast obserwować rezultaty.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:**

5. Wbudowane Komendy Magiczne:
   - Jupyter obsługuje tzw. "komendy magiczne" poprzez prefiks "%" lub "%%". Przykładowo, `%matplotlib inline` pozwala na wyświetlanie wykresów bezpośrednio w notatniku.

6. Wizualizacje:
   - Możesz używać bibliotek takich jak Matplotlib czy Seaborn do tworzenia wykresów i wizualizacji danych. Wykresy są wyświetlane w notatniku, co ułatwia analizę.

# Przygotowanie środowiska pracy - Jupyter Notebook

**Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:**

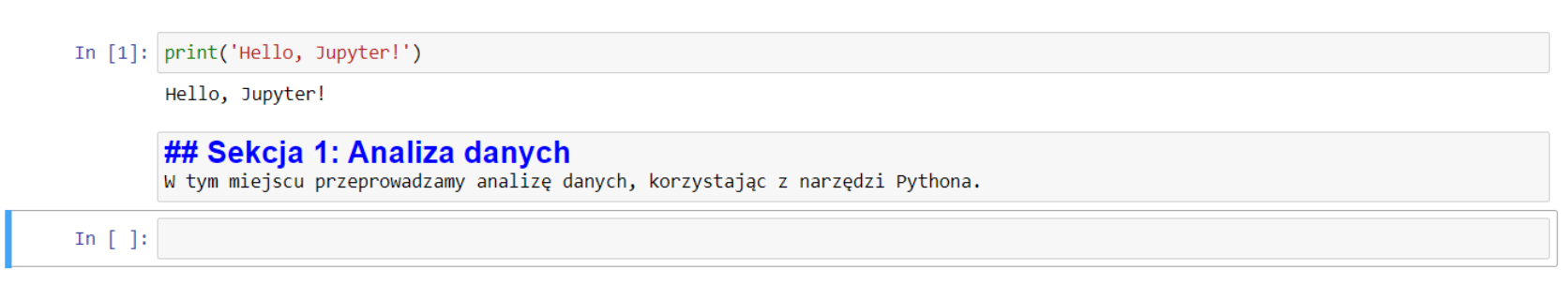
7. Zapisywanie i Eksport:
   - Notatniki można zapisywać w formatach .ipynb (Jupyter Notebook), .html, .pdf, .py (skrypt Pythona) i innych.
   - Można eksportować notatniki do różnych formatów, co ułatwia ich udostępnianie.

8. Obsługa Notatnika:
   - Notatnik zachowuje wyniki i zmienne nawet po ponownym uruchomieniu komórki. To ułatwia analizę i eksperymentowanie bez konieczności ponownego uruchamiania wszystkich komórek.

# Omówienie środowiska pracy - Jupyter Notebook

Przykład:

Poniżej znajduje się prosty przykład notatnika Jupyter, składającego się z komórek kodu i tekstu:




- Zadanie:
    1. Otwórz Jupyter Notebook.
    2. Utwórz nowy notatnik.
    3. Dodaj kilka komórek kodu i tekstu.
    4. Uruchom kod w komórkach i zobacz, jak zmieniają się wyniki.

# Omówienie środowiska pracy - Jupyter Notebook

Kilka przydatnych skrótów klawiszowych w Jupyter Notebook, które mogą zwiększyć efektywność pracy:

- Shift + Enter: Uruchamia aktualną komórkę i przechodzi do następnej. Jeśli ostatnia komórka, to dodaje nową komórkę poniżej.
- Ctrl + Enter: Uruchamia aktualną komórkę, ale nie przechodzi do następnej, pozostając w bieżącej komórce.
- Esc + A: Dodaje nową komórkę powyżej aktualnej.
- Esc + B: Dodaje nową komórkę poniżej aktualnej.
- Esc + M: Zmienia typ komórki na Markdown (tekst).
- Esc + Y: Zmienia typ komórki na Code (kod).
- Esc + D, D: Kasuje aktualną komórkę.
- Esc + Z: Cofa ostatnią zmianę (przywraca skasowaną komórkę).

# Omówienie środowiska pracy - Jupyter Notebook

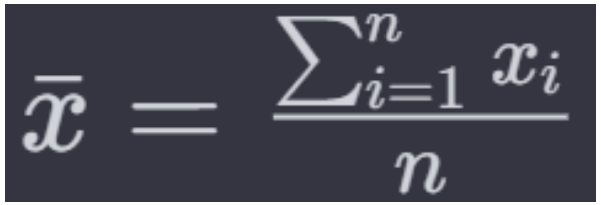
Kilka przydatnych skrótów klawiszowych w Jupyter Notebook, które mogą zwiększyć efektywność pracy:

- Shift + Tab: Pokazuje podpowiedzi dotyczące funkcji lub metody (po umieszczeniu kursora wewnątrz nawiasów).
- Ctrl + S: Zapisuje notatnik.
- Esc + 1-6: Zmienia poziom nagłówka w komórce Markdown (1- najwyższy, 6- najniższy).
- Esc + H: Wyświetla listę wszystkich dostępnych skrótów klawiszowych.
- Ctrl + Shift + P: Otwiera polecenie paska wyszukiwania, pozwalając na szybkie wyszukiwanie i uruchamianie poleceń.
- Shift + M: Łączy zaznaczone komórki w jedną.
- Ctrl + Z: Cofa ostatnią zmianę.

# Omówienie środowiska pracy - Jupyter Notebook

Te skróty klawiszowe mogą być bardzo przydatne podczas korzystania z Jupyter Notebook, przyspieszając pisanie kodu i nawigację w notatniku.

# Podstawy statystyki – podstawowe pojęcia statystyczne

1. Średnia arytmetyczna (Mean):
   - Średnia arytmetyczna to suma wszystkich liczb podzielona przez ich ilość.
   - Wzór: 
$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$
2. Mediana:
   - Mediana to środkowa wartość w uporządkowanym zbiorze danych.
     - W przypadku nieparzystej liczby elementów, mediana to wartość środkowa.
     - W przypadku parzystej liczby elementów, mediana to średnia arytmetyczna dwóch wartości środkowych.

# Omówienie środowiska pracy - Jupyter Notebook

3. Dominanta:
   - Dominanta to najczęściej występująca wartość w zbiorze danych.

4. Rozstęp:
   - Rozstęp to różnica między największą a najmniejszą wartością w zbiorze danych.

# Podstawy statystyki – podstawowe pojęcia statystyczne

5. Odchylenie standardowe:

- Odchylenie standardowe mierzy, jak bardzo dane różnią się od średniej arytmetycznej.
- Im większe odchylenie standardowe, tym większa zmienność danych
- Wzór:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

# Podstawy statystyki - Podstawowe pojęcia statystyczne

6. Kwartyle:
   - Kwartyle dzielą uporządkowany zbiór danych na cztery równe części:
     - Q1 (pierwszy kwartyl) to mediana pierwszej połowy danych.
     - Q2 (drugi kwartyl) to mediana całego zbioru danych.
     - Q3 (trzeci kwartyl) to mediana drugiej połowy danych.

7. Wartości odstające (Outliers):
   - Wartości odstające to wartości, które znacznie odbiegają od reszty danych w zbiorze.

# Podstawy statystyki – podstawowe pojęcia statystyczne

Przykład:

- Rozważmy zestaw danych dotyczący wieku osób w grupie:

  {21, 22, 23, 24, 24, 25, 26, 26, 27, 27, 28, 30, 30, 30, 31}

- Średnia arytmetyczna: $\bar{x} = \frac{21+22+\ldots+31}{15}$
- Mediana: 26 (wartość środkowa)
- Dominanta: 30 (najczęściej występująca wartość)
- Odchylenie standardowe: $\sigma = \sqrt{\frac{(21-\bar{x})^2+(22-\bar{x})^2+\ldots+(31-\bar{x})^2}{15}}$
- Rozstęp: 31 − 21 = 10
- Kwartyle: Q1 = 24, Q2 = 26, Q3 = 30

# Podstawy statystyki - Podstawowe pojęcia statystyczne

Ćwiczenie 1: Średnia arytmetyczna, Mediana, Dominanta

- Zadanie:

  Mając dany zbiór danych dotyczący wieku osób, oblicz średnią arytmetyczną, medianę i dominantę.

```
In [1]: # Dane dotyczące wieku osób

        wiek = [21, 22, 23, 24, 24, 25, 26, 26, 27, 27, 28, 30, 30, 30, 31]
```

# Podstawy statystyki – podstawowe pojęcia statystyczne

Rozwiązanie:

In [2]:
```
import statistics

# Średnia arytmetyczna
srednia = statistics.mean(wiek)

# Mediana
mediana = statistics.median(wiek)

# Dominanta
dominanta = statistics.mode(wiek)
```

In [3]:
```
print(srednia, mediana, dominanta)
```

```
26.266666666666666 26 30
```

In [4]:
```
print(srednia)
print(mediana)
print(dominanta)
```

```
26.266666666666666
26
30
```

# Podstawy statystyki - Podstawowe pojęcia statystyczne

Ćwiczenie 2: Odchylenie Standardowe

- Zadanie:

  Oblicz odchylenie standardowe dla zbioru danych dotyczącego wieku osób.

```
In [1]: # Dane dotyczące wieku osób

        wiek = [21, 22, 23, 24, 24, 25, 26, 26, 27, 27, 28, 30, 30, 30, 31]
```

# Podstawy statystyki – podstawowe pojęcia statystyczne

Rozwiązanie:

```
In [6]: # Odchylenie standardowe
        odchylenie_std = statistics.stdev(wiek)

In [7]: print(odchylenie_std)

        3.1274514194392182
```

# Podstawy statystyki - Podstawowe pojęcia statystyczne

Ćwiczenie 3: Rozstęp, Kwartyle

- Zadanie:

  Oblicz rozstęp oraz kwartyle dla zbioru danych dotyczącego wieku osób.

```
In [1]: # Dane dotyczące wieku osób

        wiek = [21, 22, 23, 24, 24, 25, 26, 26, 27, 27, 28, 30, 30, 30, 31]
```

# Podstawy statystyki – podstawowe pojęcia statystyczne

Rozwiązanie:

In [8]:
```
# Rozstęp
rozstep = max(wiek) - min(wiek)

# Kwartyle
q1 = statistics.quantiles(wiek, n=4)[0]
q2 = statistics.quantiles(wiek, n=4)[1]
q3 = statistics.quantiles(wiek, n=4)[2]
```

In [9]:
```
print(rozstep)
print(q1)
print(q2)
print(q3)
```

```
10
24.0
26.0
30.0
```

Te ćwiczenia pomogą w zrozumieniu, jak używać funkcji statystycznych w Pythonie do analizy danych.

# Podstawy statystyki - Podstawowe pojęcia statystyczne

Wyjaśnienie:

1. Rozstęp:
   - Rozstęp to różnica między największą a najmniejszą wartością w zbiorze danych.
   - W tym przypadku rozstęp będzie równy 31 - 21 = 10.

# Podstawy statystyki – podstawowe pojęcia statystyczne

2. Kwartyle:
   - Kwartyle dzielą uporządkowany zbiór danych na cztery równe części.
   - Funkcja quantiles z modułu statistics pozwala na obliczenie kwartyli.
   - Parametr n=4 oznacza, że dzielimy zbiór na 4 równoliczne części.
   - Wartości q1, q2, i q3 oznaczają odpowiednio pierwszy, drugi (mediana) i trzeci kwartyl.

```
In [10]: q1 = statistics.quantiles(wiek, n=4)[0]  # Pierwszy kwartyl
         q2 = statistics.quantiles(wiek, n=4)[1]  # Drugi kwartyl (mediana)
         q3 = statistics.quantiles(wiek, n=4)[2]  # Trzeci kwartyl
```

# Podstawy statystyki - Podstawowe pojęcia statystyczne

W przypadku naszego zbioru danych:

- q1 = 24
- q2 = 26 (mediana)
- q3 = 30

Metoda quantiles jest używana do obliczania kwantyli dla danego zbioru danych. Kwartyle są to specjalne przypadki kwantyli, gdzie dzielimy zbiór na 4 równe części. Funkcja ta pozwala na elastyczne dostosowywanie liczby równolicznych przedziałów (kwantyli).

# Podstawy statystyki – podstawowe pojęcia statystyczne

W nawiasie kwadratowym umieszczana jest liczba, która określa na ile równych części chcemy podzielić zbiór danych. W przypadku metody quantiles z modułu statistics, ta liczba oznacza ilość punktów podziału, czyli ilość części, na jakie chcemy podzielić zbiór.

```
In [10]: q1 = statistics.quantiles(wiek, n=4)[0]  # Pierwszy kwartyl
         q2 = statistics.quantiles(wiek, n=4)[1]  # Drugi kwartyl (mediana)
         q3 = statistics.quantiles(wiek, n=4)[2]  # Trzeci kwartyl
```

# Podstawy statystyki - Podstawowe pojęcia statystyczne

W naszym przykładzie, ustawiając n=4, określamy, że chcemy podzielić zbiór na 4 równoliczne części, co odpowiada kwartylom. W praktyce, ustawiając `n=100`, moglibyśmy podzielić zbiór na percentyle, czyli 100 równolicznych części.

```
In [10]: q1 = statistics.quantiles(wiek, n=4)[0]   # Pierwszy kwartyl
         q2 = statistics.quantiles(wiek, n=4)[1]   # Drugi kwartyl (mediana)
         q3 = statistics.quantiles(wiek, n=4)[2]   # Trzeci kwartyl
```

# Podstawy statystyki – podstawowe pojęcia statystyczne

- Przykład z n=100:

```
In [11]: percentyl_25 = statistics.quantiles(wiek, n=100)[24]  # 25 percentyl
         percentyl_50 = statistics.quantiles(wiek, n=100)[49]  # 50 percentyl (mediana)
         percentyl_75 = statistics.quantiles(wiek, n=100)[74]  # 75 percentyl
```

- Wartość w nawiasie kwadratowym definiuje, na ile części chcemy podzielić zbiór danych, a liczby w nawiasie okrągłym wskazują konkretny punkt podziału. W przypadku kwartyli, mamy 4 części, stąd n=4.

# Podstawy statystyki – podstawowe miary rozkładu

1. Rozkład Jednostajny:
   - Rozkład jednostajny charakteryzuje się równomiernym rozkładem prawdopodobieństwa w określonym zakresie.
   - Wszystkie wartości w danym przedziale mają równe prawdopodobieństwo wystąpienia.

# Podstawy statystyki – podstawowe miary rozkładu

2. Rozkład Normalny (Gaussowski):
   - Rozkład normalny jest najczęściej spotykanym rozkładem w statystyce.
   - Charakteryzuje się kształtem dzwonu (bell-shaped curve).
   - Posiada dwie ważne miary:
     - średnią $(\mu)$
     - odchylenie standardowe $(\sigma)$

# Podstawy statystyki – podstawowe miary rozkładu

3. Rozkład Skośny:
   - Rozkład skośny (przechylony) ma asymetryczny kształt.
   - Wartości skośne w lewo mają długi ogon w lewo, a wartości skośne w prawo mają długi ogon w prawo.

4. Rozkład Kurtozy:
   - Mierzy "spiczastość" lub "płaskość" rozkładu.
   - Rozkłady leptokurtyczne (kurtoza > 3) mają dłuższe i węższe ogony, podczas gdy rozkłady płaskokurtyczne (kurtoza < 3) mają krótsze ogony.

5. Rozkład Dwumodalny:
   - Rozkład dwumodalny składa się z dwóch różnych modów (szczytów), co oznacza, że dane zawierają dwie różne dominujące wartości.

# Podstawy statystyki – podstawowe miary rozkładu

Opis wykorzystywanych metod:

1. np.random.uniform()
    - Metoda np.random.uniform() generuje losowe dane z rozkładu jednostajnego.
        - low: Dolny zakres wartości.
        - high: Górny zakres wartości.
        - size: Kształt generowanego zestawu danych.

# Podstawy statystyki – podstawowe miary rozkładu

Opis wykorzystywanych metod:

2. np.random.normal()

- Metoda np.random.normal() generuje losowe dane z rozkładu normalnego (Gaussowskiego).
  - loc: Średnia rozkładu (wartość oczekiwana).
  - scale: Odchylenie standardowe rozkładu.
  - size: Kształt generowanego zestawu danych.

# Podstawy statystyki – podstawowe miary rozkładu

Opis wykorzystywanych metod:

3. np.random.gamma()

- Metoda np.random.gamma() generuje losowe dane z rozkładu gamma.
  - shape: Parametr kształtu (kształt rozkładu).
  - scale: Skalowanie rozkładu (opcjonalne, domyślnie 1.0).
  - size: Kształt generowanego zestawu danych.

# Podstawy statystyki – podstawowe miary rozkładu

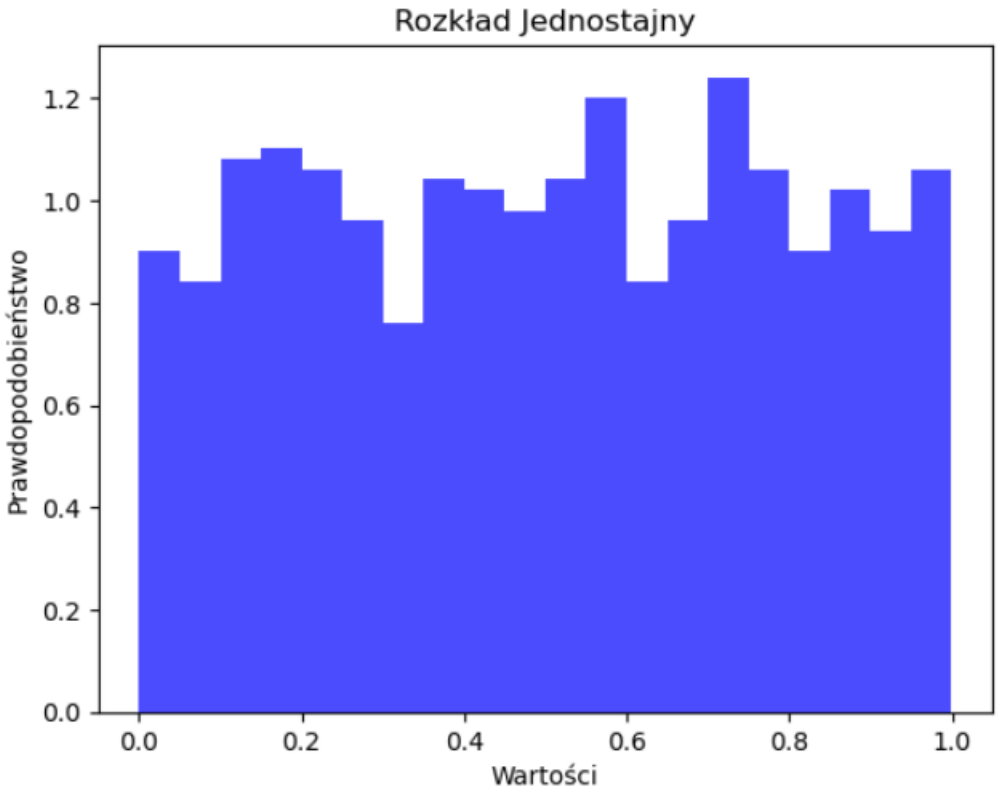
## Przykład 1: Rozkład Jednostajny

In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt

# Generowanie danych z rozkładu jednostajnego
data_uniform = np.random.uniform(0, 1, 1000)

# Wykres rozkładu jednostajnego
plt.hist(data_uniform, bins=20, density=True, alpha=0.7, color='b')
plt.title('Rozkład Jednostajny')
plt.xlabel('Wartości')
plt.ylabel('Prawdopodobieństwo')
plt.show()
```

# Podstawy statystyki – podstawowe miary rozkładu

Wizualizacja Rozkładu Jednostajnego



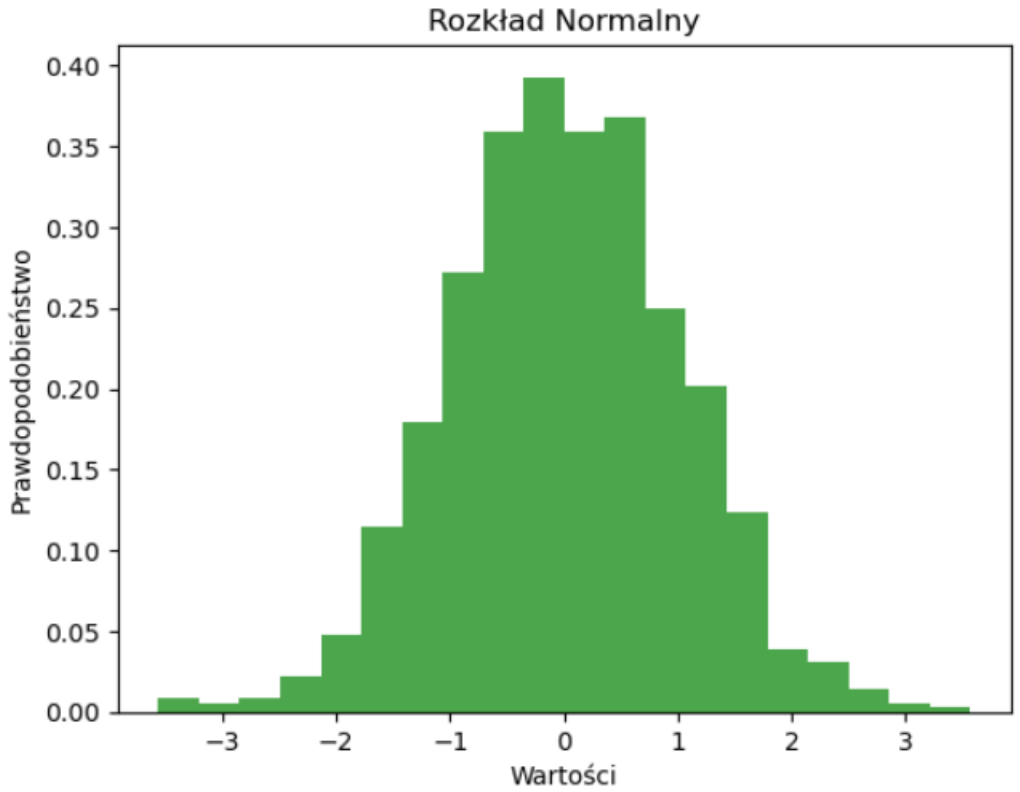
# Podstawy statystyki – podstawowe miary rozkładu

## Przykład 2: Rozkład Normalny

In [2]:
```
# Generowanie danych z rozkładu normalnego
data_normal = np.random.normal(0, 1, 1000)

# Wykres rozkładu normalnego
plt.hist(data_normal, bins=20, density=True, alpha=0.7, color='g')
plt.title('Rozkład Normalny')
plt.xlabel('Wartości')
plt.ylabel('Prawdopodobieństwo')
plt.show()
```

# Podstawy statystyki – podstawowe miary rozkładu

Wizualizacja Rozkładu Normalnego



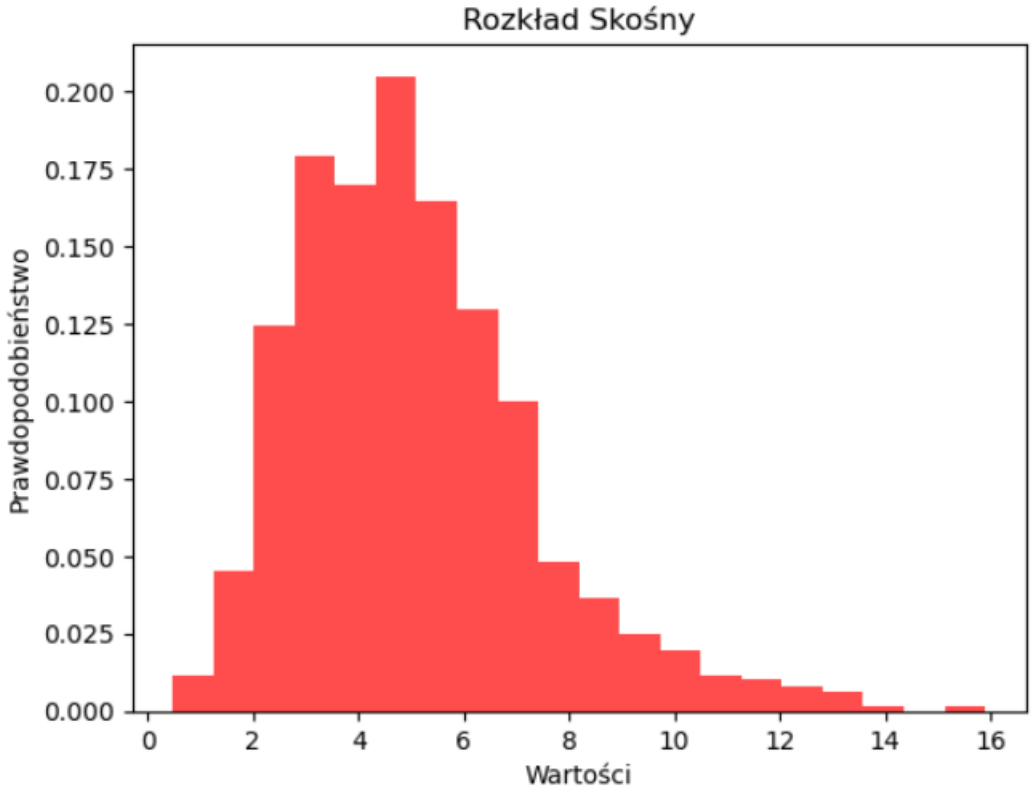
# Podstawy statystyki – podstawowe miary rozkładu

## Przykład 3: Rozkład Skośny

In [3]:
```
# Generowanie danych z rozkładu skośnego
data_skewed = np.random.gamma(5, size=1000)

# Wykres rozkładu skośnego
plt.hist(data_skewed, bins=20, density=True, alpha=0.7, color='r')
plt.title('Rozkład Skośny')
plt.xlabel('Wartości')
plt.ylabel('Prawdopodobieństwo')
plt.show()
```

# Podstawy statystyki – podstawowe miary rozkładu

Wizualizacja Rozkładu Skośnego



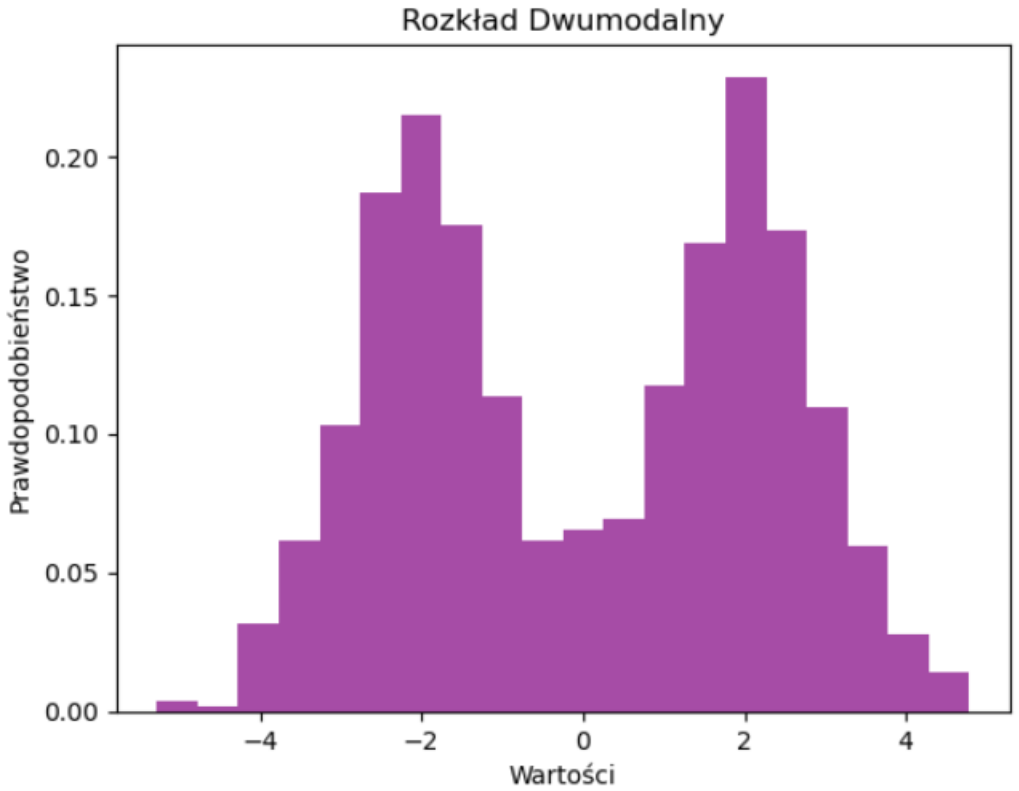
# Podstawy statystyki – podstawowe miary rozkładu

## Przykład 4: Rozkład dwumodalny

In [4]:
```
# Generowanie danych z rozkładu dwumodalnego
data_bimodal = np.concatenate([np.random.normal(-2, 1, 500), np.random.normal(2, 1, 500)])

# Wykres rozkładu dwumodalnego
plt.hist(data_bimodal, bins=20, density=True, alpha=0.7, color='purple')
plt.title('Rozkład Dwumodalny')
plt.xlabel('Wartości')
plt.ylabel('Prawdopodobieństwo')
plt.show()
```

# Podstawy statystyki – podstawowe miary rozkładu

Wizualizacja Rozkładu Dwumodalny



# Podstawy statystyki – podstawowe miary rozkładu

Ćwiczenie 1: Rozkład Jednostajny

- Zadanie:

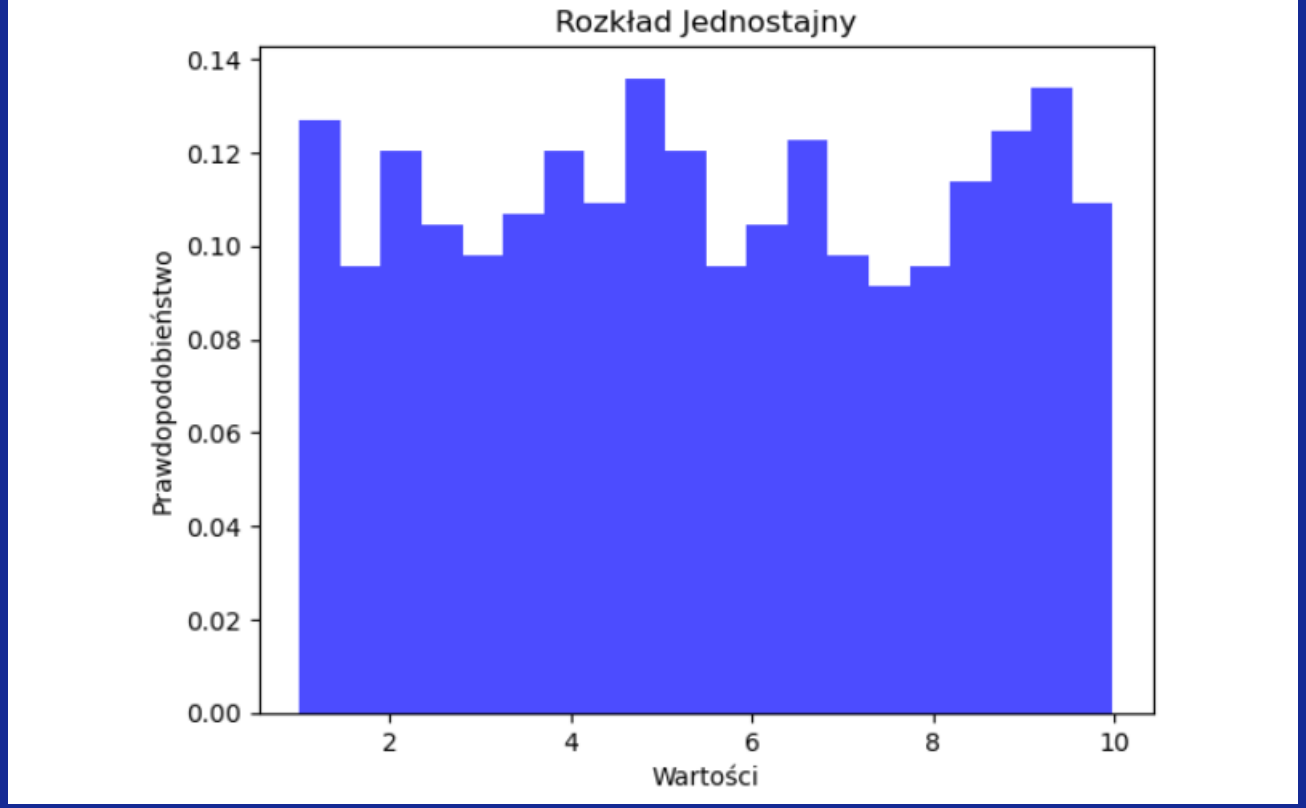
  Stwórz histogram dla zestawu danych o rozkładzie jednostajnym. Wykorzystaj funkcję generującą losowe liczby z rozkładu jednostajnego w zakresie od 1 do 10.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Wygeneruj losowe dane z rozkładu jednostajnego
data_uniform = np.random.uniform(1, 10, 1000)

# Stwórz histogram
# (Pamiętaj o dodaniu odpowiednich etykiet i tytułu wykresu)
```

# Podstawy statystyki – podstawowe miary rozkładu

Rozwiązanie:

```
In [2]: plt.hist(data_uniform, bins=20, density=True, alpha=0.7, color='b')
        plt.title('Rozkład Jednostajny')
        plt.xlabel('Wartości')
        plt.ylabel('Prawdopodobieństwo')
        plt.show()
```



# Podstawy statystyki – podstawowe miary rozkładu

Ćwiczenie 2: Rozkład Normalny

- Zadanie:

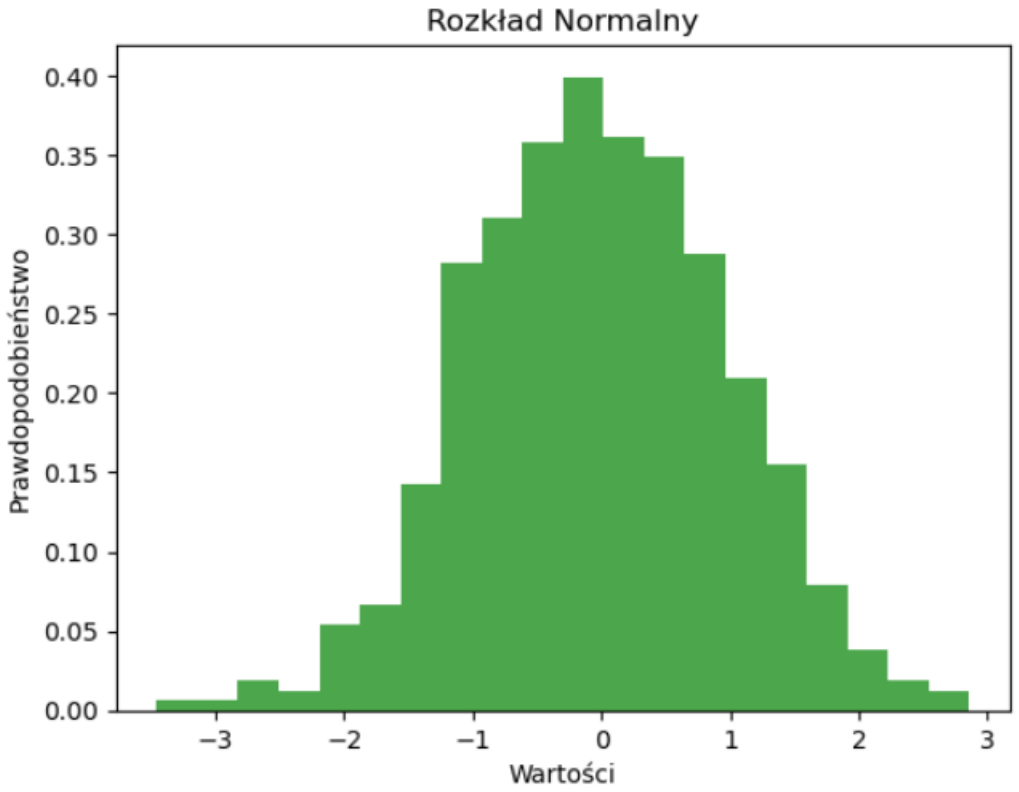
  Stwórz histogram dla zestawu danych o rozkładzie normalnym. Wykorzystaj funkcję generującą losowe liczby z rozkładu normalnego.

In [3]:
```
# Wygeneruj losowe dane z rozkładu normalnego
data_normal = np.random.normal(0, 1, 1000)

# Stwórz histogram
# (Pamiętaj o dodaniu odpowiednich etykiet i tytułu wykresu)
```

# Podstawy statystyki – podstawowe miary rozkładu

Rozwiązanie:

```
In [4]: plt.hist(data_normal, bins=20, density=True, alpha=0.7, color='g')
        plt.title('Rozkład Normalny')
        plt.xlabel('Wartości')
        plt.ylabel('Prawdopodobieństwo')
        plt.show()
```



# Podstawy statystyki – podstawowe miary rozkładu

Ćwiczenie 3: Rozkład Skośny

- Zadanie:

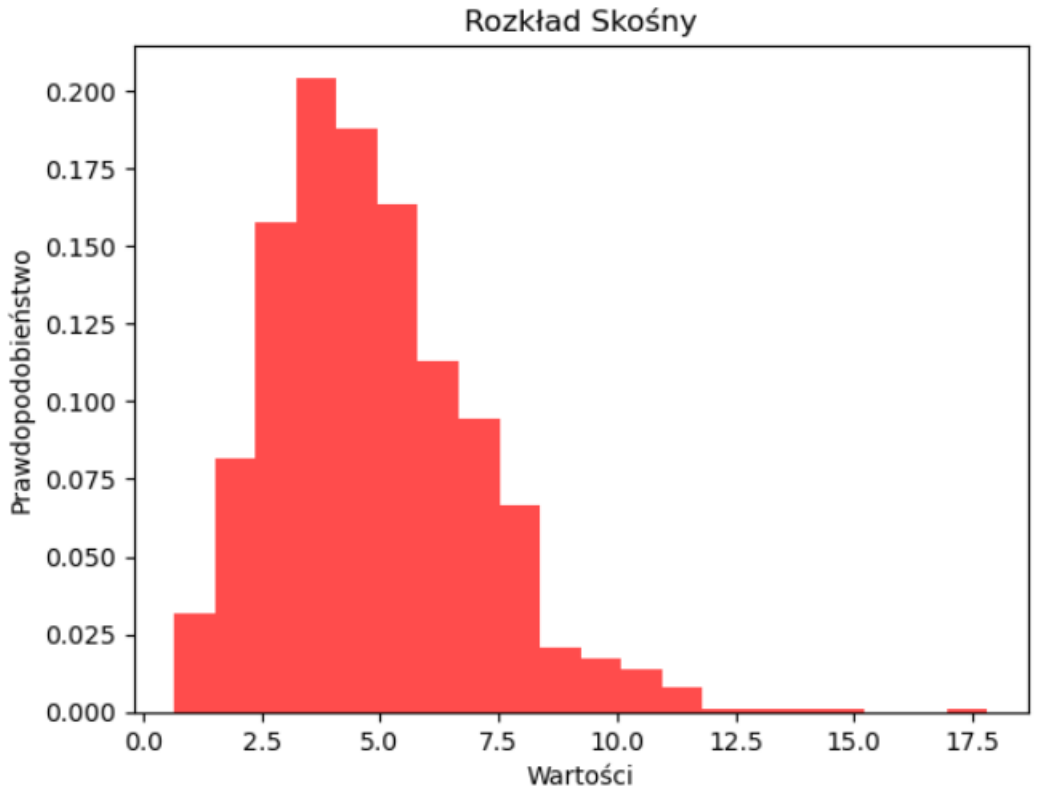
  Stwórz histogram dla zestawu danych o rozkładzie skośnym. Wykorzystaj funkcję generującą losowe liczby z rozkładu gamma.

In [3]:
```
# Wygeneruj losowe dane z rozkładu skośnego
data_skewed = np.random.gamma(5, size=1000)

# Stwórz histogram
# (Pamiętaj o dodaniu odpowiednich etykiet i tytułu wykresu)
```

# Podstawy statystyki – podstawowe miary rozkładu

Rozwiązanie:

```
In [4]: plt.hist(data_skewed, bins=20, density=True, alpha=0.7, color='r')
        plt.title('Rozkład Skośny')
        plt.xlabel('Wartości')
        plt.ylabel('Prawdopodobieństwo')
        plt.show()
```



# Podstawy statystyki – podstawowe miary rozkładu

Ćwiczenie 4: Rozkład Dwumodalny

- Zadanie:

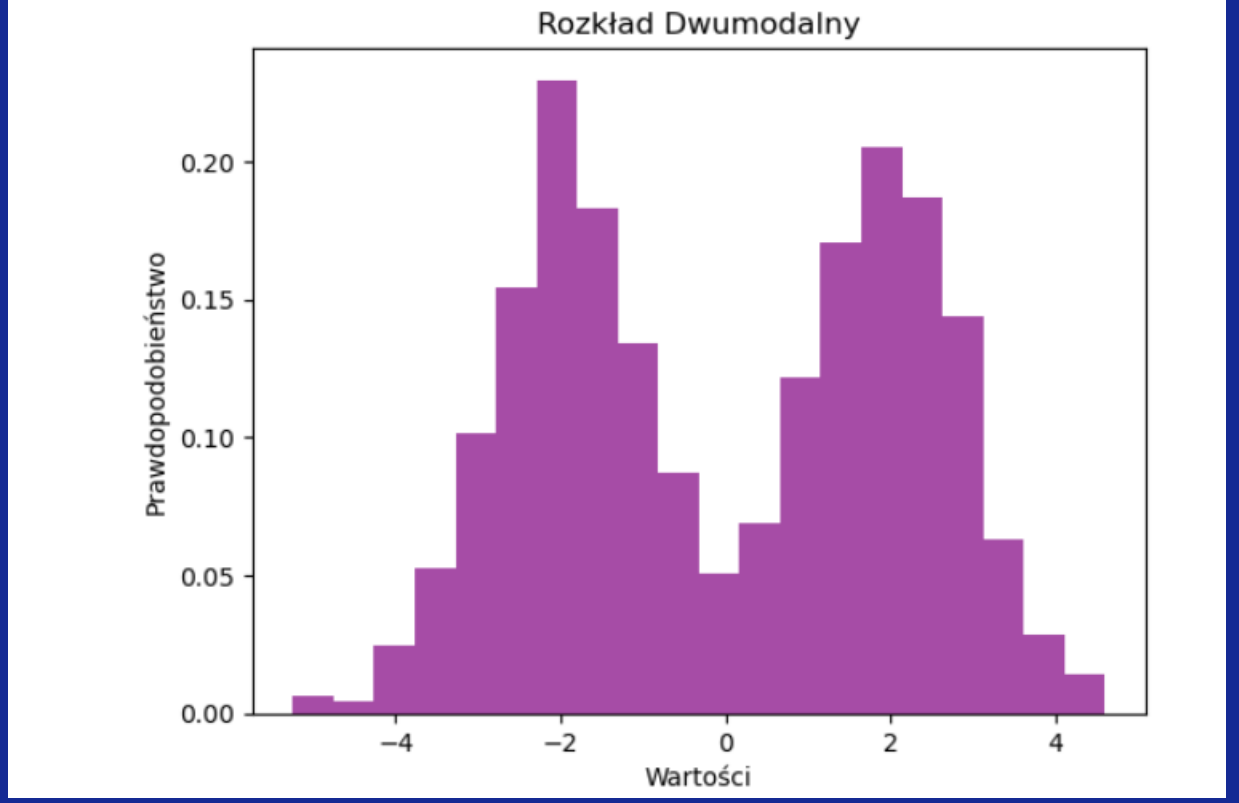
  Stwórz histogram dla zestawu danych o rozkładzie dwumodalnym. Wykorzystaj funkcję generującą losowe liczby z dwóch różnych rozkładów normalnych.

In [5]:
```
# Wygeneruj losowe dane z rozkładu dwumodalnego
data_bimodal = np.concatenate([np.random.normal(-2, 1, 500), np.random.normal(2, 1, 500)])

# Stwórz histogram
# (Pamiętaj o dodaniu odpowiednich etykiet i tytułu wykresu)
```

# Podstawy statystyki – podstawowe miary rozkładu

Rozwiązanie:

```
In [6]: plt.hist(data_bimodal, bins=20, density=True, alpha=0.7, color='purple')
        plt.title('Rozkład Dwumodalny')
        plt.xlabel('Wartości')
        plt.ylabel('Prawdopodobieństwo')
        plt.show()
```



# Python - Wybrane Elementy

W tej sekcji skupimy się na kilku kluczowych elementach języka Python, które są istotne w kontekście analizy danych i programowania w obszarze Data Science.

# Podstawy statystyki – podstawowe miary rozkładu

Różnica między metodą a funkcją w Pythonie dotyczy przede wszystkim kontekstu, w którym są one używane:

# Python - Wybrane Elementy

Metoda:

- Metoda jest funkcją, która jest związana z konkretnym obiektem lub typem danych.
- Jest wywoływana na rzecz konkretnego obiektu przy użyciu notacji kropkowej (.).
- Metoda może mieć dostęp do danych zawartych w obiekcie, na którym jest wywoływana, za pomocą argumentu self.

Przykładem metody jest .append() dla list, która dodaje element do listy, lub .capitalize() dla łańcuchów, która zmienia pierwszą literę na wielką.

# Podstawy statystyki – podstawowe miary rozkładu

Funkcja:

- Funkcja jest blokiem kodu, który wykonuje określone zadanie i może być wywoływany z dowolnego miejsca w programie.
- Nie jest związana z konkretnym obiektem ani typem danych.
- Funkcje są definiowane przy użyciu słowa kluczowego def i mogą przyjmować argumenty.

Przykładem funkcji jest print(), która wypisuje wartość na standardowe wyjście, lub len(), która zwraca długość obiektu.

# Python - Wybrane Elementy

- W skrócie, metoda jest specjalnym rodzajem funkcji, która jest związana z konkretnym obiektem, podczas gdy funkcja jest niezależnym blokiem kodu, który może być wywoływany w dowolnym kontekście.

- Metody mają dostęp do danych obiektu, na którym są wywoływane, podczas gdy funkcje nie mają takiego dostępu, chyba że dane są przekazywane jako argumenty.

# Python - Wybrane Elementy

1. Podstawowe Typy i Struktury Danych

- Liczby całkowite (int) i liczby zmiennoprzecinkowe (float):
  - Przykład:

```
In [1]: # Int & Float

        x = 5 # Int
        y = 3.14 # Float
```

# Python - Wybrane Elementy

1. Przypomnienie Podstawowych Typów i Struktur Danych

- Napisy (str):
  - Przykład:

```
In [2]: # String

        text = "Hello, Data Science!"
```

# Python - Wybrane Elementy

1. Przypomnienie Podstawowych Typów i Struktur Danych

Listy:

- Przykład:

```
In [3]: # Lista

        numbers = [1, 2, 3, 4, 5]
```

# Python - Wybrane Elementy

1. Przypomnienie Podstawowych Typów i Struktur Danych

Krotki (tuple):

- Przykład:

```
In [4]: # Tuple

        coordinates = (3, 4)
```

# Python - Wybrane Elementy

1. Przypomnienie Podstawowych Typów i Struktur Danych

Słowniki (dict):

- Przykład:

```
In [5]: # Dictionary

        student = {'name': 'John',
                   'age': 20,
                   'grade': 'A'
                  }
```

# Python - Wybrane Elementy

2. Indeksowanie, Slicing i Iteracja

- Indeksowanie:
  -  Indeksowanie w Pythonie zaczyna się od 0. 
    - Przykład:

```
In [6]: # Indeksowanie

        numbers = [10, 20, 30, 40, 50]
        first_element = numbers[0]  # Pobiera pierwszy element (10)
```

# Python - Wybrane Elementy

2. Indeksowanie, Slicing i Iteracja

Slicing:

- Pozwala na pobieranie fragmentów listy lub napisu.
  - Przykład:

```
In [7]: # Slicing

        numbers = [10, 20, 30, 40, 50]
        sliced_numbers = numbers[1:4]  # Pobiera elementy od indeksu 1 do 3
```

# Python - Wybrane Elementy

2. Indeksowanie, Slicing i Iteracja

Iteracja:

- Przykład:

```
In [8]: # Iteracja

        numbers = [10, 20, 30, 40, 50]
        for num in numbers:
            print(num)

        10
        20
        30
        40
        50
```

# Python - Wybrane Elementy

3. Funkcje, Funkcje Anonimowe (Lambda)

- Definiowanie Funkcji:
  - Przykład:

```
In [9]: # Definiowanie funkcji

        def add_numbers(x, y):
            return x + y

In [10]: print(add_numbers(3, 2))

         5
```

# Python - Wybrane Elementy

Funkcja Anonimowa (Lambda):

- Python Lambda, jest jednolinijkową, anonimową funkcją. Nie jest skomplikowana. Jest to funkcja która nie ma nazwy. Poprzez użycie słowa kluczowego 'lambda' informujemy Python, że właśnie taką anonimową funkcję chcemy utworzyć. Następnie podajemy listę parametrów, które chcemy aby przyjmowała, używamy „:", oraz definiujemy jej zawartość.

- W przeciwieństwie do poprzednich funkcji funkcja lambda nie jest funkcją wyższego rzędu, aby definiować funkcje „na stałe". Służy do wykorzystania ad hoc i do ułatwienia sobie życia. Jeżeli natomiast przypiszemy funkcję do zmiennej, tak jak w przykładzie będziemy mogli się do niej później odwołać.

# Python - Wybrane Elementy

Funkcja Anonimowa (Lambda):

- Przykład:

```
In [11]: # Funkcja Lambda

         multiply = lambda x, y: x * y

In [12]: print(multiply(2, 2))
         4

In [13]: print(multiply(2, 4))
         8

In [14]: (lambda x, y: x * y)(3, 3)
Out[14]: 9

In [15]: print((lambda x, y: x * y)(3, 3))
         9
```

# Python - Wybrane Elementy

4. List Comprehension
   - Umożliwia zwięzłe tworzenie list w jednej linii.
     - Przykład:

```
In [16]: # List Comprehension

         squares = [x**2 for x in range(1, 6)]

In [17]: print(squares)

         [1, 4, 9, 16, 25]
```

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane

- Funkcje matematyczne:
  - abs(), round(), max(), min(), sum()

- Funkcje tekstowe:
  - len(), str(), upper(), lower(), startswith(), endswith()

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane

- Funkcje Statystyczne, **należy zrobić import statistics**:
  - mean(), median(), mode(), stdev()

- Funkcje Data i Czas:
  - datetime.now(), strftime(), strptime()

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane
   - Funkcje Data i Czas:
     - Dlaczego importujemy najpierw z from datetime a później import datetime?

       - Moduł datetime zawiera głównie jedną klasę o nazwie datetime, która jest powszechnie używana.
       - Importując jedynie tę klasę za pomocą from datetime import datetime, można bezpośrednio odwoływać się do niej bez konieczności używania prefiksu nazwy modułu.
       - Użycie datetime.now() jest bardziej zwięzłe niż datetime.datetime.now(), co poprawia czytelność kodu.

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane

W wielu przypadkach importowanie tylko potrzebnej klasy za pomocą from ... import ... jest bardziej praktyczne, ponieważ ogranicza ilość pisania kodu i poprawia czytelność. Jednakże, w niektórych sytuacjach, zwłaszcza gdy moduł datetime zawiera więcej niż jedną interesującą klasę lub funkcję, można zdecydować się na import całego modułu import datetime.

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane

- Funkcje matematyczne:
  - Przykład:

In [18]:
```
# Funkcje matematyczne

a = abs(-7.25) # Zwraca wartość bezwzględną.

b = round(5.76543, 2) # Zaokrągla liczbę do wybranej ilości miejsc po przecinku (nawias)

c = max(2, 3, 4, 5, 10, 15, 20) # Zwraca największą wartość z podanej tupli, listy.

d = min(2, 3, 4, 5, 10, 15, 20) # Zwraca najmniejszą wartość z podanej tupli, listy.

e = (2, 3, 4, 5, 10, 15, 20)

f = sum(e) # Sumuje wszystkie wartości z podanej zmiennej 'e'
```

In [19]:
```
print(f'{a},\n{b},\n{c},\n{d},\n{f}')
```

```
7.25,
5.77,
20,
2,
59
```

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane

- Funkcje tekstowe:
  - Przykład:

```
In [22]: # Funkcje tekstowe

tekst = 'Przykładowy tekst do sprawdzenia.'
int = 4

g = len(tekst) # Zwraca długość znaków w podanym tekście.

h = str(int) # Zmienia wartość liczbową lub teks którego nie jesteśmy pewni na stringa

i = tekst.upper() # Zwraca tekst z powiększonymi znakami.

j = tekst.lower() # Zwraca tekst z pomniejszonymi znakami.

k = tekst.startswith('Przykład') # Zwraca wartość Prawda/Fałsz (Bolean)

l = tekst.startswith('Hello') # Zwraca wartość Prawda/Fałsz (Bolean)

m = tekst.endswith('.') # Zwraca wartość Prawda/Fałsz (Bolean)

n = tekst.endswith('World!') # Zwraca wartość Prawda/Fałsz (Bolean)
```

```
In [24]: print(f'{g},\n{h},\n{i},\n{j},\n{k},\n{l},\n{m},\n{n}')
```

```
33,
4,
PRZYKŁADOWY TEKST DO SPRAWDZENIA.,
przykładowy tekst do sprawdzenia.,
True,
False,
True,
False
```

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane

- Funkcje statystyczne:
  - Przykład:

In [51]:
```
# Funkcje statystyczne

import statistics

numbersLong = [2, 4, 2, 5 ,6 ,7 , 8, 8 ,8 ,8 ,8 , 9, 10, 11, 11, 11, 11, 11, 11, 11, 12, 13, 2, 5, 6, 7]

print(f'''{statistics.mean(numbersLong)},
{statistics.median(numbersLong)},
{statistics.mode(numbersLong)},
{statistics.stdev(numbersLong)}''')

# .mean() - Średnia
# .median() - Mediana
# .mode() - Dominanta wartość najczęściej występująca w zbiorze.
# .stdev() - Odchylenie standardowe
```

```
7.961538461538462,
8.0,
11,
3.23086080456301
```

# Python - Wybrane Elementy

5. Wybrane Funkcje Wbudowane
    - Funkcje data/czas:
        - Przykład:

    (ze względu na ograniczone miejsce, przykłady na kolejnym slajdzie)

# Python - Wybrane Elementy

## Funkcje data/czas: Strftime()

In [54]:
```python
# Funkcje data i czas
# https://docs.python.org/3/library/datetime.html#format-codes

# Różnica pomiędzy strftime() a strptime - z strptime() stringa robimy obiekt, a z strftime() tworzymy stringa

from datetime import datetime

now = datetime.now() # current date and time

year = now.strftime("%Y")
print("year:", year)

month = now.strftime("%m")
print("month:", month)

day = now.strftime("%d")
print("day:", day)

time = now.strftime("%H:%M:%S")
print("time:", time)

date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date and time:", date_time)
print(now)
```

```
year: 2024
month: 01
day: 22
time: 21:06:22
date and time: 01/22/2024, 21:06:22
2024-01-22 21:06:22.394635
```

# Python - Wybrane Elementy

## Funkcje data/czas: Strptime()

```
In [55]: # Funkcje data i czas

         date_string = "21 June, 2018"

         print("date_string =", date_string)
         print("type of date_string =", type(date_string))

         date_object = datetime.strptime(date_string, "%d %B, %Y")

         print("date_object =", date_object)
         print("type of date_object =", type(date_object))
```

```
date_string = 21 June, 2018
type of date_string = <class 'str'>
date_object = 2018-06-21 00:00:00
type of date_object = <class 'datetime.datetime'>
```

# Python - Wybrane Elementy

## Funkcje data/czas: Strptime()

In [56]:
```
# Funkcje data i czas

dt_string = "12/11/2018 09:15:32"

# Considering date is in dd/mm/yyyy format
dt_object1 = datetime.strptime(dt_string, "%d/%m/%Y %H:%M:%S")
print("dt_object1 =", dt_object1)

# Considering date is in mm/dd/yyyy format
dt_object2 = datetime.strptime(dt_string, "%m/%d/%Y %H:%M:%S")
print("dt_object2 =", dt_object2)
```

```
dt_object1 = 2018-11-12 09:15:32
dt_object2 = 2018-12-11 09:15:32
```

# Python - Wybrane Elementy

6. Kontrola Przepływu
    - Pętla While:
        - Przykład:

In [2]:
```
# Pętla While

i = 0
while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

# Python - Wybrane Elementy

6. Kontrola Przepływu
   - Pętla For:

```
In [2]: # Pętla For

        numbers = [1, 2, 3, 4, 5]
        for i in numbers:
            print(i)

        1
        2
        3
        4
        5
```

# Python - Wybrane Elementy

6. Kontrola Przepływu
    - Instrukcje Warunkowe (if, elif, else):
        - Przykład

In [6]:
```
# Instrukcje warunkowe (if, elif, else)

x = 10
if x > 0:
    print("Dodatnie")
elif x < 0:
    print("Ujemne")
else:
    print("Zero")
```

```
Dodatnie
```

# Podstawowe typy i struktura danych

37. Funkcja **enumerate**:

Funkcja **enumerate** jest przydatnym narzędziem w Pythonie, pozwalającym na łatwe iterowanie przez elementy iterowalnego obiektu (np. listy, krotki) jednocześnie śledząc ich indeksy. Funkcja ta zwraca obiekt enumeracyjny, który składa się z pary (indeks, element) dla każdego elementu w oryginalnym obiekcie.

# Podstawowe typy i struktura danych

Przykład:

```
:
# Przykład 1
produkty = ["Laptop", "Smartfon", "Kamera"]

# Użycie enumerate do iteracji przez listę z indeksami
for indeks, produkt in enumerate(produkty):
    print(f"Indeks: {indeks}, Produkt: {produkt}")
# Wynik:
# Indeks: 0, Produkt: Laptop
# Indeks: 1, Produkt: Smartfon
# Indeks: 2, Produkt: Kamera
```

```
Indeks: 0, Produkt: Laptop
Indeks: 1, Produkt: Smartfon
Indeks: 2, Produkt: Kamera
```

# Podstawowe typy i struktura danych

37. Funkcja **enumerate** – dodatkowe wyjaśnienie:

Funkcja **enumerate** jest przydatna zwłaszcza w przypadku, gdy potrzebujemy jednoczesnego dostępu do wartości i ich indeksów podczas iteracji przez obiekty iterowalne.

# Podstawowe typy i struktura danych

Rozwinięcie tematu:

- Funkcja **zip** to wbudowana funkcja, która służy do łączenia dwóch lub więcej iterowalnych obiektów (na przykład list, krotek czy napisów) w jednoiterowalny obiekt. Każdy element wynikowego obiektu jest tuplą, zawierającą elementy na odpowiednich pozycjach z oryginalnych obiektów.

# Podstawowe typy i struktura danych

Rozwinięcie tematu:

- Funkcja **zip** jest bardzo przydatna, gdy chcemy jednocześnie przeglądać elementy kilku iterowalnych obiektów. Pozwala to na eleganckie i zwięzłe operacje na danych, takie jak tworzenie słowników czy par krotek.

# Podstawowe typy i struktura danych

Przykład:

In [7]:
```
# Ćwiczenie 32
imiona = ["Anna", "Jan", "Ewa"]
wieki = [25, 30, 28]

# Użyj funkcji zip do łączenia dwóch list
lista_krotek = list(zip(imiona, wieki))

print("Lista imion:", imiona)
print("Lista wieków:", wieki)
print("Lista krotek (imię, wiek):", lista_krotek)
```

```
Lista imion: ['Anna', 'Jan', 'Ewa']
Lista wieków: [25, 30, 28]
Lista krotek (imię, wiek): [('Anna', 25), ('Jan', 30), ('Ewa', 28)]
```

# Podstawowe typy i struktura danych

1. Liczby całkowite (int) i liczby zmiennoprzecinkowe (float)
    - Ćwiczenie:

Stwórz dwie zmienne, jedną przechowującą liczbę całkowitą, a drugą liczbę zmiennoprzecinkową. Następnie wykonaj operacje matematyczne (dodawanie, odejmowanie, mnożenie, dzielenie) na tych zmiennych.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [1]:
```
# Ćwiczenie 1
liczba_calkowita = 5
liczba_zmiennoprzecinkowa = 3.14

# Operacje matematyczne
suma = liczba_calkowita + liczba_zmiennoprzecinkowa
roznica = liczba_calkowita - liczba_zmiennoprzecinkowa
iloczyn = liczba_calkowita * liczba_zmiennoprzecinkowa
iloraz = liczba_calkowita / liczba_zmiennoprzecinkowa

print("Suma:", suma)
print("Różnica:", roznica)
print("Iloczyn:", iloczyn)
print("Iloraz:", iloraz)
```

```
Suma: 8.14
Różnica: 1.8599999999999999
Iloczyn: 15.700000000000001
Iloraz: 1.592356687898089
```

# Podstawowe typy i struktura danych

2. Napisy (str)
    - Ćwiczenie:

Stwórz zmienną przechowującą napis "Hello, Python!". Następnie użyj różnych operacji na napisach, takich jak indeksowanie, wycinanie fragmentów (slicing) i łączenie napisów.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [5]:
```
# Ćwiczenie 2
napis = "Hello, Python!"

# Operacje na napisach
pierwszy_znak = napis[0]
fragment = napis[7:13]
polaczony_napis = napis + " Welcome!"

print("Pierwszy znak:", pierwszy_znak)
print("Fragment:", fragment)
print("Połączony napis:", polaczony_napis)
```

```
Pierwszy znak: H
Fragment: Python
Połączony napis: Hello, Python! Welcome!
```

# Podstawowe typy i struktura danych

3. Listy

- Ćwiczenie:

Stwórz listę zawierającą kilka różnych typów danych (liczby, napisy). Następnie wykonaj operacje takie jak dodawanie elementów, usuwanie elementów i dostęp do elementów za pomocą indeksów.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [6]:
```
# Ćwiczenie 3
lista = [1, "Python", 3.14, "Data Science"]

# Operacje na listach
lista.append("New Element")
lista.remove("Python")
element = lista[2]

print("Lista po dodaniu:", lista)
print("Element o indeksie 2:", element)
```

```
Lista po dodaniu: [1, 3.14, 'Data Science', 'New Element']
Element o indeksie 2: Data Science
```

# Podstawowe typy i struktura danych

4. Krotki (tuple)
   - Ćwiczenie:

Stwórz krotkę zawierającą co najmniej trzy elementy. Następnie spróbuj zmienić wartość jednego z elementów (będzie to niemożliwe z powodu niemodyfikowalności krotek).

# Podstawowe typy i struktura danych

Rozwiązanie:

In [7]:
```
# Ćwiczenie 4
krotka = (10, "Java", 2.71)

# Próba zmiany wartości (spowoduje błąd)
try:
    krotka[0] = 20
except TypeError as e:
    print("Błąd:", e)
```

```
Błąd: 'tuple' object does not support item assignment
```

# Podstawowe typy i struktura danych

5. Słowniki (dict)
   - Ćwiczenie:

Stwórz słownik reprezentujący informacje o studencie (imię, wiek, ocena). Następnie dodaj nowy klucz i zmień wartość jednego z kluczy.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [8]:
```
# Ćwiczenie 5
student = {'imie': 'Anna', 'wiek': 22, 'ocena': 'A'}

# Dodanie nowego klucza
student['kierunek'] = 'Informatyka'

# Zmiana wartości klucza 'wiek'
student['wiek'] = 23

print("Słownik po zmianach:", student)
```

```
Słownik po zmianach: {'imie': 'Anna', 'wiek': 23, 'ocena': 'A', 'kierunek': 'Informatyka'}
```

# Podstawowe typy i struktura danych

6. Praca z List Comprehension

- Ćwiczenie:

Stwórz listę liczb od 1 do 10. Następnie używając list comprehension, utwórz nową listę zawierającą kwadraty liczb.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [1]:
```
# Ćwiczenie 6
liczby = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Utwórz listę kwadratów używając List Comprehension
kwadraty = [x**2 for x in liczby]

print("Oryginalne liczby:", liczby)
print("Kwadraty:", kwadraty)
```

```
Oryginalne liczby: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Kwadraty: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

# Podstawowe typy i struktura danych

7. Funkcje Matematyczne i Tekstowe
    - Ćwiczenie:

Stwórz funkcję, która przyjmuje dwie liczby i zwraca ich sumę. Następnie użyj tej funkcji do dodawania dwóch liczb.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [2]:
```
# Ćwiczenie 7
def dodaj_liczby(a, b):
    return a + b

# Użycie funkcji do dodawania dwóch liczb
wynik_dodawania = dodaj_liczby(8, 4)

print("Wynik dodawania:", wynik_dodawania)
```

```
Wynik dodawania: 12
```

# Podstawowe typy i struktura danych

8. Iteracja i warunki
    - Ćwiczenie:

Stwórz listę liczb od 1 do 10. Następnie użyj pętli for do iteracji przez listę i wyświetlenia tylko parzystych liczb.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [3]:
```
# Ćwiczenie 8
liczby = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Iteracja przez listę i wyświetlenie parzystych liczb
print("Parzyste liczby:")
for liczba in liczby:
    if liczba % 2 == 0:
        print(liczba)
```

```
Parzyste liczby:
2
4
6
8
10
```

# Podstawowe typy i struktura danych

9. Pętla While
    - Ćwiczenie:

Używając pętli while, stwórz program, który dodaje kolejne liczby całkowite, aż suma przekroczy 100.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [7]:
```
# Ćwiczenie 9
suma = 0
licznik = 1

while suma <= 100:
    suma += licznik
    licznik += 1

print("Suma przekroczyła 100 po dodaniu", licznik-1, "liczb.")
```

```
Suma przekroczyła 100 po dodaniu 14 liczb.
```

# Podstawowe typy i struktura danych

10. Slicing napisów

- Ćwiczenie:

Stwórz napis zawierający dowolne zdanie. Następnie użyj slicingu, aby wyświetlić co trzeci znak w napisie.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [8]:
```
# Ćwiczenie 10
zdanie = "To jest przykładowe zdanie do ćwiczenia slicingu"

# Wyświetl co trzeci znak w napisie
co_trzeci_znak = zdanie[2::3]

print("Oryginalne zdanie:", zdanie)
print("Co trzeci znak:", co_trzeci_znak)
```

```
Oryginalne zdanie: To jest przykładowe zdanie do ćwiczenia slicingu
Co trzeci znak:  spyawzn  iealiu
```

# Podstawowe typy i struktura danych

11. Funkcje Anonimowe (Lambda) i List Comprehension
    - Ćwiczenie:

Utwórz funkcję anonimową (lambda), która podnosi liczbę do kwadratu. Następnie użyj List Comprehension, aby utworzyć listę kwadratów liczb od 1 do 5.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [9]:
```
# Ćwiczenie 11
kwadrat = lambda x: x**2

# Użyj List Comprehension do utworzenia listy kwadratów
kwadraty_list = [kwadrat(x) for x in range(1, 6)]

print("Kwadraty liczb od 1 do 5:", kwadraty_list)
```

```
Kwadraty liczb od 1 do 5: [1, 4, 9, 16, 25]
```

# Podstawowe typy i struktura danych

12. Instrukcje Warunkowe i Funkcje Tekstowe

- Ćwiczenie:

Stwórz funkcję, która przyjmuje napis i sprawdza, czy zaczyna się od litery "A". Jeśli tak, wyświetl komunikat "Zaczyna się od A", w przeciwnym razie "Nie zaczyna się od A".

# Podstawowe typy i struktura danych

Rozwiązanie:

In [10]:
```python
# Ćwiczenie 12
def sprawdzaczaczynasieodA(napis):
    if napis.startswith('A'):
        return "Zaczyna się od A"
    else:
        return "Nie zaczyna się od A"

# Przykłady użycia funkcji
print(sprawdzaczaczynasieodA("Python"))
print(sprawdzaczaczynasieodA("Anakonda"))
```

```
Nie zaczyna się od A
Zaczyna się od A
```

# Podstawowe typy i struktura danych

13. Iteracja i Operacje na Liście

- Ćwiczenie:

Stwórz listę zawierającą kilka słów. Następnie użyj pętli for do wyświetlenia długości każdego słowa.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [11]:
```
# Ćwiczenie 13
slowa = ["Python", "Data", "Analysis", "Machine", "Learning"]

# Użyj pętli for do wyświetlenia długości każdego słowa
for slowo in slowa:
    print(f"Długość słowa {slowo}: {len(slowo)}")
```

```
Długość słowa Python: 6
Długość słowa Data: 4
Długość słowa Analysis: 8
Długość słowa Machine: 7
Długość słowa Learning: 8
```

# Podstawowe typy i struktura danych

14. Praca z datą i czasem

- Ćwiczenie:

Zaimportuj moduł **datetime** i użyj go do uzyskania aktualnej daty i godziny. Następnie przekształć datę do postaci stringa i wyświetl.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [12]:
```
# Ćwiczenie 14
from datetime import datetime

# Uzyskaj aktualną datę i godzinę
aktualna_data_i_czas = datetime.now()

# Przekształć datę do postaci napisu i wyświetl
data_napis = aktualna_data_i_czas.strftime("%Y-%m-%d %H:%M:%S")

print("Aktualna data i czas:", data_napis)
```

```
Aktualna data i czas: 2024-01-25 21:54:18
```

# Podstawowe typy i struktura danych

15. Pętla While i Praca z Listami

- Ćwiczenie:

Użyj pętli while, aby utworzyć listę kolejnych potęg liczby 2, dopóki potęga nie przekroczy 1000.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [13]:
```
# Ćwiczenie 15
potega = 1
potegi_dwojki = []

while potega <= 1000:
    potegi_dwojki.append(potega)
    potega *= 2

print("Potęgi liczby 2 do 1000:", potegi_dwojki)
```

```
Potęgi liczby 2 do 1000: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

# Podstawowe typy i struktura danych

16. Operacje na napisach

- Ćwiczenie:

Stwórz napis zawierający co najmniej dwie różne litery. Następnie użyj różnych operacji na napisach, takich jak zamiana na małe/duże litery, odwracanie napisu.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [14]:
```
# Ćwiczenie 16
napis = "Python"

# Różne operacje na napisach
napis_male_litery = napis.lower()
napis_duze_litery = napis.upper()
odwrocony_napis = napis[::-1]

print("Oryginalny napis:", napis)
print("Napis z małymi literami:", napis_male_litery)
print("Napis z dużymi literami:", napis_duze_litery)
print("Odwrocony napis:", odwrocony_napis)
```

```
Oryginalny napis: Python
Napis z małymi literami: python
Napis z dużymi literami: PYTHON
Odwrocony napis: nohtyP
```

# Podstawowe typy i struktura danych

17. Wybór Elementów z Listy

- Ćwiczenie:

Stwórz listę zawierającą co najmniej 5 elementów. Następnie użyj warunku, aby utworzyć nową listę zawierającą tylko elementy spełniające określone kryterium.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [15]:
```
# Ćwiczenie 17
liczby = [10, 20, 30, 40, 50, 60]

# Utwórz nową listę zawierającą liczby większe niż 30
nowa_lista = [x for x in liczby if x > 30]

print("Oryginalna lista:", liczby)
print("Nowa lista (liczby większe niż 30):", nowa_lista)
```

```
Oryginalna lista: [10, 20, 30, 40, 50, 60]
Nowa lista (liczby większe niż 30): [40, 50, 60]
```

# Podstawowe typy i struktura danych

18. Praca z listami i słownikami

- Ćwiczenie:

Stwórz listę zawierającą słowniki reprezentujące różne osoby (imię, wiek, zawód). Następnie używając pętli for, utwórz nową listę zawierającą tylko imiona tych osób.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [16]:
```
# Ćwiczenie 18
osoby = [
    {'imie': 'Anna', 'wiek': 25, 'zawod': 'Programista'},
    {'imie': 'Jan', 'wiek': 30, 'zawod': 'Analityk'},
    {'imie': 'Ewa', 'wiek': 28, 'zawod': 'Projektant'}
]

# Utwórz listę zawierającą imiona osób
imiona_osob = [osoba['imie'] for osoba in osoby]

print("Lista osób:", osoby)
print("Imiona osób:", imiona_osob)
```

```
Lista osób: [{'imie': 'Anna', 'wiek': 25, 'zawod': 'Programista'}, {'imie': 'Jan', 'wiek': 30, 'zawod': 'Analityk'}, {'imie':
'Ewa', 'wiek': 28, 'zawod': 'Projektant'}]
Imiona osób: ['Anna', 'Jan', 'Ewa']
```

# Podstawowe typy i struktura danych

19. Funkcje Anonimowe (Lambda) i Sortowanie

- Ćwiczenie:

Stwórz listę liczb całkowitych. Następnie posortuj tę listę rosnąco za pomocą funkcji **sorted()** i funkcji anonimowej (lambda).

# Podstawowe typy i struktura danych

Rozwiązanie:

In [17]:
```
# Ćwiczenie 19
liczby = [4, 2, 7, 1, 9, 5]

# Posortuj listę rosnąco za pomocą funkcji sorted i funkcji anonimowej (lambda)
posortowane_liczby = sorted(liczby, key=lambda x: x)

print("Oryginalna lista:", liczby)
print("Posortowana lista:", posortowane_liczby)
```

```
Oryginalna lista: [4, 2, 7, 1, 9, 5]
Posortowana lista: [1, 2, 4, 5, 7, 9]
```

# Podstawowe typy i struktura danych

20. Praca z Tuple i Slicing

- Ćwiczenie:

Stwórz krotkę zawierającą co najmniej 5 elementów. Następnie użyj slicingu, aby utworzyć nową krotkę zawierającą tylko pewien fragment oryginalnej krotki.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [18]:
```
# Ćwiczenie 20
krotka = (10, 20, 30, 40, 50, 60)

# Użyj slicingu, aby utworzyć nową krotkę
nowa_krotka = krotka[1:4]

print("Oryginalna krotka:", krotka)
print("Nowa krotka:", nowa_krotka)
```

```
Oryginalna krotka: (10, 20, 30, 40, 50, 60)
Nowa krotka: (20, 30, 40)
```

# Podstawowe typy i struktura danych

21. Grupowanie i Przetwarzanie Listy Słowników

- Ćwiczenie:

Stwórz listę słowników, gdzie każdy słownik reprezentuje informacje o jednym produkcie (nazwa, cena, ilość). Następnie użyj list comprehension, aby utworzyć nową listę zawierającą tylko produkty, których cena jest mniejsza niż 100.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [19]:
```python
# Ćwiczenie 21
produkty = [
    {'nazwa': 'Laptop', 'cena': 1500, 'ilosc': 5},
    {'nazwa': 'Smartfon', 'cena': 800, 'ilosc': 10},
    {'nazwa': 'Kamera', 'cena': 120, 'ilosc': 3},
    {'nazwa': 'Słuchawki', 'cena': 50, 'ilosc': 8}
]

# Utwórz nową listę zawierającą produkty o cenie mniejszej niż 100
tanie_produkty = [produkt for produkt in produkty if produkt['cena'] < 100]

print("Lista produktów:", produkty)
print("Tanie produkty:", tanie_produkty)
```

```
Lista produktów: [{'nazwa': 'Laptop', 'cena': 1500, 'ilosc': 5}, {'nazwa': 'Smartfon', 'cena': 800, 'ilosc': 10}, {'nazwa': 'Ka
mera', 'cena': 120, 'ilosc': 3}, {'nazwa': 'Słuchawki', 'cena': 50, 'ilosc': 8}]
Tanie produkty: [{'nazwa': 'Słuchawki', 'cena': 50, 'ilosc': 8}]
```

# Podstawowe typy i struktura danych

22. Łączenie List i Słowników

- Ćwiczenie:

Stwórz dwie listy zawierające nazwy produktów i ich ceny. Następnie użyj funkcji zip, aby połączyć listy w listę krotek. Na koniec stwórz słownik, gdzie nazwy produktów są kluczami, a ceny są wartościami.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [20]:
```
# Ćwiczenie 22
nazwy_produktow = ['Laptop', 'Smartfon', 'Kamera', 'Słuchawki']
ceny_produktow = [1500, 800, 120, 50]

# Połącz listy w listę krotek za pomocą funkcji zip
produkty_i_ceny = list(zip(nazwy_produktow, ceny_produktow))

# Utwórz słownik z produktami i ich cenami
słownik_cen = dict(produkty_i_ceny)

print("Nazwy produktów:", nazwy_produktow)
print("Ceny produktów:", ceny_produktow)
print("Słownik cen:", słownik_cen)
```

```
Nazwy produktów: ['Laptop', 'Smartfon', 'Kamera', 'Słuchawki']
Ceny produktów: [1500, 800, 120, 50]
Słownik cen: {'Laptop': 1500, 'Smartfon': 800, 'Kamera': 120, 'Słuchawki': 50}
```

# Podstawowe typy i struktura danych

23. Generatory i Iteratory

- Ćwiczenie:

Stwórz generator liczb parzystych od 0 do 10. Następnie użyj pętli for do iteracji przez te liczby.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [21]:
```
# Ćwiczenie 23
generator_parzystych = (x for x in range(11) if x % 2 == 0)

# Iteruj przez liczby parzyste
print("Liczby parzyste:")
for liczba in generator_parzystych:
    print(liczba)
```

```
Liczby parzyste:
0
2
4
6
8
10
```

# Podstawowe typy i struktura danych

24. Praca z Datą i Czasem (Część 2)

- Ćwiczenie:

Stwórz funkcję, która przyjmuje dwie daty i zwraca różnicę między nimi w formie obiektu **timedelta**. Następnie użyj funkcji do obliczenia różnicy między dwiema datami.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [22]:
```python
# Ćwiczenie 24
from datetime import datetime

def roznica_dat(data1, data2):
    delta = data2 - data1
    return delta

# Przykłady użycia funkcji
data_poczatkowa = datetime(2022, 1, 1)
data_koncowa = datetime(2022, 12, 31)

roznica = roznica_dat(data_poczatkowa, data_koncowa)

print("Data początkowa:", data_poczatkowa)
print("Data końcowa:", data_koncowa)
print("Różnica między datami:", roznica)
```

```
Data początkowa: 2022-01-01 00:00:00
Data końcowa: 2022-12-31 00:00:00
Różnica między datami: 364 days, 0:00:00
```

# Podstawowe typy i struktura danych

25. Praca z Modułem **math**

- Ćwiczenie:

Stwórz funkcję, która przyjmuje liczbę i zwraca jej pierwiastek kwadratowy za pomocą modułu **math**. Następnie użyj funkcji do obliczenia pierwiastka kwadratowego z kilku liczb.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [23]:
```
# Ćwiczenie 25
import math

def pierwiastek_kwadratowy(liczba):
    return math.sqrt(liczba)

# Przykłady użycia funkcji
liczba1 = 16
liczba2 = 25
liczba3 = 9

pierwiastek1 = pierwiastek_kwadratowy(liczba1)
pierwiastek2 = pierwiastek_kwadratowy(liczba2)
pierwiastek3 = pierwiastek_kwadratowy(liczba3)

print("Pierwiastek kwadratowy z", liczba1, "to", pierwiastek1)
print("Pierwiastek kwadratowy z", liczba2, "to", pierwiastek2)
print("Pierwiastek kwadratowy z", liczba3, "to", pierwiastek3)
```

```
Pierwiastek kwadratowy z 16 to 4.0
Pierwiastek kwadratowy z 25 to 5.0
Pierwiastek kwadratowy z 9 to 3.0
```

# Podstawowe typy i struktura danych

26. Przetwarzanie Tekstu za pomocą List Comprehension

- Ćwiczenie:

Stwórz listę słów, a następnie użyj list comprehension do utworzenia nowej listy, która zawiera długości każdego słowa.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [1]:
```
# Ćwiczenie 26
lista_slow = ["Python", "Programowanie", "Analiza", "Dane", "Szkolenie"]

# Użyj list comprehension do utworzenia listy długości słów
dlugosci_slow = [len(slowo) for slowo in lista_slow]

print("Lista słów:", lista_slow)
print("Długości słów:", dlugosci_slow)
```

```
Lista słów: ['Python', 'Programowanie', 'Analiza', 'Dane', 'Szkolenie']
Długości słów: [6, 13, 7, 4, 9]
```

# Podstawowe typy i struktura danych

27. Wykorzystanie Funkcji Lambda do Sortowania

- Ćwiczenie:

Stwórz listę słów, a następnie użyj list comprehension do utworzenia nowej listy, która zawiera długości każdego słowa.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [2]:
```
# Ćwiczenie 27
lista_liczb = [8, 3, 5, 1, 9, 2, 7]

# Użyj funkcji sorted z funkcją lambda do sortowania rosnącego
posortowane_liczby = sorted(lista_liczb, key=lambda x: x)

print("Lista liczb:", lista_liczb)
print("Posortowane liczby:", posortowane_liczby)
```

```
Lista liczb: [8, 3, 5, 1, 9, 2, 7]
Posortowane liczby: [1, 2, 3, 5, 7, 8, 9]
```

# Podstawowe typy i struktura danych

28. Praca z Pętlą While i Wyjątkami

- Ćwiczenie:

Stwórz pętlę while, która prosi użytkownika o wprowadzenie liczby całkowitej. Obsłuż wyjątek, jeśli użytkownik wprowadzi coś innego niż liczba całkowita.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [3]:
```
# Ćwiczenie 28
while True:
    try:
        liczba = int(input("Wprowadź liczbę całkowitą: "))
        break
    except ValueError:
        print("Błąd! Wprowadź poprawną liczbę całkowitą.")

print("Wprowadzona liczba:", liczba)
```

```
Wprowadź liczbę całkowitą: 5
Wprowadzona liczba: 5
```

# Podstawowe typy i struktura danych

29. Operacje na Napisach

- Ćwiczenie:

Stwórz napis zawierający zdanie. Następnie użyj różnych operacji na napisie, takich jak odwracanie, zamiana liter na wielkie, podział na słowa.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [4]:
```
# Ćwiczenie 29
zdanie = "Python jest potężnym językiem programowania."

# Odwróć napis
odwrocone_zdanie = zdanie[::-1]

# Zamień na wielkie litery
wielkie_litery = zdanie.upper()

# Podziel na słowa
slowa = zdanie.split()

print("Oryginalne zdanie:", zdanie)
print("Odwrócone zdanie:", odwrocone_zdanie)
print("Wielkie litery:", wielkie_litery)
print("Słowa:", slowa)
```

```
Oryginalne zdanie: Python jest potężnym językiem programowania.
Odwrócone zdanie: .ainawomargorp meikyzęj mynżętop tsej nohtyP
Wielkie litery: PYTHON JEST POTĘŻNYM JĘZYKIEM PROGRAMOWANIA.
Słowa: ['Python', 'jest', 'potężnym', 'językiem', 'programowania.']
```

# Podstawowe typy i struktura danych

30. Użycie Modułu random do Losowania Liczb

- Ćwiczenie:

Stwórz funkcję, która losuje liczbę całkowitą z zakresu od 1 do 10. Następnie zapytaj użytkownika o odgadnięcie wylosowanej liczby.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [5]:
```python
# Ćwiczenie 30
import random

def losuj_liczbe():
    return random.randint(1, 10)

wylosowana_liczba = losuj_liczbe()

while True:
    odgadnij = int(input("Odgadnij liczbę od 1 do 10: "))
    if odgadnij == wylosowana_liczba:
        print("Gratulacje! Odgadłeś właściwą liczbę.")
        break
    else:
        print("Nieprawidłowa liczba. Spróbuj ponownie.")
```

# Podstawowe typy i struktura danych

31. Wykorzystanie List Comprehension do Filtrowania
    - Ćwiczenie:

Stwórz listę liczb całkowitych. Następnie użyj list comprehension, aby utworzyć nową listę, która zawiera tylko parzyste liczby.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [6]:
```
# Ćwiczenie 31
liczby_calkowite = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Użyj list comprehension do utworzenia listy parzystych liczb
parzyste_liczby = [liczba for liczba in liczby_calkowite if liczba % 2 == 0]

print("Lista liczb całkowitych:", liczby_calkowite)
print("Parzyste liczby:", parzyste_liczby)
```

```
Lista liczb całkowitych: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Parzyste liczby: [2, 4, 6, 8, 10]
```

# Podstawowe typy i struktura danych

32. Wykorzystanie Funkcji **zip** do Łączenia Dwóch List

- Ćwiczenie:

Stwórz dwie listy o równej długości, reprezentujące imiona i wieki osób. Następnie użyj funkcji **zip**, aby utworzyć listę krotek (imię, wiek).

# Podstawowe typy i struktura danych

Rozwinięcie tematu:

- Funkcja **zip** to wbudowana funkcja, która służy do łączenia dwóch lub więcej iterowalnych obiektów (na przykład list, krotek czy napisów) w jednoiterowalny obiekt. Każdy element wynikowego obiektu jest tuplą, zawierającą elementy na odpowiednich pozycjach z oryginalnych obiektów.

# Podstawowe typy i struktura danych

Rozwinięcie tematu:

- Funkcja **zip** jest bardzo przydatna, gdy chcemy jednocześnie przeglądać elementy kilku iterowalnych obiektów. Pozwala to na eleganckie i zwięzłe operacje na danych, takie jak tworzenie słowników czy par krotek.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [7]:
```
# Ćwiczenie 32
imiona = ["Anna", "Jan", "Ewa"]
wieki = [25, 30, 28]

# Użyj funkcji zip do łączenia dwóch list
lista_krotek = list(zip(imiona, wieki))

print("Lista imion:", imiona)
print("Lista wieków:", wieki)
print("Lista krotek (imię, wiek):", lista_krotek)
```

```
Lista imion: ['Anna', 'Jan', 'Ewa']
Lista wieków: [25, 30, 28]
Lista krotek (imię, wiek): [('Anna', 25), ('Jan', 30), ('Ewa', 28)]
```

# Podstawowe typy i struktura danych

33. Użycie Funkcji **map** do Przekształcania Elementów Listy

- Ćwiczenie:

Stwórz listę liczb całkowitych. Następnie użyj funkcji **map**, aby podnieść każdą liczbę do kwadratu.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [8]:
```
# Ćwiczenie 33
liczby_calkowite = [2, 4, 6, 8, 10]

# Użyj funkcji map do podniesienia każdej liczby do kwadratu
kwadraty = list(map(lambda x: x**2, liczby_calkowite))

print("Lista liczb całkowitych:", liczby_calkowite)
print("Kwadraty liczb:", kwadraty)
```

```
Lista liczb całkowitych: [2, 4, 6, 8, 10]
Kwadraty liczb: [4, 16, 36, 64, 100]
```

# Podstawowe typy i struktura danych

34. Praca z Datą i Czasem

- Ćwiczenie:

Stwórz funkcję, która przyjmuje aktualną datę i czas, a następnie dodaje do niej 7 dni. Wykorzystaj moduł **datetime**.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [9]:
```
# Ćwiczenie 34
from datetime import datetime, timedelta

def dodaj_7_dni():
    obecna_data_czas = datetime.now()
    nowa_data_czas = obecna_data_czas + timedelta(days=7)
    return nowa_data_czas

wynik = dodaj_7_dni()

print("Obecna data i czas:", datetime.now())
print("Data i czas po dodaniu 7 dni:", wynik)
```

```
Obecna data i czas: 2024-01-26 21:55:47.412698
Data i czas po dodaniu 7 dni: 2024-02-02 21:55:47.412177
```

# Podstawowe typy i struktura danych

35. Praca z Modułem **math** - Obliczanie Pierwiastka Kwadratowego

- Ćwiczenie:

Stwórz funkcję, która przyjmuje liczbę i używa modułu **math**, aby obliczyć jej pierwiastek kwadratowy.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [10]:
```python
# Ćwiczenie 35
import math

def pierwiastek_kwadratowy(liczba):
    return math.sqrt(liczba)

liczba_testowa = 16
wynik = pierwiastek_kwadratowy(liczba_testowa)

print(f"Pierwiastek kwadratowy z {liczba_testowa}:", wynik)
```

```
Pierwiastek kwadratowy z 16: 4.0
```

# Podstawowe typy i struktura danych

36. Funkcja **zip**:

- Ćwiczenie – Tworzenie słownika z dwóch list:

Stwórz dwie listy: jedną zawierającą nazwy przedmiotów, a drugą zawierającą odpowiadające im oceny. Następnie użyj funkcji zip, aby połączyć obie listy i utworzyć słownik, gdzie nazwy przedmiotów są kluczami, a oceny są wartościami.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [4]:
```
# Ćwiczenie
przedmioty = ["Matematyka", "Fizyka", "Chemia", "Informatyka"]
oceny = [4.5, 3.8, 5.0, 4.2]

# Użyj funkcji zip do połączenia list w słownik
słownik_ocen = dict(zip(przedmioty, oceny))

# Wyświetl utworzony słownik
print("Słownik ocen:", słownik_ocen)
```

```
Słownik ocen: {'Matematyka': 4.5, 'Fizyka': 3.8, 'Chemia': 5.0, 'Informatyka': 4.2}
```

# Podstawowe typy i struktura danych

37. Funkcja **enumerate**:

Funkcja **enumerate** jest przydatnym narzędziem w Pythonie, pozwalającym na łatwe iterowanie przez elementy iterowalnego obiektu (np. listy, krotki) jednocześnie śledząc ich indeksy. Funkcja ta zwraca obiekt enumeracyjny, który składa się z pary (indeks, element) dla każdego elementu w oryginalnym obiekcie.

# Podstawowe typy i struktura danych

Przykład:

```
:
# Przykład 1
produkty = ["Laptop", "Smartfon", "Kamera"]

# Użycie enumerate do iteracji przez listę z indeksami
for indeks, produkt in enumerate(produkty):
    print(f"Indeks: {indeks}, Produkt: {produkt}")
# Wynik:
# Indeks: 0, Produkt: Laptop
# Indeks: 1, Produkt: Smartfon
# Indeks: 2, Produkt: Kamera
```

```
Indeks: 0, Produkt: Laptop
Indeks: 1, Produkt: Smartfon
Indeks: 2, Produkt: Kamera
```

# Podstawowe typy i struktura danych

37. Funkcja **enumerate** – dodatkowe wyjaśnienie:

Funkcja **enumerate** jest przydatna zwłaszcza w przypadku, gdy potrzebujemy jednoczesnego dostępu do wartości i ich indeksów podczas iteracji przez obiekty iterowalne.

# Podstawowe typy i struktura danych

Przykład:

In [3]:
```
# Przykład 3
slowa = ["jabłko", "banan", "gruszka"]

# Użycie enumerate do wygenerowania słownika z indeksami
slownik_indeksow = {indeks: slowo for indeks, slowo in enumerate(slowa)}

print("Słownik indeksów:", slownik_indeksow)
# Wynik: {0: 'jabłko', 1: 'banan', 2: 'gruszka'}
```

```
Słownik indeksów: {0: 'jabłko', 1: 'banan', 2: 'gruszka'}
```

# Podstawowe typy i struktura danych

37. Funkcja **enumerate** – dodatkowe ćwiczenie:

- Ćwiczenie – Numerowanie elementów listy:

Stwórz listę zawierającą imiona osób. Następnie użyj funkcji enumerate, aby wygenerować słownik, gdzie kluczami są indeksy, a wartościami są odpowiadające imiona.

# Podstawowe typy i struktura danych

Rozwiązanie:

In [4]:
```
# Ćwiczenie
imiona_osob = ["Anna", "Jan", "Ewa", "Tomasz", "Magda"]

# Użyj funkcji enumerate do wygenerowania słownika z indeksami
slownik_indeksow = {indeks: imie for indeks, imie in enumerate(imiona_osob)}

# Wyświetl utworzony słownik
print("Słownik indeksów:", slownik_indeksow)
```

```
Słownik indeksów: {0: 'Anna', 1: 'Jan', 2: 'Ewa', 3: 'Tomasz', 4: 'Magda'}
```

# Podstawowe typy i struktura danych

37. Funkcja **enumerate** – dodatkowe wyjaśnienie do ćwiczenia:

- W tym ćwiczeniu **enumerate** jest używane do stworzenia słownika, w którym kluczami są indeksy elementów z listy imiona_osob, a wartościami są odpowiadające imiona. Po wykonaniu tego kodu, slownik_indeksow powinien zawierać coś podobnego do {0: 'Anna', 1: 'Jan', 2: 'Ewa', 3: 'Tomasz', 4: 'Magda'}.

# Dzień 2

# Analiza z NumPy i Pandas

Analiza z NumPy

1. Tablice Jedno i Dwuwymiarowe w NumPy oraz Podstawowe Operacje

Omówienie NumPy:

- NumPy (Numerical Python) to biblioteka w języku Python dedykowana do pracy z operacjami numerycznymi. Jednym z kluczowych elementów NumPy są tablice, które umożliwiają efektywne wykonywanie operacji na danych numerycznych.

# Analiza z NumPy i Pandas

Przykład:

In [1]:
```
# Tworzenie Tablic w NumPy:
import numpy as np

# Jednowymiarowa tablica
tablica_1d = np.array([1, 2, 3, 4, 5])

# Dwuwymiarowa tablica
tablica_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

In [2]:
```
print(tablica_1d)
```

```
[1 2 3 4 5]
```

In [3]:
```
print(tablica_2d)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

# Analiza z NumPy i Pandas

Analiza z NumPy

- Podstawowe Operacje na Tablicach NumPy:
  - Podstawowe operacje matematyczne

# Analiza z NumPy i Pandas

Przykład:

In [4]:
```
# Podstawowe Operacje Matematyczne:
# Dodawanie
wynik_dodawania = tablica_1d + 10

# Mnożenie
wynik_mnożenia = tablica_2d * 2
```

In [7]:
```
print(wynik_dodawania)
```

```
[11 12 13 14 15]
```

In [8]:
```
print(wynik_mnożenia)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

# Analiza z NumPy i Pandas

Analiza z NumPy

- Podstawowe Operacje na Tablicach NumPy:
  - Indeksowanie i Wycinanie:

# Analiza z NumPy i Pandas

Przykład:

In [9]:
```
# Indeksowanie i Wycinanie:
# Indeksowanie
element = tablica_1d[2]  # Pobiera trzeci element tablicy

# Wycinanie
fragment_tablicy = tablica_2d[:, 1:3]  # Pobiera drugą i trzecią kolumnę
```

In [10]:
```
print(element)
print(fragment_tablicy)
```

```
3
[[2 3]
 [5 6]
 [8 9]]
```

# Analiza z NumPy i Pandas

Analiza z NumPy

- Podstawowe Operacje na Tablicach NumPy:
  - Operacje Statystyczne:

# Analiza z NumPy i Pandas

Przykład:

In [11]:
```
# Operacje statystyczne
# Suma elementów
suma = np.sum(tablica_2d)

# Średnia
srednia = np.mean(tablica_1d)

# Maksimum i minimum
maksimum = np.max(tablica_2d)
minimum = np.min(tablica_1d)
```

In [12]:
```
print(suma)
print(srednia)
print(maksimum)
print(minimum)
```

```
45
3.0
9
1
```

# Analiza z NumPy i Pandas

Analiza z NumPy

- Podstawowe Operacje na Tablicach NumPy:
  - Operacje na Wymiarach:

# Analiza z NumPy i Pandas

Przykład:

In [13]:
```
# Operacje na wymiarach
# Transpozycja
transponowana_tablica = np.transpose(tablica_2d)

# Wyznacznik
wyznacznik = np.linalg.det(tablica_2d)
```

In [17]:
```
print(tablica_2d)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [15]:
```
print(transponowana_tablica)
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

In [16]:
```
print(wyznacznik)
```

```
6.66133814775094e-16
```

# Analiza z NumPy i Pandas

Analiza z NumPy

- Podstawowe Operacje na Tablicach NumPy:
  - Przykładowe ćwiczenie:

# Analiza z NumPy i Pandas

Ćwiczenie 1.

```
import numpy as np

# Ćwiczenie: Stworzenie dwuwymiarowej tablicy i wykonanie operacji
tablica_2d_cw = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Zadanie: Pomnóż każdy element tablicy przez 2
wynik_mnozenia_cw = # Tu dodaj kod

# Zadanie: Oblicz średnią dla każdej kolumny
srednia_kolumn_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Tablica po mnożeniu:", wynik_mnozenia_cw)
print("Średnie kolumn:", srednia_kolumn_cw)
```

# Analiza z NumPy i Pandas

Rozwiązanie:

In [2]:
```
import numpy as np

# Stworzenie dwuwymiarowej tablicy
tablica_2d_cw = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Zadanie: Pomnóż każdy element tablicy przez 2
wynik_mnozenia_cw = tablica_2d_cw * 2

# Zadanie: Oblicz średnią dla każdej kolumny
srednia_kolumn_cw = np.mean(tablica_2d_cw, axis=0)

# Wyświetlenie wyników
print("Tablica po mnożeniu przez 2:")
print(wynik_mnozenia_cw)

print("\nŚrednie kolumn:")
print(srednia_kolumn_cw)
```

```
Tablica po mnożeniu przez 2:
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]

Średnie kolumn:
[4. 5. 6.]
```

COMARCH
Szkolenia

# Analiza z NumPy i Pandas

Analiza z Pandas

2. Series i DataFrame w Pandas

- Omówienie Pandas
  - Pandas to potężna biblioteka w języku Python przeznaczona do manipulacji i analizy danych. Dwa główne obiekty w Pandas to Series i DataFrame. Series reprezentuje jednowymiarową strukturę danych, podczas gdy DataFrame to dwuwymiarowa tabela danych.

# Analiza z NumPy i Pandas

Tworzenie Series i DataFrame w Pandas:

In [16]:
```
import pandas as pd

# Tworzenie Series
seria = pd.Series([1, 3, 5, np.nan, 6, 8])

# Tworzenie DataFrame z tablicy NumPy
df_tablica = pd.DataFrame(np.random.randn(6, 4), columns=list('ABCD'))

# Tworzenie DataFrame z Dictionary
df_slownik = pd.DataFrame({'A': 1.0,
                           'B': pd.Timestamp('20220101'),
                           'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                           'D': np.array([3] * 4, dtype='int32'),
                           'E': pd.Categorical(["test", "train", "test", "train"]),
                           'F': 'foo'})
```

# Analiza z NumPy i Pandas

Series i DataFrame w Pandas

- Struktura Series:
  - Jednowymiarowy obiekt zawierający dane, indeks i etykiety.

- Struktura DataFrame:
  - Dwuwymiarowa tabela danych z etykietowanymi kolumnami i indeksem.

# Analiza z NumPy i Pandas

Ćwiczenie 2.

```
import pandas as pd
import numpy as np

# Ćwiczenie: Stworzenie DataFrame z danymi losowymi
df_cw = pd.DataFrame(np.random.randint(0, 100, size=(5, 3)), columns=['A', 'B', 'C'])

# Zadanie: Wyświetlenie informacji o danych w DataFrame
info_dane_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie trzech pierwszych wierszy DataFrame
pierwsze_wiersze_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Informacje o danych:", info_dane_cw)
print("Trzy pierwsze wiersze DataFrame:\n", pierwsze_wiersze_cw)
```

# Analiza z NumPy i Pandas

Rozwiązanie:

In [2]:
```
import pandas as pd
import numpy as np

# Stworzenie DataFrame z danymi losowymi
df_cw = pd.DataFrame(np.random.randint(0, 100, size=(5, 3)), columns=['A', 'B', 'C'])

# Wyświetlenie informacji o danych w DataFrame
info_dane_cw = df_cw.info()

# Wyświetlenie trzech pierwszych wierszy DataFrame
pierwsze_wiersze_cw = df_cw.head(3)

# Wyświetlenie wyników
print("Informacje o danych:", info_dane_cw)
print("Trzy pierwsze wiersze DataFrame:\n", pierwsze_wiersze_cw)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       5 non-null      int32
 1   B       5 non-null      int32
 2   C       5 non-null      int32
dtypes: int32(3)
memory usage: 192.0 bytes
Informacje o danych: None
Trzy pierwsze wiersze DataFrame:
     A   B   C
0  85  31  77
1  67  42  61
2  83  24  98
```

# Analiza z NumPy i Pandas

Ćwiczenie 2.

```
import pandas as pd
import numpy as np

# Ćwiczenie: Stworzenie DataFrame z danymi losowymi
df_cw = pd.DataFrame(np.random.randint(0, 100, size=(5, 3)), columns=['A', 'B', 'C'])

# Zadanie: Wyświetlenie informacji o danych w DataFrame
info_dane_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie trzech pierwszych wierszy DataFrame
pierwsze_wiersze_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Informacje o danych:", info_dane_cw)
print("Trzy pierwsze wiersze DataFrame:\n", pierwsze_wiersze_cw)
```

# Analiza z NumPy i Pandas

3. Wczytywanie i Zapis Danych w Różnych Formatach

Wczytywanie Danych w Pandas:

- Pandas oferuje wiele funkcji do wczytywania danych z różnych źródeł, takich jak pliki CSV, Excel, SQL, a nawet strony internetowe. Poniżej znajdują się przykłady wczytywania danych z pliku CSV i Excel:

# Analiza z NumPy i Pandas

Przykład:

Dane w csv (link):

https://www.kaggle.com/datasets/willianoliveiragibin/qs-top-100-universities

https://www.kaggle.com/datasets/hummaamqaasim/jobs-in-data

In [ ]:
```python
import pandas as pd

# Wczytywanie danych z pliku CSV
df_csv = pd.read_csv('nazwa_pliku.csv')

# Wczytywanie danych z pliku Excel
df_excel = pd.read_excel('nazwa_pliku.xlsx', sheet_name='Arkusz1')
```

# Analiza z NumPy i Pandas

3. Wczytywanie i Zapis Danych w Różnych Formatach

Zapis Danych w Pandas:

- Podobnie jak z wczytywaniem, Pandas umożliwia zapisywanie danych do różnych formatów. Poniżej znajdują się przykłady zapisywania danych do pliku CSV i Excel:

# Analiza z NumPy i Pandas

Przykład:

In [22]:
```
df_slownik.to_csv('NowyPlikSlownik.csv', index=False)
df_slownik.to_excel('NowyPlikExcel.xlsx', sheet_name='Slownik', index=False)
```

In [23]:
```
# Sprawdzenie Current Working Directory (CWD) w JupyterNotebook
# metoda 1
import os

notebook_path = os.getcwd()
print(notebook_path)

# metoda 2
from ipykernel import get_connection_file
# import os - to już zaimportowaliśmy więc nie ma potrzeby powielać, natomiast jest niezbędnę do wykorzystania metody 2

connection_file = get_connection_file()
notebook_path = os.path.dirname(connection_file)
print(notebook_path)
```

```
C:\Users\PabloPapito\PythonWAnalizieDanych\DayTwo
C:\Users\PabloPapito\AppData\Roaming\jupyter\runtime
```

# Analiza z NumPy i Pandas

Ćwiczenie 3.

- Wczytywanie danych z plików CSV i Excel.
- Zapisywanie danych do plików w różnych formatach.

# Analiza z NumPy i Pandas

## Ćwiczenie 3.

```
import pandas as pd

# Ćwiczenie: Wczytanie danych z pliku CSV
df_wczytane_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie pięciu pierwszych wierszy wczytanych danych
pierwsze_wiersze_wczytane_cw = # Tu dodaj kod

# Zadanie: Zapisanie wczytanych danych do pliku Excel
# (Uwaga: Dla potrzeb ćwiczenia, możesz użyć tych samych wczytanych danych)
df_wczytane_cw.to_excel('dane_wczytane.xlsx', index=False)

# Wyświetlenie wyników
print("Pięć pierwszych wierszy wczytanych danych:\n", pierwsze_wiersze_wczytane_cw)
print("Dane zostały zapisane do pliku Excel.")
```

# Analiza z NumPy i Pandas

4. Podstawowe Atrybuty DataFrame w Pandas

- Struktura DataFrame w Pandas:
  - DataFrame w Pandas to dwuwymiarowa struktura danych, która składa się z wierszy i kolumn. Przedstawmy kilka podstawowych atrybutów, które pozwalają zrozumieć strukturę danych w ramach DataFrame.

# Analiza z NumPy i Pandas

Atrybut shape:

- Atrybut shape zwraca krotkę reprezentującą liczbę wierszy i kolumn w DataFrame.

```
In [23]: import pandas as pd

         # Przykładowe utworzenie DataFrame
         data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
         df = pd.DataFrame(data)

         # Sprawdzenie kształtu DataFrame
         ksztalt = df.shape
         print("Kształt DataFrame:", ksztalt)

         Kształt DataFrame: (3, 2)
```

# Analiza z NumPy i Pandas

Atrybut index:

- Atrybut index zwraca indeksy wierszy (etykiety).

```
In [24]: # Sprawdzenie indeksów wierszy DataFrame
         indeksy = df.index
         print("Indeksy wierszy DataFrame:", indeksy)

         Indeksy wierszy DataFrame: RangeIndex(start=0, stop=3, step=1)
```

# Analiza z NumPy i Pandas

Atrybut columns:

- Atrybut columns zwraca etykiety kolumn.

```
In [25]: # Sprawdzenie etykiet kolumn DataFrame
         etykiety_kolumn = df.columns
         print("Etykiety kolumn DataFrame:", etykiety_kolumn)

         Etykiety kolumn DataFrame: Index(['A', 'B'], dtype='object')
```

# Analiza z NumPy i Pandas

Atrybut dtypes:

- Atrybut dtypes zwraca informacje o typach danych w poszczególnych kolumnach.

```
In [26]: # Sprawdzenie typów danych kolumn DataFrame
         typy_danych = df.dtypes
         print("Typy danych kolumn DataFrame:\n", typy_danych)
```

```
Typy danych kolumn DataFrame:
 A    int64
B    int64
dtype: object
```

# Analiza z NumPy i Pandas

Ćwiczenie 4.

- Sprawdzanie kształtu i struktury danych w DataFrame.
- Eksploracja indeksów, etykiet kolumn i typów danych.

# Analiza z NumPy i Pandas

Ćwiczenie 4.

```
import pandas as pd

# Ćwiczenie: Sprawdzenie kształtu i struktury DataFrame
df_cw = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Zadanie: Wyświetlenie kształtu DataFrame
ksztalt_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie indeksów wierszy DataFrame
indeksy_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie etykiet kolumn DataFrame
etykiety_kolumn_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie typów danych kolumn DataFrame
typy_danych_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Kształt DataFrame:", ksztalt_cw)
print("Indeksy wierszy DataFrame:", indeksy_cw)
print("Etykiety kolumn DataFrame:", etykiety_kolumn_cw)
print("Typy danych kolumn DataFrame:\n", typy_danych_cw)
```

# Analiza z NumPy i Pandas

5. Przydatne Funkcje w Pandas

- W Pandas istnieje wiele przydatnych funkcji do szybkiego podglądu i analizy danych. Poniżej omówione są niektóre z tych funkcji:

# Analiza z NumPy i Pandas

Funkcja describe()

- Funkcja describe() dostarcza podsumowania statystyczne dla kolumn w DataFrame, takie jak średnia, odchylenie standardowe, minimum, maksimum i kwartyle.

```
In [27]: # Przykład użycia describe
         opis_statystyczny = df.describe()
         print("Opis statystyczny DataFrame:\n", opis_statystyczny)
```

```
Opis statystyczny DataFrame:
              A    B
count  3.0  3.0
mean   2.0  5.0
std    1.0  1.0
min    1.0  4.0
25%    1.5  4.5
50%    2.0  5.0
75%    2.5  5.5
max    3.0  6.0
```

# Analiza z NumPy i Pandas

## Funkcja info()

- Funkcja info() wyświetla podstawowe informacje o DataFrame, takie jak ilość niepustych wartości i typy danych dla każdej kolumny.

```
In [28]: # Przykład użycia info
informacje_o_danych = df.info()
print("Informacje o danych w DataFrame:\n", informacje_o_danych)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       3 non-null      int64
 1   B       3 non-null      int64
dtypes: int64(2)
memory usage: 180.0 bytes
Informacje o danych w DataFrame:
 None
```

# Analiza z NumPy i Pandas

Funkcja head(n)

- Funkcja head(n) zwraca pierwsze n wierszy DataFrame, co jest przydatne do szybkiego podglądu danych.

```
In [29]: # Przykład użycia head
         pierwsze_wiersze = df.head(3)  # Zwróci trzy pierwsze wiersze
         print("Pierwsze trzy wiersze DataFrame:\n", pierwsze_wiersze)
```

```
Pierwsze trzy wiersze DataFrame:
    A  B
0  1  4
1  2  5
2  3  6
```

# Analiza z NumPy i Pandas

Funkcja tail(n)

- Funkcja tail(n) zwraca ostatnie n wierszy DataFrame, co pomaga przy weryfikacji danych na końcu zbioru.

```
In [30]: # Przykład użycia tail
         ostatnie_wiersze = df.tail(3)  # Zwróci trzy ostatnie wiersze
         print("Ostatnie trzy wiersze DataFrame:\n", ostatnie_wiersze)

         Ostatnie trzy wiersze DataFrame:
             A  B
         0   1  4
         1   2  5
         2   3  6
```

# Analiza z NumPy i Pandas

## Funkcja sample(n)

- Funkcja sample(n) zwraca losowe n wierszy z DataFrame.

```
In [31]: # Przykład użycia sample
losowe_wiersze = df.sample(3)  # Zwróci trzy losowe wiersze
print("Losowe trzy wiersze DataFrame:\n", losowe_wiersze)
```

```
Losowe trzy wiersze DataFrame:
    A  B
1   2  5
2   3  6
0   1  4
```

# Analiza z NumPy i Pandas

Ćwiczenie 5.

- Korzystanie z funkcji describe() do uzyskania statystyk opisowych.
- Wykorzystywanie funkcji info() do analizy struktury danych.
- Sprawdzanie pierwszych, ostatnich i losowych wierszy za pomocą head(), tail() i sample().

# Analiza z NumPy i Pandas

Ćwiczenie 5.

```
# Ćwiczenie: Użycie funkcji describe, info, head, tail, sample
df_cw = pd.DataFrame({'A': [1, 2, 3, 4, 5],
                      'B': ['a', 'b', 'c', 'd', 'e'],
                      'C': [0.1, 0.5, 0.8, 0.3, 0.9]})

# Zadanie: Uzyskanie opisu statystycznego dla DataFrame
opis_statystyczny_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie informacji o danych w DataFrame
informacje_o_danych_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie trzech pierwszych wierszy DataFrame
pierwsze_wiersze_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie trzech ostatnich wierszy DataFrame
ostatnie_wiersze_cw = # Tu dodaj kod

# Zadanie: Wyświetlenie trzech losowych wierszy DataFrame
losowe_wiersze_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Opis statystyczny DataFrame:\n", opis_statystyczny_cw)
print("Informacje o danych w DataFrame:\n", informacje_o_danych_cw)
print("Trzy pierwsze wiersze DataFrame:\n", pierwsze_wiersze_cw)
print("Trzy ostatnie wiersze DataFrame:\n", ostatnie_wiersze_cw)
print("Trzy losowe wiersze DataFrame:\n", losowe_wiersze_cw)
```

# Analiza z NumPy i Pandas

6. Czyszczenie Wartości Zduplikowanych w Pandas

Sprawdzanie Duplikatów:

- W analizie danych często konieczne jest sprawdzenie i usunięcie zduplikowanych wierszy. Pandas dostarcza funkcji do tego celu.

# Analiza z NumPy i Pandas

## 6. Czyszczenie Wartości Zduplikowanych w Pandas

In [64]:
```python
import pandas as pd

# Przykładowe utworzenie DataFrame z zduplikowanymi danymi
data = {'A': [1, 2, 2, 3, 4],
        'B': ['a', 'b', 'b', 'c', 'd']}
df2 = pd.DataFrame(data)

# Sprawdzenie duplikatów w całych wierszach
duplikaty_wiersze = df2.duplicated()

# Sprawdzenie duplikatów w kolumnie 'A'
duplikaty_kolumna_A = df2['A'].duplicated()

print("Duplikaty wierszy:\n", duplikaty_wiersze)
print("Duplikaty w kolumnie 'A':\n", duplikaty_kolumna_A)
```

# Analiza z NumPy i Pandas

## 6. Czyszczenie Wartości Zduplikowanych w Pandas

In [72]:
```
# Usunięcie zduplikowanych wierszy (zostawiając pierwszy wystąpienie)
df_bez_duplikatow = df2.drop_duplicates()

# Usunięcie zduplikowanych wierszy bazując na konkretnej kolumnie
df_bez_duplikatow_kolumna_A = df2.drop_duplicates(subset='A')

print("DataFrame bez duplikatów:\n", df_bez_duplikatow)
print("DataFrame bez duplikatów w kolumnie 'A':\n", df_bez_duplikatow_kolumna_A)
```

```
DataFrame bez duplikatów:
    A  B
0  1  a
1  2  b
3  3  c
4  4  d
DataFrame bez duplikatów w kolumnie 'A':
    A  B
0  1  a
1  2  b
3  3  c
4  4  d
```

# Analiza z NumPy i Pandas

Ćwiczenie 6.

- Sprawdzanie duplikatów w danych.
- Usuwanie zduplikowanych wierszy.

# Analiza z NumPy i Pandas

Ćwiczenie 6.

```
# Ćwiczenie: Sprawdzenie duplikatów i usunięcie ich
df_cw = pd.DataFrame({'A': [1, 2, 2, 3, 4],
                      'B': ['a', 'b', 'b', 'c', 'd']})

# Zadanie: Sprawdzenie duplikatów w całych wierszach
duplikaty_wiersze_cw = # Tu dodaj kod

# Zadanie: Sprawdzenie duplikatów w kolumnie 'A'
duplikaty_kolumna_A_cw = # Tu dodaj kod

# Zadanie: Usunięcie zduplikowanych wierszy
df_bez_duplikatow_cw = # Tu dodaj kod

# Zadanie: Usunięcie zduplikowanych wierszy bazując na kolumnie 'A'
df_bez_duplikatow_kolumna_A_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Duplikaty wierszy:\n", duplikaty_wiersze_cw)
print("Duplikaty w kolumnie 'A':\n", duplikaty_kolumna_A_cw)
print("DataFrame bez duplikatów:\n", df_bez_duplikatow_cw)
print("DataFrame bez duplikatów w kolumnie 'A':\n", df_bez_duplikatow_kolumna_A_cw)
```

# Analiza z NumPy i Pandas

7. Wartości Brakujące - Różne Podejścia do Radzenia Sobie z Nimi w Pandas

- W analizie danych często spotykamy się z wartościami brakującymi (NaN lub None). Pandas oferuje różne metody radzenia sobie z tymi wartościami.

# Analiza z NumPy i Pandas

7. Wartości Brakujące (None/Null) - Różne Podejścia do Radzenia Sobie z Nimi

- Sprawdzenie Wartości Brakujących:

In [73]:
```
import pandas as pd

# Przykładowe utworzenie DataFrame z wartościami brakującymi
data = {'A': [1, 2, None, 4],
        'B': ['a', 'b', 'c', None]}
df = pd.DataFrame(data)

# Sprawdzenie, czy istnieją wartości brakujące w DataFrame
brakujace_wartosci = df.isnull()

print("Wartości brakujące w DataFrame:\n", brakujace_wartosci)
```

```
Wartości brakujące w DataFrame:
        A      B
0  False  False
1  False  False
2   True  False
3  False   True
```

# Analiza z NumPy i Pandas

7. Wartości Brakujące (None/Null) - Różne Podejścia do Radzenia Sobie z Nimi

- Usuwanie Wartości Brakujących:

In [74]:
```
# Usunięcie wierszy zawierających przynajmniej jedną wartość brakującą
df_bez_brakujacych_wierszy = df.dropna()

# Usunięcie kolumn zawierających przynajmniej jedną wartość brakującą
df_bez_brakujacych_kolumn = df.dropna(axis=1)

print("DataFrame bez wierszy zawierających wartości brakujące:\n", df_bez_brakujacych_wierszy)
print("DataFrame bez kolumn zawierających wartości brakujące:\n", df_bez_brakujacych_kolumn)
```

```
DataFrame bez wierszy zawierających wartości brakujące:
      A  B
0  1.0  a
1  2.0  b
DataFrame bez kolumn zawierających wartości brakujące:
 Empty DataFrame
Columns: []
Index: [0, 1, 2, 3]
```

# Analiza z NumPy i Pandas

7. Wartości Brakujące (None/Null) - Różne Podejścia do Radzenia Sobie z Nimi

- Uzupełnianie Wartości Brakujących:

```
In [75]: # Uzupełnienie wartości brakujących określoną wartością (np. średnią)
srednia_wartosc_A = df['A'].mean()
df_uzupelnione = df.fillna({'A': srednia_wartosc_A, 'B': 'brak_danych'})

print("DataFrame po uzupełnieniu wartości brakujących:\n", df_uzupelnione)
```

```
DataFrame po uzupełnieniu wartości brakujących:
          A            B
0  1.000000            a
1  2.000000            b
2  2.333333            c
3  4.000000  brak_danych
```

COMARCH
Szkolenia

# Analiza z NumPy i Pandas

Ćwiczenie 7.

- Sprawdzanie wartości brakujących w danych.
- Usuwanie wierszy i kolumn zawierających wartości brakujące.
- Uzupełnianie wartości brakujących.

# Analiza z NumPy i Pandas

Ćwiczenie 7.

```
# Ćwiczenie: Radzenie sobie z wartościami brakującymi
df_cw = pd.DataFrame({'A': [1, 2, None, 4],
                      'B': ['a', 'b', 'c', None]})

# Zadanie: Sprawdzenie wartości brakujących w DataFrame
brakujace_wartosci_cw = # Tu dodaj kod

# Zadanie: Usunięcie wierszy zawierających wartości brakujące
df_bez_brakujacych_wierszy_cw = # Tu dodaj kod

# Zadanie: Uzupełnienie wartości brakujących w kolumnie 'A' średnią wartością
srednia_wartosc_A_cw = # Tu dodaj kod
df_uzupelnione_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Wartości brakujące w DataFrame:\n", brakujace_wartosci_cw)
print("DataFrame bez wierszy zawierających wartości brakujące:\n", df_bez_brakujacych_wierszy_cw)
print("DataFrame po uzupełnieniu wartości brakujących:\n", df_uzupelnione_cw)
```

# Analiza z NumPy i Pandas

8. Wykrywanie Wartości Odstających w Pandas

Wartości odstające (ang. outliers) są danymi, które znacząco różnią się od reszty zbioru danych i mogą wpływać na wyniki analizy. Pandas pozwala na wykrywanie tych wartości i podejmowanie odpowiednich działań.

# Analiza z NumPy i Pandas

## Wykrywanie Wartości Odstających:

In [40]:
```python
import pandas as pd

# Przykładowe utworzenie DataFrame z wartościami odstającymi
data = {'A': [1, 2, 3, 100],
        'B': [4, 5, 6, 200]}
df = pd.DataFrame(data)

# Wykrywanie wartości odstających na podstawie kwantyli
kwantyl_25 = df.quantile(0.25)
kwantyl_75 = df.quantile(0.75)
rozstep_miedzykwartylowy = kwantyl_75 - kwantyl_25

# Definiowanie zakresu wartości "normalnych"
dolny_limit = kwantyl_25 - 1.5 * rozstep_miedzykwartylowy
gorny_limit = kwantyl_75 + 1.5 * rozstep_miedzykwartylowy

# Wykrywanie wartości odstających
odstajace_wartosci = (df < dolny_limit) | (df > gorny_limit)

print("Wartości odstające w DataFrame:\n", odstajace_wartosci)
```

```
Wartości odstające w DataFrame:
       A      B
0  False  False
1  False  False
2  False  False
3   True   True
```

# Analiza z NumPy i Pandas

Zastępowanie Wartości Odstających:

In [41]:
```
# Zastępowanie wartości odstających wartością graniczną
df_bez_odstajacych = df.mask(odstajace_wartosci, gorny_limit, axis=1)

print("DataFrame po zastąpieniu wartości odstających:\n", df_bez_odstajacych)
```

```
DataFrame po zastąpieniu wartości odstających:
       A        B
0   1.0    4.000
1   2.0    5.000
2   3.0    6.000
3  65.5  129.125
```

# Analiza z NumPy i Pandas

Ćwiczenie 8.

- Wykrywanie wartości odstających w danych.
- Zastępowanie lub usuwanie wartości odstających.

# Analiza z NumPy i Pandas

Ćwiczenie 8.

```
# Ćwiczenie: Wykrywanie i radzenie sobie z wartościami odstającymi
df_cw = pd.DataFrame({'A': [1, 2, 3, 100],
                      'B': [4, 5, 6, 200]})

# Zadanie: Wykrycie wartości odstających na podstawie kwantyli
kwantyl_25_cw = # Tu dodaj kod
kwantyl_75_cw = # Tu dodaj kod
rozstep_miedzykwartylowy_cw = # Tu dodaj kod

# Zadanie: Definiowanie zakresu wartości "normalnych"
dolny_limit_cw = # Tu dodaj kod
gorny_limit_cw = # Tu dodaj kod

# Zadanie: Wykrywanie wartości odstających
odstajace_wartosci_cw = # Tu dodaj kod

# Zadanie: Zastępowanie wartości odstających wartością graniczną
df_bez_odstajacych_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Wartości odstające w DataFrame:\n", odstajace_wartosci_cw)
print("DataFrame po zastąpieniu wartości odstających:\n", df_bez_odstajacych_cw)
```

# Analiza z NumPy i Pandas

9. Sortowanie Danych w Pandas

Sortowanie danych jest ważnym krokiem w analizie danych, umożliwiając uporządkowanie danych według określonych kryteriów. Pandas oferuje funkcje umożliwiające sortowanie zarówno wierszy, jak i kolumn.

# Analiza z NumPy i Pandas

Sortowanie wierszy:

In [42]:
```
import pandas as pd

# Przykładowe utworzenie DataFrame do sortowania wierszy
data = {'A': [3, 1, 4, 2],
        'B': ['c', 'a', 'd', 'b']}
df = pd.DataFrame(data)

# Sortowanie wierszy według kolumny 'A'
df_posortowany = df.sort_values(by='A')

print("DataFrame po posortowaniu wierszy według kolumny 'A':\n", df_posortowany)
```

```
DataFrame po posortowaniu wierszy według kolumny 'A':
    A  B
1  1  a
3  2  b
0  3  c
2  4  d
```

# Analiza z NumPy i Pandas

Sortowanie kolumn:

In [43]:
```
# Sortowanie kolumn według etykiet kolumn
df_kolumny_posortowane = df.sort_index(axis=1)

print("DataFrame po posortowaniu kolumn według etykiet:\n", df_kolumny_posortowane)
```

```
DataFrame po posortowaniu kolumn według etykiet:
    A  B
0  3  c
1  1  a
2  4  d
3  2  b
```

# Analiza z NumPy i Pandas

Sortowanie malejące:

In [44]:
```
# Sortowanie wierszy malejąco według kolumny 'A'
df_posortowany_malejaco = df.sort_values(by='A', ascending=False)

print("DataFrame po posortowaniu malejąco według kolumny 'A':\n", df_posortowany_malejaco)
```

```
DataFrame po posortowaniu malejąco według kolumny 'A':
    A  B
2  4  d
0  3  c
3  2  b
1  1  a
```

# Analiza z NumPy i Pandas

Ćwiczenie 9.

- Sortowanie wierszy i kolumn w różnych kierunkach.
- Sortowanie danych według różnych kryteriów.

# Analiza z NumPy i Pandas

Ćwiczenie 9.

```
# Ćwiczenie: Sortowanie danych w Pandas
df_cw = pd.DataFrame({'A': [3, 1, 4, 2],
                      'B': ['c', 'a', 'd', 'b']})

# Zadanie: Sortowanie wierszy według kolumny 'A'
df_posortowany_cw = # Tu dodaj kod

# Zadanie: Sortowanie kolumn według etykiet
df_kolumny_posortowane_cw = # Tu dodaj kod

# Zadanie: Sortowanie wierszy malejąco według kolumny 'A'
df_posortowany_malejaco_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("DataFrame po posortowaniu wierszy według kolumny 'A':\n", df_posortowany_cw)
print("DataFrame po posortowaniu kolumn według etykiet:\n", df_kolumny_posortowane_cw)
print("DataFrame po posortowaniu malejąco według kolumny 'A':\n", df_posortowany_malejaco_cw)
```

# Analiza z NumPy i Pandas

## 10. Filtrowanie Danych w Pandas

Filtrowanie danych to proces wybierania jedynie tych danych, które spełniają określone kryteria. W Pandas istnieje kilka metod do filtrowania danych, a każda z nich ma swoje zastosowanie.

# Analiza z NumPy i Pandas

Filtrowanie za pomocą loc:

In [70]:
```
import pandas as pd

# Przykładowe utworzenie DataFrame do filtrowania
data = {'A': [1, 2, 3, 4],
        'B': ['a', 'b', 'c', 'd']}

dfBezIndeksow = pd.DataFrame(data)

dfBezIndeksow.index += 1 # by default indeksy są od 0 - tutaj możemy ustawić od 1

df = pd.DataFrame(data, index=['x', 'y', 'z', 'w'])

# Filtrowanie wierszy o indeksach 'x' i 'y' oraz kolumny 'A'
df_filtr_loc = df.loc[['x', 'y'], 'A']


print('Nasz zbiór danych:\n', data, '\n\n----------------------\n')

print('Nasz zbiór danych bez indeksów literowych tylko cyfrowe od 0:\n', dfBezIndeksow, '\n\n----------------------\n')



print('Dodanie indeksów wierszy do danych:\n', df, '\n\n----------------------\n')

print("Wynik filtrowania za pomocą loc:\n", df_filtr_loc)
```

# Analiza z NumPy i Pandas

Filtrowanie za pomocą iloc:

In [81]:
```
# Filtrowanie pierwszych dwóch wierszy oraz pierwszej kolumny
df_filtr_iloc = df.iloc[:2, 0] # pierwszy argument to wiersz, a drugi to kolumna

print("Wynik filtrowania za pomocą iloc:\n", df_filtr_iloc)
```

```
Wynik filtrowania za pomocą iloc:
 x    1
y    2
Name: A, dtype: int64
```

# Analiza z NumPy i Pandas

Filtrowanie za pomocą query:

In [82]:
```
# Filtrowanie wierszy, gdzie wartość w kolumnie 'A' jest większa niż 2
df_filtr_query = df.query('A > 2')

print("Wynik filtrowania za pomocą query:\n", df_filtr_query)
```

```
Wynik filtrowania za pomocą query:
    A  B
z  3  c
w  4  d
```

# Analiza z NumPy i Pandas

Filtrowanie za pomocą where:

```
In [83]: # Filtrowanie, zwracając nowy DataFrame z wartościami, które spełniają warunek, a reszta to NaN
         df_filtr_where = df.where(df['A'] > 2)

         print("Wynik filtrowania za pomocą where:\n", df_filtr_where)
```

```
Wynik filtrowania za pomocą where:
     A    B
x  NaN  NaN
y  NaN  NaN
z  3.0    c
w  4.0    d
```

# Analiza z NumPy i Pandas

Filtrowanie za pomocą isin:

In [87]:
```
# Filtrowanie wierszy, gdzie wartość w kolumnie 'B' jest jedną z określonych wartości
df_filtr_isin = df[df['B'].isin(['a', 'c'])]

print("Wynik filtrowania za pomocą isin:\n", df_filtr_isin)
```

```
Wynik filtrowania za pomocą isin:
    A  B
x   1  a
z   3  c
```

# Analiza z NumPy i Pandas

Ćwiczenie 10.

- Filtrowanie danych na podstawie różnych warunków.
- Kombinowanie różnych metod filtrowania w jednym wyrażeniu.

# Analiza z NumPy i Pandas

## Ćwiczenie 10.

```
# Ćwiczenie: Filtrowanie danych w Pandas
df_cw = pd.DataFrame({'A': [1, 2, 3, 4],
                      'B': ['a', 'b', 'c', 'd']}, index=['x', 'y', 'z', 'w'])

# Zadanie: Filtrowanie wierszy o indeksach 'x' i 'y' oraz kolumny 'A'
df_filtr_loc_cw = # Tu dodaj kod

# Zadanie: Filtrowanie pierwszych dwóch wierszy oraz pierwszej kolumny za pomocą iloc
df_filtr_iloc_cw = # Tu dodaj kod

# Zadanie: Filtrowanie wierszy, gdzie wartość w kolumnie 'A' jest większa niż 2 za pomocą query
df_filtr_query_cw = # Tu dodaj kod

# Zadanie: Filtrowanie, zwracając nowy DataFrame z wartościami, które spełniają warunek, a reszta to NaN za pomocą where
df_filtr_where_cw = # Tu dodaj kod

# Zadanie: Filtrowanie wierszy, gdzie wartość w kolumnie 'B' jest jedną z określonych wartości za pomocą isin
df_filtr_isin_cw = # Tu dodaj kod

# Zadanie: Filtrowanie wierszy, gdzie wartość w kolumnie 'A' jest NaN za pomocą isnull
df_filtr_isnull_cw = # Tu dodaj kod

# Zadanie: Filtrowanie wierszy, gdzie wartość w kolumnie 'A' nie jest NaN za pomocą notnull
df_filtr_notnull_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Wynik filtrowania za pomocą loc:\n", df_filtr_loc_cw)
print("Wynik filtrowania za pomocą iloc:\n", df_filtr_iloc_cw)
print("Wynik filtrowania za pomocą query:\n", df_filtr_query_cw)
print("Wynik filtrowania za pomocą where:\n", df_filtr_where_cw)
print("Wynik filtrowania za pomocą isin:\n", df_filtr_isin_cw)
print("Wynik filtrowania za pomocą isnull:\n", df_filtr_isnull_cw)
print("Wynik filtrowania za pomocą notnull:\n", df_filtr_notnull_cw)
```

# Analiza z NumPy i Pandas

11. Tabele Przestawne w Pandas

- Tabele przestawne to narzędzie umożliwiające podsumowanie i analizę danych w formie tabeli. Pandas oferuje funkcję do tworzenia tabel przestawnych, co ułatwia analizę różnych aspektów danych.

# Analiza z NumPy i Pandas

Tworzenie Tabeli Przestawnej:

In [51]:
```python
import pandas as pd

# Przykładowe utworzenie DataFrame do tabeli przestawnej
data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Value': [10, 20, 30, 40, 50, 60]}
df = pd.DataFrame(data)

# Tworzenie tabeli przestawnej
table_przestawna = pd.pivot_table(df, values='Value', columns='Category', aggfunc='sum')

print("Tabela przestawna:\n", table_przestawna)
```

```
Tabela przestawna:
 Category   A    B
Value      90  120
```

# Analiza z NumPy i Pandas

## Określanie Indeksów i Kolumn Tabeli Przestawnej:

In [52]:
```
# Określanie indeksów i kolumn dla tabeli przestawnej
table_przestawna_indeks_kolumny = pd.pivot_table(df, values='Value', index='Category', columns='Category', aggfunc='sum')

print("Tabela przestawna z określonymi indeksami i kolumnami:\n", table_przestawna_indeks_kolumny)
```

```
Tabela przestawna z określonymi indeksami i kolumnami:
 Category      A      B
Category
A          90.0    NaN
B           NaN  120.0
```

# Analiza z NumPy i Pandas

Określanie Funkcji Agregującej:

In [53]:
```
# Określanie funkcji agregującej dla tabeli przestawnej (np. średnia)
table_przestawna_srednia = pd.pivot_table(df, values='Value', index='Category', aggfunc='mean')

print("Tabela przestawna z określoną funkcją agregującą (średnia):\n", table_przestawna_srednia)
```

```
Tabela przestawna z określoną funkcją agregującą (średnia):
           Value
Category
A             30
B             40
```

# Analiza z NumPy i Pandas

## Określanie Indeksów i Kolumn Tabeli Przestawnej:

In [52]:
```
# Określanie indeksów i kolumn dla tabeli przestawnej
table_przestawna_indeks_kolumny = pd.pivot_table(df, values='Value', index='Category', columns='Category', aggfunc='sum')

print("Tabela przestawna z określonymi indeksami i kolumnami:\n", table_przestawna_indeks_kolumny)
```

```
Tabela przestawna z określonymi indeksami i kolumnami:
 Category      A      B
Category
A          90.0    NaN
B           NaN  120.0
```

# Analiza z NumPy i Pandas

Uzupełnianie Wartości Brakujących:

In [54]:
```
# Uzupełnianie wartości brakujących w tabeli przestawnej (np. wartością 0)
table_przestawna_uzupelniona = pd.pivot_table(df, values='Value', index='Category', fill_value=0, aggfunc='sum')

print("Tabela przestawna z uzupełnionymi wartościami brakującymi:\n", table_przestawna_uzupelniona)
```

```
Tabela przestawna z uzupełnionymi wartościami brakującymi:
           Value
Category
A             90
B            120
```

# Analiza z NumPy i Pandas

## Określanie Indeksów i Kolumn Tabeli Przestawnej:

In [52]:
```
# Określanie indeksów i kolumn dla tabeli przestawnej
table_przestawna_indeks_kolumny = pd.pivot_table(df, values='Value', index='Category', columns='Category', aggfunc='sum')

print("Tabela przestawna z określonymi indeksami i kolumnami:\n", table_przestawna_indeks_kolumny)
```

```
Tabela przestawna z określonymi indeksami i kolumnami:
 Category      A      B
Category
A          90.0    NaN
B           NaN  120.0
```

# Analiza z NumPy i Pandas

Ćwiczenie 11.

- Tworzenie różnych tabel przestawnych na podstawie dostępnych danych.
- Określanie różnych funkcji agregujących w tabelach przestawnych.
- Eksperymentowanie z różnymi ustawieniami indeksów i kolumn.

# Analiza z NumPy i Pandas

## Ćwiczenie 11.

```
# Ćwiczenie: Tworzenie tabeli przestawnej w Pandas
df_cw = pd.DataFrame({'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
                      'Value': [10, 20, 30, 40, 50, 60]})

# Zadanie: Stwórz tabelę przestawną na podstawie danych w df_cw
table_przestawna_cw = # Tu dodaj kod

# Zadanie: Określ indeksy i kolumny dla tabeli przestawnej
table_przestawna_indeks_kolumny_cw = # Tu dodaj kod

# Zadanie: Określ funkcję agregującą (np. suma) dla tabeli przestawnej
table_przestawna_suma_cw = # Tu dodaj kod

# Zadanie: Uzupełnij wartości brakujące w tabeli przestawnej (np. wartością 0)
table_przestawna_uzupelniona_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Tabela przestawna:\n", table_przestawna_cw)
print("Tabela przestawna z określonymi indeksami i kolumnami:\n", table_przestawna_indeks_kolumny_cw)
print("Tabela przestawna z określoną funkcją agregującą (suma):\n", table_przestawna_suma_cw)
print("Tabela przestawna z uzupełnionymi wartościami brakującymi:\n", table_przestawna_uzupelniona_cw)
```

# Analiza z NumPy i Pandas

12. Grupowanie Danych w Pandas

- Grupowanie danych to proces dzielenia danych na grupy w oparciu o określone kryteria, a następnie wykonywania operacji na każdej z tych grup. W Pandas, do grupowania danych używamy funkcji groupby.

# Analiza z NumPy i Pandas

Grupowanie na Podstawie Jednej Kolumny:

In [55]:
```
import pandas as pd

# Przykładowe utworzenie DataFrame do grupowania
data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Value': [10, 20, 30, 40, 50, 60]}
df = pd.DataFrame(data)

# Grupowanie na podstawie kolumny 'Category' i obliczanie sumy dla każdej grupy
grupowanie_jedna_kolumna = df.groupby('Category').sum()

print("Wynik grupowania na podstawie jednej kolumny:\n", grupowanie_jedna_kolumna)
```

```
Wynik grupowania na podstawie jednej kolumny:
          Value
Category
A            90
B           120
```

# Analiza z NumPy i Pandas

Grupowanie na Podstawie Wielu Kolumn:

In [56]:
```
# Grupowanie na podstawie wielu kolumn i obliczanie sumy dla każdej grupy
grupowanie_wiele_kolumn = df.groupby(['Category', 'Value']).size().reset_index(name='Count')

print("Wynik grupowania na podstawie wielu kolumn:\n", grupowanie_wiele_kolumn)
```

```
Wynik grupowania na podstawie wielu kolumn:
   Category  Value  Count
0         A     10      1
1         A     30      1
2         A     50      1
3         B     20      1
4         B     40      1
5         B     60      1
```

# Analiza z NumPy i Pandas

Wykonywanie Różnych Operacji dla Każdej Grupy:

In [57]:
```
# Grupowanie i obliczanie różnych statystyk dla każdej grupy
grupowanie_statystyki = df.groupby('Category').agg({'Value': ['sum', 'mean', 'count']})

print("Wynik grupowania z różnymi operacjami dla każdej grupy:\n", grupowanie_statystyki)
```

```
Wynik grupowania z różnymi operacjami dla każdej grupy:
           Value
             sum  mean count
Category
A             90  30.0     3
B            120  40.0     3
```

# Analiza z NumPy i Pandas

Grupowanie na Podstawie Wielu Kolumn:

In [56]:
```
# Grupowanie na podstawie wielu kolumn i obliczanie sumy dla każdej grupy
grupowanie_wiele_kolumn = df.groupby(['Category', 'Value']).size().reset_index(name='Count')

print("Wynik grupowania na podstawie wielu kolumn:\n", grupowanie_wiele_kolumn)
```

```
Wynik grupowania na podstawie wielu kolumn:
   Category  Value  Count
0         A     10      1
1         A     30      1
2         A     50      1
3         B     20      1
4         B     40      1
5         B     60      1
```

# Analiza z NumPy i Pandas

Ćwiczenie 12.

- Grupowanie danych na podstawie różnych kryteriów.
- Wykonywanie różnych operacji na grupach danych.

# Analiza z NumPy i Pandas

Ćwiczenie 12.

```
# Ćwiczenie: Grupowanie danych w Pandas
df_cw = pd.DataFrame({'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
                      'Value': [10, 20, 30, 40, 50, 60]})

# Zadanie: Grupowanie na podstawie kolumny 'Category' i obliczanie sumy dla każdej grupy
grupowanie_jedna_kolumna_cw = # Tu dodaj kod

# Zadanie: Grupowanie na podstawie wielu kolumn i obliczanie sumy dla każdej grupy
grupowanie_wiele_kolumn_cw = # Tu dodaj kod

# Zadanie: Grupowanie i obliczanie różnych statystyk dla każdej grupy
grupowanie_statystyki_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Wynik grupowania na podstawie jednej kolumny:\n", grupowanie_jedna_kolumna_cw)
print("Wynik grupowania na podstawie wielu kolumn:\n", grupowanie_wiele_kolumn_cw)
print("Wynik grupowania z różnymi operacjami dla każdej grupy:\n", grupowanie_statystyki_cw)
```

# Analiza z NumPy i Pandas

13. Łączenie Danych w Pandas

- Łączenie danych to proces łączenia dwóch lub więcej DataFrame'ów w celu utworzenia jednego DataFrame'u na podstawie wspólnych kolumn lub indeksów. W Pandas, do łączenia danych używamy funkcji merge lub concat.

# Analiza z NumPy i Pandas

Łączenie na Podstawie Wspólnej Kolumny:

In [58]:
```
import pandas as pd

# Przykładowe utworzenie dwóch DataFrame'ów do łączenia
df1 = pd.DataFrame({'ID': [1, 2, 3],
                    'Value1': ['A', 'B', 'C']})

df2 = pd.DataFrame({'ID': [2, 3, 4],
                    'Value2': ['X', 'Y', 'Z']})

# łączenie na podstawie wspólnej kolumny 'ID'
laczenie_kolumna = pd.merge(df1, df2, on='ID')

print("Wynik łączenia na podstawie wspólnej kolumny:\n", laczenie_kolumna)
```

```
Wynik łączenia na podstawie wspólnej kolumny:
    ID Value1 Value2
0   2      B      X
1   3      C      Y
```

# Analiza z NumPy i Pandas

Łączenie na Podstawie Wspólnego Indeksu:

In [59]:
```
# Przykładowe utworzenie dwóch DataFrame'ów do łączenia na podstawie indeksu
df3 = pd.DataFrame({'Value3': ['M', 'N', 'O']}, index=[1, 2, 3])

# łączenie na podstawie wspólnego indeksu
laczenie_indeks = pd.concat([df1, df3], axis=1)

print("Wynik łączenia na podstawie wspólnego indeksu:\n", laczenie_indeks)
```

```
Wynik łączenia na podstawie wspólnego indeksu:
     ID Value1 Value3
0  1.0      A    NaN
1  2.0      B      M
2  3.0      C      N
3  NaN    NaN      O
```

# Analiza z NumPy i Pandas

Łączenie na Podstawie Wspólnego Indeksu w Kierunku Pionowym:

In [60]:
```
# łączenie na podstawie wspólnego indeksu w kierunku pionowym
laczenie_pionowe = pd.concat([df1, df3], axis=0)

print("Wynik łączenia na podstawie wspólnego indeksu w kierunku pionowym:\n", laczenie_pionowe)
```

```
Wynik łączenia na podstawie wspólnego indeksu w kierunku pionowym:
     ID Value1 Value3
0   1.0      A    NaN
1   2.0      B    NaN
2   3.0      C    NaN
1   NaN    NaN      M
2   NaN    NaN      N
3   NaN    NaN      O
```

# Analiza z NumPy i Pandas

Ćwiczenie 13.

- Łączenie danych na podstawie różnych kolumn lub indeksów.
- Eksperymentowanie z różnymi typami łączenia, takimi jak inner, outer, left, right.

# Analiza z NumPy i Pandas

Ćwiczenie 13.

```
# Ćwiczenie: Łączenie danych w Pandas
df1_cw = pd.DataFrame({'ID': [1, 2, 3],
                       'Value1': ['A', 'B', 'C']})

df2_cw = pd.DataFrame({'ID': [2, 3, 4],
                       'Value2': ['X', 'Y', 'Z']})

# Zadanie: Łączenie na podstawie wspólnej kolumny 'ID'
laczenie_kolumna_cw = # Tu dodaj kod

# Zadanie: Przykładowe utworzenie DataFrame'ów do łączenia na podstawie indeksu
df3_cw = pd.DataFrame({'Value3': ['M', 'N', 'O']}, index=[1, 2, 3])

# Zadanie: Łączenie na podstawie wspólnego indeksu
laczenie_indeks_cw = # Tu dodaj kod

# Zadanie: Łączenie na podstawie wspólnego indeksu w kierunku pionowym
laczenie_pionowe_cw = # Tu dodaj kod

# Wyświetlenie wyników
print("Wynik łączenia na podstawie wspólnej kolumny:\n", laczenie_kolumna_cw)
print("Wynik łączenia na podstawie wspólnego indeksu:\n", laczenie_indeks_cw)
print("Wynik łączenia na podstawie wspólnego indeksu w kierunku pionowym:\n", laczenie_pionowe_cw)
```

# Analiza z NumPy i Pandas

14. Tworzenie Nowych Atrybutów w Pandas

- Tworzenie nowych atrybutów (kolumn) to kluczowy krok w analizie danych, który pozwala na dostosowywanie danych do konkretnych potrzeb i celów analizy. W Pandas, nowe atrybuty można dodawać poprzez różne operacje na istniejących danych.

# Analiza z NumPy i Pandas

Dodawanie Nowego Atrybutu na Podstawie Istniejących Kolumn:

In [61]:
```python
import pandas as pd

# Przykładowe utworzenie DataFrame
df = pd.DataFrame({'Value1': [10, 20, 30],
                   'Value2': [5, 15, 25]})

# Dodawanie nowego atrybutu 'Sum' na podstawie istniejących kolumn
df['Sum'] = df['Value1'] + df['Value2']

print("DataFrame z dodanym atrybutem 'Sum':\n", df)
```

```
DataFrame z dodanym atrybutem 'Sum':
    Value1  Value2  Sum
0       10       5   15
1       20      15   35
2       30      25   55
```

# Analiza z NumPy i Pandas

Dodawanie Nowego Atrybutu na Podstawie Warunków Logicznych:

In [62]:
```
# Dodawanie nowego atrybutu 'Category' na podstawie warunków logicznych
df['Category'] = ['High' if x > 15 else 'Low' for x in df['Sum']]

print("DataFrame z dodanym atrybutem 'Category':\n", df)
```

```
DataFrame z dodanym atrybutem 'Category':
    Value1  Value2  Sum Category
0      10       5   15      Low
1      20      15   35     High
2      30      25   55     High
```

# Analiza z NumPy i Pandas

Utworzenie Nowego Atrybutu przy Użyciu Funkcji:

In [63]:
```python
# Utworzenie nowego atrybutu 'Product' przy użyciu funkcji
def categorize_product(sum_value):
    if sum_value > 30:
        return 'Premium'
    elif sum_value > 20:
        return 'Standard'
    else:
        return 'Basic'

df['Product'] = df['Sum'].apply(categorize_product)

print("DataFrame z dodanym atrybutem 'Product':\n", df)
```

```
DataFrame z dodanym atrybutem 'Product':
    Value1  Value2  Sum Category  Product
0      10       5   15      Low    Basic
1      20      15   35     High  Premium
2      30      25   55     High  Premium
```

# Analiza z NumPy i Pandas

Ćwiczenie 14.

- Dodawanie nowych atrybutów na podstawie istniejących danych.
- Utworzenie nowych atrybutów przy użyciu warunków logicznych.
- Wykorzystanie funkcji do tworzenia nowych atrybutów.

# Analiza z NumPy i Pandas

Ćwiczenie 14.

```
# Ćwiczenie: Tworzenie nowych atrybutów w Pandas
df_cw = pd.DataFrame({'Value1': [10, 20, 30],
                      'Value2': [5, 15, 25]})

# Zadanie: Dodaj nowy atrybut 'Multiply' na podstawie iloczynu istniejących kolumn
df_cw['Multiply'] = # Tu dodaj kod

# Zadanie: Dodaj nowy atrybut 'Status' na podstawie warunku logicznego (np. > 50)
df_cw['Status'] = # Tu dodaj kod

# Zadanie: Utwórz nowy atrybut 'Grade' przy użyciu funkcji (np. A, B, C)
def categorize_grade(multiply_value):
    # Tu dodaj kod

df_cw['Grade'] = # Tu dodaj kod

# Wyświetlenie wyników
print("DataFrame z dodanym atrybutem 'Multiply':\n", df_cw)
print("DataFrame z dodanym atrybutem 'Status':\n", df_cw)
print("DataFrame z dodanym atrybutem 'Grade':\n", df_cw)
```

# Wizualizacja Danych z Matplotlib i Seaborn

Wizualizacja danych to kluczowy element analizy danych, pozwalający przedstawić wnioski i trendy w sposób czytelny i zrozumiały. Matplotlib i Seaborn to dwie popularne biblioteki w języku Python używane do tworzenia wykresów i wizualizacji danych. Poniżej omówię podstawowe elementy związane z wizualizacją danych, zgodnie z programem szkolenia.

# Wizualizacja Danych z Matplotlib i Seaborn

Rodzaje Wykresów i Wizualizacji:

W Matplotlib i Seaborn istnieje wiele rodzajów wykresów, takich jak:

- histogramy,
- wykresy słupkowe,
- wykresy punktowe,
- wykresy liniowe,
- heatmapy,
- pairploty itp.

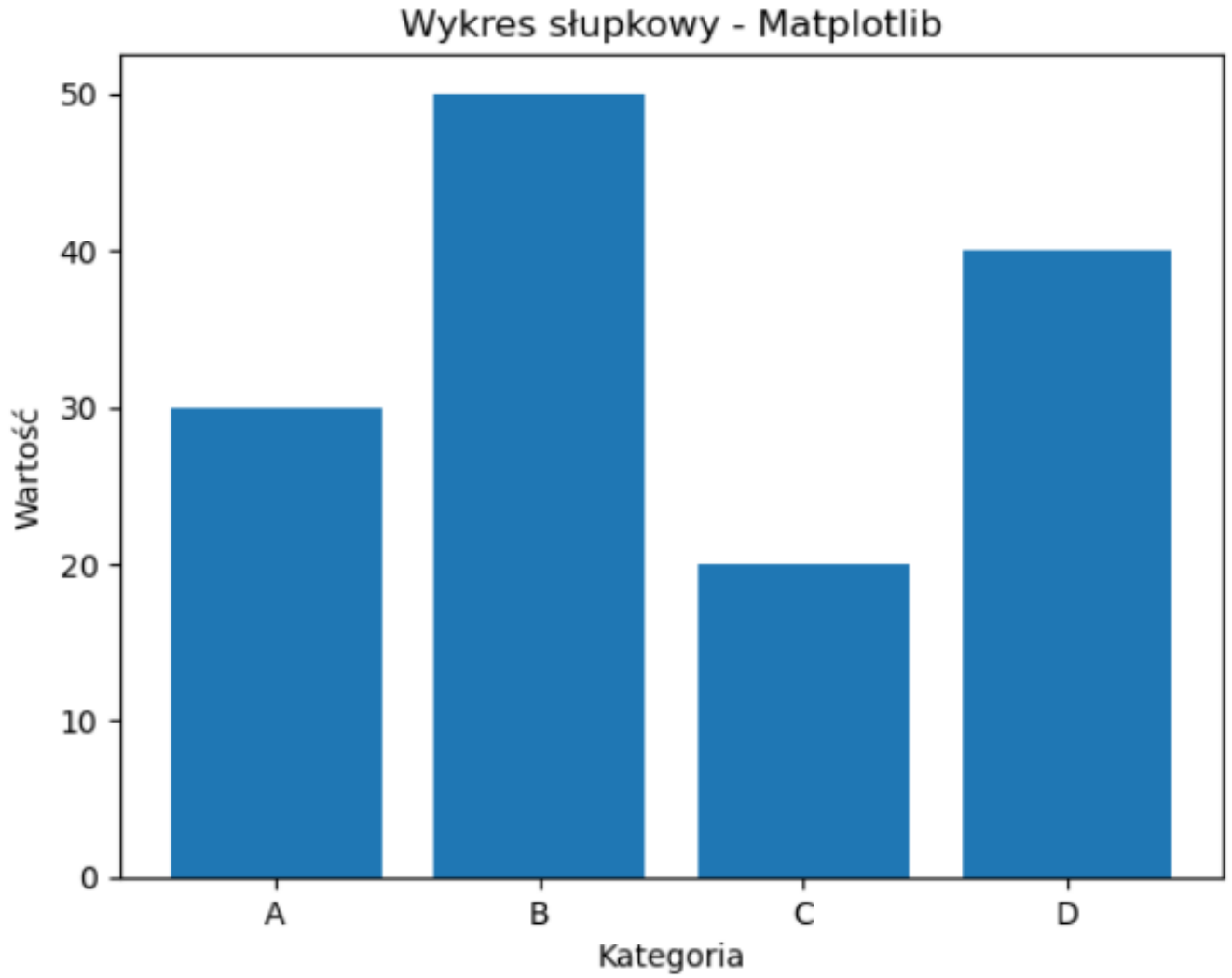
Każdy z tych rodzajów wykresów może być dostosowany do konkretnego celu analizy danych.

# Wizualizacja Danych z Matplotlib i Seaborn

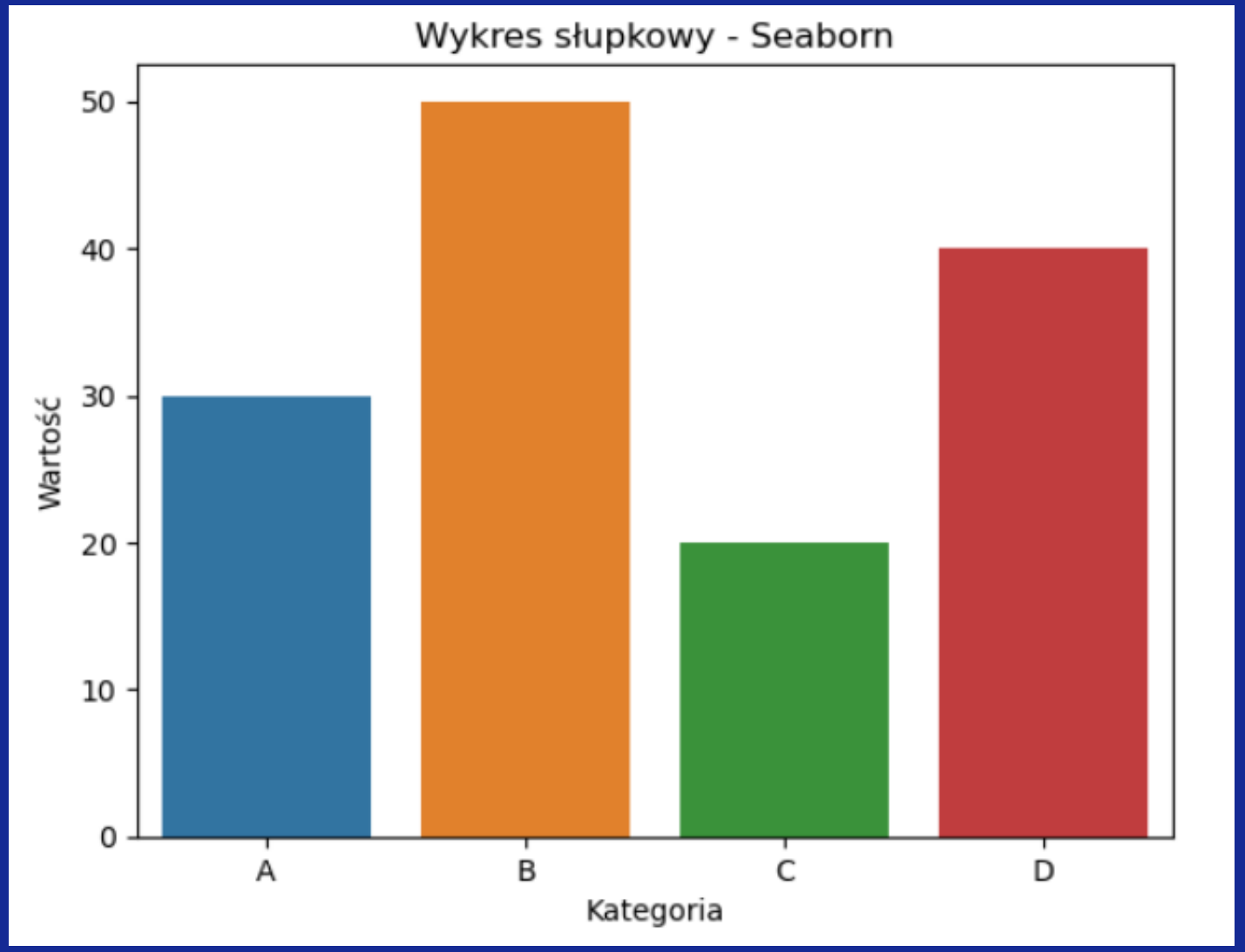
In [2]:

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

# Przykładowe dane
data = {'Category': ['A', 'B', 'C', 'D'],
        'Value': [30, 50, 20, 40]}

df = pd.DataFrame(data)

# Wykres słupkowy w Matplotlib
plt.bar(df['Category'], df['Value'])
plt.title('Wykres słupkowy - Matplotlib')
plt.xlabel('Kategoria')
plt.ylabel('Wartość')
plt.show()

# Wykres słupkowy w Seaborn
sns.barplot(x='Category', y='Value', data=df)
plt.title('Wykres słupkowy - Seaborn')
plt.xlabel('Kategoria')
plt.ylabel('Wartość')
plt.show()
```

# Wizualizacja Danych z Matplotlib i Seaborn



# Wizualizacja Danych z Matplotlib i Seaborn



# Wizualizacja Danych z Matplotlib i Seaborn

Przykład:

- Rozważmy dwa zestawy danych dotyczące temperatury w dwóch różnych miejscach (w stopniach Celsiusza):
- Zestaw 1: [20, 21, 22, 22, 23, 24, 25, 25, 25, 26, 26, 26, 27, 27, 27]
- Zestaw 2: [15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22]
- Zbadajmy podstawowe miary rozkładu dla obu zestawów danych, takie jak średnia, odchylenie standardowe, kurtoza itp.

Rozwiązania do ćwiczenia można przedstawić w formie kodu w Jupyter Notebook, aby lepiej zobaczyć, jak te miary są obliczane dla konkretnych danych.

# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie:

In [2]:
```
import numpy as np
import scipy.stats as stats

# Zestaw 1
zestaw_1 = [20, 21, 22, 22, 23, 24, 25, 25, 25, 26, 26, 26, 27, 27, 27]

# Zestaw 2
zestaw_2 = [15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22]

# Średnia arytmetyczna
srednia_z1 = np.mean(zestaw_1)
srednia_z2 = np.mean(zestaw_2)

# Mediana
mediana_z1 = np.median(zestaw_1)
mediana_z2 = np.median(zestaw_2)

# Odchylenie standardowe
std_z1 = np.std(zestaw_1)
std_z2 = np.std(zestaw_2)

# Kurtoza
kurtosis_z1 = stats.kurtosis(zestaw_1)
kurtosis_z2 = stats.kurtosis(zestaw_2)
```

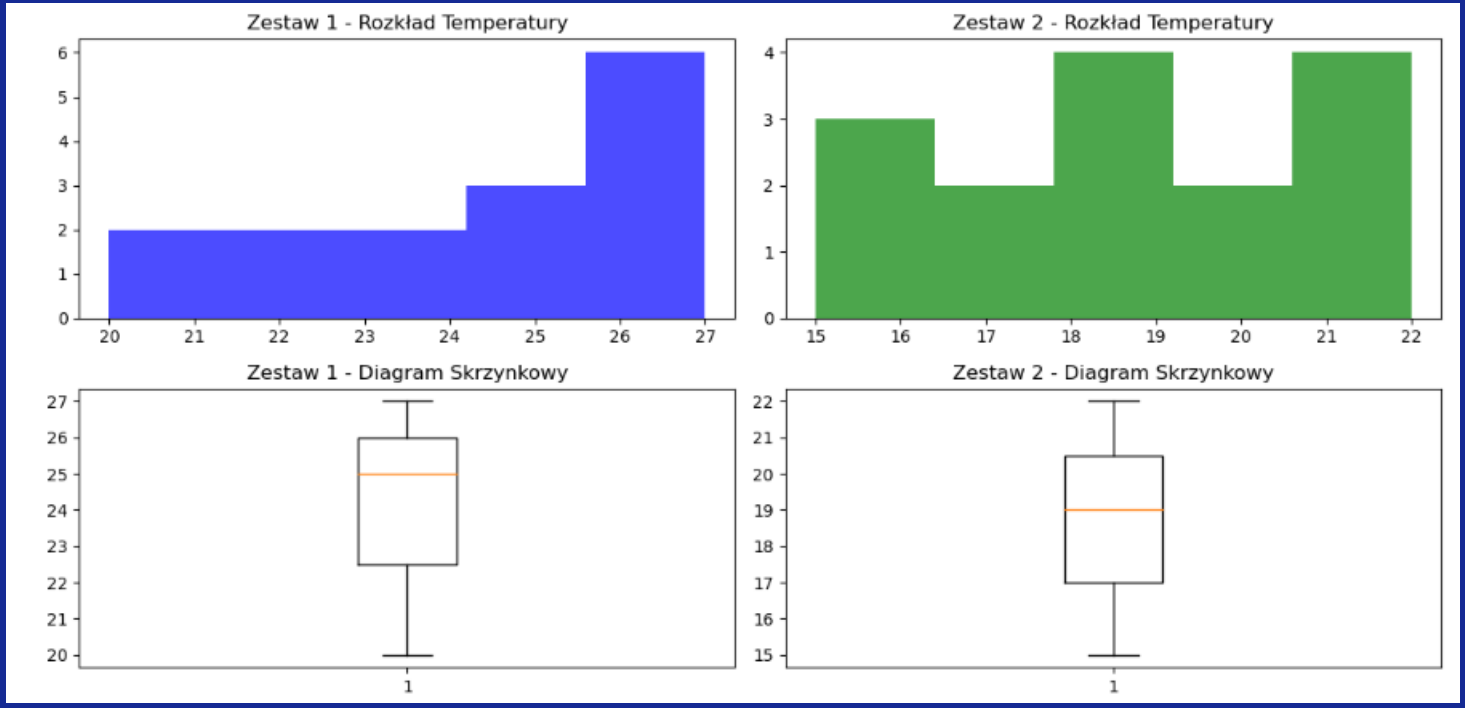
# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie - wizualizacja:

In [3]:
```
# Wizualizacja
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.hist(zestaw_1, bins=5, color='blue', alpha=0.7)
plt.title('Zestaw 1 - Rozkład Temperatury')

plt.subplot(2, 2, 2)
plt.hist(zestaw_2, bins=5, color='green', alpha=0.7)
plt.title('Zestaw 2 - Rozkład Temperatury')

plt.subplot(2, 2, 3)
plt.boxplot(zestaw_1)
plt.title('Zestaw 1 - Diagram Skrzynkowy')

plt.subplot(2, 2, 4)
plt.boxplot(zestaw_2)
plt.title('Zestaw 2 - Diagram Skrzynkowy')

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

# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie – wizualizacja (wykresy):



# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie - omówienie:

- W kodzie użyto dwóch głównych rodzajów wizualizacji danych: histogramów i diagramów skrzynkowych (boxplotów).

**Histogramy:**

1. Zestaw 1 - Rozkład Temperatury:
   - Histogram przedstawia rozkład wartości temperatury w zestawie 1.
   - Parametr bins=5 oznacza, że dane są podzielone na pięć przedziałów.
   - Wartości są reprezentowane na osi x, a liczność wystąpień w każdym przedziale na osi y.

2. Zestaw 2 - Rozkład Temperatury:
   - Analogicznie do zestawu 1, histogram przedstawia rozkład wartości temperatury w zestawie 2.

# Wizualizacja Danych z Matplotlib i Seaborn

**Diagramy Skrzynkowe (Boxplots):**

1. Zestaw 1 - Diagram Skrzynkowy:
   - Diagram skrzynkowy prezentuje rozkład wartości w zestawie 1.
   - Linia środkowa w pudełku to mediana, a górna i dolna krawędź pudełka oznaczają pierwszy i trzeci kwartyl.
   - "Wąsy" na końcach pudełka reprezentują zakres danych, a potencjalne wartości odstające są oznaczone punktami.

2. Zestaw 2 - Diagram Skrzynkowy:
   - Analogicznie do zestawu 1, diagram skrzynkowy prezentuje rozkład wartości w zestawie 2.

# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie - interpretacja:

- Histogramy pozwalają na wizualną ocenę rozkładu danych, identyfikację modów, a także ocenę symetrii i skośności rozkładu.
- Diagramy skrzynkowe pozwalają na szybkie zrozumienie centralnej tendencji (mediana) i rozproszenia danych, a także na identyfikację wartości odstających.

# Wizualizacja Danych z Matplotlib i Seaborn

Wnioski:

- Zestaw 1 wydaje się mieć bardziej jednostajny rozkład temperatur, z mniejszą zmiennością.
- Zestaw 2 ma bardziej skośny rozkład, z większym zakresem temperatur.

Wizualizacja danych pozwala szybko zrozumieć charakterystykę rozkładu, co jest istotne w analizie statystycznej.

# Wizualizacja Danych z Matplotlib i Seaborn

- Omówienie kodu wizualizacji

In [3]:
```
# Wizualizacja
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.hist(zestaw_1, bins=5, color='blue', alpha=0.7)
plt.title('Zestaw 1 - Rozkład Temperatury')
```

# Wizualizacja Danych z Matplotlib i Seaborn

Omówienie kodu wizualizacji

- Histogramy:
  - plt.figure(figsize=(12, 6)): Ustawienie rozmiaru całego wykresu.
  - plt.subplot(2, 2, 1): Tworzenie siatki 2x2 i ustawienie pierwszego subplotu.
  - plt.hist(zestaw_1, bins=5, color='blue', alpha=0.7): Tworzenie histogramu dla zestawu 1 z 5 przedziałami, niebieskim kolorem i 70% przezroczystością.
  - plt.title('Zestaw 1 - Rozkład Temperatury'): Dodanie tytułu do subplotu.

- Analogiczne kroki są powtarzane dla drugiego zestawu danych.

# Wizualizacja Danych z Matplotlib i Seaborn

Omówienie kodu wizualizacji

```
plt.subplot(2, 2, 3)
plt.boxplot(zestaw_1)
plt.title('Zestaw 1 - Diagram Skrzynkowy')
```

# Wizualizacja Danych z Matplotlib i Seaborn

**Diagramy Skrzynkowe:**

- plt.subplot(2, 2, 3): Ustawienie trzeciego subplotu.
- plt.boxplot(zestaw_1): Tworzenie diagramu skrzynkowego dla zestawu 1.
- plt.title('Zestaw 1 - Diagram Skrzynkowy'): Dodanie tytułu do subplotu.

Podobne kroki są powtarzane dla drugiego zestawu danych.

# Wizualizacja Danych z Matplotlib i Seaborn

Wnioski:

- Wizualizacja jest przygotowywana w formie siatki 2x2, gdzie górny wiersz zawiera histogramy, a dolny wiersz diagramy skrzynkowe.
- Każdy subplot jest tworzony przy użyciu funkcji `plt.subplot`, a następnie dodawane są odpowiednie wizualizacje danych.
- plt.tight_layout() pomaga w ułożeniu wykresów, aby uniknąć nakładania się elementów.

# Wizualizacja Danych z Matplotlib i Seaborn

**Diagramy Skrzynkowe:**

- plt.subplot(2, 2, 3): Ustawienie trzeciego subplotu.
- plt.boxplot(zestaw_1): Tworzenie diagramu skrzynkowego dla zestawu 1.
- plt.title('Zestaw 1 - Diagram Skrzynkowy'): Dodanie tytułu do subplotu.

Podobne kroki są powtarzane dla drugiego zestawu danych.

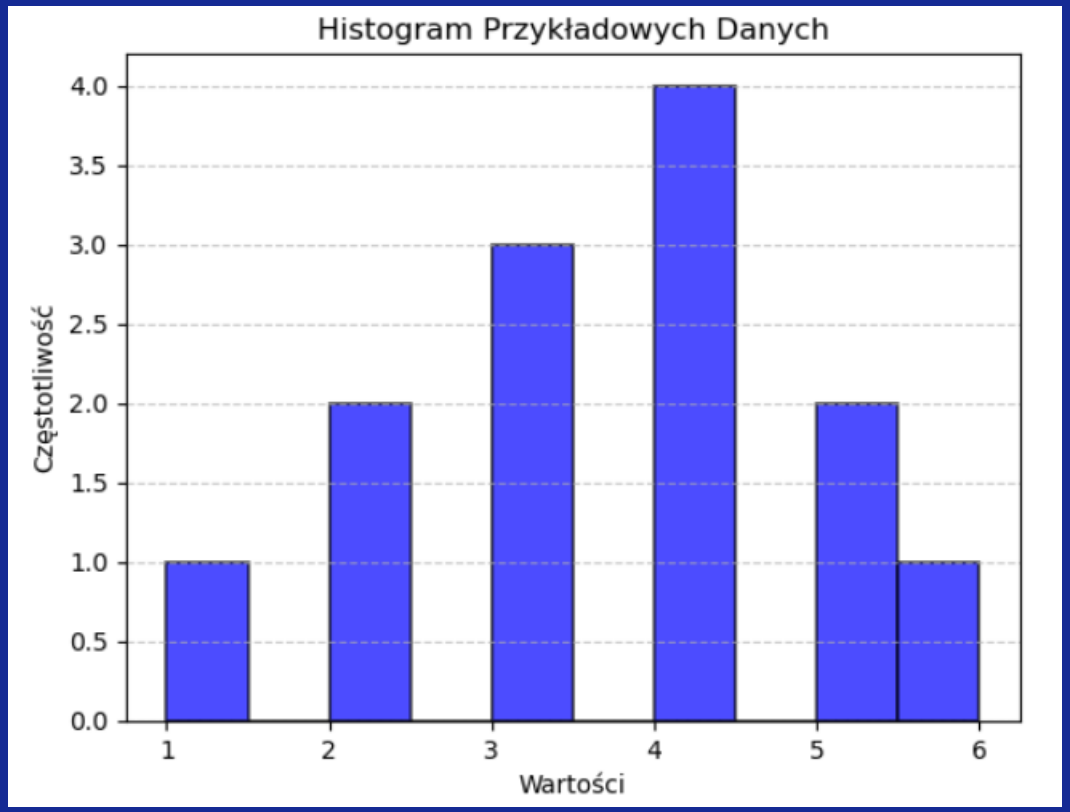
# Podstawy statystyki – podstawowe miary rozkładu

## Opis metody plt.hist()

In [5]:
```python
# Rozłożenie metody plt.hist()
import matplotlib.pyplot as plt

# Dane do stworzenia histogramu
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 6]

# Tworzenie histogramu
plt.hist(
    x=data,              # Dane wejściowe (lista lub tablica)
    bins=10,             # Liczba przedziałów (słupków)
    range=(1, 6),        # Zakres wartości, które chcemy uwzględnić
    density=False,       # Jeśli True, znormalizuje histogram do formy gęstości prawdopodobieństwa
    cumulative=False,    # Jeśli True, zwróci histogram kumulacyjny
    color='blue',        # Kolor histogramu
    alpha=0.7,           # Przezroczystość histogramu (0 - całkowicie przeźroczysty, 1 - całkowicie nieprzeźroczysty)
    edgecolor='black',   # Kolor krawędzi słupków
    linewidth=1.2,       # Grubość krawędzi słupków
    histtype='bar',      # Typ histogramu ('bar', 'barstacked', 'step', 'stepfilled')
    align='mid',         # Wyrównanie słupków ('left', 'mid', 'right')
    orientation='vertical' # Orientacja histogramu ('horizontal', 'vertical')
)

# Dodatkowe opcje do dostosowania wykresu
plt.title('Histogram Przykładowych Danych')
plt.xlabel('Wartości')
plt.ylabel('Częstotliwość')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Wyświetlenie histogramu
plt.show()
```

# Podstawy statystyki – podstawowe miary rozkładu

Wizualizacja przykładowych danych



# Wizualizacja Danych z Matplotlib i Seaborn

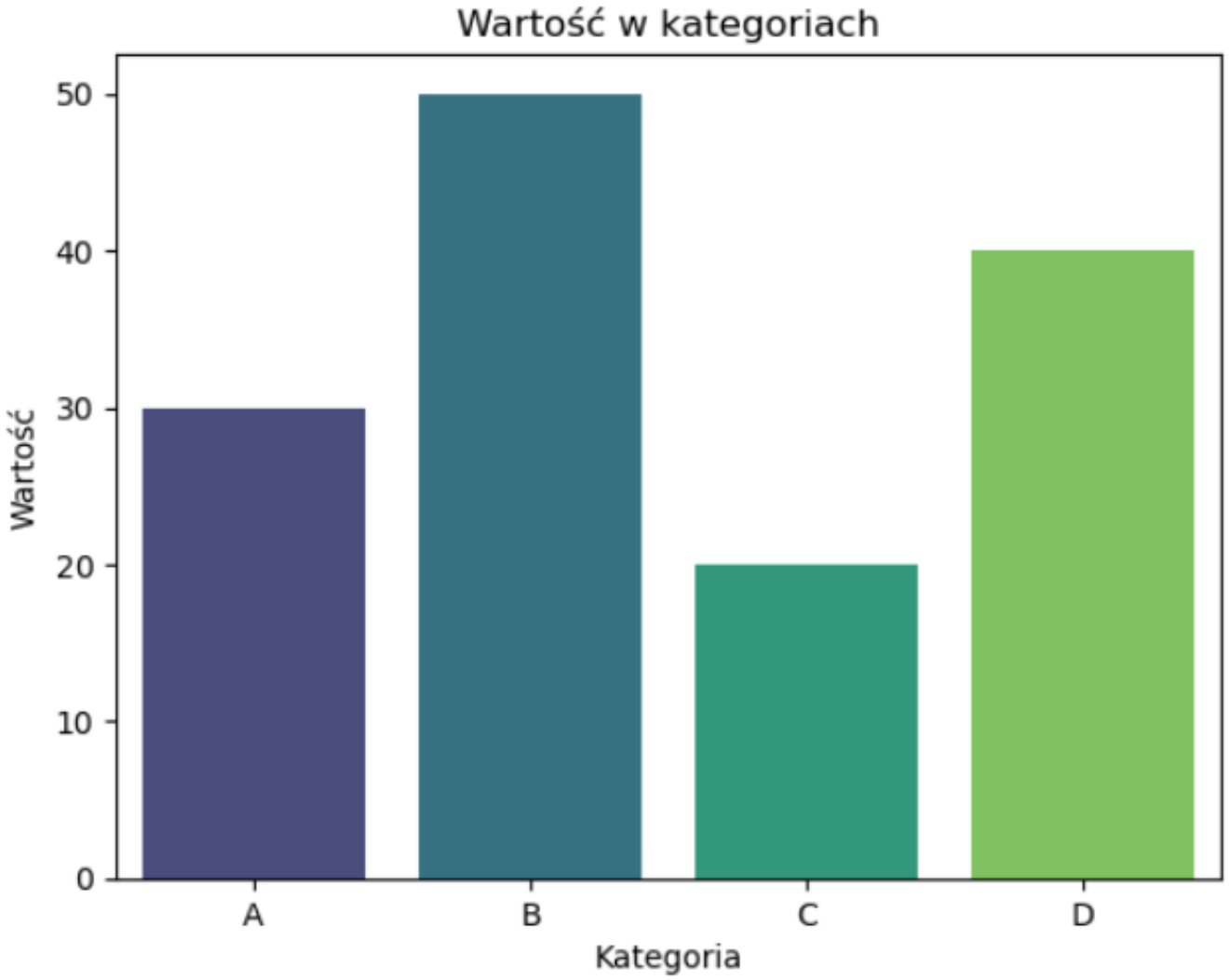
Elementy Storytelling w Wizualizacji:

Tworzenie wykresów nie polega tylko na przedstawieniu danych. Ważne jest także uwzględnienie elementów storytellingu, czyli takiego przedstawienia danych, które łatwo zrozumie każdy odbiorca.

# Wizualizacja Danych z Matplotlib i Seaborn

In [3]:
```
# Wykres z elementami storytellingu - Seaborn
sns.barplot(x='Category', y='Value', data=df, palette='viridis')
plt.title('Wartość w kategoriach')
plt.xlabel('Kategoria')
plt.ylabel('Wartość')
plt.show()
```

# Wizualizacja Danych z Matplotlib i Seaborn



# Wizualizacja Danych z Matplotlib i Seaborn

Storytelling w kontekście wizualizacji danych odnosi się do umiejętności opowiadania historii za pomocą wykresów i grafik.

Oto kilka elementów storytellingu, które można zauważyć w wykresie z punktu 2:

- Nagłówek zawierający informację: Tytuł "Wartość w kategoriach" informuje użytkownika o głównym temacie wykresu i o tym, co próbuje przedstawić.

- Kontekst i Cel: Wykres wskazuje, że ma przedstawić wartości w różnych kategoriach. Cel wykresu może być jasno określony przez opisanie, co oznaczają te wartości i dlaczego są ważne.

- Zrozumiałe i Przystępne Dla Odbiorcy: Wykres jest czytelny i łatwy do zrozumienia nawet dla osób, które nie są specjalistami w dziedzinie analizy danych. Każda kategoria jest oznaczona i jest jasne, jakie wartości są przedstawione.

# Wizualizacja Danych z Matplotlib i Seaborn

- Narracja poprzez wizualizację: Wykres sam w sobie stanowi część narracji. Dzięki klarownemu przedstawieniu danych użytkownik może zrozumieć trend lub zależności między różnymi kategoriami.

- Podkreślenie punktu centralnego: Wykres może podkreślać konkretne trendy, różnice lub podobieństwa między kategoriami. Może to być zrobione poprzez różne elementy wizualne, takie jak kolorystyka, wielkość, czy tekst dodany do wykresu.

- Inspirująca akcja lub refleksja: Wykres może inspirować do dalszej analizy danych lub działań, lub może prowokować refleksję na temat prezentowanych informacji.

Wszystkie te elementy pomagają w skutecznym storytellingu danych, czyli opowiadaniu historii, która jest zrozumiała, angażująca i inspirująca dla odbiorcy.

# Wizualizacja Danych z Matplotlib i Seaborn

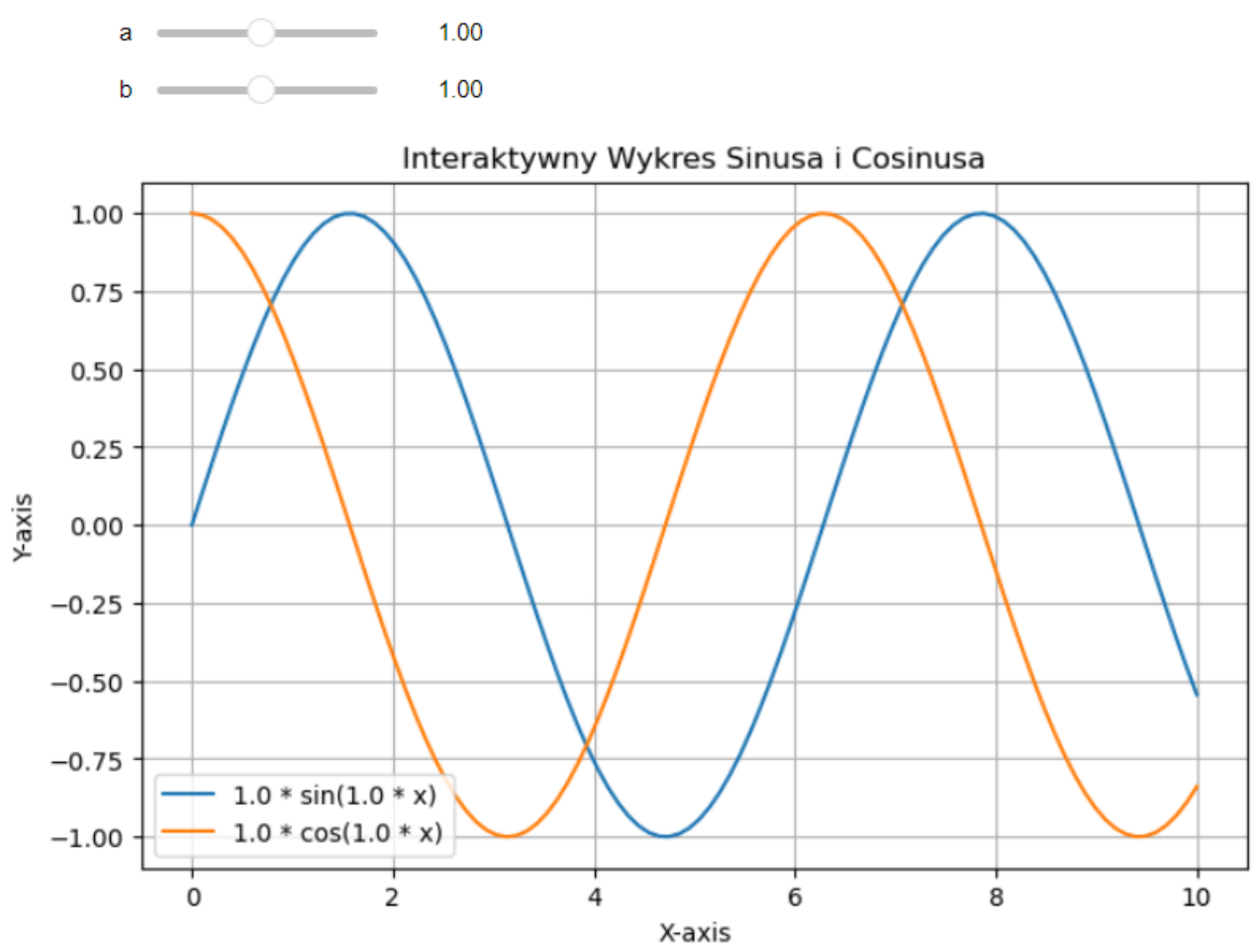
Przykład 1.

```
# Ćwiczenie: Tworzenie interaktywnych wykresów
import numpy as np

# Przykładowe dane
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Zadanie: Stwórz interaktywny wykres, na którym będą widoczne obie funkcje (sinus i cosinus)
plt.figure(figsize=(8, 5))
# Tu dodaj kod do utworzenia interaktywnego wykresu
plt.title('Interaktywny Wykres Sinusa i Cosinusa')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(['sin(x)', 'cos(x)'])
plt.grid(True)
plt.show()
```

# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie.

Out[5]:



# Wizualizacja Danych z Matplotlib i Seaborn

4. Przydatne Funkcje i Skróty:

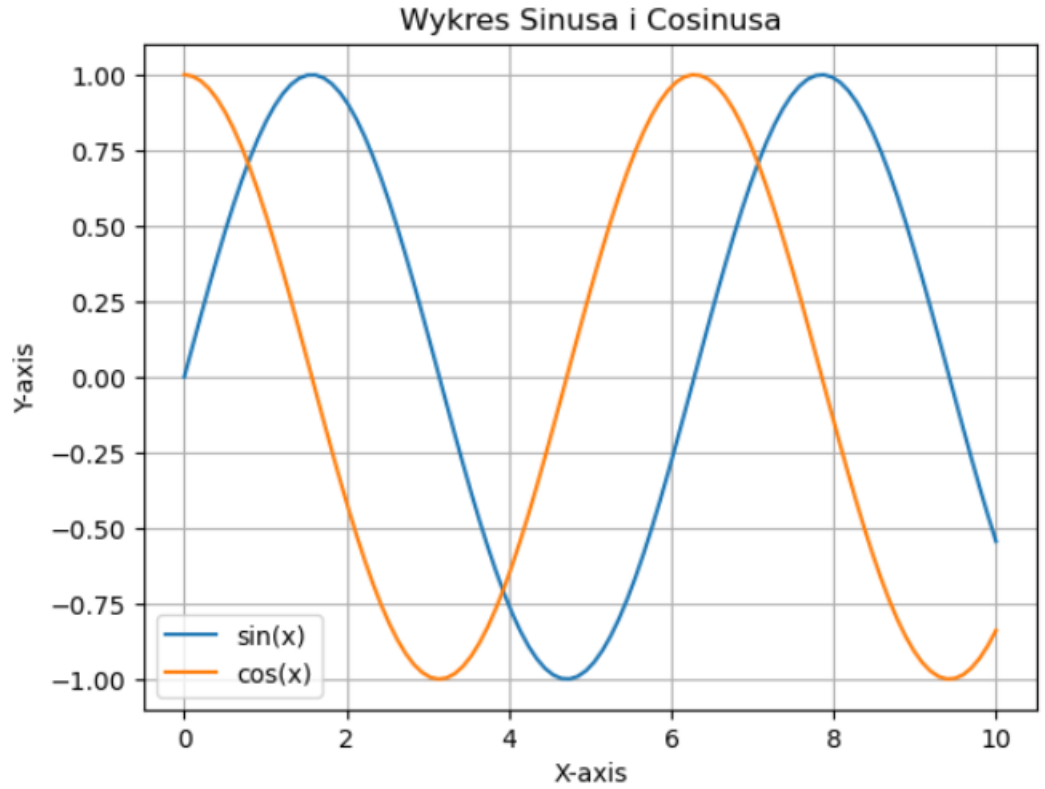
W Matplotlib i Seaborn istnieje wiele przydatnych funkcji i skrótów klawiszowych ułatwiających pracę. Na przykład:

- plt.title('Tytuł') - Dodanie tytułu do wykresu.
- plt.xlabel('Oś X') - Dodanie opisu osi X.
- plt.ylabel('Oś Y') - Dodanie opisu osi Y.
- plt.legend() - Dodanie legendy do wykresu.
- plt.grid(True) - Dodanie siatki.

# Wizualizacja Danych z Matplotlib i Seaborn

Przykład:

```
In [4]: # Przykład użycia przydatnych funkcji
plt.plot(x, y1, label='sin(x)')
plt.plot(x, y2, label='cos(x)')
plt.title('Wykres Sinusa i Cosinusa')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
plt.show()
```



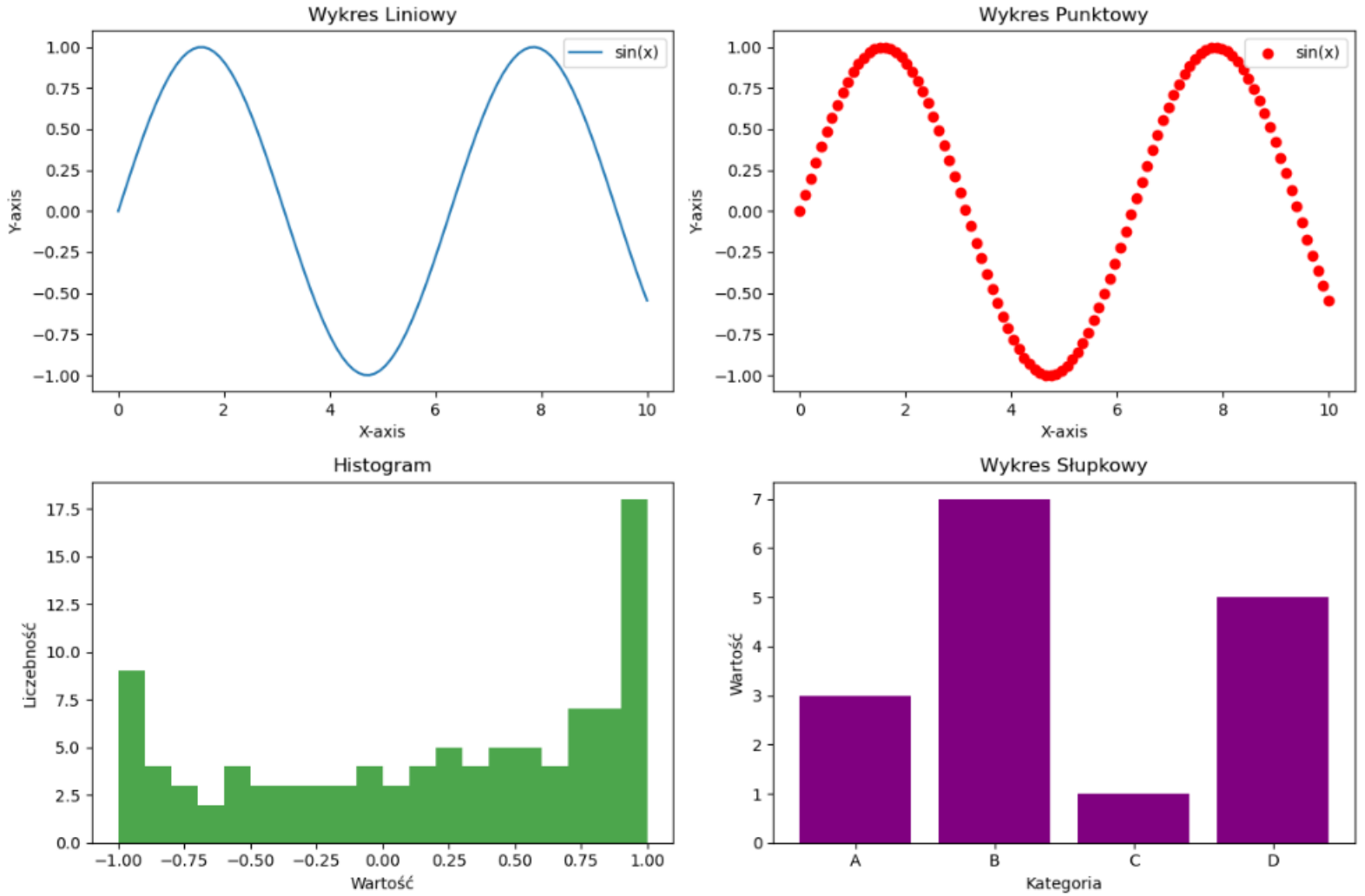
# Wizualizacja Danych z Matplotlib i Seaborn

Przykład 2.

```
# Ćwiczenie: Eksperymentowanie z różnymi wykresami
import numpy as np

# Przykładowe dane
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Zadanie: Stwórz różne wykresy na podstawie danych (np. wykres liniowy, punktowy, histogram)
plt.figure(figsize=(12, 8))
# Tu dodaj kod do utworzenia różnych wykresów
plt.title('Eksperymenty z Wykresami')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

# Wizualizacja Danych z Matplotlib i Seaborn

Rozwiązanie:



# Ankieta

Adres strony:

https://www.erp.comarch.pl/Szkolenia/Ankiety/survey/MDKZNM

COMARCH

# Ścieżka kształcenia

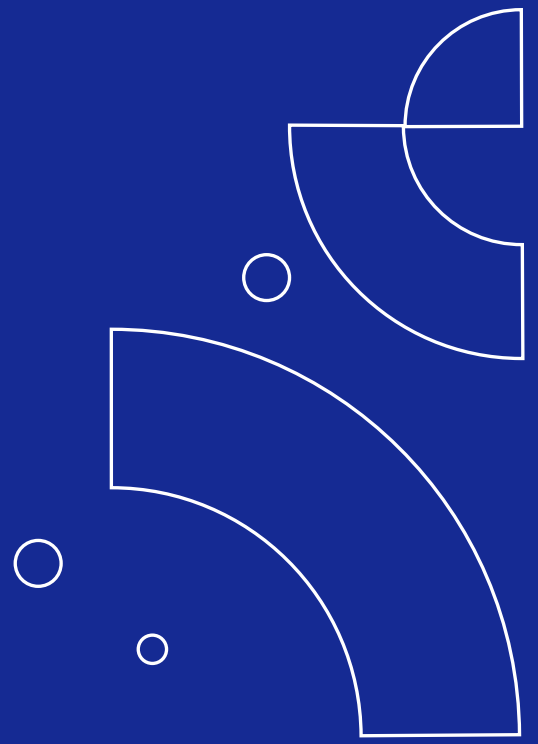
1. Wstęp do Machine Learning i Deep Learning w języku Python
   - https://www.comarch.pl/szkolenia/programowanie/python/wstep-do-machine-learning-i-deep-learning/
2. Wstęp do Machine Learning i Deep Learning w języku Python
   - https://www.comarch.pl/szkolenia/programowanie/python/wstep-do-machine-learning-i-deep-learning/
3. Machine Learning z użyciem języka Python. Zagadnienia zaawansowane
   - https://www.comarch.pl/szkolenia/programowanie/python/machine-learning-z-uzyciem-jezyka-python/

COMARCH
Szkolenia

# Literatura przedmiotu

1. Źródła z wykorzystaniem których została stworzona niniejsza prezentacja:
   - https://www.kaggle.com/ - zbiory danych
   - AUTOMATYZACJA NUDNYCH ZADAŃ Z PYTHONEM: Nauka programowania, Sweigart Al.
   - PROGRAMOWANIE W PYTHONIE DLA ŚREDNIO ZAAWANSOWANY: Najlepsze praktyki tworzenia czystego kodu, Sweigart Al.
   - https://www.python.org/dev/
   - https://www.geeksforgeeks.org/python-programming-language/
   - https://www.w3schools.com/python/python_intro.asp

2. Dalsza ścieżka kształcenia
   - W głównej mierze można skupić się na darmowych rozwiązaniach aby pogłębiać wiedzę.
   - Sololearn – interesująca platforma do zdobywania wiedzy
   - Udemy
   - Coursera
   - Comarch



COMARCH

COMARCH
Szkolenia

# Dziękujemy za udział w szkoleniu

Python w analizie danych.

Wstęp do Data Science

**Paweł Goleń**

Trener

# Centrum Szkoleniowe Comarch

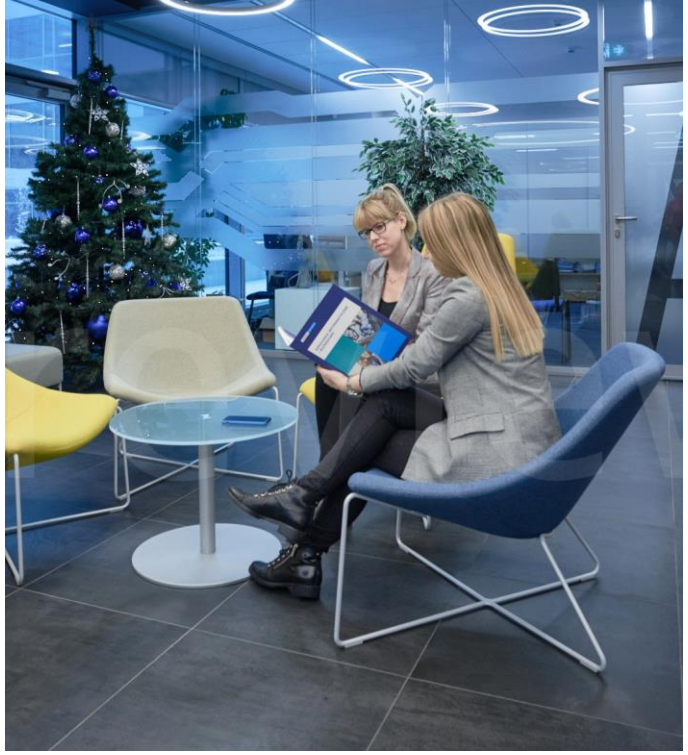
ul. Prof. M.Życzkowskiego 33

31-864 Kraków

Tel. +48 (12) 687 78 11

E-Mail: szkolenia@comarch.pl

www.szkolenia.comarch.pl




COMARCH
Szkolenia

COMARCH
Szkolenia

www.szkolenia.comarch.pl