



Wizualizacja Danych 2024

Piotr Jastrzębski

2024-04-27

Spis treści

1	Wizualizacja Danych 2024	5
2	Trochę teorii...	6
2.1	Test racjonalnego myślenia	6
2.2	Analiza danych - podstawowe pojęcia	6
2.2.1	Współczesne znaczenia słowa “statystyka”:	6
2.2.2	“Masowość”	7
2.2.3	Podział statystyki	7
2.2.4	Zbiorowość/populacja	7
2.2.5	Jednostka statyczna	7
2.2.6	Cechy statystyczne	8
2.2.7	Skale	9
2.3	Rodzaje badań statystycznych	11
2.4	Etapy badania statystycznego	11
2.5	Analiza danych zastanych	11
2.6	Proces analizy danych	12
2.6.1	Zdefiniowanie wymagań	12
2.6.2	Gromadzenie danych	13
2.6.3	Przetwarzanie danych	13
2.6.4	Właściwa analiza danych	13
2.6.5	Raportowanie i dystrybucja wyników	13
2.7	Skąd brać dane?	14
2.8	Koncepcja “Tidy data”	14
2.8.1	Zasady “czystych danych”	14
2.8.2	Przykłady nieuporządkowanych danych	15
2.8.3	Długie czy szerokie dane?	15
2.9	Parę rad na dobre prezentacje	15
2.9.1	Współczynnik kłamstwa	15
2.9.2	Współczynnik kłamstwa	16
2.10	Jak tworzyć?	17
2.11	Bibliografia	17
3	NumPy	18
3.1	Import biblioteki NumPy	18
3.2	Lista a tablica	19

3.3	Atrybuty tablic <code>ndarray</code>	20
3.4	Typy danych	21
3.5	Tworzenie tablic	22
3.6	Indeksowanie, “krojenie”	30
3.7	Modyfikacja kształtu i rozmiaru	34
3.8	Broadcasting	40
3.9	Funkcje uniwersalne	43
3.10	Statystyka i agregacja	43
3.11	Wyrażenia warunkowe	44
3.12	Działania na zbiorach	44
3.13	Operacje tablicowe	44
3.14	Alegbra liniowa	44
3.15	Funkcja na stringach	44
3.16	Data i czas	44
3.17	Pseudolosowe	45
4	NumPy - zadania	46
5	Pandas	47
5.1	Podstawowe byty	47
5.2	Uzupełnianie braków	53
5.3	Obsługa plików csv	54
5.4	Obsługa plików z Excela	54
5.5	Repozytorium z testowymi plikami	55
5.6	Operacje manipulacyjne	55
5.7	“Tidy data”	65
5.8	Obsługa brakujących danych	65
5.9	Usuwanie duplikatów	67
5.10	Zastępowanie wartościami	68
5.11	Dyskretyzacja i podział na koszyki	69
5.12	Wykrywanie i filtrowanie elementów odstających	71
5.13	Zmiana typu w kolumnie	72
5.14	Zmiana znaku kategoriach	74
6	Matplotlib	76
6.1	Galerie wykresów	77
6.2	Wykres liniowy	77
6.3	Parametry legendy	80
6.4	Style, kolory linii	81
6.5	Wykresy jako obiekty	83
6.6	Wykres liniowy i punktowy	84
6.7	Kolory	88
6.8	Mapy kolorów	90

6.9	Markery	95
6.10	Zapis do pliku	96
6.11	Linie poziome i pionowe	98
6.12	Adnotacje (tekst) na wykresie	100
6.13	Etykiety osi	102
6.14	Etykiety podziałki osi	104
6.15	Wykres kołowy	106
6.16	Podwykresy	111
6.17	Siatka	114
6.18	Wykres dwuosiowy	116
6.19	Wykres słupkowy	118
6.20	Wykres pudełkowy	129
6.21	Histogram	133
6.22	Wykres warstwowy	139
6.23	Wykres pierścieniowy	140
6.24	Wykresy w przestrzeni	141
6.24.1	Helisa	141
6.24.2	Torus	142
7	Seaborn	144
7.1	Ładowanie wbudowanych danych	145
7.2	Wykres punktowy	146
7.3	Wykres liniowy	150
7.4	Style	153

1 Wizualizacja Danych 2024

Materiały na semestr letni - rok akademicki 2023/24.

2 Trochę teorii...

2.1 Test racjonalnego myślenia

- Jeśli 5 maszyn w ciągu 5 minut produkuje 5 urządzeń, ile czasu zajmie 100 maszynom zrobienie 100 urządzeń?
- Na stawie rozrasta się kępa lilii wodnych. Codziennie kępa staje się dwukrotnie większa. Jeśli zarośnięcie całego stawu zajmie liliom 48 dni, to ile dni potrzeba, żeby zarosły połowę stawu?
- Kij bejsbolowy i piłka kosztują razem 1 dolar i 10 centów. Kij kosztuje o dolara więcej niż piłka. Ile kosztuje piłka?

Wizualizacja – ogólna nazwa graficznych metod tworzenia, analizy i przekazywania informacji. Za pomocą środków wizualnych ludzie wymieniają się zarówno ideami abstrakcyjnymi, jak i komunikatami mającymi bezpośrednie oparcie w rzeczywistości. W dzisiejszych czasach wizualizacja wpływa na sposób prowadzenia badań naukowych, jest rutynowo wykorzystywana w dyscyplinach technicznych i medycynie, służy celom dydaktycznym, a także bywa pojmowana jako środek wyrazu artystycznego.

2.2 Analiza danych - podstawowe pojęcia

2.2.1 Współczesne znaczenia słowa “statystyka”:

- zbiór danych liczbowych pokazujący kształtowanie procesów i zjawisk np. statystyka ludności.
- wszelkie czynności związane z gromadzeniem i opracowywaniem danych liczbowych np. statystyka pewnego problemu dokonywana przez GUS.
- charakterystyki liczbowe np. statystyki próby np. średnia arytmetyczna, odchylenie standardowe itp.
- dyscyplina naukowa - nauka o metodach badania zjawisk masowych.

2.2.2 “Masowość”

Zjawiska/procesy masowe - badaniu podlega duża liczba jednostek. Dzieli się na:

- gospodarcze (np. produkcja, konsumpcja, usługi reklama),
- społeczne (np. wypadki drogowe, poglądy polityczne),
- demograficzne (np. urodzenia, starzenie, migracje).

2.2.3 Podział statystyki

Statystyka - dyscyplina naukowa - podział:

- statystyka opisowa - zajmuje się sprawami związanymi z gromadzeniem, prezentacją, analizą i interpretacją danych liczbowych. Obserwacja obejmuje całą badaną zbiorowość.
- statystyka matematyczna - uogólnienie wyników badania części zbiorowości (próby) na całą zbiorowość.

2.2.4 Zbiorowość/populacja

Zbiorowość statystyczna, populacja statystyczna: zbiór obiektów podlegających badaniu statystycznemu. Tworzą je jednostki podobne do siebie, logicznie powiązane, lecz nie identyczne. Mają pewne cechy wspólne oraz pewne właściwości pozwalające je różnicować.

- przykłady:
 - badanie wzrostu Polaków - mieszkańcy Polski
 - poziom nauczania w szkołach woj. warmińsko-mazurskiego - szkoły woj. warmińsko-mazurskiego.
- podział:
 - zbiorowość/populacja generalna - obejmuje całość,
 - zbiorowość/populacja próbna (próba) - obejmuje część populacji.

2.2.5 Jednostka statyczna

Jednostka statystyczna: każdy z elementów zbiorowości statystycznej.

- przykłady:
 - studenci UWM - student UWM
 - mieszkańcy Polski - każda osoba mieszkająca w Polsce
 - maszyny produkowane w fabryce - każda maszyna

2.2.6 Cechy statystyczne

Cechy statystyczne

- właściwości charakteryzujące jednostki statystyczne w danej zbiorowości statystycznej.
- dzielimy je na stałe i zmienne.

Cechy stałe

- takie właściwości, które są wspólne wszystkim jednostkom danej zbiorowości statystycznej.
- podział:
 - rzeczowe - kto lub co jest przedmiotem badania statystycznego,
 - czasowe - kiedy zostało przeprowadzone badanie lub jakiego okresu czasu dotyczy badanie,
 - przestrzenne - jakiego terytorium (miejsce lub obszar) dotyczy badanie.
- przykład: studenci WMiI UWM w Olsztynie w roku akad. 2017/2018:
 - cecha rzeczowa: posiadanie legitymacji studenckiej,
 - cecha czasowa - studenci studiujący w roku akad. 2017/2018
 - cecha przestrzenna - miejsce: WMiI UWM w Olsztynie.

Cechy zmienne

- właściwości różnicujące jednostki statystyczne w danej zbiorowości.
- przykład: studenci UWM - cechy zmienne: wiek, płeć, rodzaj ukończonej szkoły średniej, kolor oczu, wzrost.

Ważne:

- obserwacji podlegają tylko cechy zmienne,
- cecha stała w jednej zbiorowości może być cechą zmienną w innej zbiorowości.

Przykład: studenci UWM mają legitymację wydaną przez UWM. Studenci wszystkich uczelni w Polsce mają legitymacje wydane przez różne szkoły.

Podział cech zmiennych:

- cechy mierzalne (ilościowe) - można je wyrazić liczbą wraz z określoną jednostką miary.
- cechy niemierzalne (jakościowe) - określane słownie, reprezentują pewne kategorie.

Przykład: zbiorowość studentów. Cechy mierzalne: wiek, waga, wzrost, liczba nieobecności. Cechy niemierzalne: płeć, kolor oczu, kierunek studiów.

Często ze względów praktycznych cechom niemierzalnym przypisywane są kody liczbowe. Nie należy ich jednak mylić z cechami mierzalnymi. Np. 1 - wykształcenie podstawowe, 2 - wykształcenie zasadnicze, itd...

Podział cech mierzalnych:

- ciągłe - mogące przybrać każdą wartość z określonego przedziału, np. wzrost, wiek, powierzchnia mieszkania.
- skokowe - mogące przyjmować konkretne (dyskretne) wartości liczbowe bez wartości pośrednich np. liczba osób w gospodarstwie domowych, liczba osób zatrudnionych w danej firmie.

Cechy skokowe zazwyczaj mają wartości całkowite choć nie zawsze jest to wymagane np. liczba etatów w firmie (z uwzględnieniem części etatów).

2.2.7 Skale

Skala pomiarowa

- to system, pozwalający w pewien sposób usystematyzować wyniki pomiarów statystycznych.
- podział:
 - skala nominalna,
 - skala porządkowa,
 - skala przedziałowa (interwałowa),
 - skala ilorazowa (stosunkowa).

Skala nominalna

- skala, w której klasyfikujemy jednostkę statystyczną do określonej kategorii.
- wartość w tej skali nie ma żadnego uporządkowania.
- przykład:

Religia	Kod
Chrześcijaństwo	1
Islam	2
Buddyzm	3

Skala porządkowa

- wartości mają jasno określony porządek, ale nie są dane odległości między nimi,
- pozwala na uszeregowanie elementów.
- przykłady:

Wykształcenie	Kod
Podstawowe	1
Średnie	2
Wyższe	3

Dochód	Kod
Niski	1
Średni	2
Wysoki	3

Skala przedziałowa (interwałowa)

- wartości cechy wyrażone są poprzez konkretne wartości liczbowe,
- pozwala na porównywanie jednostek (coś jest większe lub mniejsze),
- nie możliwe jest badanie ilorazów (określenie ile razy dana wartość jest większa lub mniejsza od drugiej).
- przykład:

Miasto	Temperatura w $^{\circ}C$	Temperatura w $^{\circ}F$
Warszawa	15	59
Olsztyn	10	50
Gdańsk	5	41
Szczecin	20	68

Skala ilorazowa (stosunkowa)

- wartości wyrażone są przez wartości liczbowe,
- możliwe określenie jest relacji mniejsza lub większa między wartościami,
- możliwe jest określenie stosunku (ilorazu) między wartościami,
- występuje zero absolutne.
- przykład:

Produkt	Cena w zł
Chleb	3
Masło	8
Gruszki	5

2.3 Rodzaje badań statystycznych

- badanie pełne - obejmują wszystkie jednostki zbiorowości statystycznej.
 - spis statystyczny,
 - rejestracja bieżąca,
 - sprawozdawczość statystyczna.
- badania częściowe - obserwowana jest część populacji. Przeprowadza się wtedy gdy badanie pełne jest niecelowe lub niemożliwe.
 - metoda monograficzna,
 - metoda reprezentacyjna.

2.4 Etapy badania statystycznego

- projektowanie i organizacja badania: ustalenie celu, podmiotu, przedmiotu, zakresu, źródła i czasu trwania badania;
- obserwacja statystyczna;
- opracowanie materiału statystycznego: kontrola materiału statystycznego, grupowanie uzyskanych danych, prezentacja wyników danych;
- analiza statystyczna.

2.5 Analiza danych zastanych

Analiza danych zastanych – proces przetwarzania danych w celu uzyskania na ich podstawie użytecznych informacji i wniosków. W zależności od rodzaju danych i stawianych problemów, może to oznaczać użycie metod statystycznych, eksploracyjnych i innych.

Korzystanie z danych zastanych jest przykładem badań niereaktywnych - metod badań zachowań społecznych, które nie wpływają na te zachowania. Dane takie to: dokumenty, archiwa, sprawozdania, kroniki, spisy ludności, księgi parafialne, dzienniki, pamiętniki, blogi internetowe, audio-pamiętniki, archiwa historii mówionej i inne. (Wikipedia)

Dane zastane możemy podzielić ze względu na (Makowska red. 2013):

- Charakter: Ilościowe, Jakościowe
- Formę: Dane opracowane, Dane surowe
- Sposób powstania: Pierwotne, Wtórne
- Dynamikę: Ciągła rejestracja zdarzeń, Rejestracja w interwałach czasowych, Rejestracja jednorazowa
- Poziom obiektywizmu: Obiektywne, Subiektywne
- Źródła pochodzenia: Dane publiczne, Dane prywatne

Analiza danych to proces polegający na sprawdzaniu, porządkowaniu, przekształcaniu i modelowaniu danych w celu zdobycia użytecznych informacji, wypracowania wniosków i wspierania procesu decyzyjnego. Analiza danych ma wiele aspektów i podejść, obejmujących różne techniki pod różnymi nazwami, w różnych obszarach biznesowych, naukowych i społecznych. Praktyczne podejście do definiowania danych polega na tym, że dane to liczby, znaki, obrazy lub inne metody zapisu, w formie, którą można ocenić w celu określenia lub podjęcia decyzji o konkretnym działaniu. Wiele osób uważa, że dane same w sobie nie mają znaczenia – dopiero dane przetworzone i zinterpretowane stają się informacją.

2.6 Proces analizy danych

Analiza odnosi się do rozbicia całości posiadanych informacji na jej odrębne komponenty w celu indywidualnego badania. Analiza danych to proces uzyskiwania nieprzetworzonych danych i przekształcania ich w informacje przydatne do podejmowania decyzji przez użytkowników. Dane są zbierane i analizowane, aby odpowiadać na pytania, testować hipotezy lub obalać teorie. Istnieje kilka faz, które można wyszczególnić w procesie analizy danych. Fazy są iteracyjne, ponieważ informacje zwrotne z faz kolejnych mogą spowodować dodatkową pracę w fazach wcześniejszych.

2.6.1 Zdefiniowanie wymagań

Przed przystąpieniem do analizy danych, należy dokładnie określić wymagania jakościowe dotyczące danych. Dane wejściowe, które mają być przedmiotem analizy, są określone na podstawie wymagań osób kierujących analizą lub klientów (którzy będą używać finalnego produktu analizy). Ogólny typ jednostki, na podstawie której dane będą zbierane, jest określany jako jednostka eksperymentalna (np. osoba lub populacja ludzi). Dane mogą być liczbowe lub kategoriowe (tj. Etykiety tekstowe). Faza definiowania wymagań powinna dać odpowiedź na 2 zasadnicze pytania:

- co chcemy zmierzyć?
- w jaki sposób chcemy to zmierzyć?

2.6.2 Gromadzenie danych

Dane są gromadzone z różnych źródeł. Wymogi, co do rodzaju i jakości danych mogą być przekazywane przez analityków do “opiekunów danych”, takich jak personel technologii informacyjnych w organizacji. Dane ponadto mogą być również gromadzone automatycznie z różnego rodzaju czujników znajdujących się w otoczeniu - takich jak kamery drogowe, satelity, urządzenia rejestrujące obraz, dźwięk oraz parametry fizyczne. Kolejną metodą jest również pozyskiwanie danych w drodze wywiadów, gromadzenie ze źródeł internetowych lub bezpośrednio z dokumentacji.

2.6.3 Przetwarzanie danych

Zgromadzone dane muszą zostać przetworzone lub zorganizowane w sposób logiczny do analizy. Na przykład, mogą one zostać umieszczone w tabelach w celu dalszej analizy - w arkuszu kalkulacyjnym lub innym oprogramowaniu. Oczyszczanie danych Po fazie przetworzenia i uporządkowania, dane mogą być niekompletne, zawierać duplikaty lub zawierać błędy. Konieczność czyszczenia danych wynika z problemów związanych z wprowadzaniem i przechowywaniem danych. Czyszczenie danych to proces zapobiegania powstawaniu i korygowania wykrytych błędów. Typowe zadania obejmują dopasowywanie rekordów, identyfikowanie nieścisłości, ogólny przegląd jakości istniejących danych, usuwanie duplikatów i segmentację kolumn. Niezwykle istotne jest też zwracanie uwagi na dane których wartości są powyżej lub poniżej ustalonych wcześniej progów (ekstrema).

2.6.4 Właściwa analiza danych

Istnieje kilka metod, które można wykorzystać do tego celu, na przykład data mining, business intelligence, wizualizacja danych lub badania eksploracyjne. Ta ostatnia metoda jest sposobem analizowania zbiorów informacji w celu określenia ich odrębnych cech. W ten sposób dane mogą zostać wykorzystane do przetestowania pierwotnej hipotezy. Statystyki opisowe to kolejna metoda analizy zebranych informacji. Dane są badane, aby znaleźć najważniejsze ich cechy. W statystykach opisowych analitycy używają kilku podstawowych narzędzi - można użyć średniej lub średniej z zestawu liczb. Pomaga to określić ogólny trend aczkolwiek nie zapewnia to dużej dokładności przy ocenie ogólnego obrazu zebranych danych. W tej fazie ma miejsce również modelowanie i tworzenie formuł matematycznych - stosowane są w celu identyfikacji zależności między zmiennymi, takich jak korelacja lub przyczynowość.

2.6.5 Raportowanie i dystrybucja wyników

Ta faza polega na ustalaniu w jakiej formie przekazywać wyniki. Analityk może rozważyć różne techniki wizualizacji danych, aby w sposób wyraźnym i skuteczny przekazać wnioski z

analizy odbiorcom. Wizualizacja danych wykorzystuje formy graficzne jak wykresy i tabele. Tabele są przydatne dla użytkownika, który może wyszukiwać konkretne rekordy, podczas gdy wykresy (np. wykresy słupkowe lub liniowe) dają spojrzenie ilościowych na zbiór analizowanych danych.

2.7 Skąd brać dane?

Darmowa repozytoria danych:

- Bank danych lokalnych GUS - [link](#)
- Otwarte dane - [link](#)
- Bank Światowy - [link](#)

Przydatne strony:

- <https://www.kaggle.com/>
- <https://archive.ics.uci.edu/ml/index.php>

2.8 Koncepcja “Tidy data”

Koncepcja czyszczenia danych (ang. tidy data):

- WICKHAM, Hadley . Tidy Data. Journal of Statistical Software, [S.l.], v. 59, Issue 10, p. 1 - 23, sep. 2014. ISSN 1548-7660. Available at: <https://www.jstatsoft.org/v059/i10>. Date accessed: 25 oct. 2018. doi:<http://dx.doi.org/10.18637/jss.v059.i10>.

2.8.1 Zasady “czystych danych”

Idealne dane są zaprezentowane w tabeli:

Imię	Wiek	Wzrost	Kolor oczu
Adam	26	167	Brązowe
Sylwia	34	164	Piwnie
Tomasz	42	183	Niebieskie

Na co powinniśmy zwrócić uwagę?

- jedna obserwacja (jednostka statystyczna) = jeden wiersz w tabeli/macierzy/ramce danych

- wartości danej cechy znajdują się w kolumnach
- jeden typ/rodzaj obserwacji w jednej tabeli/macierzy/ramce danych

2.8.2 Przykłady nieuporządkowanych danych

Imię	Wiek	Wzrost	Brązowe	Niebieskie	Piwne
Adam	26	167	1	0	0
Sylwia	34	164	0	0	1
Tomasz	42	183	0	1	0

Nagłówki kolumn muszą odpowiadać cechom, a nie wartościom zmiennych.

2.8.3 Długie czy szerokie dane?

https://seaborn.pydata.org/tutorial/data_structure.html#long-form-vs-wide-form-data

2.9 Parę rad na dobre prezentacje

Edward Tufte, prof z Yale, <https://www.edwardtufte.com/>

1. Prezentuj dane “na bogato”.
2. Nie ukrywaj danych, pokazuj prawdę.
3. Nie używaj wykresów śmieciowych.
4. Pokazuj zmienność danych, a nie projektuj jej.
5. Wykres ma posiadać jak najmniejszy współczynnik kłamstwa (lie-factor).
6. Powerpoint to zło!

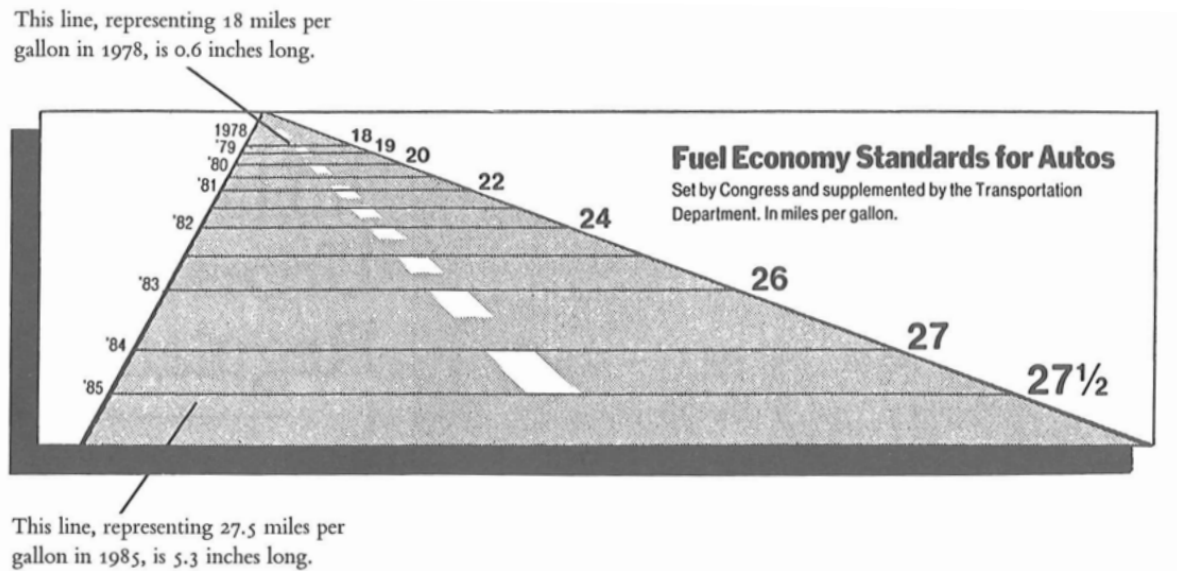
2.9.1 Współczynnik kłamstwa

https://www.facebook.com/janinadaily/photos/a.1524649467770881/2836063543296127/?pav=0&eav=AfbVIDx5un8ZOklKI9c-B1jP4nOoNa2QMmJmjoA-291JNNgM1L_NmoCGMS_mJOy4xjo&_rdr

- stosunek efektu widocznego na wykresie do efektu wykazywanego przez dane, na podstawie których ten wykres narysowaliśmy.

https://infovis-wiki.net/wiki/Lie_Factor

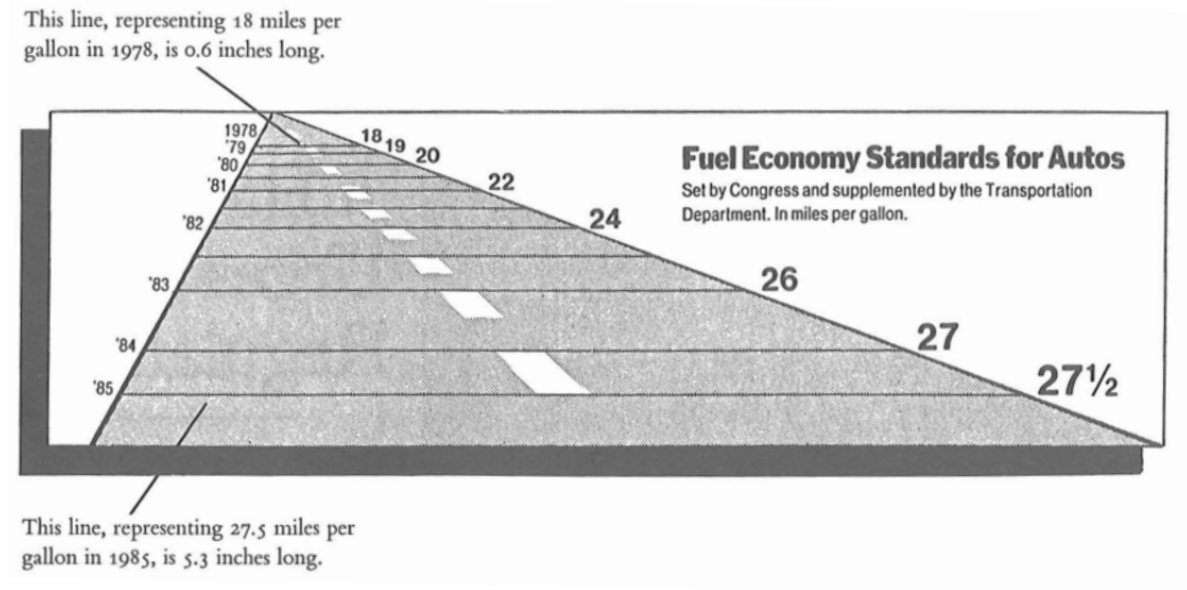
2.9.2 Współczynnik kłamstwa



[Tufte, 1991] Edward Tufte, The Visual Display of Quantitative Information, Second Edition, Graphics Press, USA, 1991, p. 57 – 69.

$$\text{LieFactor} = \frac{\text{rozmiar efektu widocznego na wykresie}}{\text{rozmiar efektu wynikającego z danych}}$$

$$\text{rozmiar efektu} = \frac{|\text{druga wartość} - \text{pierwsza wartość}|}{\text{pierwsza wartość}}$$



$$\text{LieFactor} = \frac{\frac{5.3-0.6}{0.6}}{\frac{27.5-18}{18}} \approx 14.8$$

2.10 Jak tworzyć?

- https://bookdown.org/rudolf_von_ems/jak_sie_nie_dac/stats_graphs.html
- <https://www.data-to-viz.com/>
- <https://100.datavizproject.com/>

2.11 Bibliografia

- <https://pl.wikipedia.org/wiki/Wizualizacja>
- https://mfiles.pl/pl/index.php/Analiza_danych, dostęp online 1.04.2019.
- Walesiak M., Gatnar E., Statystyczna analiza danych z wykorzystaniem programu R, PWN, Warszawa, 2009.
- Wasilewska E., Statystyka opisowa od podstaw, Podręcznik z zadaniami, Wydawnictwo SGGW, Warszawa, 2009.
- https://en.wikipedia.org/wiki/Cognitive_reflection_test, dostęp online 20.03.2023.
- <https://qlikblog.pl/edward-tufte-dobre-praktyki-prezentacji-danych/>, dostęp online 20.03.2023.

3 NumPy

NumPy jest biblioteką Pythona służącą do obliczeń naukowych.

Zastosowania:

- algebra liniowa
- zaawansowane obliczenia matematyczne (numeryczne)
- całkowania
- rozwiązywanie równań
- ...

3.1 Import biblioteki NumPy

```
import numpy as np
```

Podstawowym bytem w bibliotece NumPy jest N-wymiarowa tablica zwana `ndarray`. Każdy element na tablicy traktowany jest jako typ `dtype`.

```
numpy.array(object, dtype=None, *, copy=True, order='K', subok=False, ndmin=0, like=None)
```

- `object` - to co ma być wrzucone do tablicy
- `dtype` - typ
- `copy` - czy obiekty mają być skopiowane, domyślne `True`
- `order` - sposób układania: C (rzędy), F (kolumny), A, K
- `subok` - realizowane przez podklasy (jeśli `True`), domyślnie `False`
- `ndmin` - minimalny rozmiar (wymiar) tablicy
- `like` - tworzenie na podstawie tablic referencyjnej

```
import numpy as np
```

```
a = np.array([1, 2, 3])
```

```
print("a:", a)
```

```
print("typ a:", type(a))
```

①

②

```

b = np.array([1, 2, 3.0]) ③
print("b:", b)
c = np.array([[1, 2], [3, 4]]) ④
print("c:", c)
d = np.array([1, 2, 3], ndmin=2) ⑤
print("d:", d)
e = np.array([1, 2, 3], dtype=complex) ⑥
print("e:", e)
f = np.array(np.mat('1 2; 3 4')) ⑦
print("f:", f)
g = np.array(np.mat('1 2; 3 4'), subok=True) ⑧
print("g:", g)
print(type(g))

```

- ① Standardowe domyślne.
- ② Sprawdzenie typu.
- ③ Jeden z elementów jest innego typu. Tu następuje zatem rozszerzenie do typu “największego”.
- ④ Tu otrzymamy tablicę 2x2.
- ⑤ W tej linijce otrzymana będzie tablica 2x1.
- ⑥ Ustalenie innego typu - większego.
- ⑦ Skorzystanie z podtypu macierzowego.
- ⑧ Zachowanie typu macierzowego.

```

a: [1 2 3]
typ a: <class 'numpy.ndarray'>
b: [1. 2. 3.]
c: [[1 2]
     [3 4]]
d: [[1 2 3]]
e: [1.+0.j 2.+0.j 3.+0.j]
f: [[1 2]
     [3 4]]
g: [[1 2]
     [3 4]]
<class 'numpy.matrix'>

```

3.2 Lista a tablica

```

import numpy as np
import time

start_time = time.time()
my_arr = np.arange(1000000)
my_list = list(range(1000000))
start_time = time.time()
my_arr2 = my_arr * 2
print("--- %s seconds ---" % (time.time() - start_time))
start_time = time.time()
my_list2 = [x * 2 for x in my_list]
print("--- %s seconds ---" % (time.time() - start_time))

```

```

--- 0.0020003318786621094 seconds ---
--- 0.09847068786621094 seconds ---

```

3.3 Atrybuty tablic ndarray

Atrybut	Opis
shape	krotka z informacją liczbę elementów dla każdego z wymiarów
size	liczba elementów w tablicy (łącznie)
ndim	liczba wymiarów tablicy
nbytes	liczba bajtów jaką tablica zajmuje w pamięci
dtype	typ danych

<https://numpy.org/doc/stable/reference/arrays.ndarray.html#array-attributes>

```

import numpy as np

tab1 = np.array([2, -3, 4, -8, 1])
print("typ:", type(tab1))
print("shape:", tab1.shape)
print("size:", tab1.size)
print("ndim:", tab1.ndim)
print("nbytes:", tab1.nbytes)
print("dtype:", tab1.dtype)

```

```
typ: <class 'numpy.ndarray'>
shape: (5,)
size: 5
ndim: 1
nbytes: 20
dtype: int32
```

```
import numpy as np

tab2 = np.array([[2, -3], [4, -8]])
print("typ:", type(tab2))
print("shape:", tab2.shape)
print("size:", tab2.size)
print("ndim:", tab2.ndim)
print("nbytes:", tab2.nbytes)
print("dtype:", tab2.dtype)
```

```
typ: <class 'numpy.ndarray'>
shape: (2, 2)
size: 4
ndim: 2
nbytes: 16
dtype: int32
```

NumPy nie wspiera postrzępionych tablic! Poniższy kod wygeneruje błąd:

```
import numpy as np

tab3 = np.array([[2, -3], [4, -8, 5], [3]])
```

3.4 Typy danych

<https://numpy.org/doc/stable/reference/arrays.scalars.html>

<https://numpy.org/doc/stable/reference/arrays.dtypes.html#arrays-dtypes-constructing>

Typy całkowitoliczbowe	int,int8,int16,int32,int64
Typy całkowitoliczbowe (bez znaku)	uint,uint8,uint16,uint32,uint64
Typ logiczny	bool

Typy zmiennoprzecinkowe	float, float16, float32, float64, float128
Typy zmiennoprzecinkowe zespolone	complex, complex64, complex128, complex256
Napis	str

```
import numpy as np

tab = np.array([[2, -3], [4, -8]])
print(tab)
tab2 = np.array([[2, -3], [4, -8]], dtype=int)
print(tab2)
tab3 = np.array([[2, -3], [4, -8]], dtype=float)
print(tab3)
tab4 = np.array([[2, -3], [4, -8]], dtype=complex)
print(tab4)
```

```
[[ 2 -3]
 [ 4 -8]]
[[ 2 -3]
 [ 4 -8]]
[[ 2. -3.]
 [ 4. -8.]]
[[ 2.+0.j -3.+0.j]
 [ 4.+0.j -8.+0.j]]
```

3.5 Tworzenie tablic

`np.array` - argumenty rzutowany na tablicę (coś po czym można iterować) - warto sprawdzić rozmiar/kształt

```
import numpy as np

tab = np.array([2, -3, 4])
print(tab)
print("size:", tab.size)
tab2 = np.array((4, -3, 3, 2))
print(tab2)
print("size:", tab2.size)
```

```

tab3 = np.array({3, 3, 2, 5, 2})
print(tab3)
print("size:", tab3.size)
tab4 = np.array({"pl": 344, "en": 22})
print(tab4)
print("size:", tab4.size)

```

```

[ 2 -3  4]
size: 3
[ 4 -3  3  2]
size: 4
{2, 3, 5}
size: 1
{'pl': 344, 'en': 22}
size: 1

```

`np.zeros` - tworzy tablicę wypełnioną zerami

```

import numpy as np

tab = np.zeros(4)
print(tab)
print(tab.shape)
tab2 = np.zeros([2, 3])
print(tab2)
print(tab2.shape)
tab3 = np.zeros([2, 3, 4])
print(tab3)
print(tab3.shape)

```

```

[0.  0.  0.  0.]
(4,)
[[0.  0.  0.]
 [0.  0.  0.]]
(2, 3)
[[[0.  0.  0.  0.]
   [0.  0.  0.  0.]
   [0.  0.  0.  0.]]

 [[0.  0.  0.  0.]
   [0.  0.  0.  0.]
   [0.  0.  0.  0.]]

```

```
[0. 0. 0. 0.]]]
(2, 3, 4)
```

`np.ones` - tworzy tablicę wypełnioną jedynkami (to nie odpowiednik macierzy jednostkowej!)

```
import numpy as np

tab = np.ones(4)
print(tab)
print(tab.shape)
tab2 = np.ones([2, 3])
print(tab2)
print(tab2.shape)
tab3 = np.ones([2, 3, 4])
print(tab3)
print(tab3.shape)
```

```
[1. 1. 1. 1.]
(4,)
[[1. 1. 1.]
 [1. 1. 1.]]
(2, 3)
[[[1. 1. 1. 1.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]]
 (2, 3, 4)]
```

`np.diag` - tworzy tablicę odpowiadającą macierzy diagonalnej

```
import numpy as np

print("tab0")
tab0 = np.diag([3, 4, 5])
print(tab0)
print("tab1")
tab1 = np.array([[2, 3, 4], [3, -4, 5], [3, 4, -5]])
print(tab1)
```



```

tab2 = np.diag(tab1)
print("tab2")
print(tab2)
tab3 = np.diag(tab1, k=1)
print("tab3")
print(tab3)
print("tab4")
tab4 = np.diag(tab1, k=-2)
print(tab4)
print("tab5")
tab5 = np.diag(np.diag(tab1))
print(tab5)

```

```

tab0
[[3 0 0]
 [0 4 0]
 [0 0 5]]
tab1
[[ 2  3  4]
 [ 3 -4  5]
 [ 3  4 -5]]
tab2
[ 2 -4 -5]
tab3
[3 5]
tab4
[3]
tab5
[[ 2  0  0]
 [ 0 -4  0]
 [ 0  0 -5]]

```

`np.arange` - tablica wypełniona równomiernymi wartościami

Składnia: `numpy.arange([start,]stop, [step,]dtype=None)`

Zasada działania jest podobna jak w funkcji `range`, ale dopuszczamy liczby “z ułamkiem”.

```

import numpy as np

a = np.arange(3)
print(a)

```

```

b = np.arange(3.0)
print(b)
c = np.arange(3, 7)
print(c)
d = np.arange(3, 11, 2)
print(d)
e = np.arange(0, 1, 0.1)
print(e)
f = np.arange(3, 11, 2, dtype=float)
print(f)
g = np.arange(3, 10, 2)
print(g)

```

```

[0 1 2]
[0. 1. 2.]
[3 4 5 6]
[3 5 7 9]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
[3. 5. 7. 9.]
[3 5 7 9]

```

`np.linspace` - tablica wypełniona równomiernymi wartościami wg skali liniowej

```

import numpy as np

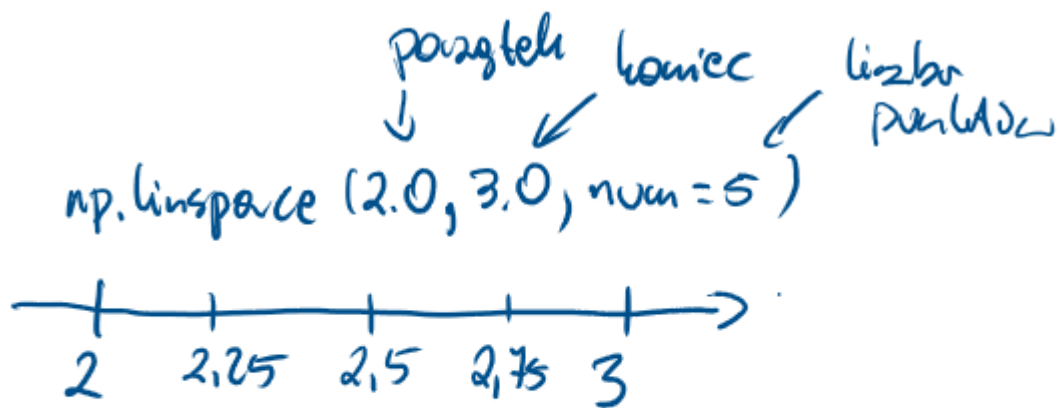
a = np.linspace(2.0, 3.0, num=5)
print(a)
b = np.linspace(2.0, 3.0, num=5, endpoint=False)
print(b)
c = np.linspace(10, 20, num=4)
print(c)
d = np.linspace(10, 20, num=4, dtype=int)
print(d)

```

```

[2.   2.25 2.5   2.75 3.   ]
[2.   2.2 2.4 2.6 2.8]
[10.          13.33333333 16.66666667 20.          ]
[10 13 16 20]

```



Wzrost: odcinek jest dzielony
na **num-1** części!

endpoint = False

- wyłącza ostatni punkt
(z prawej strony)

Wtedy podział odbywa się
na **num** części.

dtype ← ustala typ

zwykle używa się int
(wyniki mają uciętą część
ułamkową)

np.logspace - tablica wypełniona wartościami wg skali logarytmicznej

Składnia: `numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None,`

axis=0)

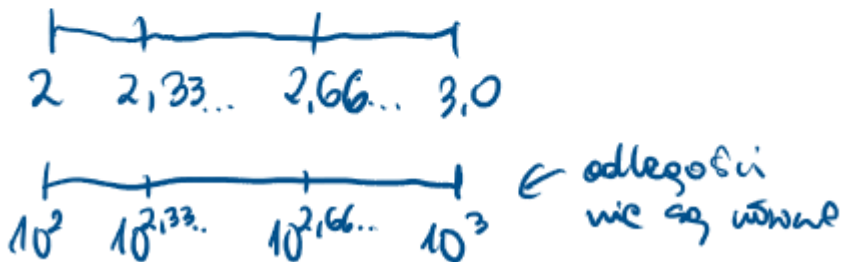
```
import numpy as np

a = np.logspace(2.0, 3.0, num=4)
print(a)
b = np.logspace(2.0, 3.0, num=4, endpoint=False)
print(b)
c = np.logspace(2.0, 3.0, num=4, base=2.0)
print(c)
```

```
[ 100.          215.443469   464.15888336 1000.          ]
[100.          177.827941   316.22776602 562.34132519]
[4.           5.0396842   6.34960421 8.           ]
```

np.logspace(2.0, 3.0, num=4)

Domyślnie podstawa logarytmu 10



`np.empty` - pusta (niezainicjowana) tablica - konkretne wartości nie są “gwarantowane”

```
import numpy as np

a = np.empty(3)
print(a)
b = np.empty(3, dtype=int)
print(b)
```

```
[0. 1. 2.]  
[0 1 2]
```

`np.identity` - tablica przypominająca macierz jednostkową

`np.eye` - tablica z jedynkami na przekątnej (pozostałe zera)

```
import numpy as np  
  
print("a")  
a = np.identity(4)  
print(a)  
print("b")  
b = np.eye(4, k=1)  
print(b)  
print("c")  
c = np.eye(4, k=2)  
print(c)  
print("d")  
d = np.eye(4, k=-1)  
print(d)
```

```
a  
[[1. 0. 0. 0.]  
 [0. 1. 0. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 0. 1.]]  
b  
[[0. 1. 0. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 0. 1.]  
 [0. 0. 0. 0.]]  
c  
[[0. 0. 1. 0.]  
 [0. 0. 0. 1.]  
 [0. 0. 0. 0.]  
 [0. 0. 0. 0.]]  
d  
[[0. 0. 0. 0.]  
 [1. 0. 0. 0.]  
 [0. 1. 0. 0.]  
 [0. 0. 1. 0.]]
```

3.6 Indeksowanie, “krojenie”

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 16, 1])
print("1:", a[5])
print("2:", a[-2])
print("3:", a[3:6])
print("4:", a[:])
print("5:", a[0:-1])
print("6:", a[:5])
```

①
②
③
④
⑤
⑥

- ① Dostęp do elementu o indeksie 5.
- ② Dostęp do elementu drugiego od tyłu.
- ③ Dostęp do elementów o indeksach od 3 do 5 (włącznie) - zasada przedziałów lewostronnie domkniętych, prawostronnie otwartych.
- ④ Dostęp do wszystkich elementów.
- ⑤ Dostęp do wszystkich elementów z wyłączeniem ostatniego.
- ⑥ Dostęp od początku do elementu o indeksie 4.

```
1: 8
2: 16
3: [ 4 -7  8]
4: [  2  5 -2  4 -7  8  9 11 -23 -4 -7 16  1]
5: [  2  5 -2  4 -7  8  9 11 -23 -4 -7 16]
6: [ 2  5 -2  4 -7]
```

```
import numpy as np

print("1:", a[4:])
print("2:", a[4:-1])
print("3:", a[4:10:2])
print("4:", a[::-1])
print("5:", a[:2])
print("6:", a[::-2])
```

①
②
③
④
⑤
⑥

- ① Dostęp do elementów od indeksu 4 do końca.
- ② Dostęp do elementów od indeksu 4 do końca bez ostatniego.
- ③ Dostęp do elementów o indeksach stanowiących ciąg arytmetyczny od 4 do 10 (z czówrką, ale bez dziesiątki) z krokiem równym 2

- ④ Dostęp do elementów od tyłu do początku.
- ⑤ Dostęp do elementów o indeksach parzystych od początku.
- ⑥ Dostęp do elementów o indeksach “nieparzystych ujemnych” od początku.

```
1: [ -7   8   9  11 -23  -4  -7  16   1]
2: [ -7   8   9  11 -23  -4  -7  16]
3: [ -7   9 -23]
4: [  1  16  -7  -4 -23  11   9   8  -7   4  -2   5   2]
5: [  2  -2  -7   9 -23  -7   1]
6: [  1  -7 -23   9  -7  -2   2]
```

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = a[:2, 1:]
print(b)
print(np.shape(b))
c = a[1]
print(c)
print(np.shape(c))
d = a[1, :]
print(d)
print(np.shape(d))
```

```
[[4 5]
 [4 8]]
(2, 2)
[-3  4  8]
(3,)
[-3  4  8]
(3,)
```

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
e = a[1:2, :]
print(e)
print(np.shape(e))
f = a[:, :2]
print(f)
print(np.shape(f))
```

```

g = a[1, :2]
print(g)
print(np.shape(g))
h = a[1:2, :2]
print(h)
print(np.shape(h))

```

```

[[-3  4  8]]
(1, 3)
[[ 3  4]
 [-3  4]
 [ 3  2]]
(3, 2)
[-3  4]
(2,)
[[-3  4]]
(1, 2)

```

**Uwaga - takie “krojenie” to tzw “widok”.

```

import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = a[1:2, 1:]
print(b)
a[1][1] = 9
print(a)
print(b)
b[0][0] = -11
print(a)
print(b)

```

```

[[4 8]]
[[ 3  4  5]
 [-3  9  8]
 [ 3  2  9]]
[[9 8]]
[[ 3  4  5]
 [-3 -11  8]
 [ 3  2  9]]
[[-11  8]]

```


Naprawa:

```
import numpy as np

a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
b = a[1:2, 1:].copy()
print(b)
a[1][1] = 9
print(a)
print(b)
b[0][0] = -11
print(a)
print(b)
```

```
[[4 8]]
[[ 3  4  5]
 [-3  9  8]
 [ 3  2  9]]
[[4 8]]
[[ 3  4  5]
 [-3  9  8]
 [ 3  2  9]]
[[-11  8]]
```

Indeksowanie logiczne (fancy indexing)

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
b = a[np.array([1, 3, 7])]
print(b)
c = a[[1, 3, 7]]
print(c)
```

```
[ 5  4 11]
[ 5  4 11]
```

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
b = a > 0
```

```
print(b)
c = a[a > 0]
print(c)
```

```
[ True  True False  True False  True  True  True False False False  True
  True]
[ 2  5  4  8  9 11  8  1]
```

```
import numpy as np

a = np.array([2, 5, -2, 4, -7, 8, 9, 11, -23, -4, -7, 8, 1])
b = a[a > 0]
print(b)
b[0] = -5
print(a)
print(b)
a[1] = 20
print(a)
print(b)
```

```
[ 2  5  4  8  9 11  8  1]
[ 2  5 -2  4 -7  8  9 11 -23 -4 -7  8  1]
[-5  5  4  8  9 11  8  1]
[ 2 20 -2  4 -7  8  9 11 -23 -4 -7  8  1]
[-5  5  4  8  9 11  8  1]
```

3.7 Modyfikacja kształtu i rozmiaru

<https://numpy.org/doc/stable/reference/routines.array-manipulation.html>

```
import numpy as np

print("a")
a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
print(a)
print("b")
b = np.reshape(a, (1, 9))
print(b)
print("c")
```

```
c = a.reshape(9)
print(c)
```

```
a
[[ 3  4  5]
 [-3  4  8]
 [ 3  2  9]]
b
[[ 3  4  5 -3  4  8  3  2  9]]
c
[ 3  4  5 -3  4  8  3  2  9]
```

```
import numpy as np

print("a")
a = np.array([[3, 4, 5], [-3, 4, 8], [3, 2, 9]])
print(a)
print("d")
d = a.flatten()
print(d)
print("e")
e = a.ravel()
print(e)
print("f")
f = np.ravel(a)
print(f)
```

```
a
[[ 3  4  5]
 [-3  4  8]
 [ 3  2  9]]
d
[ 3  4  5 -3  4  8  3  2  9]
e
[ 3  4  5 -3  4  8  3  2  9]
f
[ 3  4  5 -3  4  8  3  2  9]
```

```
import numpy as np
```

```

print("g")
g = [[1, 3, 4]]
print(g)
print("h")
h = np.squeeze(g)
print(h)
print("i")
i = a.T
print(i)
print("j")
j = np.transpose(a)
print(j)

```

```

g
[[1, 3, 4]]
h
[1 3 4]
i
[[ 3 -3  3]
 [ 4  4  2]
 [ 5  8  9]]
j
[[ 3 -3  3]
 [ 4  4  2]
 [ 5  8  9]]

```

```

import numpy as np

print("h")
h = [3, -4, 5, -2]
print(h)
print("k")
k = np.hstack((h, h, h))
print(k)
print("l")
l = np.vstack((h, h, h))
print(l)
print("m")
m = np.dstack((h, h, h))
print(m)

```

```

h

```

```

[3, -4, 5, -2]
k
[ 3 -4  5 -2  3 -4  5 -2  3 -4  5 -2]
l
[[ 3 -4  5 -2]
 [ 3 -4  5 -2]
 [ 3 -4  5 -2]]
m
[[[ 3  3  3]
  [-4 -4 -4]
  [ 5  5  5]
  [-2 -2 -2]]]

```

```

import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
print("r1")
r1 = np.concatenate((a, b))
print(r1)
print("r2")
r2 = np.concatenate((a, b), axis=0)
print(r2)
print("r3")
r3 = np.concatenate((a, b.T), axis=1)
print(r3)
print("r4")
r4 = np.concatenate((a, b), axis=None)
print(r4)

```

```

r1
[[1 2]
 [3 4]
 [5 6]]
r2
[[1 2]
 [3 4]
 [5 6]]
r3
[[1 2 5]
 [3 4 6]]
r4

```

```
[1 2 3 4 5 6]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
print("r1")
r1 = np.resize(a, (2, 3))
print(r1)
print("r2")
r2 = np.resize(a, (1, 4))
print(r2)
print("r3")
r3 = np.resize(a, (2, 4))
print(r3)
```

```
r1
[[1 2 3]
 [4 1 2]]
r2
[[1 2 3 4]]
r3
[[1 2 3 4]
 [1 2 3 4]]
```

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
print("r1")
r1 = np.append(a, b)
print(r1)
print("r2")
r2 = np.append(a, b, axis=0)
print(r2)
```

```
r1
[1 2 3 4 5 6]
r2
[[1 2]
 [3 4]
 [5 6]]
```

```
import numpy as np

a = np.array([[1, 2], [3, 7]])
print("r1")
r1 = np.insert(a, 1, 4)
print(r1)
print("r2")
r2 = np.insert(a, 2, 4)
print(r2)
print("r3")
r3 = np.insert(a, 1, 4, axis=0)
print(r3)
print("r4")
r4 = np.insert(a, 1, 4, axis=1)
print(r4)
```

```
r1
[1 4 2 3 7]
r2
[1 2 4 3 7]
r3
[[1 2]
 [4 4]
 [3 7]]
r4
[[1 4 2]
 [3 4 7]]
```

```
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print("r1")
r1 = np.delete(a, 1, axis=1)
print(r1)
print("r2")
r2 = np.delete(a, 2, axis=0)
print(r2)
```

```
r1
[[ 1  3  4]
 [ 5  7  8]]
```

```
[ 9 11 12]]
r2
[[1 2 3 4]
 [5 6 7 8]]
```

3.8 Broadcasting

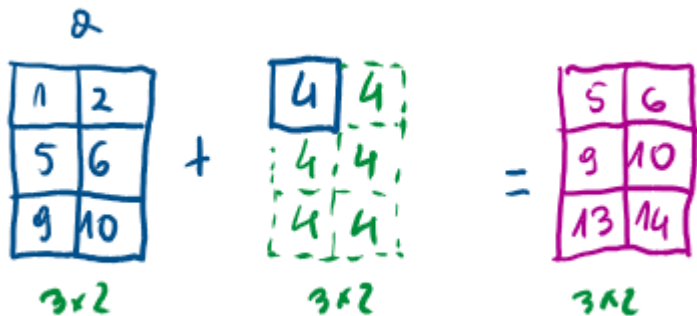
Rozważane warianty są przykładowe.

Wariant 1 - skalar-tablica - wykonanie operacji na każdym elemencie tablicy

```
import numpy as np

a = np.array([[1, 2], [5, 6], [9, 10]])
b = a + 4
print(b)
c = 2 ** a
print(c)
```

```
[[ 5  6]
 [ 9 10]
 [13 14]]
[[  2   4]
 [ 32  64]
 [512 1024]]
```



Wariant 2 - dwie tablice - “gdy jedna z tablic może być rozszerzona” (oba wymiary są równe lub jeden z nich jest równy 1)

<https://numpy.org/doc/stable/user/basics.broadcasting.html>


```

import numpy as np

a = np.array([[1, 2], [5, 6]])
b = np.array([9, 2])
r1 = a + b
print(r1)
r2 = a / b
print(r2)
c = np.array([[4], [-2]])
r3 = a + c
print(r3)
r4 = c / a
print(r4)

```

```

[[10  4]
 [14  8]]
[[0.11111111 1.          ]
 [0.55555556 3.          ]]
[[5 6]
 [3 4]]
[[ 4.          2.          ]
 [-0.4        -0.33333333]]

```

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 6 \\ \hline \end{array}
 \quad 2 \times 2
 \quad + \quad
 \begin{array}{|c|c|} \hline 8 & 2 \\ \hline 9 & 2 \\ \hline \end{array}
 \quad 2 \times 1
 \quad = \quad
 \begin{array}{|c|c|} \hline 10 & 4 \\ \hline 14 & 8 \\ \hline \end{array}
 \quad 2 \times 2$$

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 6 \\ \hline \end{array}
 \quad 2 \times 2
 \quad + \quad
 \begin{array}{|c|c|} \hline 4 & 4 \\ \hline -2 & -2 \\ \hline \end{array}
 \quad 2 \times 1
 \quad = \quad
 \begin{array}{|c|c|} \hline 5 & 6 \\ \hline 3 & 4 \\ \hline \end{array}
 \quad 2 \times 2$$

Wariant 3 - "kolumna" i "wiersz"

```
import numpy as np

a = np.array([[5, 2, -3]]).T
b = np.array([3, -2, 1, 2, 4])
print(a+b)
print(b+a)
print(a*b)
```

```
[[ 8  3  6  7  9]
 [ 5  0  3  4  6]
 [ 0 -5 -2 -1  1]]
[[ 8  3  6  7  9]
 [ 5  0  3  4  6]
 [ 0 -5 -2 -1  1]]
[[ 15 -10  5  10  20]
```

```
[ 6 -4  2  4  8]
[-9  6 -3 -6 -12]]
```

Diagram illustrating matrix multiplication:

Matrix a (3x1) is multiplied by Matrix b (1x5) to result in a 3x5 matrix.

Matrix a (3x1):

5
2
-3

Matrix b (1x5):

3	-2	1	2	4
---	----	---	---	---

Resulting matrix (3x5):

8	3	6	7	9
5	0	3	4	6
0	-5	-2	-1	1

3.9 Funkcje uniwersalne

<https://numpy.org/doc/stable/reference/ufuncs.html#methods>

3.10 Statystyka i agregacja

Funkcja	Opis
np.mean	Średnia wszystkich wartości w tablicy.
np.std	Odchylenie standardowe.
np.var	Wariancja.
np.sum	Suma wszystkich elementów.
np.prod	Iloczyn wszystkich elementów.
np.cumsum	Skumulowana suma wszystkich elementów.
np.cumprod	Skumulowany iloczyn wszystkich elementów.
np.min, np.max	Minimalna/maksymalna wartość w tablicy.
np.argmin, np.argmax	Indeks minimalnej/maksymalnej wartości w tablicy.
np.all	Sprawdza czy wszystkie elementy są różne od zera.
np.any	Sprawdza czy co najmniej jeden z elementów jest różny od zera.

3.11 Wyrażenia warunkowe

<https://numpy.org/doc/stable/reference/generated/numpy.where> <https://numpy.org/doc/stable/reference/generated/numpy.choose> <https://numpy.org/doc/stable/reference/generated/numpy.select> <https://numpy.org/doc/stable/reference/generated/numpy.nonzero>

3.12 Działania na zbiorach

<https://numpy.org/doc/stable/reference/routines.set.html>

3.13 Operacje tablicowe

<https://numpy.org/doc/stable/reference/generated/numpy.transpose>

<https://numpy.org/doc/stable/reference/generated/numpy.flip> <https://numpy.org/doc/stable/reference/generated/numpy.fliplr> <https://numpy.org/doc/stable/reference/generated/numpy.flipud>

<https://numpy.org/doc/stable/reference/generated/numpy.sort>

3.14 Algebra liniowa

<https://numpy.org/doc/stable/reference/routines.linalg.html>

3.15 Funkcja na stringach

<https://numpy.org/doc/stable/reference/routines.char.html>

3.16 Data i czas

<https://numpy.org/doc/stable/reference/arrays.datetime.html>

3.17 Pseudolosowe

<https://numpy.org/doc/stable/reference/random/index.html>

Bibliografia:

- Dokumentacja biblioteki, <https://numpy.org/doc/stable/>, dostęp online 5.03.2021.
- Robert Jahansson, Matematyczny Python. Obliczenia naukowe i analiza danych z użyciem NumPy, SciPy i Matplotlib, Wyd. Helion, 2021.
- <https://www.tutorialspoint.com/numpy/index.htm>, dostęp online 20.03.2019.

4 NumPy - zadania

1. Utwórz tablicę NumPy o wymiarach 3x2, a następnie zmień jej kształt na 2x3 bez zmiany danych.
2. Dla danej tablicy NumPy zawierającej co najmniej 10 elementów, wykonaj indeksowanie, aby uzyskać trzeci element, a następnie “krojenie”, aby uzyskać elementy od trzeciego do szóstego.
3. Utwórz tablicę zawierającą 10 równo rozmieszczonych punktów między 0 a 100. Następnie, wykorzystując utworzoną tablicę, oblicz wartości funkcji kwadratowej $y = x^2$ dla każdego punktu. Wyniki zapisz w nowej tablicy.
4. Wygeneruj tablicę zawierającą 20 punktów równomiernie rozłożonych w zakresie od π do 2π i użyj tej tablicy do obliczenia i wyświetlenia sinusa dla każdego punktu. Wyniki zapisz w osobnej tablicy.
5. Stwórz tablicę składającą się z 15 punktów równomiernie rozłożonych między -5 a 5. Następnie, na podstawie tej tablicy, utwórz dwie nowe tablice: jedną zawierającą wartości funkcji eksponencjalnej e^x dla każdego z punktów, a drugą zawierającą logarytm naturalny dla tych punktów, gdzie punkty równoznaczne z wartością mniejszą lub równą 0 są pomijane.
6. Stwórz tablicę `logArray`, używając funkcji `logspace`, która zawiera 30 punktów rozłożonych logarytmicznie między 10^1 a 10^5 . Następnie oblicz średnią wartość wszystkich elementów w tej tablicy.
7. Wygeneruj tablicę `frequencies`, korzystając z funkcji `logspace`, aby otrzymać 25 punktów logarytmicznie równomiernie rozłożonych między częstotliwościami 10^2 Hz a 10^6 Hz. Użyj tej tablicy do symulacji wartości pewnego sygnału w zależności od częstotliwości i zapisz wyniki w nowej tablicy `signalValues`.
8. Korzystając z funkcji `logspace`, utwórz tablicę `resistances` reprezentującą wartości rezystancji, które są rozłożone logarytmicznie w zakresie od 1Ω do $1M\Omega$ włącznie, z 40 punktami.

5 Pandas

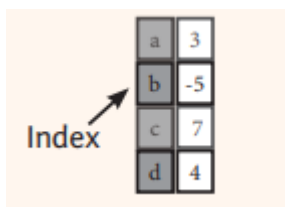
Pandas jest biblioteką Pythona służącą do analizy i manipulowania danymi

Import:

```
import pandas as pd
```

5.1 Podstawowe byty

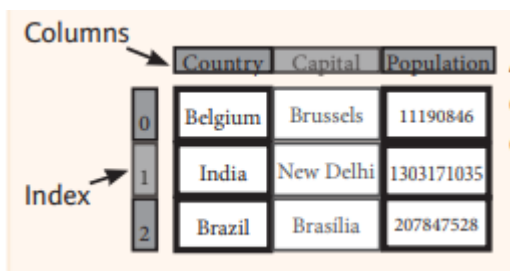
Seria - Series



The diagram illustrates a Pandas Series as a vertical column of data. It consists of four rows, each with a label in the first column and a numerical value in the second column. The labels are 'a', 'b', 'c', and 'd', while the values are 3, -5, 7, and 4. An arrow labeled 'Index' points to the first row, indicating that the labels serve as the index for the data.

a	3
b	-5
c	7
d	4

Ramka danych - DataFrame



The diagram illustrates a Pandas DataFrame as a table with multiple columns and rows. The columns are labeled 'Country', 'Capital', and 'Population'. The rows are indexed from 0 to 2. The data is as follows:

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
import pandas as pd
import numpy as np

s = pd.Series([3, -5, 7, 4])
```

```

print(s)
print("values")
print(s.values)
print(type(s.values))
t = np.sort(s.values)
print(t)
print(s.index)
print(type(s.index))

```

```

0    3
1   -5
2    7
3    4
dtype: int64
values
[ 3 -5  7  4]
<class 'numpy.ndarray'>
[-5  3  4  7]
RangeIndex(start=0, stop=4, step=1)
<class 'pandas.core.indexes.range.RangeIndex'>

```

```

import pandas as pd
import numpy as np

s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
print(s)
print(s['b'])
s['b'] = 8
print(s)
print(s[s > 5])
print(s * 2)
print(np.sin(s))

```

```

a    3
b   -5
c    7
d    4
dtype: int64
-5
a    3
b    8

```



```

c      7
d      4
dtype: int64
b      8
c      7
dtype: int64
a      6
b     16
c     14
d      8
dtype: int64
a    0.141120
b    0.989358
c    0.656987
d   -0.756802
dtype: float64

```

```

import pandas as pd

d = {'key1': 350, 'key2': 700, 'key3': 70}
s = pd.Series(d)
print(s)

```

```

key1    350
key2    700
key3     70
dtype: int64

```

```

import pandas as pd

d = {'key1': 350, 'key2': 700, 'key3': 70}
k = ['key0', 'key2', 'key3', 'key1']
s = pd.Series(d, index=k)
print(s)
pd.isnull(s)
pd.notnull(s)
s.isnull()
s.notnull()
s.name = "Wartosc"
s.index.name = "Klucz"
print(s)

```

```

key0      NaN
key2      700.0
key3       70.0
key1      350.0
dtype: float64
Klucz
key0      NaN
key2      700.0
key3       70.0
key1      350.0
Name: Wartosc, dtype: float64

```

```

import pandas as pd

data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],
        'Population': [11190846, 1303171035, 207847528]}
frame = pd.DataFrame(data)
print(frame)
df = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print(df)

```

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

	Country	Population	Capital
0	Belgium	11190846	Brussels
1	India	1303171035	New Delhi
2	Brazil	207847528	Brasília

```

import pandas as pd

data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],
        'Population': [11190846, 1303171035, 207847528]}
df = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print(df.iloc[[0], [0]])
print("--")
print(df.loc[[0], ['Country']])
print("--")
print(df.loc[2])

```

```
print("--")
print(df.loc[:, 'Capital'])
print("--")
print(df.loc[1, 'Capital'])
```

```
Country
```

```
0 Belgium
```

```
--
```

```
Country
```

```
0 Belgium
```

```
--
```

```
Country          Brazil
```

```
Population      207847528
```

```
Capital         Brasília
```

```
Name: 2, dtype: object
```

```
--
```

```
0 Brussels
```

```
1 New Delhi
```

```
2 Brasília
```

```
Name: Capital, dtype: object
```

```
--
```

```
New Delhi
```

1. loc:

- To metoda indeksowania oparta na etykietach, co oznacza, że używa nazw etykiet kolumn i indeksów wierszy do wyboru danych.
- Działa na podstawie etykiet indeksu oraz etykiet kolumny, co pozwala na wygodniejsze filtrowanie danych.
- Obsługuje zarówno jednostkowe etykiety, jak i zakresy etykiet.
- Działa również z etykietami nieliczbowymi.
- Przykład użycia: `df.loc[1:3, ['A', 'B']]` - zwraca wiersze od indeksu 1 do 3 (włącznie) oraz kolumny 'A' i 'B'.

2. iloc:

- To metoda indeksowania oparta na pozycji, co oznacza, że używa liczbowych indeksów kolumn i wierszy do wyboru danych.
- Działa na podstawie liczbowych indeksów zarówno dla wierszy, jak i kolumn.
- Obsługuje jednostkowe indeksy oraz zakresy indeksów.
- W przypadku używania zakresów indeksów, zakres jest półotwarty, co oznacza, że prawy kraniec nie jest uwzględniany.

- Przykład użycia: `df.iloc[1:3, 0:2]` - zwraca wiersze od indeksu 1 do 3 (bez 3) oraz kolumny od indeksu 0 do 2 (bez 2).

```
import pandas as pd

data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],
        'Population': [11190846, 1303171035, 207847528]}
df = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print(df['Population'])
print("--")
print(df[df['Population'] > 1200000000])
print("--")
print(df.drop('Country', axis=1))
print("--")
```

```
0      11190846
1     1303171035
2     207847528
Name: Population, dtype: int64
--
   Country  Population  Capital
1   India   1303171035  New Delhi
--
   Population  Capital
0     11190846  Brussels
1    1303171035  New Delhi
2     207847528  Brasília
--
```

```
import pandas as pd

data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],
        'Population': [11190846, 1303171035, 207847528]}
df = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print("Shape:", df.shape)
print("--")
print("Index:", df.index)
print("--")
print("columns:", df.columns)
print("--")
```

```
df.info()
print("--")
print(df.count())
```

```
Shape: (3, 3)
--
Index: RangeIndex(start=0, stop=3, step=1)
--
columns: Index(['Country', 'Population', 'Capital'], dtype='object')
--
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Country     3 non-null      object
1   Population   3 non-null      int64
2   Capital      3 non-null      object
dtypes: int64(1), object(2)
memory usage: 204.0+ bytes
--
Country      3
Population    3
Capital       3
dtype: int64
```

5.2 Uzupełnianie braków

```
import pandas as pd

s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
print(s + s2)
print("--")
print(s.add(s2, fill_value=0))
print("--")
print(s.mul(s2, fill_value=2))
```

```
a    10.0
```

```

b      NaN
c      5.0
d      7.0
dtype: float64
--
a      10.0
b      -5.0
c      5.0
d      7.0
dtype: float64
--
a      21.0
b      -10.0
c      -14.0
d      12.0
dtype: float64

```

5.3 Obsługa plików csv

Funkcja `pandas.read_csv`

Dokumentacja: [link](#)

Wybrane argumenty:

- `filepath` - ścieżka dostępu
- `sep=_NoDefault.no_default, delimiter=None` - separator
- `header='infer'` - nagłówek - domyślnie nazwy kolumn, ew. `header=None` oznacza brak nagłówka
- `index_col=None` - ustalenie kolumny na indeksy (nazwy wierszy)
- `thousands=None` - separator tysięcy
- `decimal='.'` - separator dziesiętny

Zapis `pandas.DataFrame.to_csv`

Dokumentacja: [link](#)

5.4 Obsługa plików z Excela

Funkcja `pandas.read_excel`

https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html

**** Ważne:** trzeba zainstalować bibliotekę `openpyxl` do importu `.xlsx` oraz `xlrd` do importu `.xls` (nie trzeba ich importować w kodzie jawnie w większości wypadków)

Wybrane argumenty:

- `io` - ścieżka dostępu
- `sheet_name=0` - nazwa arkusza
- `header='infer'` - nagłówek - domyślnie nazwy kolumn, ew. `header=None` oznacza brak nagłówka
- `index_col=None` - ustalenie kolumny na indeksy (nazwy wierszy)
- `thousands=None` - separator tysięcy
- `decimal='.'` - separator dziesiętny

5.5 Repozytorium z testowymi plikami

- <https://github.com/pjastr/SamleTestFilesVD>

5.6 Operacje manipulacyjne

Ściągowka https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

- `merge`

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html>

Funkcja `merge` służy do łączenia dwóch ramek danych wzdłuż wspólnej kolumny, podobnie jak operacje JOIN w SQL.

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False)
```

Gdzie:

- `right`: ramka danych, którą chcesz dołączyć do oryginalnej ramki danych.
- `how`: określa typ łączenia. Dostępne są cztery typy: 'inner', 'outer', 'left' i 'right'. 'inner' to domyślna wartość, która zwraca tylko te wiersze, które mają pasujące klucze w obu ramkach danych.
- `on`: nazwa lub lista nazw, które mają być używane do łączenia. Musi to być nazwa występująca zarówno w oryginalnej, jak i prawej ramce danych.
- `left_on` i `right_on`: nazwy kolumn w lewej i prawej ramce danych, które mają być używane do łączenia. Można to użyć, jeśli nazwy kolumn nie są takie same.
- `left_index` i `right_index`: czy indeksy z lewej i prawej ramki danych mają być używane do łączenia.

- **sort**: czy wynikowa ramka danych ma być posortowana według łączonych kluczy.
- **suffixes**: sufiksy, które mają być dodane do nazw kolumn, które nachodzą na siebie. Domyślnie to ('_x', '_y').
- **copy**: czy zawsze kopiować dane, nawet jeśli nie są potrzebne.
- **indicator**: dodaj kolumnę do wynikowej ramki danych, która pokazuje źródło każdego wiersza.
- **validate**: sprawdź, czy określone zasady łączenia są spełnione.

```
import pandas as pd

df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2', 'A3'],
    'B': ['B0', 'B1', 'B2', 'B3'],
    'key': ['K0', 'K1', 'K0', 'K1']
})

df2 = pd.DataFrame({
    'C': ['C0', 'C1'],
    'D': ['D0', 'D1']},
    index=['K0', 'K1']
)

print(df1)
print(df2)
merged_df = df1.merge(df2, left_on='key', right_index=True)
print(merged_df)
```

```

      A  B key
0  A0  B0  K0
1  A1  B1  K1
2  A2  B2  K0
3  A3  B3  K1
      C  D
K0  C0  D0
K1  C1  D1
      A  B key  C  D
0  A0  B0  K0  C0  D0
1  A1  B1  K1  C1  D1
2  A2  B2  K0  C0  D0
3  A3  B3  K1  C1  D1
```



```

import pandas as pd

df1 = pd.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'A': ['A0', 'A1', 'A2', 'A3'],
    'B': ['B0', 'B1', 'B2', 'B3']
})

df2 = pd.DataFrame({
    'key': ['K0', 'K1', 'K4', 'K5'],
    'C': ['C0', 'C1', 'C2', 'C3'],
    'D': ['D0', 'D1', 'D2', 'D3']
})

print(df1)

print(df2)

inner_merged_df = df1.merge(df2, how='inner', on='key', suffixes=('_left', '_right'), indicator=True)
outer_merged_df = df1.merge(df2, how='outer', on='key', suffixes=('_left', '_right'), indicator=True)
left_merged_df = df1.merge(df2, how='left', on='key', suffixes=('_left', '_right'), indicator=True)
right_merged_df = df1.merge(df2, how='right', on='key', suffixes=('_left', '_right'), indicator=True)

print("Inner join")
print(inner_merged_df)

print("Outer join")
print(outer_merged_df)

print("Left join")
print(left_merged_df)

print("Right join")
print(right_merged_df)

```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

	key	C	D
0	K0	C0	D0

```

1  K1  C1  D1
2  K4  C2  D2
3  K5  C3  D3
Inner join
  key  A    B    C    D  _merge
0  K0  A0  B0  C0  D0    both
1  K1  A1  B1  C1  D1    both
Outer join
  key  A    B    C    D    _merge
0  K0  A0  B0  C0  D0      both
1  K1  A1  B1  C1  D1      both
2  K2  A2  B2  NaN  NaN  left_only
3  K3  A3  B3  NaN  NaN  left_only
4  K4  NaN  NaN  C2  D2  right_only
5  K5  NaN  NaN  C3  D3  right_only
Left join
  key  A    B    C    D    _merge
0  K0  A0  B0  C0  D0      both
1  K1  A1  B1  C1  D1      both
2  K2  A2  B2  NaN  NaN  left_only
3  K3  A3  B3  NaN  NaN  left_only
Right join
  key  A    B    C    D    _merge
0  K0  A0  B0  C0  D0      both
1  K1  A1  B1  C1  D1      both
2  K4  NaN  NaN  C2  D2  right_only
3  K5  NaN  NaN  C3  D3  right_only

```

- `join`

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html>

Metoda `join` jest używana do łączenia dwóch ramek danych wzdłuż osi.

Podstawowe użycie tej metody wygląda następująco:

```
DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False)
```

Gdzie:

- **other**: ramka danych, którą chcesz dołączyć do oryginalnej ramki danych.
- **on**: nazwa lub lista nazw kolumn w oryginalnej ramce danych, do których chcesz dołączyć.

- **how**: określa typ łączenia. Dostępne są cztery typy: 'inner', 'outer', 'left' i 'right'. 'left' to domyślna wartość, która zwraca wszystkie wiersze z oryginalnej ramki danych i pasujące wiersze z drugiej ramki danych. Wartości są uzupełniane wartością NaN, jeśli nie ma dopasowania.
- **lsuffix** i **rsuffix**: sufiksy do dodania do kolumn, które się powtarzają. Domyślnie jest to puste.
- **sort**: czy sortować dane według klucza.

```
import pandas as pd

df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']},
    index=['K0', 'K1', 'K2'])

df2 = pd.DataFrame({
    'C': ['C0', 'C2', 'C3'],
    'D': ['D0', 'D2', 'D3']},
    index=['K0', 'K2', 'K3'])

print(df1)

print(df2)

joined_df = df1.join(df2)
print(joined_df)
```

```

      A  B
K0  A0  B0
K1  A1  B1
K2  A2  B2
      C  D
K0  C0  D0
K2  C2  D2
K3  C3  D3
      A  B    C    D
K0  A0  B0   C0   D0
K1  A1  B1  NaN  NaN
K2  A2  B2   C2   D2
```

- **concat**

<https://pandas.pydata.org/docs/reference/api/pandas.concat.html>

Metoda `concat` jest używana do łączenia dwóch lub więcej ramek danych wzdłuż określonej osi.

Podstawowe użycie tej metody wygląda następująco:

```
pandas.concat(objs, axis=0, join='outer', ignore_index=False, keys=None, levels=None, names=None)
```

Gdzie:

- `objs`: sekwencja ramek danych, które chcesz połączyć.
- `axis`: oś, wzdłuż której chcesz łączyć ramki danych. Domyślnie to 0 (łączenie wierszy, pionowo), ale można także ustawić na 1 (łączenie kolumn, poziomo).
- `join`: określa typ łączenia. Dostępne są dwa typy: 'outer' i 'inner'. 'outer' to domyślna wartość, która zwraca wszystkie kolumny z każdej ramki danych. 'inner' zwraca tylko te kolumny, które są wspólne dla wszystkich ramek danych.
- `ignore_index`: jeśli ustawione na True, nie używa indeksów z ramek danych do tworzenia indeksu w wynikowej ramce danych. Zamiast tego tworzy nowy indeks od 0 do n-1.
- `keys`: wartości do skojarzenia z obiektami.
- `levels`: określone indeksy dla nowej ramki danych.
- `names`: nazwy dla poziomów indeksów (jeśli są wielopoziomowe).
- `verify_integrity`: sprawdza, czy nowy, skonkatenowana ramka danych nie ma powtarzających się indeksów.
- `sort`: czy sortować niekonkatenacyjną oś (np. indeksy, jeśli `axis=0`), niezależnie od danych.
- `copy`: czy zawsze kopiować dane, nawet jeśli nie są potrzebne.

```
import pandas as pd

df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
})

df2 = pd.DataFrame({
    'A': ['A3', 'A4', 'A5'],
    'B': ['B3', 'B4', 'B5']
})

print(df1)

print(df2)
```

```
concatenated_df = pd.concat([df1, df2], ignore_index=True)
print(concatenated_df)
```

```

      A  B
0  A0  B0
1  A1  B1
2  A2  B2
      A  B
0  A3  B3
1  A4  B4
2  A5  B5
      A  B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
4  A4  B4
5  A5  B5
```

```
import pandas as pd

df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
})

df2 = pd.DataFrame({
    'C': ['C0', 'C1', 'C2'],
    'D': ['D0', 'D1', 'D2']
})

print(df1)

print(df2)

concatenated_df_axis1 = pd.concat([df1, df2], axis=1)
concatenated_df_keys = pd.concat([df1, df2], keys=['df1', 'df2'])

print(concatenated_df_axis1)
print(concatenated_df_keys)
```

	A	B
0	A0	B0
1	A1	B1
2	A2	B2

	C	D
0	C0	D0
1	C1	D1
2	C2	D2

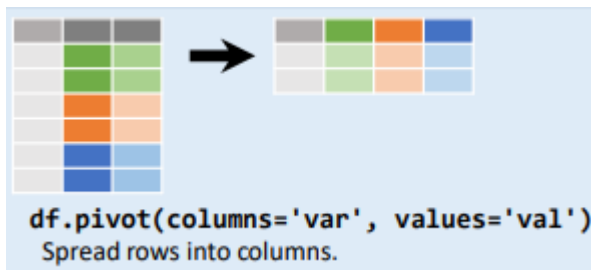
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2

	A	B	C	D
df1 0	A0	B0	NaN	NaN
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN

	A	B	C	D
df2 0	NaN	NaN	C0	D0
1	NaN	NaN	C1	D1
2	NaN	NaN	C2	D2

- `pivot`

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html>



Metoda `pivot` jest używana do przekształcenia danych z formatu “długiego” do “szerokiego”.

Podstawowe użycie tej metody wygląda następująco:

```
DataFrame.pivot(index=None, columns=None, values=None)
```

Gdzie:

- `index`: nazwa kolumny lub lista nazw kolumn, które mają stać się indeksem w nowej ramce danych.
- `columns`: nazwa kolumny, z której unikalne wartości mają stać się kolumnami w nowej ramce danych.

- **values:** nazwa kolumny lub lista nazw kolumn, które mają stać się wartościami dla nowych kolumn w nowej ramce danych.

```
import pandas as pd

df = pd.DataFrame({
    'foo': ['one', 'one', 'one', 'two', 'two', 'two'],
    'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
    'baz': [1, 2, 3, 4, 5, 6],
    'zoo': ['x', 'y', 'z', 'q', 'w', 't'],
})

print(df)

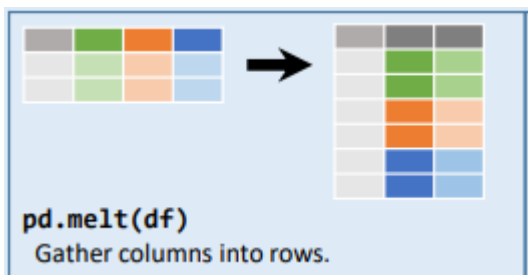
pivot_df = df.pivot(index='foo', columns='bar', values='baz')
print(pivot_df)
```

```
   foo bar  baz zoo
0  one  A    1   x
1  one  B    2   y
2  one  C    3   z
3  two  A    4   q
4  two  B    5   w
5  two  C    6   t
bar  A  B  C
foo
one  1  2  3
two  4  5  6
```

- **wide_to_long**

https://pandas.pydata.org/docs/reference/api/pandas.wide_to_long.html

- **melt**



<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.melt.html>

Funkcja `melt` służy do przekształcania danych z formatu szerokiego na długi.

Podstawowe użycie tej metody wygląda następująco:

```
pandas.melt(frame, id_vars=None, value_vars=None, var_name=None, value_name='value', col_level=None)
```

Gdzie:

- **frame**: ramka danych, którą chcesz przetworzyć.
- **id_vars**: kolumna(y), które chcesz zachować jako identyfikatory. Te kolumny nie będą zmieniane.
- **value_vars**: kolumna(y), które chcesz przekształcić na pary klucz-wartość. Jeżeli nie jest podane, wszystkie kolumny nie będące **id_vars** zostaną użyte.
- **var_name**: nazwa nowej kolumny, która będzie zawierała nazwy kolumn przekształconych na pary klucz-wartość. Domyślnie to 'variable'.
- **value_name**: nazwa nowej kolumny, która będzie zawierała wartości kolumn przekształconych na pary klucz-wartość. Domyślnie to 'value'.
- **col_level**: jeżeli kolumny są wielopoziomowe, to jest poziom, który będzie użyty do przekształcania kolumn na pary klucz-wartość.

```
import pandas as pd

df = pd.DataFrame({
    'A': ['foo', 'bar', 'baz'],
    'B': ['one', 'one', 'two'],
    'C': [2.0, 1.0, 3.0],
    'D': [3.0, 2.0, 1.0]
})
print(df)
melted_df = df.melt(id_vars=['A', 'B'], value_vars=['C', 'D'], var_name='My_Var', value_name='My_Val')
print(melted_df)
```

	A	B	C	D
0	foo	one	2.0	3.0
1	bar	one	1.0	2.0
2	baz	two	3.0	1.0

	A	B	My_Var	My_Val
0	foo	one	C	2.0
1	bar	one	C	1.0
2	baz	two	C	3.0
3	foo	one	D	3.0


```
4 bar one D 2.0
5 baz two D 1.0
```

5.7 “Tidy data”

Imię	Wiek	Wzrost	Kolor oczu
Adam	26	167	Brązowe
Sylwia	34	164	Piwnie
Tomasz	42	183	Niebieskie

- jedna obserwacja (jednostka statystyczna) = jeden wiersz w tabeli/macierzy/ramce danych
- wartości danej cechy znajdują się w kolumnach
- jeden typ/rodzaj obserwacji w jednej tabeli/macierzy/ramce danych

5.8 Obsługa brakujących danych

```
import numpy as np
import pandas as pd

string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
print(string_data)
print(string_data.isnull())
print(string_data.dropna())
```

```
0    aardvark
1    artichoke
2         NaN
3     avocado
dtype: object
0    False
1    False
2     True
3    False
dtype: bool
0    aardvark
1    artichoke
```

3 avocado
dtype: object

```
from numpy import nan as NA
import pandas as pd

data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
                    [NA, NA, NA], [NA, 6.5, 3.]])
cleaned = data.dropna()
print(cleaned)
print(data.dropna(how='all'))
data[4] = NA
print(data.dropna(how='all', axis=1))
print(data)
print(data.fillna(0))
print(data.fillna({1: 0.5, 2: 0}))
```

```
   0    1    2
0  1.0  6.5  3.0
   0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
3  NaN  6.5  3.0
   0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
2  NaN  NaN  NaN
3  NaN  6.5  3.0
   0    1    2    4
0  1.0  6.5  3.0  NaN
1  1.0  NaN  NaN  NaN
2  NaN  NaN  NaN  NaN
3  NaN  6.5  3.0  NaN
   0    1    2    4
0  1.0  6.5  3.0  0.0
1  1.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0
3  0.0  6.5  3.0  0.0
   0    1    2    4
0  1.0  6.5  3.0  NaN
```

```

1  1.0  0.5  0.0 NaN
2  NaN  0.5  0.0 NaN
3  NaN  6.5  3.0 NaN

```

5.9 Usuwanie duplikatów

```

import pandas as pd

data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                      'k2': [1, 1, 2, 3, 3, 4, 4]})

print(data)
print(data.duplicated())
print(data.drop_duplicates())

```

```

   k1 k2
0  one  1
1  two  1
2  one  2
3  two  3
4  one  3
5  two  4
6  two  4
0  False
1  False
2  False
3  False
4  False
5  False
6   True
dtype: bool
   k1 k2
0  one  1
1  two  1
2  one  2
3  two  3
4  one  3
5  two  4

```

5.10 Zastępowanie wartościami

```
import pandas as pd
import numpy as np

data = pd.Series([1., -999., 2., -999., -1000., 3.])
print(data)
print(data.replace(-999, np.nan))
print(data.replace([-999, -1000], np.nan))
print(data.replace([-999, -1000], [np.nan, 0]))
print(data.replace({-999: np.nan, -1000: 0}))
```

```
0      1.0
1    -999.0
2      2.0
3    -999.0
4   -1000.0
5      3.0
dtype: float64
0      1.0
1      NaN
2      2.0
3      NaN
4   -1000.0
5      3.0
dtype: float64
0      1.0
1      NaN
2      2.0
3      NaN
4      NaN
5      3.0
dtype: float64
0      1.0
1      NaN
2      2.0
3      NaN
4      0.0
5      3.0
dtype: float64
0      1.0
```

```

1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64

```

5.11 Dyskretyzacja i podział na koszyki

```

import pandas as pd

ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins)
print(cats)
print(cats.codes)
print(cats.categories)
print(pd.Series(cats).value_counts())

```

```

[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (35, 60], (60, 100]]
Length: 12
Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
[0 0 0 1 0 0 2 1 3 2 2 1]
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], dtype='interval[int64, right]')
(18, 25]      5
(25, 35]      3
(35, 60]      3
(60, 100]     1
Name: count, dtype: int64

```

```

import pandas as pd

ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats2 = pd.cut(ages, [18, 26, 36, 61, 100], right=False)
print(cats2)
group_names = ['Youth', 'YoungAdult',

```

```

        'MiddleAged', 'Senior']
print(pd.cut(ages, bins, labels=group_names))

```

```

[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100), [36, 61), [36, 61),
Length: 12
Categories (4, interval[int64, left]): [[18, 26) < [26, 36) < [36, 61) < [61, 100)]
['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult', 'Senior', 'MiddleAged']
Length: 12
Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']

```

```

import pandas as pd
import numpy as np

data = np.random.rand(20)
print(pd.cut(data, 4, precision=2))

```

```

[(0.51, 0.75], (0.03, 0.27], (0.75, 1.0], (0.03, 0.27], (0.75, 1.0], ..., (0.27, 0.51], (0.03, 0.27],
Length: 20
Categories (4, interval[float64, right]): [(0.03, 0.27] < (0.27, 0.51] < (0.51, 0.75] < (0.75, 1.0]]

```

```

import pandas as pd
import numpy as np

data = np.random.randn(1000)
cats = pd.qcut(data, 4)
print(cats)
print(pd.Series(cats).value_counts())

```

```

[(0.0337, 0.698], (0.698, 3.571], (-0.65, 0.0337], (-3.026999999999997, -0.65], (0.698, 3.571],
Length: 1000
Categories (4, interval[float64, right]): [(-3.026999999999997, -0.65] < (-0.65, 0.0337] < (0.0337, 0.698] < (0.698, 3.571]]
(-3.026999999999997, -0.65]      250
(-0.65, 0.0337]                  250
(0.0337, 0.698]                  250
(0.698, 3.571]                   250
Name: count, dtype: int64

```

5.12 Wykrywanie i filtrowanie elementów odstających

```
import pandas as pd
import numpy as np

data = pd.DataFrame(np.random.randn(1000, 4))
print(data.describe())
print("----")
col = data[2]
print(col[np.abs(col) > 3])
print("----")
print(data[(np.abs(data) > 3).any(axis=1)])
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	-0.001973	0.043772	-0.022898	-0.045212
std	1.028921	0.984265	1.025611	1.066126
min	-3.115899	-3.127973	-3.349511	-3.532457
25%	-0.690127	-0.621834	-0.709499	-0.743104
50%	-0.000937	0.095136	-0.050913	-0.062400
75%	0.729307	0.671406	0.684288	0.684596
max	3.347952	2.898009	3.430562	2.975965

```
66    -3.349511
435   -3.022028
448    3.430562
Name: 2, dtype: float64
----
```

	0	1	2	3
22	1.804922	-0.164692	-2.111408	-3.331254
66	-0.129128	0.141759	-3.349511	0.802074
76	3.347952	-0.395751	1.583663	2.975965
225	-3.056940	-1.640111	0.388686	0.406996
236	-0.232126	0.066572	-0.406193	-3.020402
249	-3.051663	0.909844	-0.783890	0.070252
435	0.717346	-1.779972	-3.022028	1.403453
448	-0.388693	0.803829	3.430562	0.460670
493	0.749240	0.268008	-0.232823	-3.532457
507	-3.115899	1.766609	1.525618	-0.683341
675	0.446327	-1.060099	0.197378	-3.385007
755	0.743823	-0.267848	-1.756832	-3.039496

```
764 -1.457975 -3.127973 -1.915806 1.468374
828 -3.103203 -0.120753 -0.183607 1.246635
835 -0.358757 -0.362257 0.219038 -3.103092
```

5.13 Zmiana typu w kolumnie

```
import pandas as pd

data = {
    'A': ['1', '2', '3', '4', '5', '6'],
    'B': ['7.5', '8.5', '9.5', '10.5', '11.5', '12.5'],
    'C': ['x', 'y', 'z', 'x', 'y', 'z']
}
df = pd.DataFrame(data)

# Wyświetlenie oryginalnej ramki danych
print("Oryginalna ramka danych:")
print(df)

# Zmiana typu danych kolumny 'A' na int
df['A'] = pd.Series(df['A'], dtype=int)

# Zmiana typu danych kolumny 'B' na float
df['B'] = pd.Series(df['B'], dtype=float)

# Wyświetlenie ramki danych po zmianie typów
print("\nRamka danych po zmianie typów:")
print(df)
```

Oryginalna ramka danych:

	A	B	C
0	1	7.5	x
1	2	8.5	y
2	3	9.5	z
3	4	10.5	x
4	5	11.5	y
5	6	12.5	z

Ramka danych po zmianie typów:

	A	B	C
0	1	1.0	x
1	2	2.0	y
2	3	3.0	z
3	4	4.0	x
4	5	5.0	y
5	6	6.0	z

```
import pandas as pd

data = {
    'A': ['1', '2', '3', '4', '5', '6'],
    'B': ['7.5', '8.5', '9.5', '10.5', '11.5', '12.5'],
    'C': ['x', 'y', 'z', 'x', 'y', 'z']
}
df = pd.DataFrame(data)

# Wyświetlenie oryginalnej ramki danych
print("Oryginalna ramka danych:")
print(df)

# Zmiana typu danych kolumny 'A' na int
df['A'] = df['A'].astype(int)

# Zmiana typu danych kolumny 'B' na float
df['B'] = df['B'].astype(float)

# Wyświetlenie ramki danych po zmianie typów
print("\nRamka danych po zmianie typów:")
print(df)
```

Oryginalna ramka danych:

	A	B	C
0	1	7.5	x
1	2	8.5	y
2	3	9.5	z
3	4	10.5	x
4	5	11.5	y
5	6	12.5	z

Ramka danych po zmianie typów:

	A	B	C
0	1	7.5	x
1	2	8.5	y
2	3	9.5	z
3	4	10.5	x
4	5	11.5	y
5	6	12.5	z

5.14 Zmiana znaku kategoriach

```
import pandas as pd

# Tworzenie ramki danych
data = {
    'A': ['abc', 'def', 'ghi', 'jkl', 'mno', 'pqr'],
    'B': ['1.23', '4.56', '7.89', '0.12', '3.45', '6.78'],
    'C': ['xyz', 'uvw', 'rst', 'opq', 'lmn', 'ijk']
}
df = pd.DataFrame(data)

# Wyświetlenie oryginalnej ramki danych
print("Oryginalna ramka danych:")
print(df)

# Zmiana małych liter na duże w kolumnie 'A'
df['A'] = df['A'].str.upper()

# Zastąpienie kropki przecinkiem w kolumnie 'B'
df['B'] = df['B'].str.replace('.', ',')

# Wyświetlenie ramki danych po modyfikacji
print("\nRamka danych po modyfikacji:")
print(df)
```

Oryginalna ramka danych:

	A	B	C
0	abc	1.23	xyz
1	def	4.56	uvw
2	ghi	7.89	rst
3	jkl	0.12	opq

```
4 mno 3.45 lmn
5 pqr 6.78 ijk
```

Ramka danych po modyfikacji:

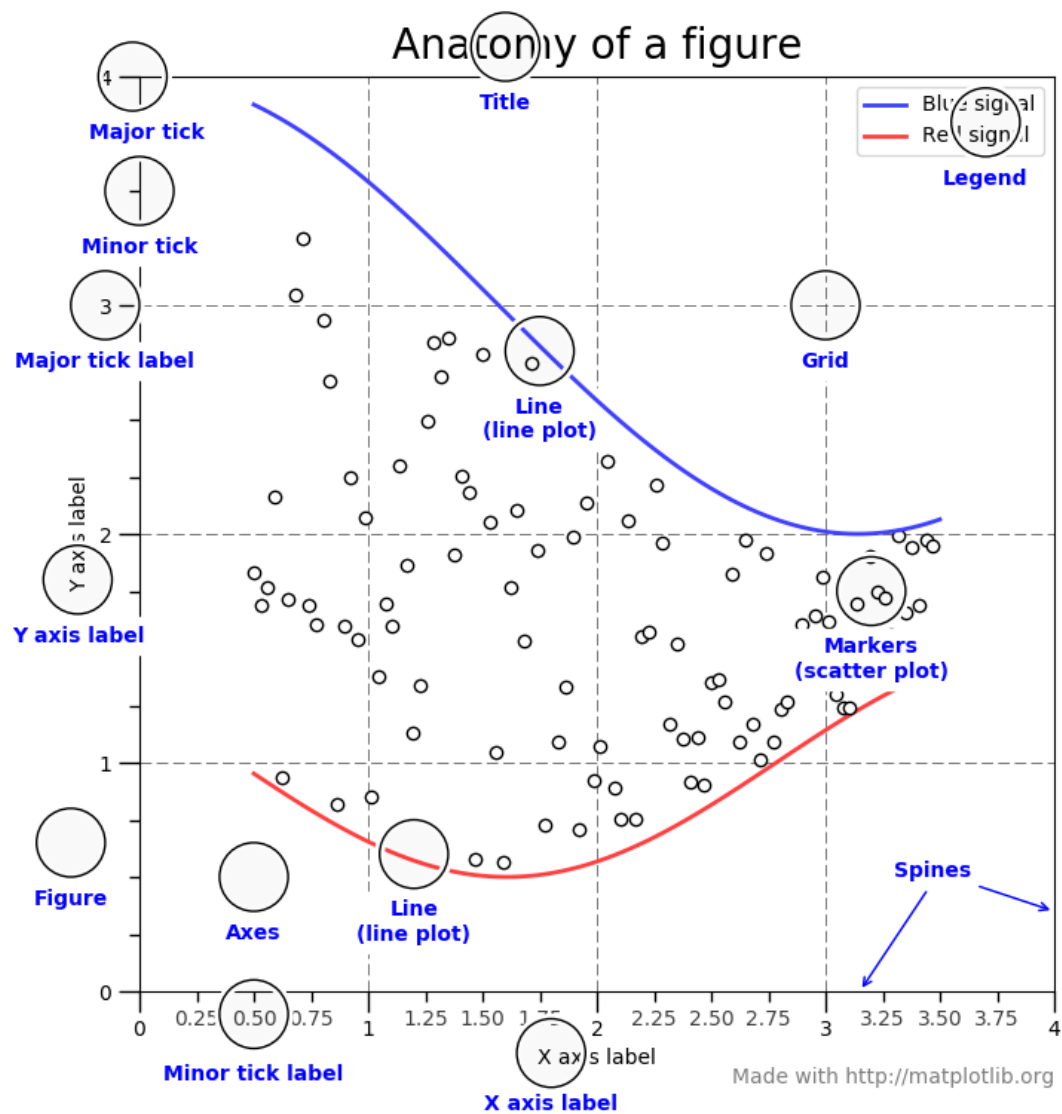
	A	B	C
0	ABC	1,23	xyz
1	DEF	4,56	uvw
2	GHI	7,89	rst
3	JKL	0,12	opq
4	MNO	3,45	lmn
5	PQR	6,78	ijk

Bibliografia:

- Dokumentacja biblioteki, <https://pandas.pydata.org/>, dostęp online 5.03.2021.
- Hannah Stepanek, Thinking in Pandas, How to Use the Python Data Analysis Library the Right Way, Apress, 2020.

6 Matplotlib

<https://matplotlib.org/>



Import

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

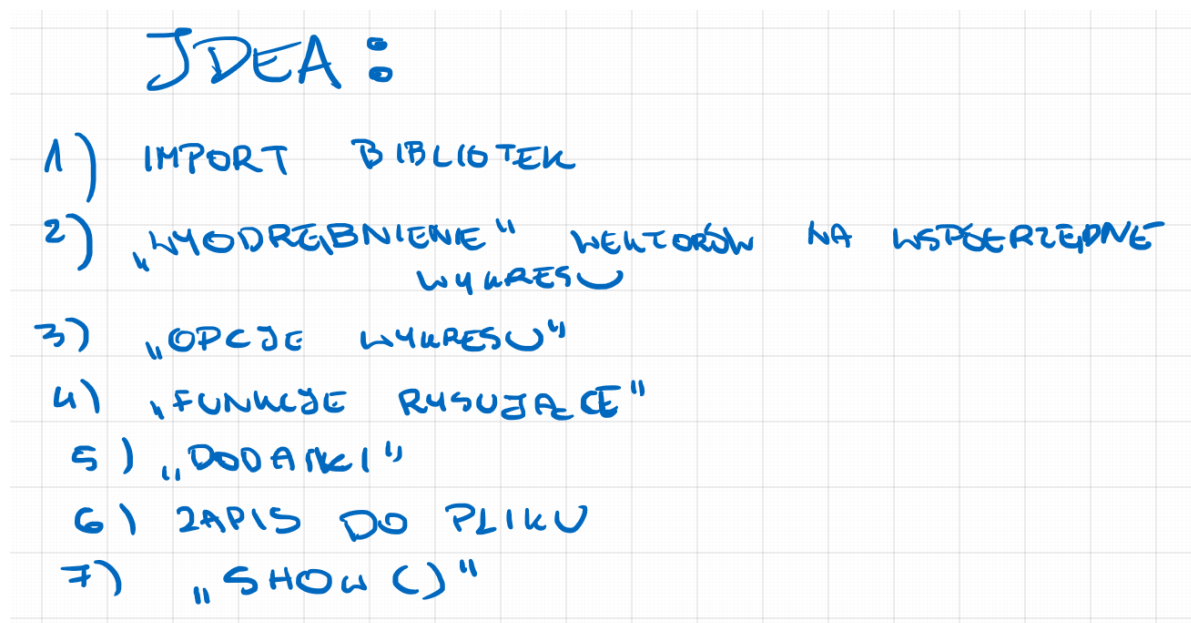
6.1 Galerie wykresów

<https://matplotlib.org/gallery/index.html>

<https://python-graph-gallery.com/>

<https://github.com/rasbt/matplotlib-gallery>

<https://seaborn.pydata.org/examples/index.html>



6.2 Wykres liniowy

Wykres liniowy jest stosowany, gdy chcemy przedstawić zmiany wartości w czasie lub w funkcji innej zmiennej. Wykres liniowy jest odpowiedni dla danych ciągłych, gdzie istnieje związek

między punktami danych. Służy do ilustrowania trendów, wzorców i porównywania między różnymi zestawami danych.

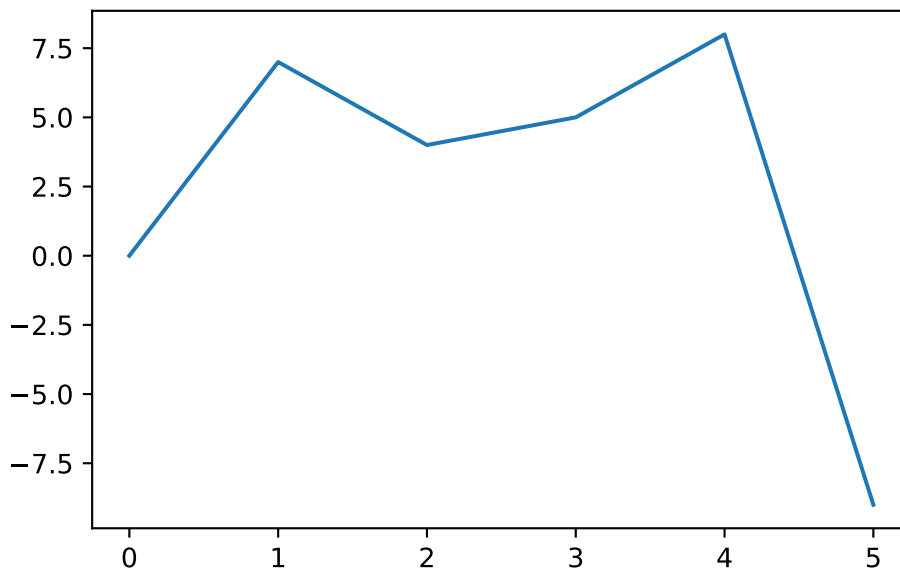
Oto niektóre sytuacje, w których wykresy liniowe są stosowane:

1. Prezentowanie zmian wartości w czasie, na przykład wzrostu gospodarczego, ceny akcji, zmiany temperatury itp.
2. Ukazywanie związku między dwiema zmiennymi, np. związek między poziomem edukacji a zarobkami.
3. Porównywanie trendów dla różnych grup lub kategorii, na przykład analiza sprzedaży różnych produktów w czasie.
4. Analiza korelacji między zmiennymi, na przykład związek między rosnącymi cenami paliwa a spadkiem sprzedaży samochodów.
5. Eksploracja danych, aby zrozumieć strukturę danych i znaleźć wzorce lub anomalie.

Wykresy liniowe są szczególnie przydatne, gdy mamy do czynienia z danymi ciągłymi, a relacje między punktami danych są istotne. Jednak mogą być również używane do prezentowania danych dyskretnych, o ile istnieje zrozumiały związek między punktami danych.

```
import matplotlib.pyplot as plt

x = [0, 7, 4, 5, 8, -9]
plt.plot(x)
plt.show()
```



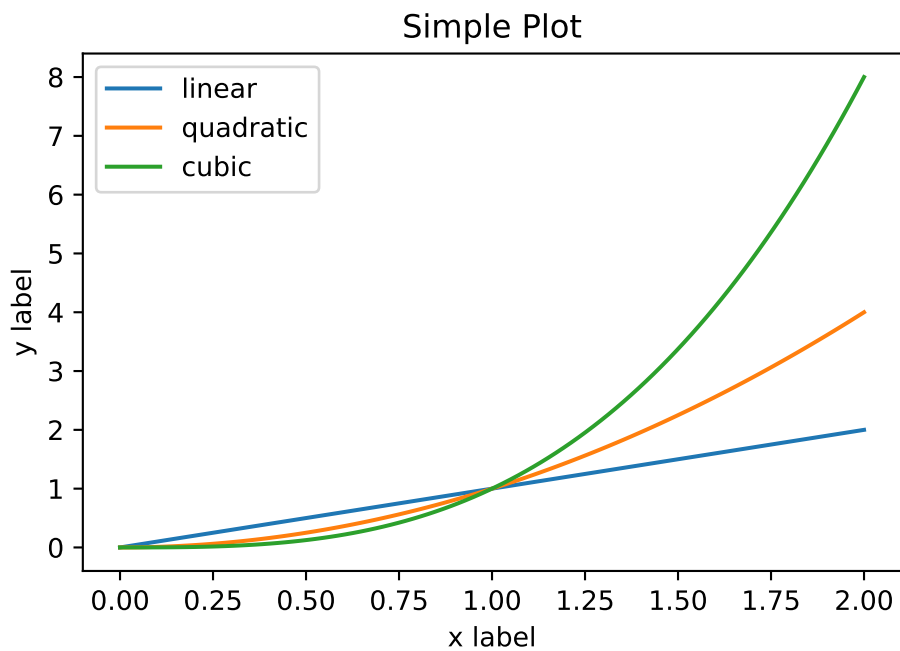
```

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)
plt.plot(x, x, label='linear')
plt.plot(x, x ** 2, label='quadratic')
plt.plot(x, x ** 3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()

```

- ① `x = np.linspace(0, 2, 100)`: tworzy tablicę `x` z 100 równomiernie rozłożonymi wartościami od 0 do 2 (włącznie), korzystając z funkcji `linspace` z biblioteki `numpy`.
- ② `plt.plot(x, x, label='linear')`: rysuje liniowy wykres ($y = x$) z wartościami z tablicy `x`.
- ③ `plt.plot(x, x**2, label='quadratic')`: rysuje wykres kwadratowy ($y = x^2$) z wartościami z tablicy `x`.
- ④ `plt.plot(x, x**3, label='cubic')`: rysuje wykres sześcienny ($y = x^3$) z wartościami z tablicy `x`.
- ⑤ `plt.xlabel('x label')`: dodaje etykietę osi X.
- ⑥ `plt.ylabel('y label')`: dodaje etykietę osi Y.
- ⑦ `plt.title("Simple Plot")`: nadaje tytuł wykresu "Simple Plot".
- ⑧ `plt.legend()`: dodaje legendę do wykresu, która pokazuje etykiety (`label`) dla poszczególnych linii.
- ⑨ `plt.show()`: wyświetla wykres.



6.3 Parametry legendy

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

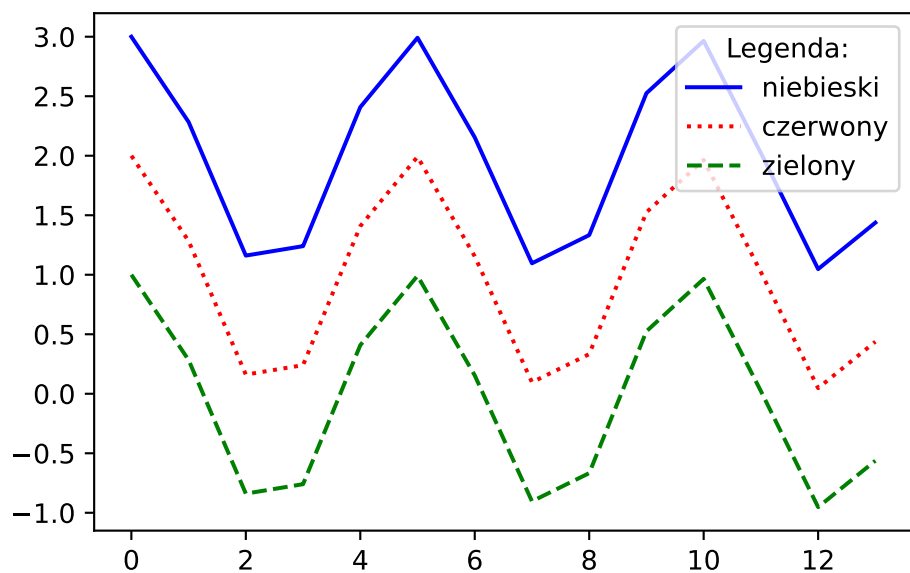
**Parametry lokalizacji
legendy**

6.4 Style, kolory linii

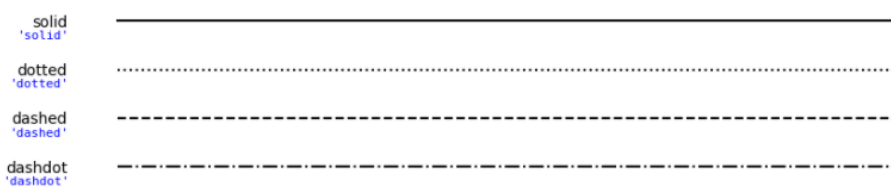
```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.arange(14) ①
y = np.cos(5 * x) ②
plt.plot(x, y + 2, 'blue', linestyle="-", label="niebieski") ③
plt.plot(x, y + 1, 'red', linestyle=":", label="czerwony") ④
plt.plot(x, y, 'green', linestyle="--", label="zielony") ⑤
plt.legend(title='Legenda:')
plt.show()
```

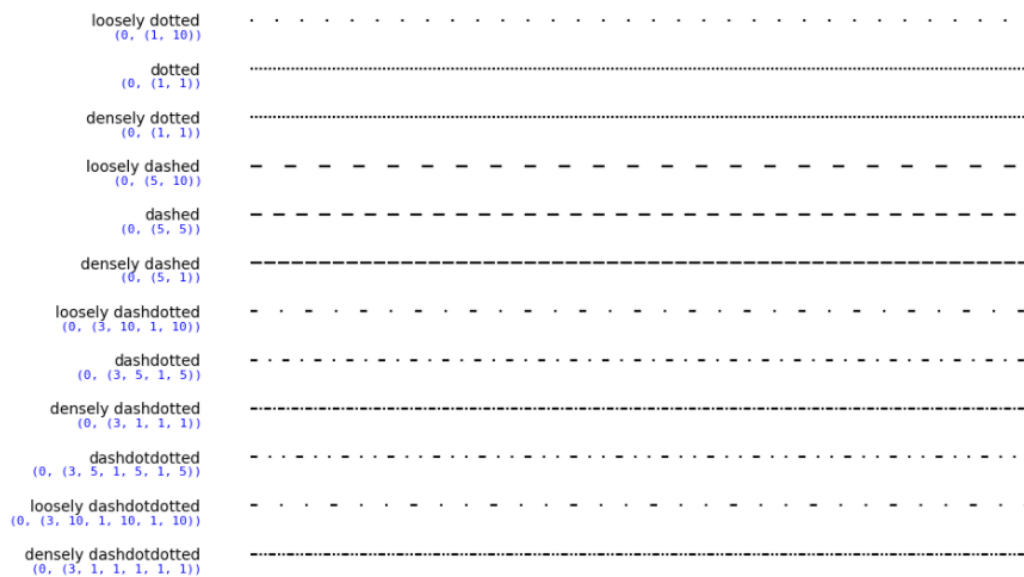
- ① `x = np.arange(14)`: tworzy tablicę `x` z wartościami od 0 do 13 (łącznie z 13), korzystając z funkcji `arange` z biblioteki `numpy`.
- ② `y = np.cos(5 * x)`: oblicza wartości funkcji cosinus dla każdej wartości `x`, przemnożonej przez 5. Wynikowe wartości są zapisane w tablicy `y`.
- ③ `plt.plot(x, y + 2, 'blue', linestyle="-", label="niebieski")`: rysuje niebieski wykres z wartościami z tablicy `x`, a wartości `y` przesunięte o 2 w górę. Linia jest ciągła (`linestyle="-"`).
- ④ `plt.plot(x, y + 1, 'red', linestyle=":", label="czerwony")`: rysuje czerwony wykres z wartościami z tablicy `x`, a wartości `y` przesunięte o 1 w górę. Linia jest punktowana (`linestyle=":"`).
- ⑤ `plt.plot(x, y, 'green', linestyle="--", label="zielony")`: rysuje zielony wykres z wartościami z tablicy `x` i wartościami `y`. Linia jest przerywana (`linestyle="--"`).



Named linestyles



Parametrized linestyles



Linestyle	Description
'-' or 'solid'	solid line
'--' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line
'None' or ' ' or ''	draw nothing

Uproszczone

(offset, (on_off_seq))

(0, (3, 10, 1, 15)) - (3pt line, 10pt space, 1pt line, 15pt space)

linia odstęp linia odstęp

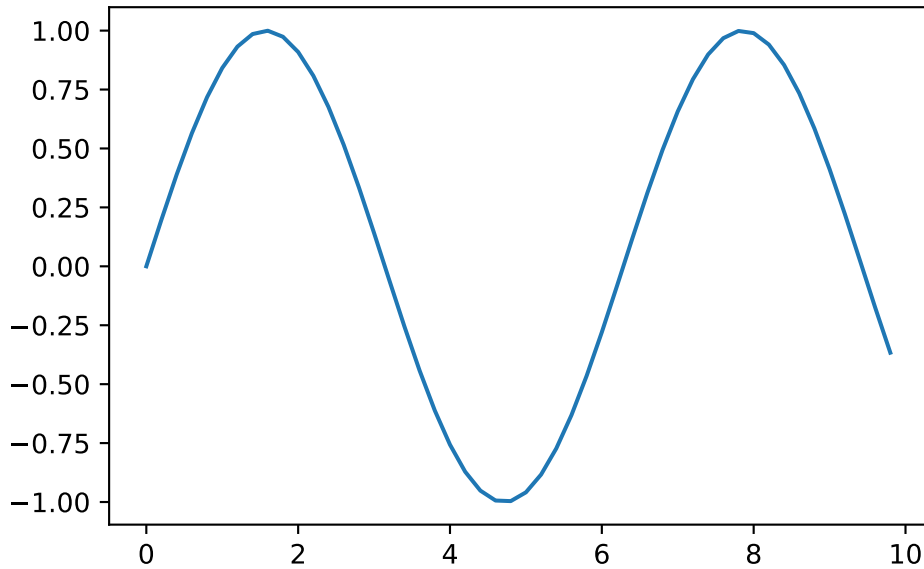
6.5 Wykresy jako obiekty

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

x = np.arange(0, 10, 0.2)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

①
②
③
④
⑤

- ① `x = np.arange(0, 10, 0.2)`: tworzy tablicę `x` z wartościami od 0 do 10 (bez 10) z krokiem 0.2, korzystając z funkcji `arange` z biblioteki `numpy`.
- ② `y = np.sin(x)`: oblicza wartości funkcji sinus dla każdej wartości `x`. Wynikowe wartości są zapisane w tablicy `y`.
- ③ `fig, ax = plt.subplots()`: tworzy nową figurę (`fig`) i osie (`ax`) za pomocą funkcji `subplots` z biblioteki `matplotlib.pyplot`. Figura to obiekt zawierający wszystkie elementy wykresu, takie jak osie, linie wykresu, tekst itp. Osie to obiekt, który definiuje układ współrzędnych, na którym rysowany jest wykres.
- ④ `ax.plot(x, y)`: rysuje wykres wartości `y` w funkcji `x` na osiach `ax` utworzonych wcześniej.
- ⑤ `plt.show()`: wyświetla wykres.



6.6 Wykres liniowy i punktowy

Wykres punktowy (scatter plot) jest stosowany, gdy chcemy przedstawić związek między dwiema zmiennymi lub rozkład punktów danych w przestrzeni dwuwymiarowej. Wykres punktowy jest odpowiedni dla danych zarówno ciągłych, jak i dyskretnych, gdy chcemy zobrazować wzory, korelację lub związki między zmiennymi.

Oto niektóre sytuacje, w których wykresy punktowe są stosowane:

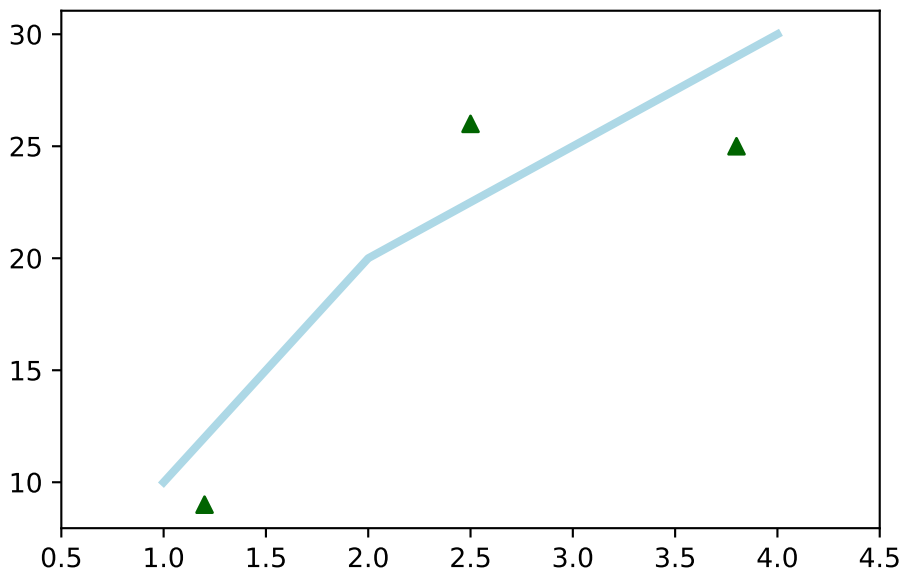
1. Analiza korelacji między dwiema zmiennymi, na przykład związek między wiekiem a dochodem.
2. Prezentowanie rozkładu punktów danych, na przykład wykazanie geograficznego rozmieszczenia sklepów w mieście.
3. Eksploracja danych, aby zrozumieć strukturę danych i znaleźć wzorce, grupy lub anomalie, na przykład w celu identyfikacji skupisk danych w analizie skupień (clustering).
4. Wykrywanie wartości odstających (outliers) w danych, na przykład dla wykrywania nietypowych obserwacji w zbiorze danych.
5. Porównywanie różnych grup lub kategorii danych, na przykład porównanie wzrostu gospodarczego różnych krajów względem ich długu publicznego.

Wykresy punktowe są szczególnie przydatne, gdy mamy do czynienia z danymi o różnym charakterze (ciągłe lub dyskretne) oraz gdy chcemy zbadać korelację, grupy, wzorce lub wartości odstające.

```
import matplotlib.pyplot as plt

fig = plt.figure() ①
ax = fig.add_subplot(111) ②
ax.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3) ③
ax.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^') ④
ax.set_xlim(0.5, 4.5) ⑤
plt.show()
```

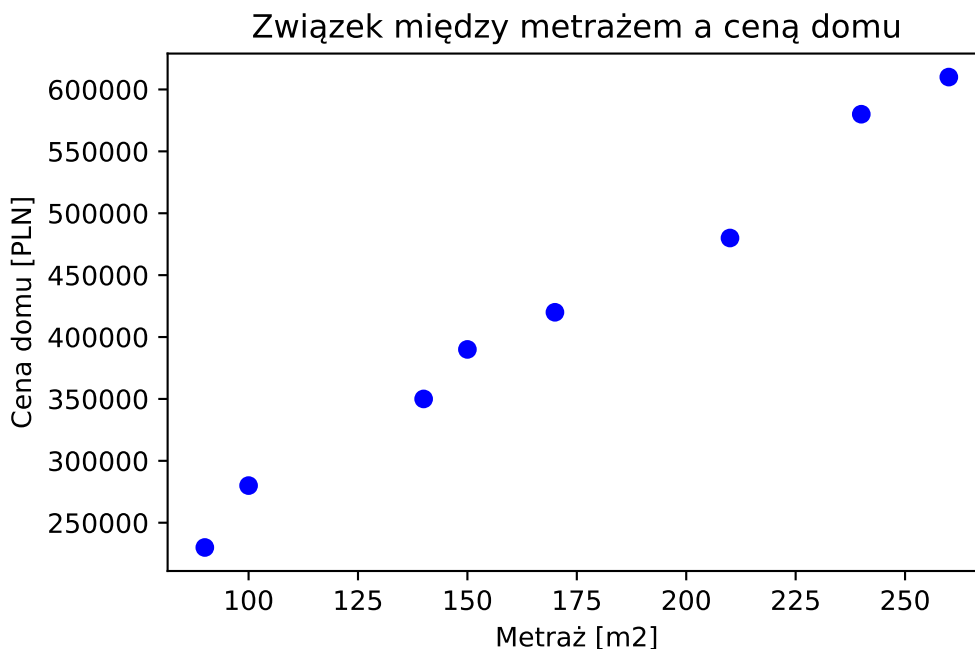
- ① `fig = plt.figure()`: tworzy nową figurę (`fig`). Figura to obiekt zawierający wszystkie elementy wykresu, takie jak osie, linie wykresu, tekst itp.
- ② `ax = fig.add_subplot(111)`: dodaje nowy zestaw osi (`ax`) do figury `fig` za pomocą metody `add_subplot`. Argument `111` oznacza, że chcemy stworzyć siatkę `1x1` i umieścić nasz wykres na pierwszym (i jedynym) polu tej siatki.
- ③ `ax.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3)`: rysuje niebieski wykres liniowy o szerokości linii równej `3` na osiach `ax`, używając listy wartości `[1, 2, 3, 4]` dla osi `X` i `[10, 20, 25, 30]` dla osi `Y`.
- ④ `ax.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^')`: dodaje punkty w kształcie trójkątów (`^`) na osiach `ax` w miejscach określonych przez listy wartości `[0.3, 3.8, 1.2, 2.5]` dla osi `X` i `[11, 25, 9, 26]` dla osi `Y`. Punkty są w kolorze ciemnozielonym.
- ⑤ `ax.set_xlim(0.5, 4.5)`: ustawia zakres wartości osi `X` na osiach `ax` od `0.5` do `4.5`.



```
import matplotlib.pyplot as plt

house_prices = [230000, 350000, 480000, 280000, 420000, 610000, 390000, 580000]
square_meters = [90, 140, 210, 100, 170, 260, 150, 240]
plt.scatter(square_meters, house_prices, color='blue', marker='o')
plt.xlabel('Metraż [m2]')
plt.ylabel('Cena domu [PLN]')
plt.title('Związek między metrażem a ceną domu')
plt.show()
```

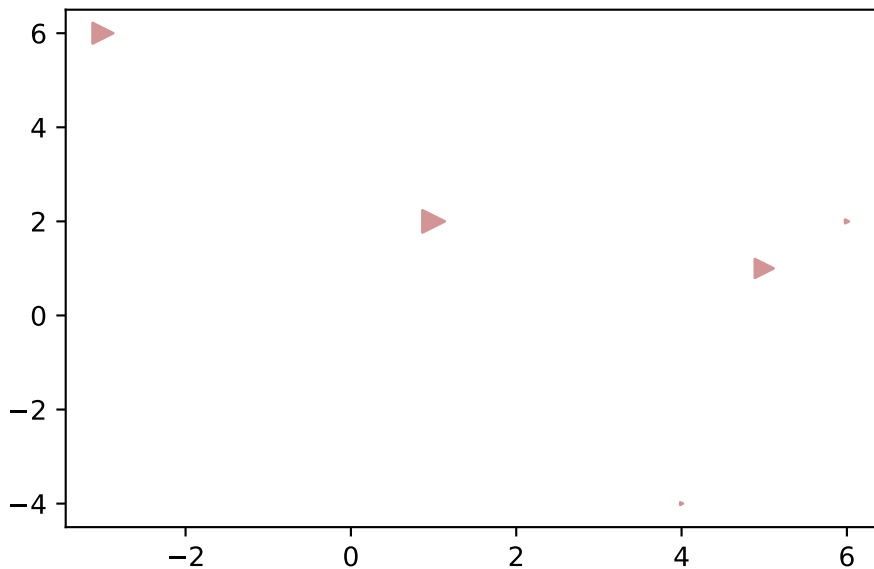
- ① `plt.scatter(square_meters, house_prices, color='blue', marker='o')`: tworzy wykres punktowy (scatter plot) z metrażem domów na osi X (`square_meters`) i cenami domów na osi Y (`house_prices`). Punkty są koloru niebieskiego (`color='blue'`) i mają kształt kółka (`marker='o'`).



```
from matplotlib import pyplot as plt

x = [1, -3, 4, 5, 6]
y = [2, 6, -4, 1, 2]
area = [70, 60, 1, 50, 2]
plt.scatter(x, y, marker=">", color="brown", alpha=0.5, s=area)
plt.show()
```

- ① Kod `plt.scatter(x, y, marker=">", color="brown", alpha=0.5, s=area)` tworzy wykres punktowy (scatter plot) `x`: lista lub tablica współrzędnych `x` punktów na wykresie. `y`: lista lub tablica współrzędnych `y` punktów na wykresie. Wartości `x` i `y` muszą mieć tę samą długość, aby przedstawić każdy punkt na wykresie. `marker`: symbol reprezentujący kształt punktów na wykresie. W tym przypadku, używamy `>` co oznacza strzałkę skierowaną w prawo. `color`: kolor punktów na wykresie. W tym przypadku, używamy koloru `"brown"` (brązowy). `alpha`: przezroczystość punktów na wykresie, gdzie wartość 1 oznacza całkowitą nieprzezroczystość, a 0 całkowitą przezroczystość. W tym przypadku, używamy wartości 0.5 co oznacza półprzezroczystość punktów. `s`: rozmiar punktów na wykresie, który może być pojedynczą wartością lub listą/tablicą wartości o długości takiej samej jak współrzędne `x` i `y`.



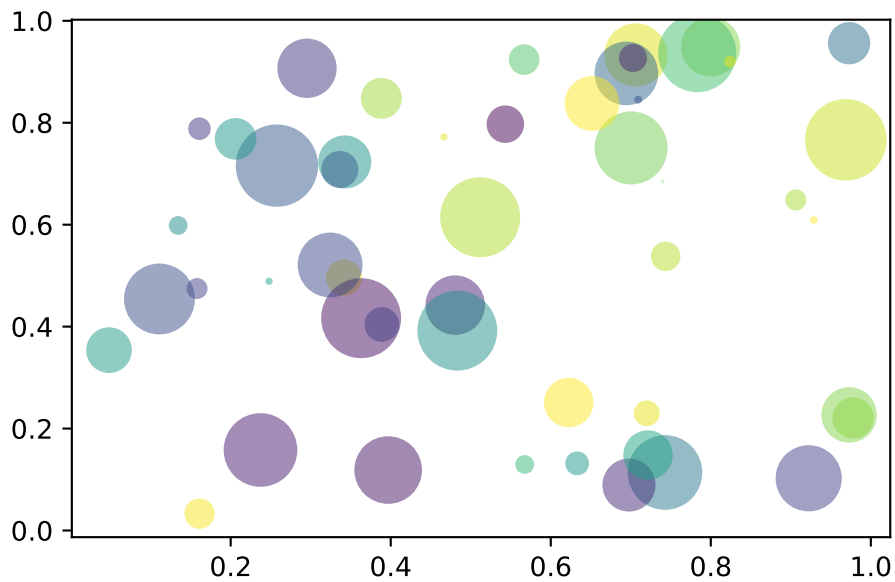
```
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N)) ** 2 # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```

```
plt.show()
```



6.7 Kolory

- https://matplotlib.org/stable/gallery/color/named_colors.html
- https://pl.wikipedia.org/wiki/Lista_kolor%C3%B3w

	b		c		k
	g		m		w
	r		y		

	tab:blue		tab:brown
	tab:orange		tab:pink
	tab:green		tab:gray
	tab:red		tab:olive
	tab:purple		tab:cyan

black	bisque	forestgreen	slategrey
dimgray	darkorange	limegreen	lightsteelblue
dimgrey	burlywood	darkgreen	cornflowerblue
gray	antiquewhite	green	royalblue
grey	tan	lime	ghostwhite
darkgray	navajowhite	seagreen	lavender
darkgrey	blanchedalmond	mediumseagreen	midnightblue
silver	papayawhip	springgreen	navy
lightgray	moccasin	mintcream	darkblue
lightgrey	orange	mediumspringgreen	mediumblue
gainsboro	wheat	mediumaquamarine	blue
whitesmoke	oldlace	aquamarine	slateblue
white	floralwhite	turquoise	darkslateblue
snow	darkgoldenrod	lightseagreen	mediumslateblue
rosybrown	goldenrod	mediumturquoise	mediumpurple
lightcoral	cornsilk	azure	rebeccapurple
indianred	gold	lightcyan	blueviolet
brown	lemonchiffon	paleturquoise	indigo
firebrick	khaki	darkslategray	darkorchid
maroon	palegoldenrod	darkslategrey	darkviolet
darkred	darkkhaki	teal	mediumorchid
red	ivory	darkcyan	thistle
mistyrose	beige	aqua	plum
salmon	lightyellow	cyan	violet
tomato	lightgoldenrodyellow	darkturquoise	purple
darksalmon	olive	cadetblue	darkmagenta
coral	yellow	powderblue	fuchsia
orangered	olivedrab	lightblue	magenta
lightsalmon	yellowgreen	deepskyblue	orchid
sienna	darkolivegreen	skyblue	mediumvioletred
seashell	greenyellow	lightskyblue	deeppink
chocolate	chartreuse	steelblue	hotpink
saddlebrown	lawngreen	aliceblue	lavenderblush
sandybrown	honeydew	dodgerblue	palevioletred
peachpuff	darkseagreen	lightslategray	crimson
peru	palegreen	lightslategrey	pink
linen	lightgreen	slategray	lightpink

```

import numpy as np
import matplotlib.pyplot as plt

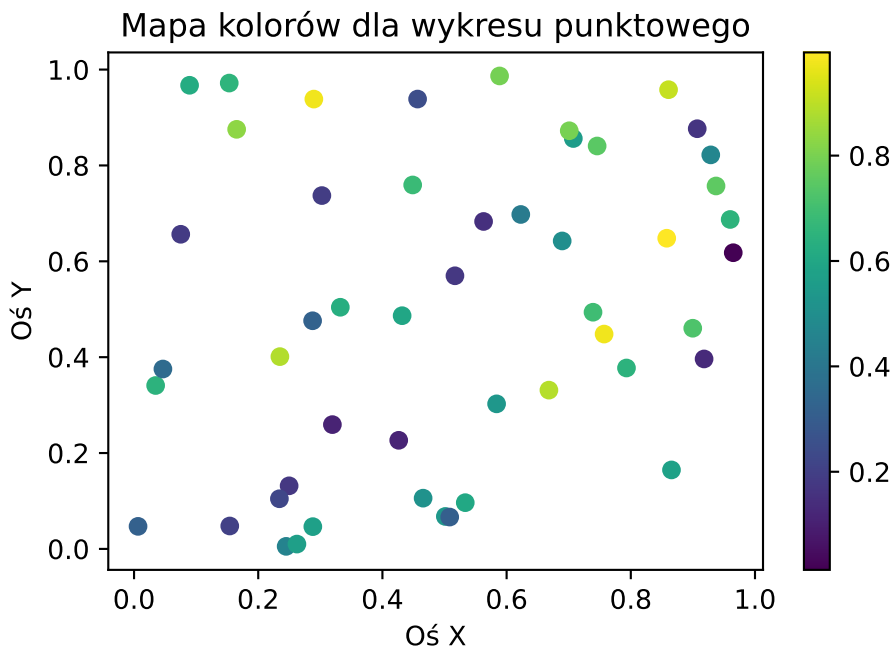
x = np.random.rand(50)
y = np.random.rand(50)
z = np.random.rand(50)
plt.scatter(x, y, c=z, cmap='viridis')
plt.colorbar()
plt.xlabel('0 ≤ X')
plt.ylabel('0 ≤ Y')

```

①
②

```
plt.title('Mapa kolorów dla wykresu punktowego')
plt.show()
```

- ① `plt.scatter(x, y, c=z, cmap='viridis')`: ta linia tworzy wykres punktowy (`scatter plot`) z danymi `x`, `y` i `z`. `x` i `y` to dane, które będą wyświetlane na osi X i Y, a `z` to dane, które będą używane do stworzenia mapy kolorów. Argument `cmap='viridis'` określa mapę kolorów, która będzie użyta do przypisania kolorów do wartości numerycznych.
- ② `plt.colorbar()`: ta linia dodaje pasek kolorów do wykresu punktowego. Pasek kolorów wskazuje, które kolory odpowiadają wartościom numerycznym na mapie kolorów.



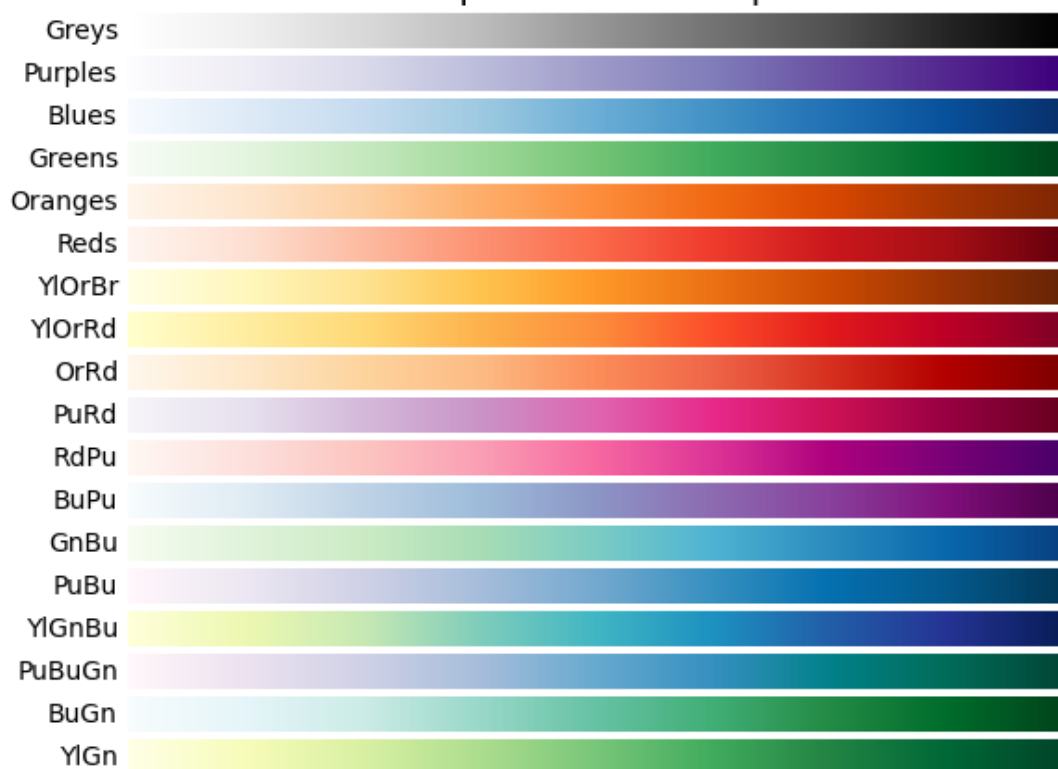
6.8 Mapy kolorów

Lista wbudowanych map kolorów: <https://matplotlib.org/stable/tutorials/colors/colormaps.html>

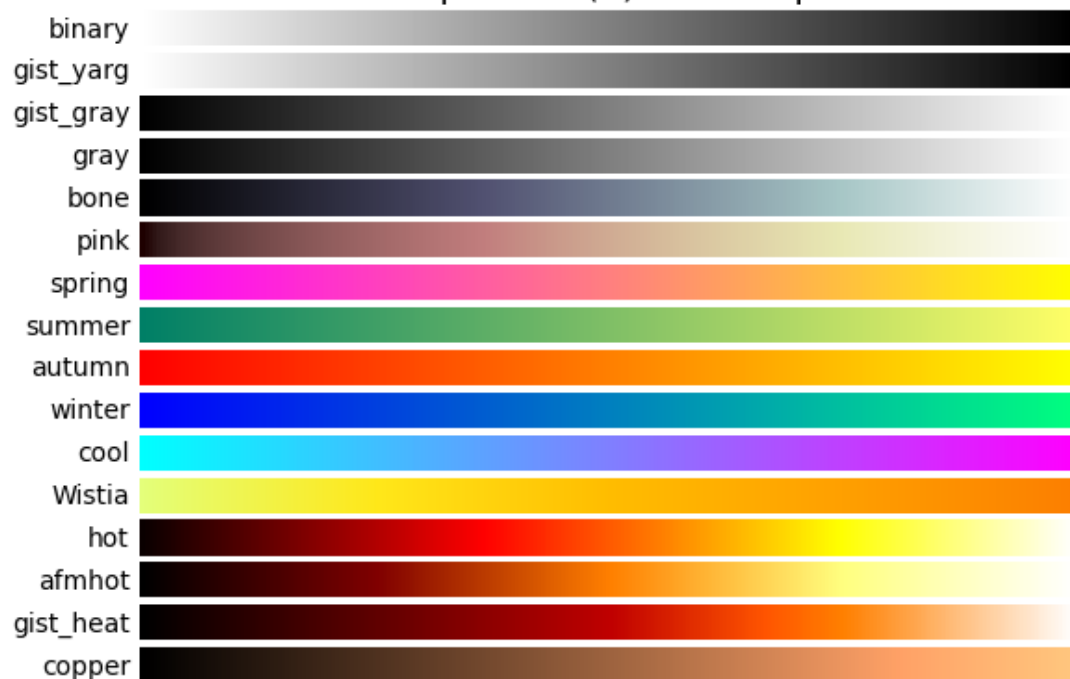
Perceptually Uniform Sequential colormaps



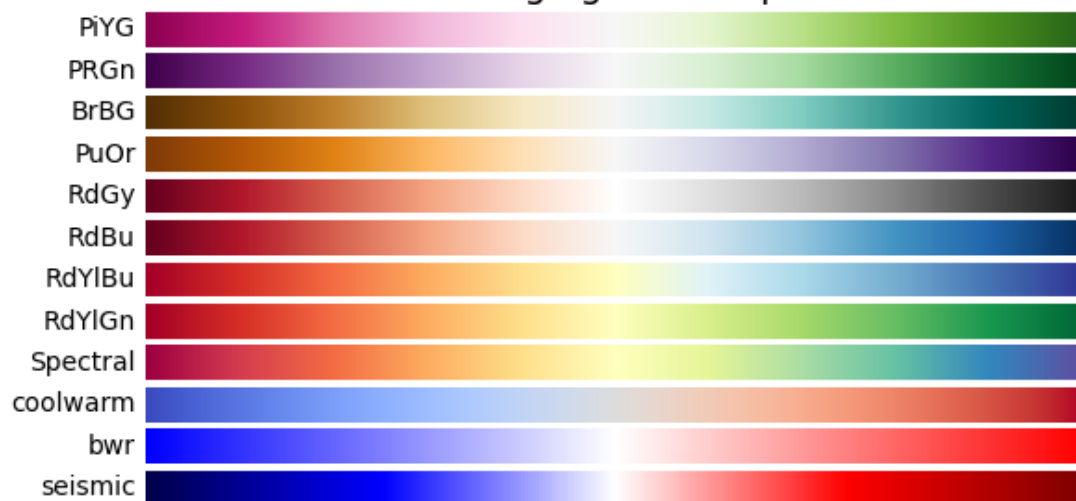
Sequential colormaps

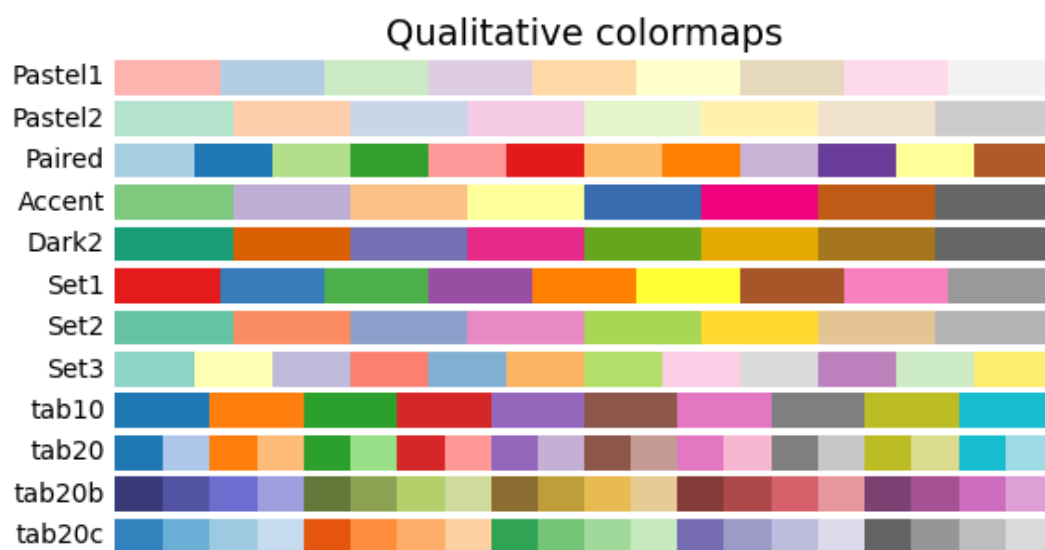
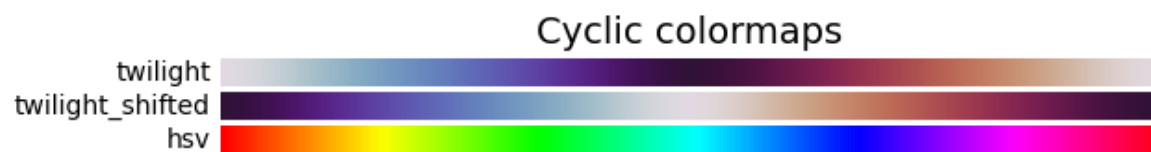


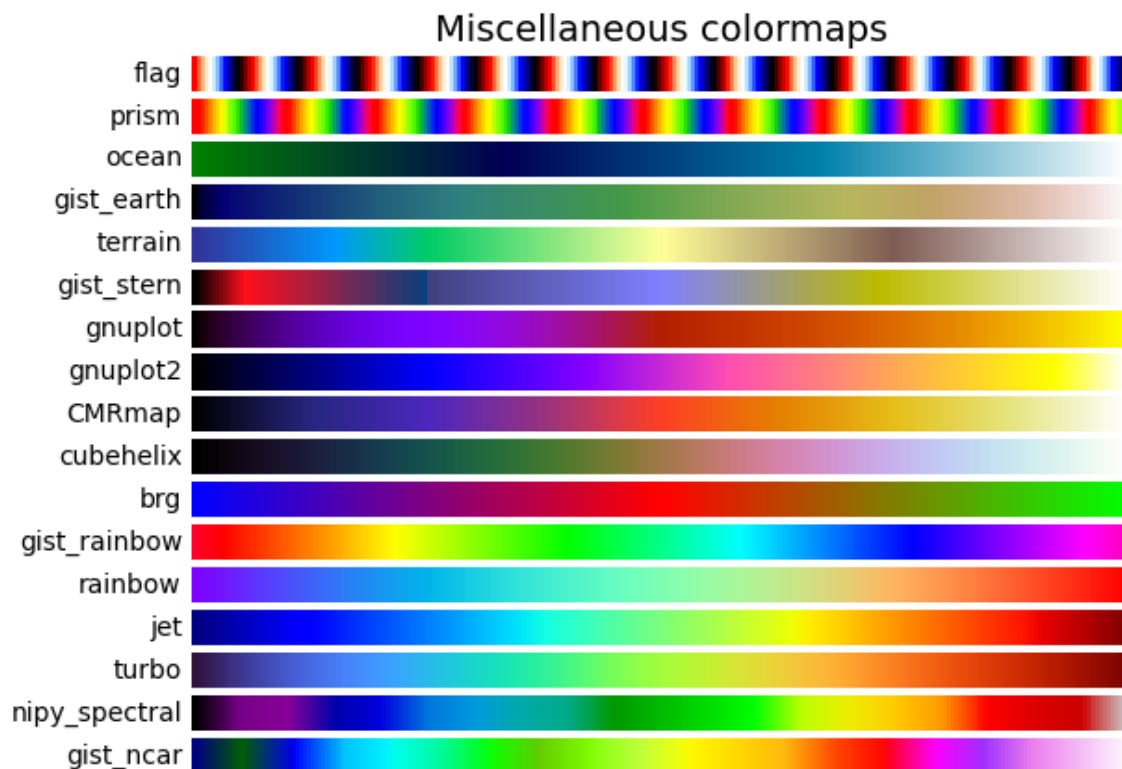
Sequential (2) colormaps



Diverging colormaps







```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

# Przykładowe dane
x = np.random.rand(50)
y = np.random.rand(50)
z = np.random.rand(50) * 100

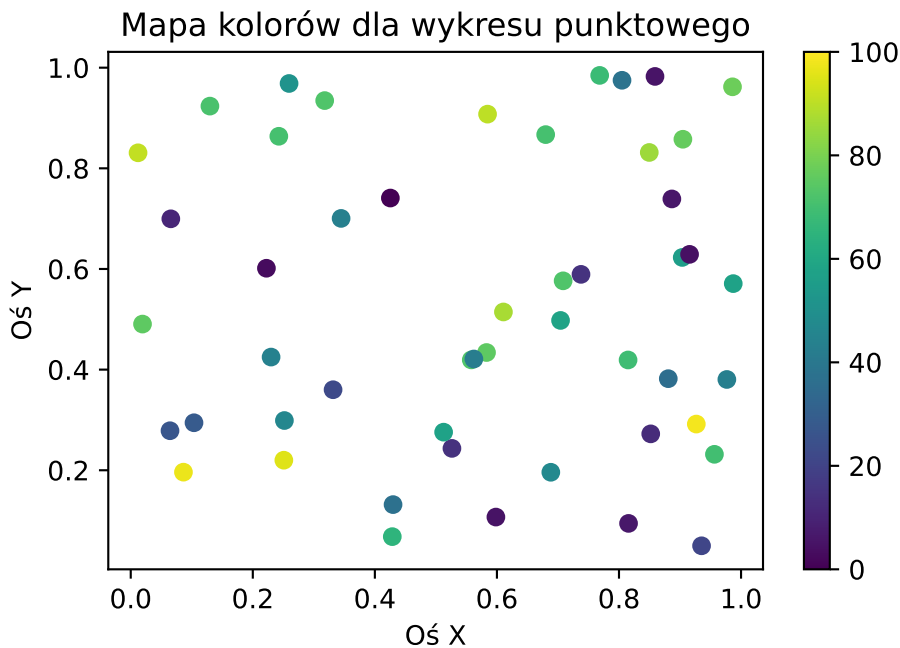
# Utworzenie mapy kolorów
norm = Normalize(vmin=0, vmax=100)
cmap = plt.cm.viridis

# Tworzenie wykresu punktowego z mapą kolorów
plt.scatter(x, y, c=z, cmap=cmap, norm=norm)
plt.colorbar()

# Dodanie etykiet osi
```

```
plt.xlabel('Oś X')
plt.ylabel('Oś Y')
plt.title('Mapa kolorów dla wykresu punktowego')

# Wyświetlenie wykresu
plt.show()
```



6.9 Markery

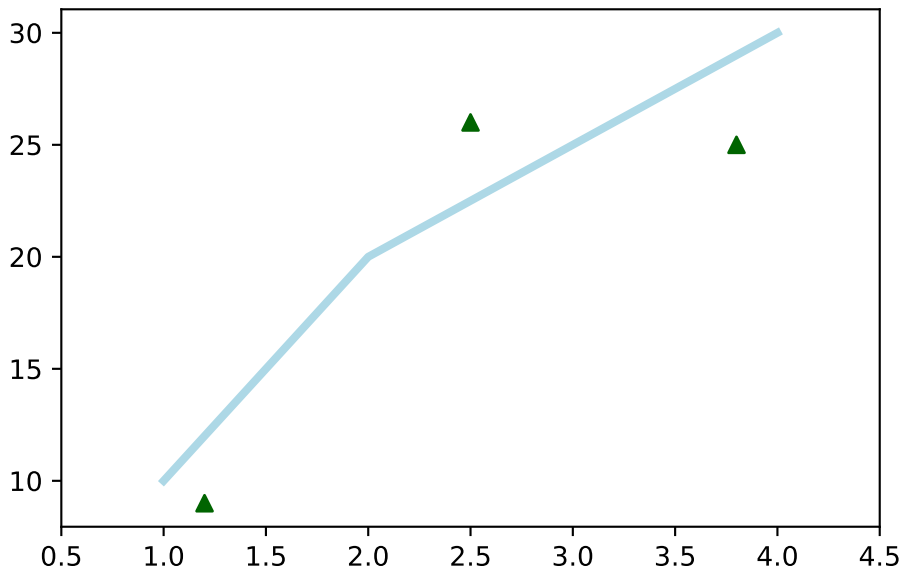
https://matplotlib.org/stable/api/markers_api.html

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3) ①
plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^') ②
plt.xlim(0.5, 4.5) ③
plt.show()
```

- ① `plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3)` - Tworzy wykres liniowy z podanymi współrzędnymi punktów (1, 10), (2, 20), (3, 25) i (4, 30). Kolor linii to jasnoniebieski (lightblue), a jej grubość wynosi 3.

- ② `plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^')` - Tworzy wykres punktowy z podanymi współrzędnymi punktów (0.3, 11), (3.8, 25), (1.2, 9) i (2.5, 26). - Kolor punktów to ciemnozielony (darkgreen), a ich kształt to trójkąty wypełnione w górę (^).
- ③ `plt.xlim(0.5, 4.5)` - Ustala zakres wartości na osi X, zaczynając od 0.5 do 4.5.



6.10 Zapis do pliku

1. PNG (Portable Network Graphics) - plik rasterowy, popularny format do zapisywania obrazów w Internecie.
2. JPEG (Joint Photographic Experts Group) - plik rasterowy, popularny format do zapisywania obrazów fotograficznych.
3. SVG (Scalable Vector Graphics) - plik wektorowy, dobrze skalujący się i zachowujący jakość na różnych rozdzielczościach.
4. PDF (Portable Document Format) - format dokumentów wektorowych, popularny w druku i przeglądaniu dokumentów.
5. EPS (Encapsulated PostScript) - plik wektorowy, często używany w publikacjach naukowych i materiałach drukowanych.
6. TIFF (Tagged Image File Format) - plik rasterowy, popularny w profesjonalnym druku i grafice.
7. WebP to nowoczesny format obrazów opracowany przez Google, który oferuje lepszą kompresję oraz niższe straty jakości w porównaniu do popularnych formatów JPEG i PNG, co przyczynia się do szybszego ładowania stron internetowych i oszczędności transferu danych.

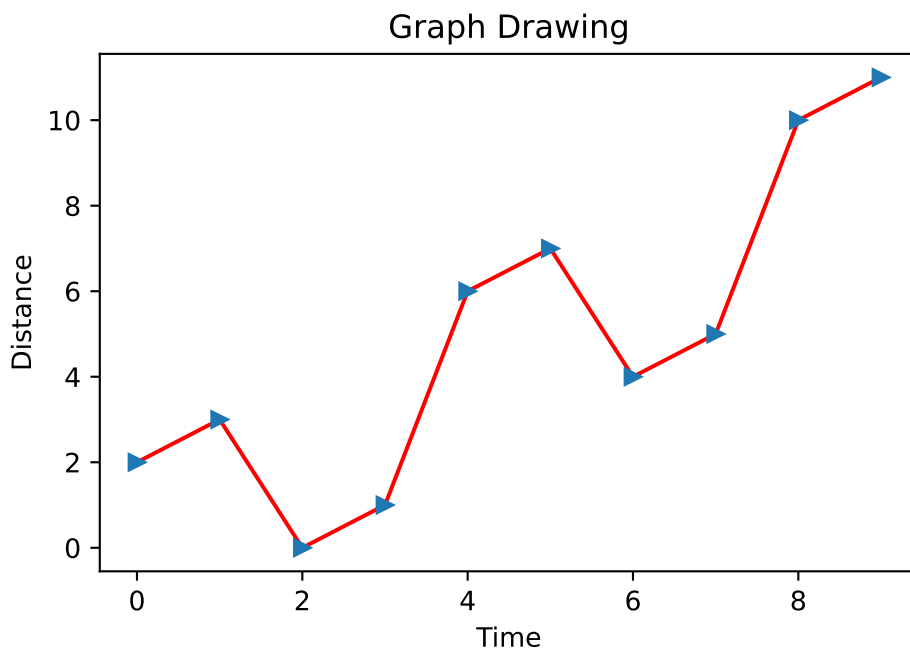

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10)
y = x ^ 2
# Labeling the Axes and Title
plt.title("Graph Drawing")
plt.xlabel("Time")
plt.ylabel("Distance")

# Formatting the line colors
plt.plot(x, y, 'r')

# Formatting the line type
plt.plot(x, y, '>')

# save in pdf formats
plt.savefig('timevsdist.pdf', format='pdf')
```



ZAPIS DO PLIKU:

```
plt.savefig('timevsdist.pdf', format='pdf')
```

↑
nazwa pliku
(jako string)

↑
rozszerzenie
(wersja bezplena)

WAŻNE!

→ DO ZAPISU JPL POTRZEBNA
BIBLIOTEKA PILLOW

→ SAVEFIG POWINIEN BYĆ WYKONYWANY
PRZED SHOW!

6.11 Linie poziome i pionowe

Funkcje `axhline` i `axvline` służą do dodawania poziomych (horyzontalnych) i pionowych (wertykalnych) linii do wykresu, odpowiednio.

`axhline` rysuje horyzontalną linię przechodzącą przez określoną wartość na osi Y, niezależnie od wartości na osi X. Składnia funkcji to `axhline(y, xmin, xmax, **kwargs)`, gdzie:

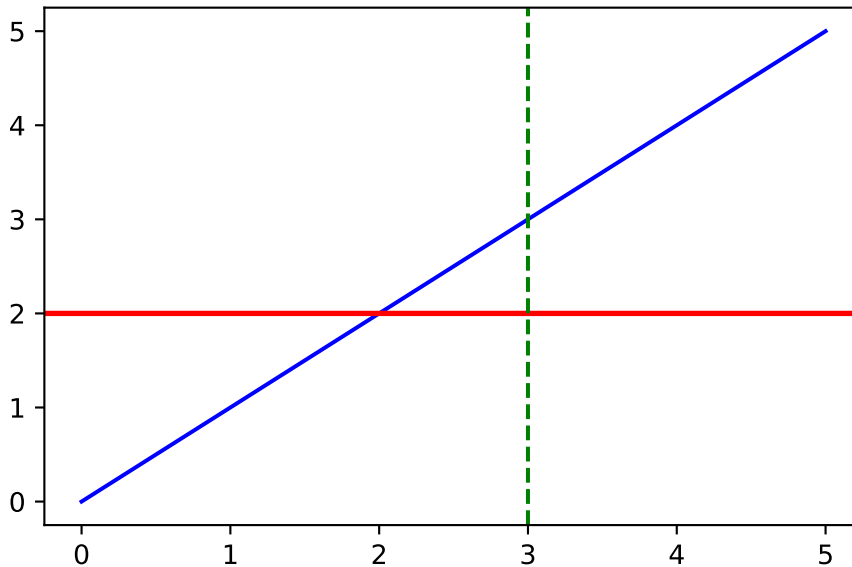
- `y` - wartość na osi Y, przez którą przechodzi linia (domyślnie 0)
- `xmin, xmax` - wartości z zakresu 0-1 określające początek i koniec linii na osi X (domyślnie 0 i 1)
- `**kwargs` - dodatkowe argumenty, takie jak `color`, `linewidth` czy `linestyle`, służące do kontrolowania wyglądu linii

`axvline` rysuje pionową linię przechodzącą przez określoną wartość na osi X, niezależnie od wartości na osi Y. Składnia funkcji to `axvline(x, ymin, ymax, **kwargs)`, gdzie:

- `x` - wartość na osi X, przez którą przechodzi linia (domyślnie 0)
- `ymin, ymax` - wartości z zakresu 0-1 określające początek i koniec linii na osi Y (domyślnie 0 i 1)
- `**kwargs` - dodatkowe argumenty, takie jak `color`, `linewidth` czy `linestyle`, służące do kontrolowania wyglądu linii

```
import matplotlib.pyplot as plt

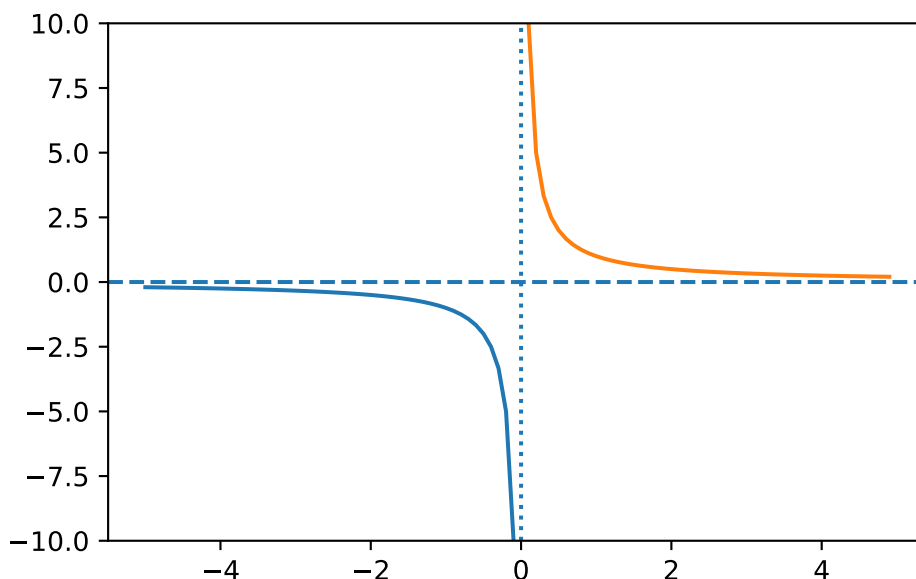
plt.plot([0, 5], [0, 5], color='blue')
plt.axhline(2, color='red', linewidth=2) # Horyzontalna linia przechodząca przez Y=2
plt.axvline(3, color='green', linestyle='--') # Pionowa linia przechodząca przez X=3, styl
plt.show()
```



W powyższym przykładzie, `axhline` rysuje czerwoną linię horyzontalną przechodzącą przez wartość 2 na osi Y, natomiast `axvline` rysuje zieloną przerywaną linię pionową przechodzącą przez wartość 3 na osi X.

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

x = np.arange(-5, 5, 0.1)
x1 = x[x < 0]
y1 = 1 / x1
plt.plot(x1, y1)
x2 = x[x > 0]
y2 = 1 / x2
plt.plot(x2, y2)
plt.ylim(-10, 10)
plt.axhline(y=0, linestyle="--")
plt.axvline(x=0, linestyle=":")
plt.show()
```



6.12 Adnotacje (tekst) na wykresie

Funkcja `annotate` służy do dodawania adnotacji (tekstu i strzałek) na wykresie w celu wyróżnienia lub zaznaczenia określonych punktów czy obszarów.

Składnia funkcji to `annotate(text, xy, xytext, arrowprops, **kwargs)`, gdzie:

- `text` - ciąg znaków reprezentujący tekst adnotacji.
- `xy` - krotka (x, y) określająca współrzędne punktu, do którego odnosimy się w adnotacji.
- `xytext` - krotka (x, y) określająca współrzędne, w których tekst adnotacji powinien się zacząć. Jeśli nie podano, tekst zostanie wyświetlony bezpośrednio przy współrzędnych xy.
- `arrowprops` - słownik zawierający opcje rysowania strzałki, takie jak `arrowstyle`, `color` czy `linewidth`. Jeśli nie podano, strzałka nie zostanie narysowana.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania tekstu, takie jak `fontsize`, `color` czy `fontweight`.

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 4, 9, 16], marker='o', linestyle='-', color='blue')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

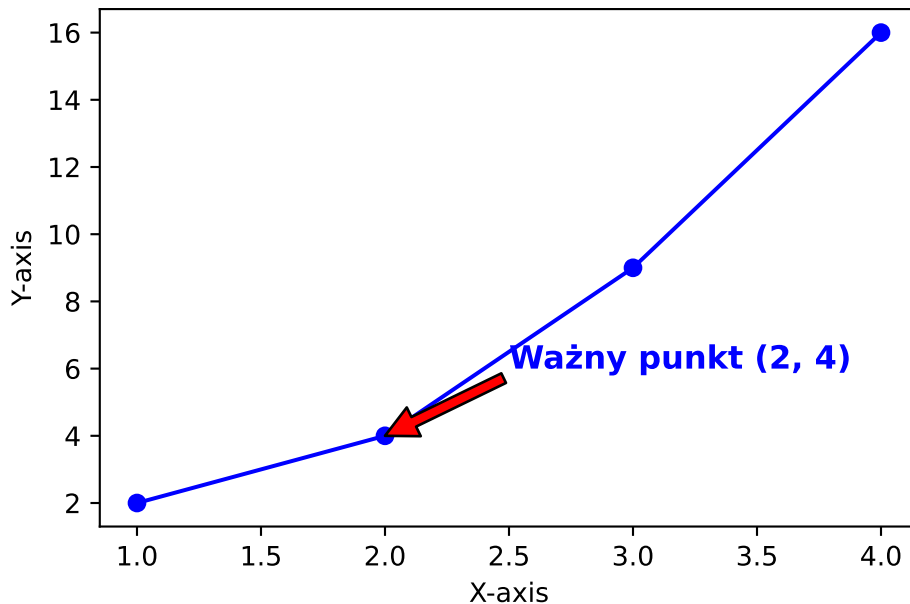
plt.annotate('Ważny punkt (2, 4)',
```

```

xy=(2, 4), # Współrzędne punktu do zaznaczenia
xytext=(2.5, 6), # Współrzędne początku tekstu
arrowprops=dict(facecolor='red'), # Właściwości strzałki (kolor)
fontsize=12, # Rozmiar czcionki
color='blue', # Kolor tekstu
fontweight='bold') # Grubość czcionki

plt.show()

```



Jeśli chcesz dodać adnotację tylko z tekstem, składnia funkcji to `annotate(text, xy, **kwargs)`, gdzie:

- `text` - ciąg znaków reprezentujący tekst adnotacji.
- `xy` - krotka (x, y) określająca współrzędne, w których tekst adnotacji powinien się zacząć.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania tekstu, takie jak `fontsize`, `color`, `fontweight` czy `horizontalalignment`.

```

import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 4, 9, 16], marker='o', linestyle='-', color='blue')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

plt.annotate('Ważny punkt (2, 4)',

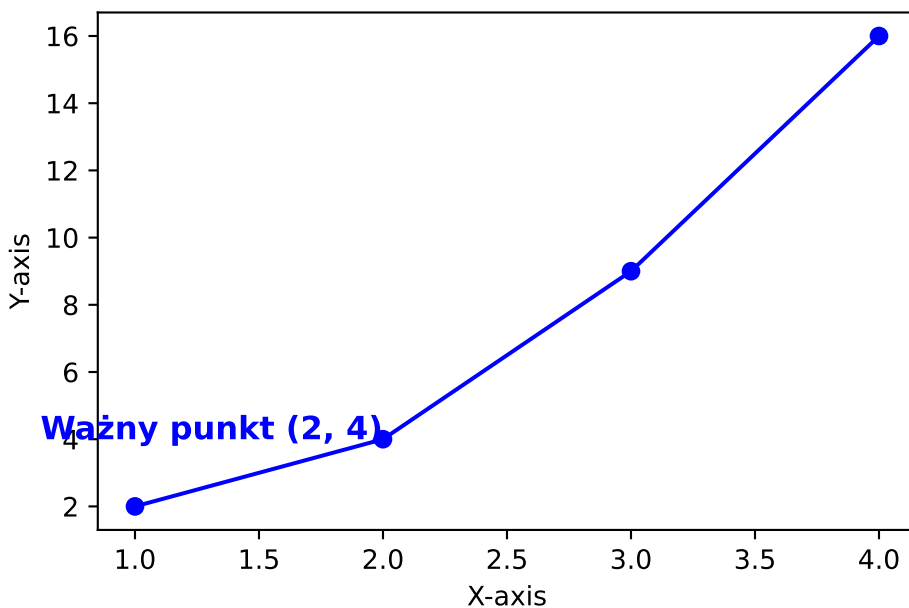
```

```

xy=(2, 4), # Współrzędne początku tekstu
fontsize=12, # Rozmiar czcionki
color='blue', # Kolor tekstu
fontweight='bold', # Grubość czcionki
horizontalalignment='right') # Wyrównanie tekstu do prawej strony

plt.show()

```



6.13 Etykiety osi

Funkcje `xlabel` i `ylabel` służą do dodawania etykiet osi X i Y na wykresie, odpowiednio. Etykiety osi pomagają w lepszym zrozumieniu prezentowanych danych, wskazując, jakie wartości są reprezentowane na poszczególnych osiach.

Składnia funkcji to `xlabel(label, **kwargs)` lub `ylabel(label, **kwargs)`, gdzie:

- `label` - ciąg znaków reprezentujący tekst etykiety osi.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania etykiety, takie jak `fontsize`, `color`, `fontweight` czy `horizontalalignment`.

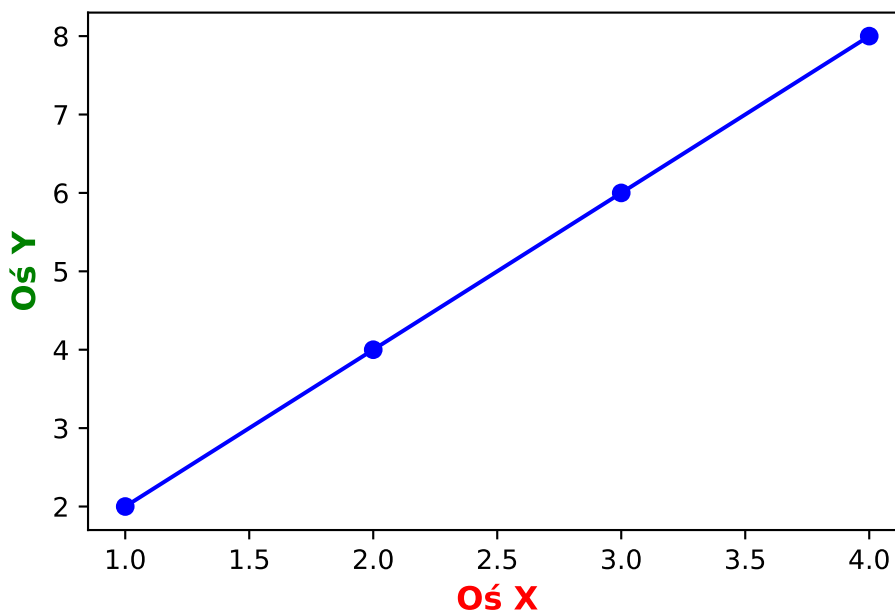
```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [2, 4, 6, 8]

plt.plot(x, y, marker='o', linestyle='-', color='blue')

plt.xlabel('Oś X', fontsize=12, color='red', fontweight='bold')
plt.ylabel('Oś Y', fontsize=12, color='green', fontweight='bold')

plt.show()
```



Funkcja `annotate` pozwala na użycie składni LaTeX w tekście adnotacji, co jest szczególnie przydatne, gdy chcemy dodać na wykresie równania matematyczne lub symbole. Aby użyć składni LaTeX, należy umieścić tekst w znacznikach dolara (\$).

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

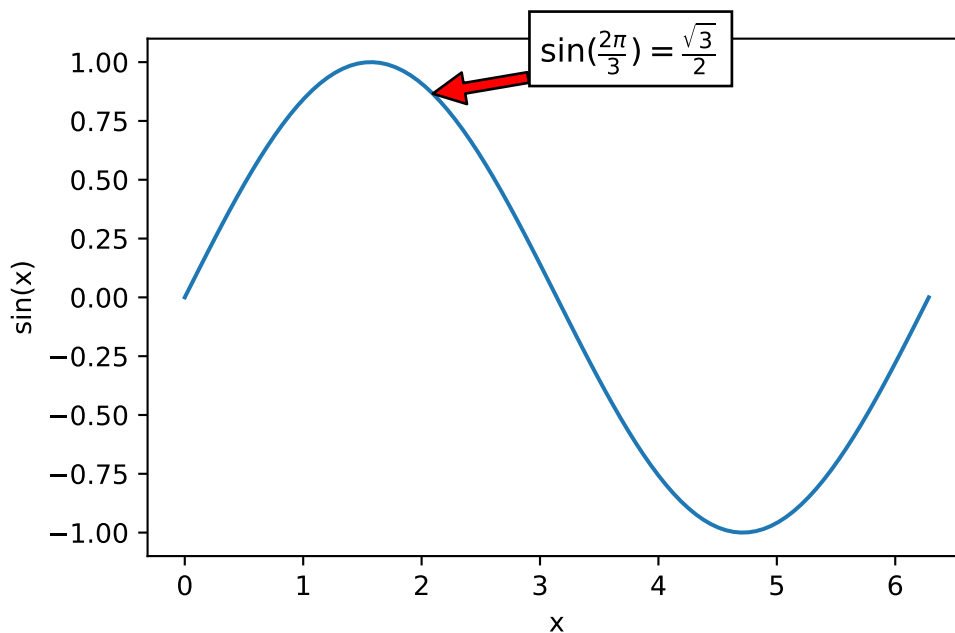
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

plt.plot(x, y)
```

```
plt.xlabel('x')
plt.ylabel('sin(x)')

# Adnotacja z tekstem w składni LaTeX
plt.annotate(r'$\sin(\frac{2\pi}{3})=\frac{\sqrt{3}}{2}$',
            xy=(2 * np.pi / 3, np.sqrt(3) / 2), # Współrzędne punktu do zaznaczenia
            xytext=(3, 1.0),                     # Współrzędne początku tekstu
            fontsize=12,                          # Rozmiar czcionki
            arrowprops=dict(facecolor='red'),      # Właściwości strzałki (kolor)
            bbox=dict(facecolor='white'))          # Ramka wokół tekstu (kolor tła)

plt.show()
```



6.14 Etykiety podziałki osi

Funkcje `xticks` i `yticks` służą do manipulowania etykietami osi X i Y oraz wartościami na osi, odpowiednio. Pozwalają na kontrolowanie wyświetlania etykiet, odstępów między nimi oraz formatowania.

`xticks` manipuluje etykietami i wartościami na osi X, a `yticks` na osi Y. Składnia funkcji to `xticks(ticks, labels, **kwargs)` lub `yticks(ticks, labels, **kwargs)`, gdzie:

- `ticks` - lista wartości, dla których mają być umieszczone etykiety na osi. Jeśli nie podano, pozostają aktualne wartości.
- `labels` - lista ciągów znaków, które mają być użyte jako etykiety dla wartości z listy `ticks`. Jeśli nie podano, zostaną użyte domyślne etykiety.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania etykiet, takie jak `fontsize`, `color`, `fontweight` czy `rotation`.

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

x = np.arange(0, 10, 0.1)
y = np.sin(x)

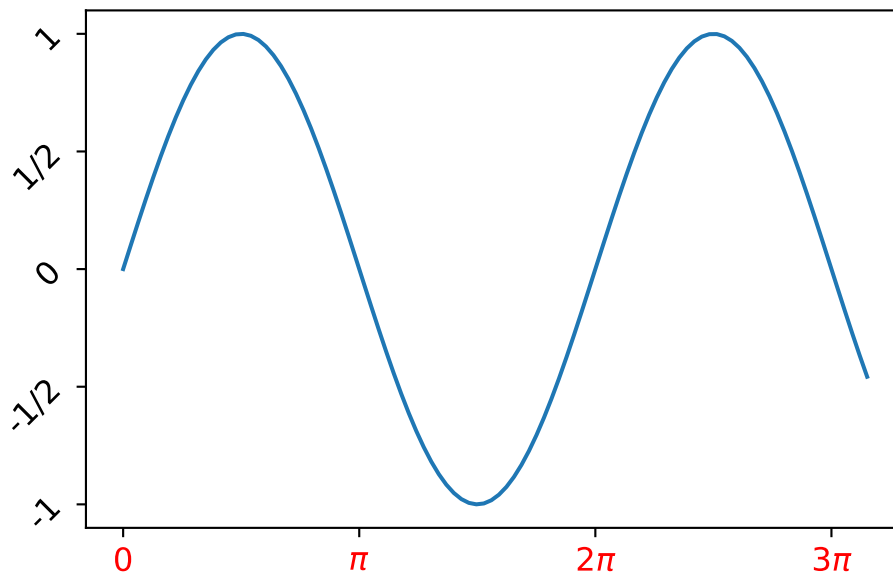
plt.plot(x, y)

xtick_vals = [0, np.pi, 2 * np.pi, 3 * np.pi]
xtick_labels = ['0', r'$\pi$', r'$2\pi$', r'$3\pi$']

ytick_vals = [-1, -0.5, 0, 0.5, 1]
ytick_labels = ['-1', '-1/2', '0', '1/2', '1']

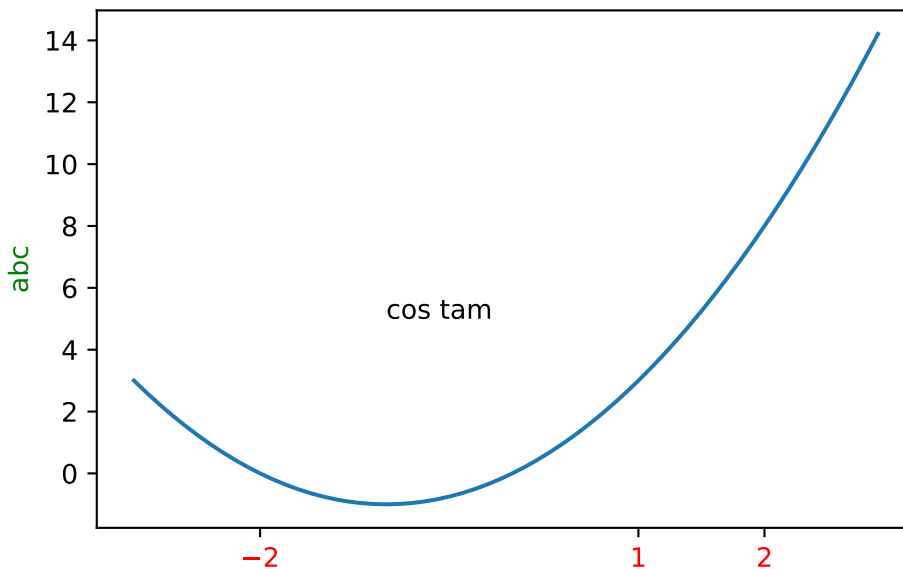
plt.xticks(xtick_vals, xtick_labels, fontsize=12, color='red')
plt.yticks(ytick_vals, ytick_labels, fontsize=12, rotation=45)

plt.show()
```



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

x = np.arange(-3, 3, 0.1)
y = x ** 2 + 2 * x
plt.plot(x, y)
plt.annotate(xy=(-1, 5), text="cos tam")
plt.xticks([-2, 1, 2], color="red")
plt.ylabel("abc", color="green")
plt.show()
```



6.15 Wykres kołowy

Wykres kołowy (pie chart) jest stosowany, gdy chcemy przedstawić proporcje różnych kategorii lub segmentów w stosunku do całości. Jest szczególnie użyteczny, gdy mamy niewielką liczbę kategorii (zazwyczaj nie więcej niż 5-7) oraz gdy dane są jakościowe (kategoryczne). Wykres kołowy pozwala na wizualne zrozumienie udziałów procentowych poszczególnych kategorii w ramach całego zbioru danych.

Przykłady danych, dla których stosuje się wykres kołowy:

1. Struktura wydatków domowych, gdzie kategorie to: mieszkanie, jedzenie, transport, rozrywka, inne.
2. Procentowy udział w rynku różnych firm w danej branży.

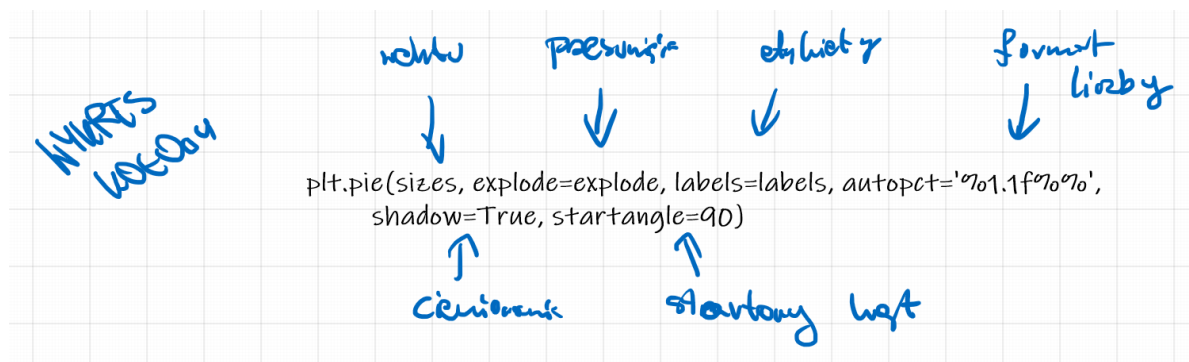
3. Rozkład głosów na partie polityczne w wyborach.
4. Procentowy udział różnych rodzajów energii w produkcji energii elektrycznej (węgiel, gaz, energia odnawialna, energia jądrowa itp.).

Chociaż wykresy kołowe mają swoje zastosowania, są również krytykowane za ograniczoną precyzję w ocenie proporcji. Dlatego często zaleca się stosowanie innych rodzajów wykresów, takich jak słupkowe (bar chart) czy stosunkowe (stacked bar chart), które mogą być bardziej przejrzyste i precyzyjne w porównywaniu wartości między kategoriami.

Funkcja `pie` służy do tworzenia wykresów kołowych. Pozwala na wizualne przedstawienie proporcji różnych segmentów względem całości.

Składnia funkcji to `plt.pie(x, explode=None, labels=None, colors=None, autopct=None, shadow=False, startangle=0, counterclock=True)`, gdzie:

- `x` - lista wartości numerycznych, reprezentująca dane dla każdego segmentu. Funkcja `pie` automatycznie obliczy procentowe udziały każdej wartości względem sumy wszystkich wartości.
- `explode` - lista wartości, które określają, czy (i jak bardzo) każdy segment ma być oddzielony od środka wykresu. Wartość 0 oznacza brak oddzielenia, a wartości większe oznaczają większe oddzielenie.
- `labels` - lista ciągów znaków, które będą używane jako etykiety segmentów.
- `colors` - lista kolorów dla poszczególnych segmentów.
- `autopct` - formatowanie procentów, które mają być wyświetlane na wykresie (np. `'%1.1f%%'`).
- `shadow` - wartość logiczna (True/False), która określa, czy wykres ma mieć cień. Domyślnie ustawione na False.
- `startangle` - kąt początkowy wykresu kołowego, mierzony w stopniach przeciwnie do ruchu wskazówek zegara od osi X.
- `counterclock` - wartość logiczna (True/False), która określa, czy segmenty mają być rysowane zgodnie z ruchem wskazówek zegara. Domyślnie ustawione na True.



```
import matplotlib.pyplot as plt

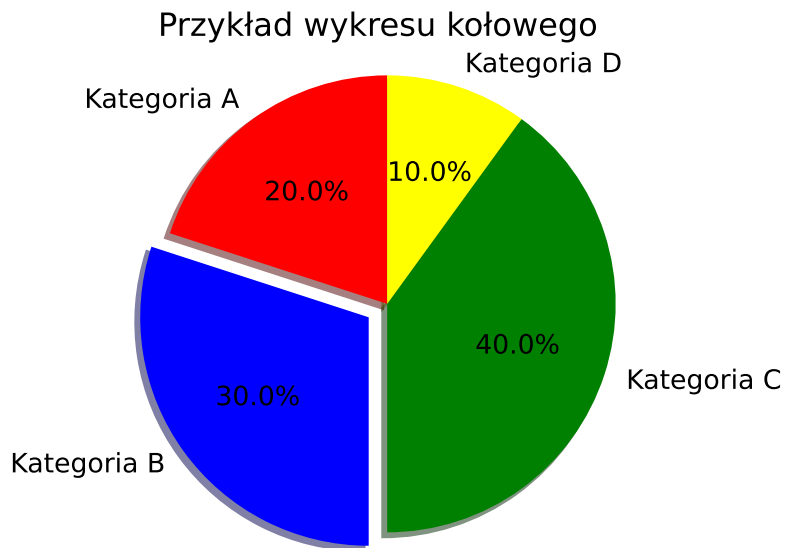
# Dane
sizes = [20, 30, 40, 10]
labels = ['Kategoria A', 'Kategoria B', 'Kategoria C', 'Kategoria D']
colors = ['red', 'blue', 'green', 'yellow']
explode = (0, 0.1, 0, 0) # Wyróżnienie segmentu Kategoria B

# Tworzenie wykresu kołowego
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True)

# Dodanie tytułu
plt.title('Przykład wykresu kołowego')

# Równomierne skalowanie osi X i Y, aby koło było okrągłe
plt.axis('equal')

plt.show()
```



```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
```

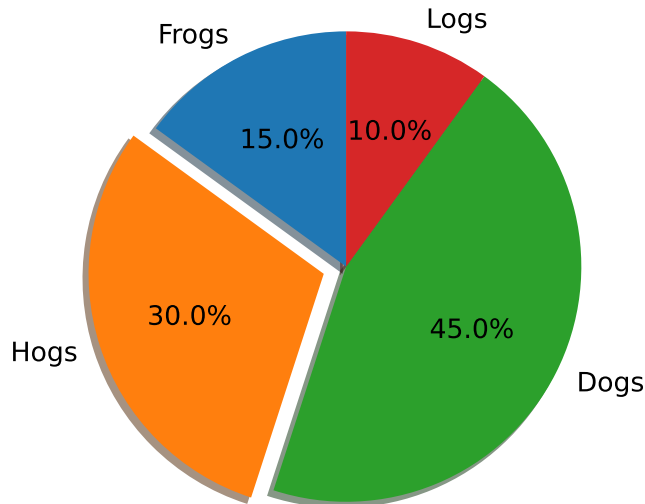
```

explode = [0, 0.1, 0, 0] # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

```



```

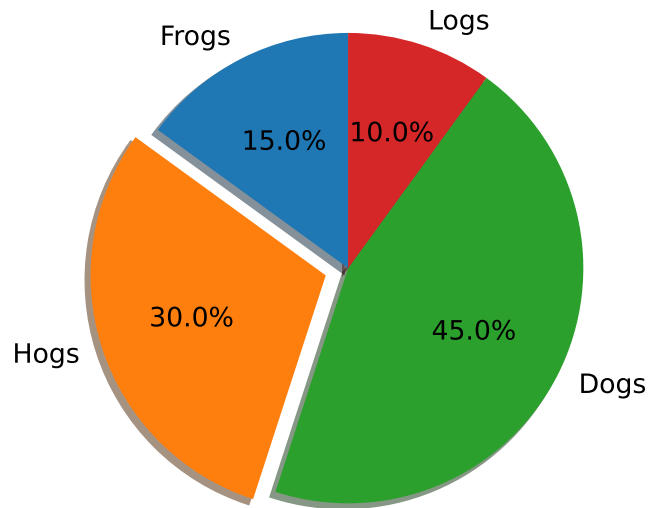
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
explode = [0, 0.1, 0, 0] # only "explode" the 2nd slice (i.e. 'Hogs')

plt.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.axis('equal')

plt.show()

```



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

# Dane
sizes = [20, 30, 40, 10]
labels = ['Kategoria A', 'Kategoria B', 'Kategoria C', 'Kategoria D']
n = len(sizes)

# Tworzenie mapy kolorów
cmap = plt.get_cmap('viridis')
colors = [cmap(i) for i in np.linspace(0, 1, n)]

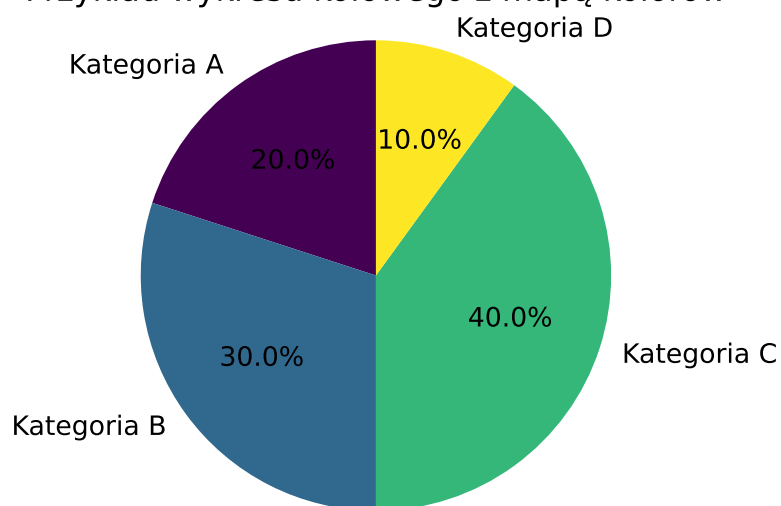
# Tworzenie wykresu kołowego
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)

# Dodanie tytułu
plt.title('Przykład wykresu kołowego z mapą kolorów')

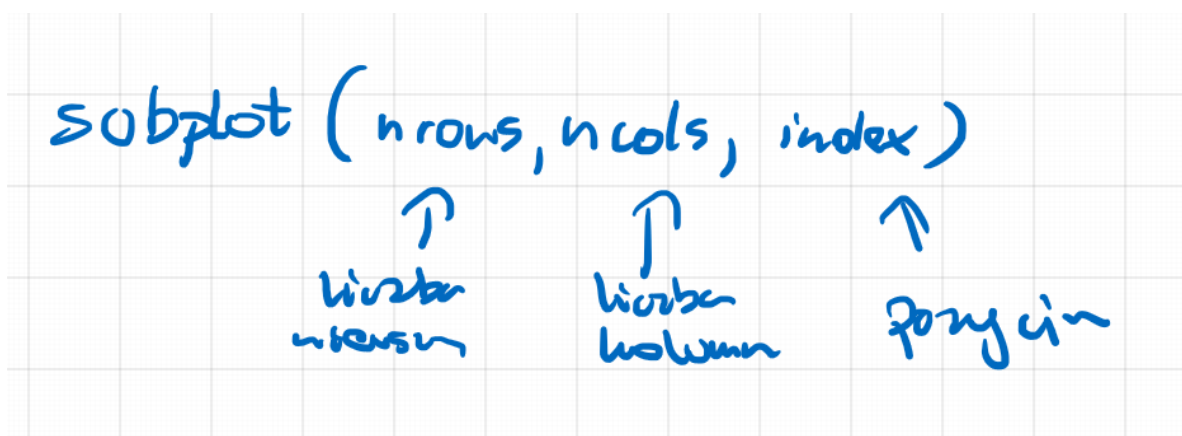
# Równomierne skalowanie osi X i Y, aby koło było okrągłe
plt.axis('equal')

plt.show()
```

Przykład wykresu kołowego z mapą kolorów



6.16 Podwykresy



Funkcja `subplot` pozwala na tworzenie wielu wykresów w pojedynczym oknie lub figurze. Dzięki temu można porównać różne wykresy, które mają wspólny kontekst lub prezentować różne aspekty danych.

Składnia funkcji to `plt.subplot(nrows, ncols, index, **kwargs)`, gdzie:

- `nrows` - liczba wierszy w siatce wykresów.
- `ncols` - liczba kolumn w siatce wykresów.

- `index` - indeks bieżącego wykresu, który ma być utworzony (indeksacja zaczyna się od 1). Indeksy są numerowane wierszami, tzn. kolejny wykres w rzędzie będzie miał indeks o jeden większy.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania wykresu.

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

x = np.linspace(0, 2 * np.pi, 100)

# Tworzenie siatki wykresów 2x2
# Pierwszy wykres (w lewym górnym rogu)
plt.subplot(2, 2, 1)
plt.plot(x, np.sin(x))
plt.title('sin(x)')

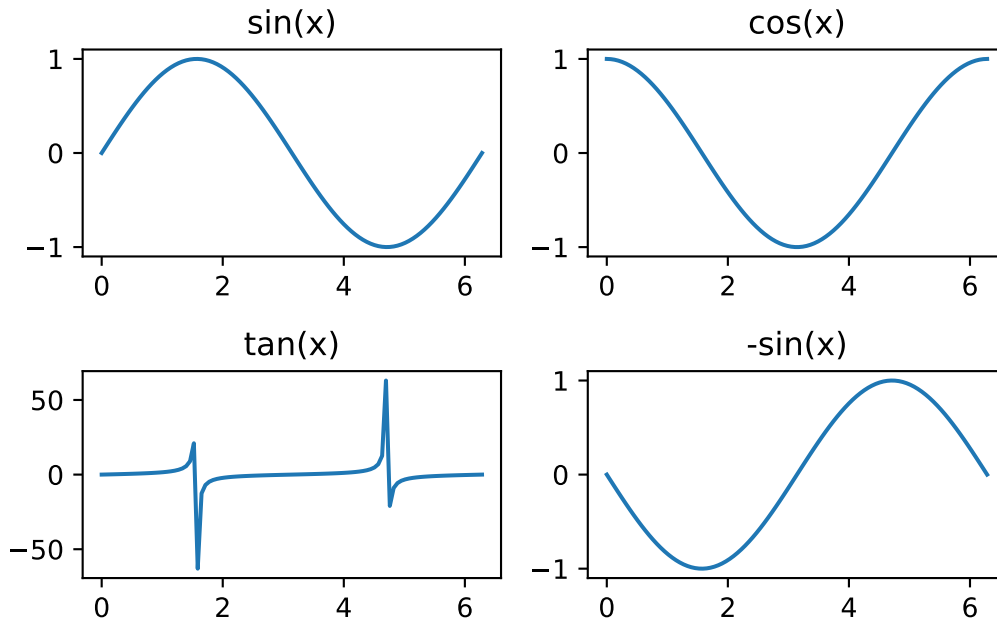
# Drugi wykres (w prawym górnym rogu)
plt.subplot(2, 2, 2)
plt.plot(x, np.cos(x))
plt.title('cos(x)')

# Trzeci wykres (w lewym dolnym rogu)
plt.subplot(2, 2, 3)
plt.plot(x, np.tan(x))
plt.title('tan(x)')

# Czwarty wykres (w prawym dolnym rogu)
plt.subplot(2, 2, 4)
plt.plot(x, -np.sin(x))
plt.title('-sin(x)')

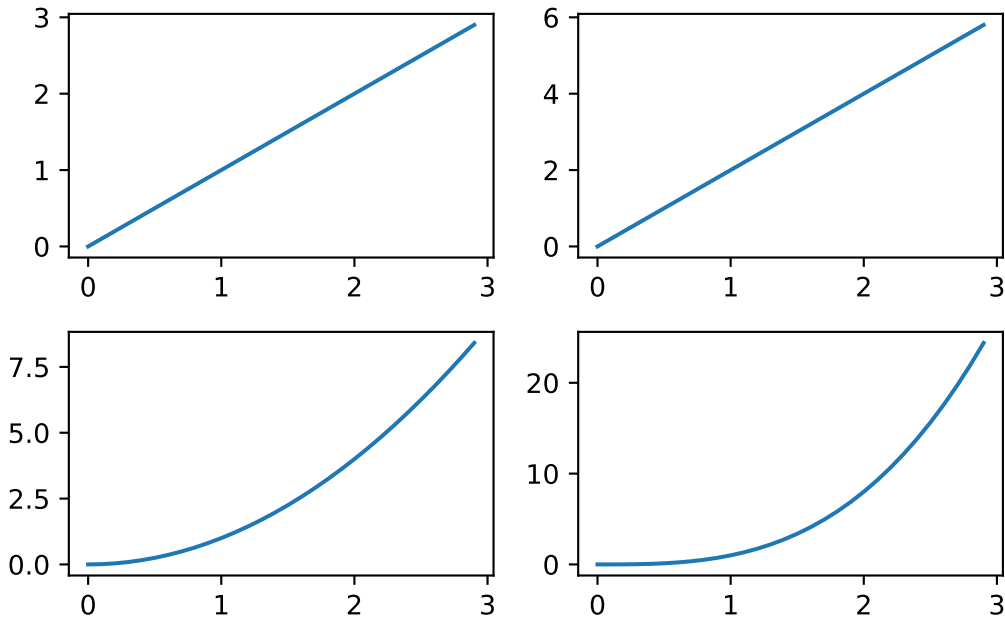
# Dopasowanie odstępów między wykresami
plt.tight_layout()

# Wyświetlenie wykresów
plt.show()
```

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

x = np.arange(0, 3, 0.1)
plt.subplot(2, 2, 1)
plt.plot(x, x)
plt.subplot(2, 2, 2)
plt.plot(x, x * 2)
plt.subplot(2, 2, 3)
plt.plot(x, x * x)
plt.subplot(2, 2, 4)
plt.plot(x, x ** 3)
plt.tight_layout()
plt.show()
```



6.17 Siatka

Funkcja `grid` pozwala na dodanie siatki na wykresie, co ułatwia ocenę wartości na osiach i ich porównywanie. Można kontrolować kolor, grubość i styl linii siatki, a także określać, które osie mają mieć siatkę.

Składnia funkcji to `plt.grid(b=None, which='major', axis='both', **kwargs)`, gdzie:

- **b** - wartość logiczna (True/False), która określa, czy siatka ma być wyświetlana. Domyślnie ustawione na `None`, co oznacza, że Matplotlib automatycznie określa, czy siatka powinna być wyświetlana na podstawie konfiguracji.
- **which** - określa, które linie siatki mają być wyświetlane: 'major' (tylko linie siatki dla głównych podziałek), 'minor' (linie siatki dla podziałek pomocniczych), lub 'both' (domyślnie - linie siatki dla obu rodzajów podziałek).
- **axis** - określa, które osie mają mieć siatkę: 'both' (obie osie), 'x' (tylko oś X), lub 'y' (tylko oś Y).
- ****kwargs** - dodatkowe argumenty dotyczące formatowania siatki.

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

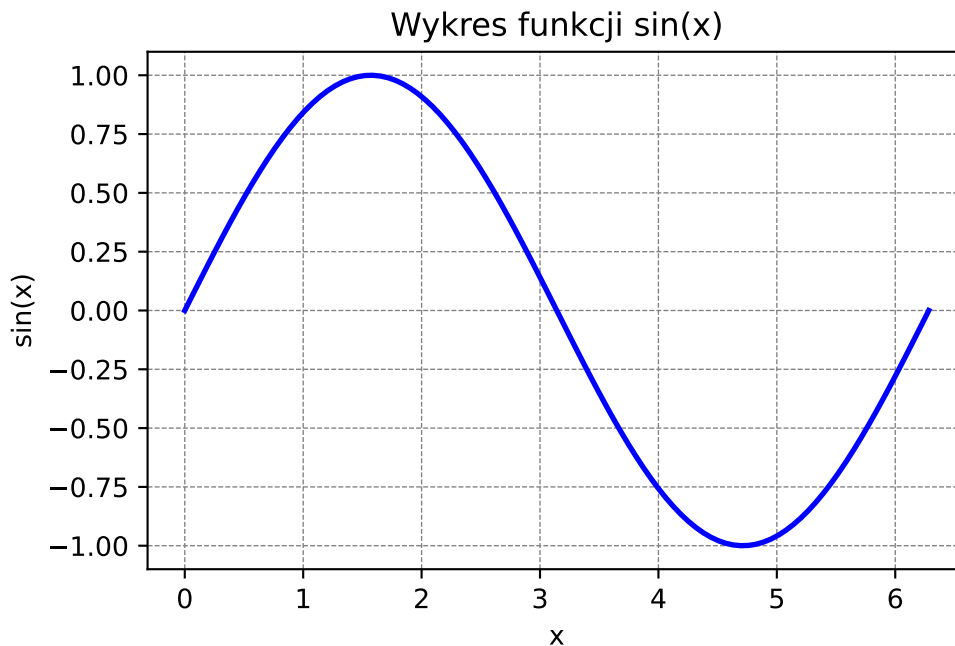
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
```

```
# Tworzenie wykresu
plt.plot(x, y, color='blue', linewidth=2)

# Dodanie siatki
plt.grid(True, which='both', color='gray', linewidth=0.5, linestyle='--')

# Dodanie tytułu i etykiet osi
plt.title('Wykres funkcji sin(x)')
plt.xlabel('x')
plt.ylabel('sin(x)')

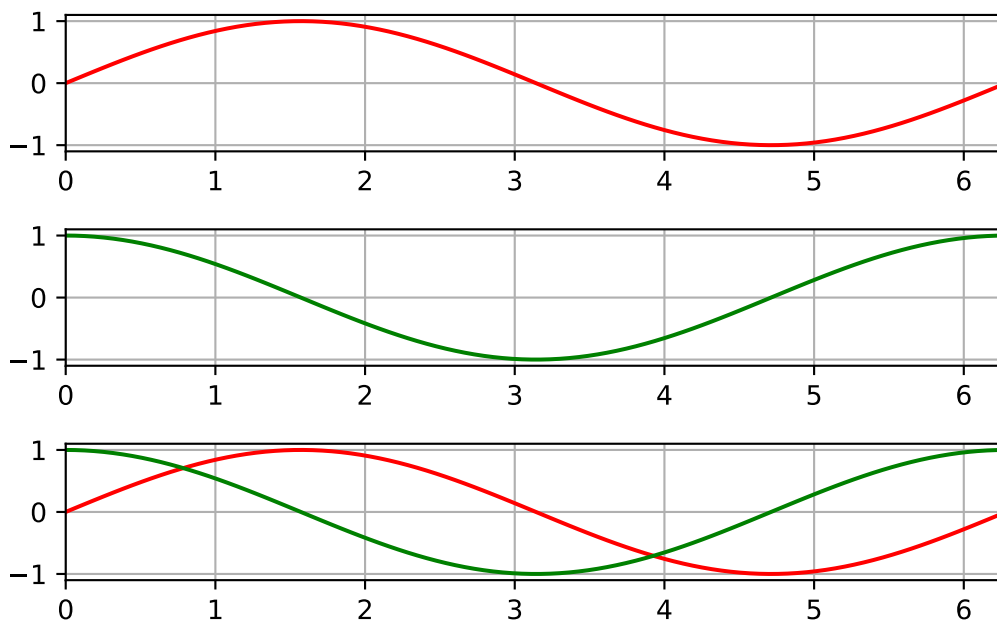
# Wyświetlenie wykresu
plt.show()
```



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

x = np.linspace(0, np.pi * 2, 100)
plt.subplot(3, 1, 1)
plt.plot(x, np.sin(x), 'r')
plt.grid(True)
plt.xlim(0, np.pi * 2)
```

```
plt.subplot(3, 1, 2)
plt.plot(x, np.cos(x), 'g')
plt.grid(True)
plt.xlim(0, np.pi * 2)
plt.subplot(3, 1, 3)
plt.plot(x, np.sin(x), 'r', x, np.cos(x), 'g')
plt.grid(True)
plt.xlim(0, np.pi * 2)
plt.tight_layout()
plt.savefig("fig3.png", dpi=72)
plt.show()
```



6.18 Wykres dwuosiowy

Funkcja `twinx` w bibliotece Matplotlib pozwala na utworzenie drugiej osi Y, która będzie współdzielić oś X z pierwszą osią Y. Dzięki temu, można w prosty sposób przedstawić dwie serie danych, które są mierzone w różnych jednostkach, ale mają wspólną zmienną niezależną.

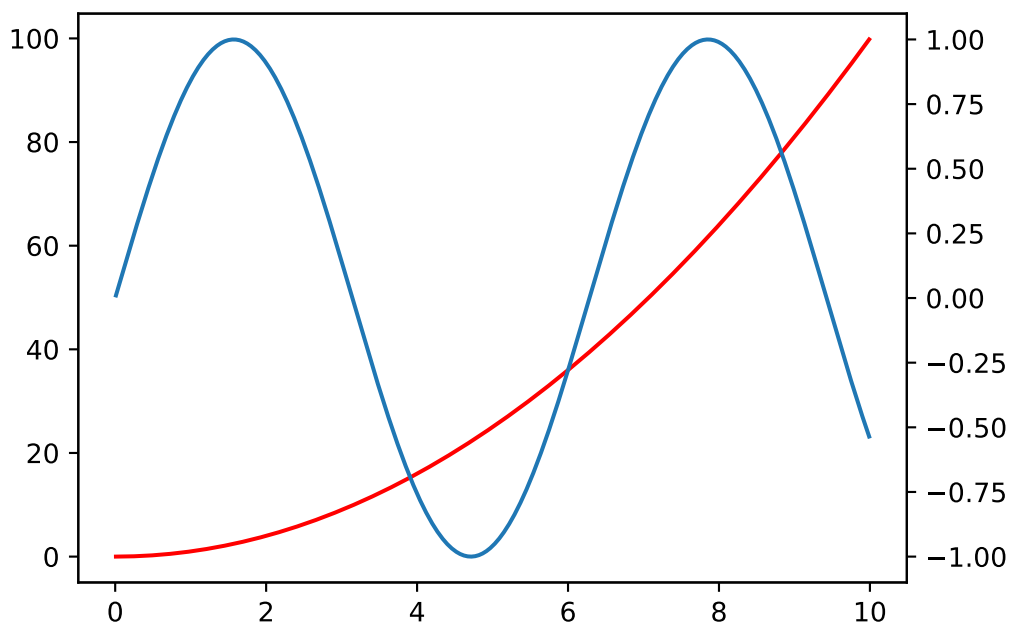
Składnia funkcji to `twinx(ax=None, **kwargs)`, gdzie:

- `ax` - obiekt Axes, który ma być użyty do tworzenia nowej osi Y. Domyślnie ustawione na `None`, co oznacza, że będzie tworzona nowa oś Y.

- ****kwargs** - dodatkowe argumenty dotyczące formatowania nowej osi Y.

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

fig, ax1 = plt.subplots()
x = np.arange(0.01, 10.0, 0.01)
y = x ** 2
ax1.plot(x, y, 'r')
ax2 = ax1.twinx()
y2 = np.sin(x)
ax2.plot(x, y2)
fig.tight_layout()
plt.show()
```



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

fig, ax1 = plt.subplots()
t = np.arange(0.01, 10.0, 0.01)
s1 = np.exp(t)
ax1.plot(t, s1, 'b-')
ax1.set_xlabel('time (s)')
```

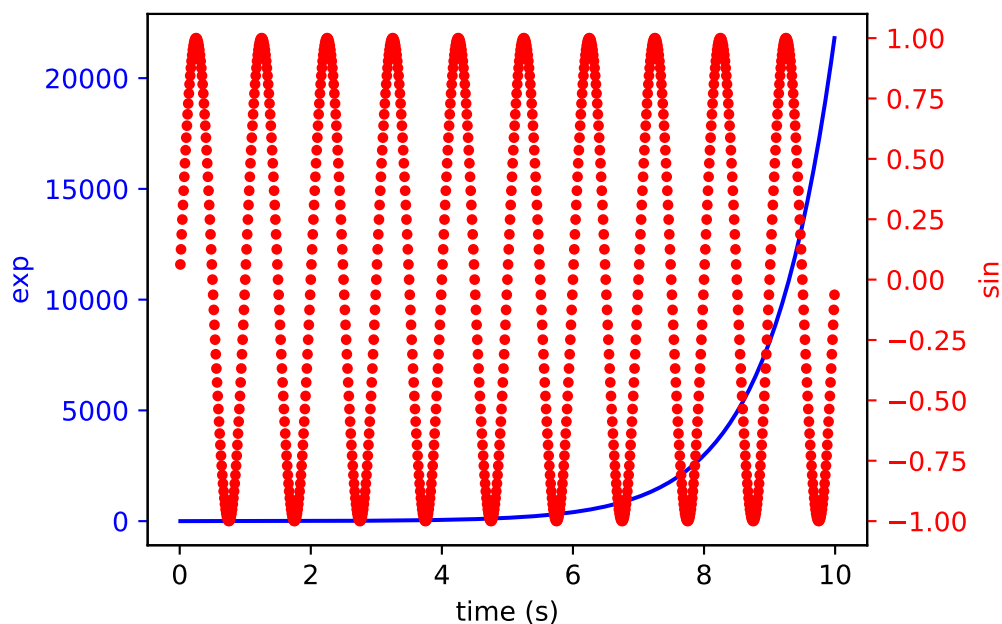
```

ax1.set_ylabel('exp', color='b')
ax1.tick_params('y', colors='b')

ax2 = ax1.twinx()
s2 = np.sin(2 * np.pi * t)
ax2.plot(t, s2, 'r.')
ax2.set_ylabel('sin', color='r')
ax2.tick_params('y', colors='r')

fig.tight_layout()
plt.show()

```



6.19 Wykres słupkowy

Wykres słupkowy jest stosowany do przedstawiania danych kategoryalnych lub dyskretnych. Jest to powszechnie używany rodzaj wykresu, który pomaga wizualnie porównać wartości lub ilości dla różnych kategorii. Oto kilka typów danych, dla których wykres słupkowy może być stosowany:

1. Częstości: Wykres słupkowy jest używany do przedstawiania liczby wystąpień różnych kategorii, takich jak wyniki ankiety, preferencje konsumentów lub różne grupy ludności.

2. Proporcje: Można go stosować do przedstawiania udziału procentowego poszczególnych kategorii w całości, np. udział rynkowy różnych firm, procentowe wyniki testów czy procentowy rozkład ludności według wieku.
3. Wartości liczbowe: Wykres słupkowy może przedstawiać wartości liczbowe związane z różnymi kategoriami, np. sprzedaż produktów, przychody z różnych źródeł czy średnią temperaturę w różnych miastach.
4. Danych szeregów czasowych: Wykres słupkowy może być również używany do przedstawiania danych szeregów czasowych w przypadku, gdy zmiany występują w regularnych odstępach czasu, np. roczna sprzedaż, miesięczne opady czy tygodniowe przychody.

Warto zauważyć, że wykresy słupkowe są odpowiednie, gdy mamy do czynienia z niewielką liczbą kategorii, ponieważ zbyt wiele słupków na wykresie może sprawić, że stanie się on trudny do interpretacji. W takich przypadkach warto rozważyć inne typy wykresów, takie jak wykres kołowy lub stosunkowy.

Funkcja `bar` w bibliotece Matplotlib służy do tworzenia wykresów słupkowych (bar chart). Wykresy słupkowe są często stosowane, gdy chcemy porównać wartości różnych kategorii.

Składnia funkcji to `plt.bar(x, height, width=0.8, bottom=None, align='center', data=None, **kwargs)`, gdzie:

- `x` - pozycje słupków na osi X. Może to być sekwencja wartości numerycznych lub lista etykiet, które będą umieszczone na osi X.
- `height` - wysokość słupków.
- `width` - szerokość słupków.
- `bottom` - położenie dolnej krawędzi słupków. Domyślnie ustawione na `None`, co oznacza, że słupki zaczynają się od zera.
- `align` - sposób wyśrodkowania słupków wzdłuż osi X. Domyślnie ustawione na `'center'`.
- `data` - obiekt `DataFrame`, który zawiera dane do wykresu.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania wykresu, takie jak kolor, przezroczystość, etykiety osi, tytuł i legendę.

```
import matplotlib.pyplot as plt

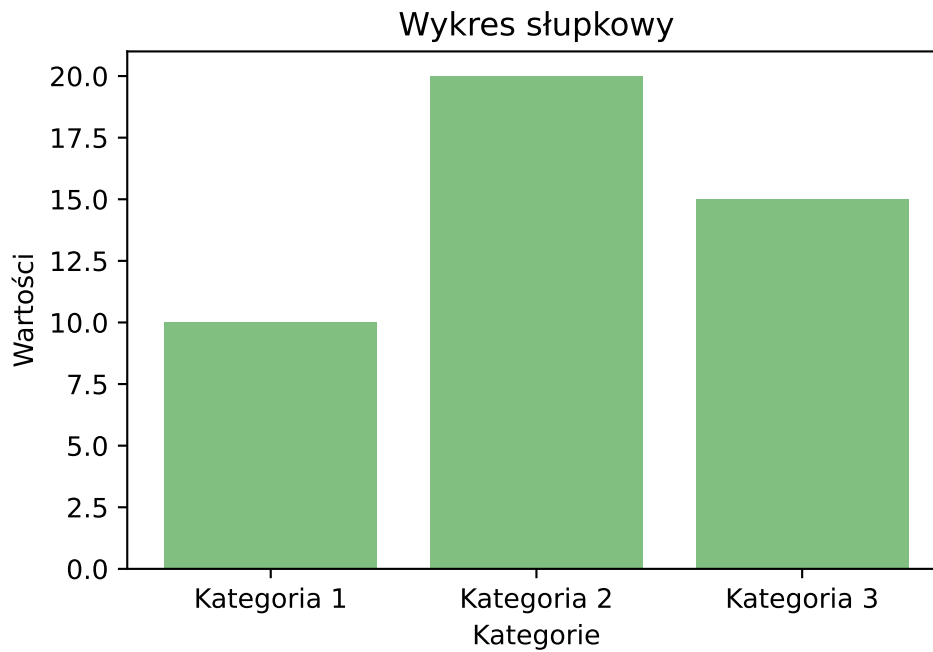
# Dane
kategorie = ['Kategoria 1', 'Kategoria 2', 'Kategoria 3']
wartosci = [10, 20, 15]

# Tworzenie wykresu słupkowego
plt.bar(kategorie, wartosci, color='green', alpha=0.5)

# Dodanie tytułu i etykiet osi
plt.title('Wykres słupkowy')
plt.xlabel('Kategorie')
```

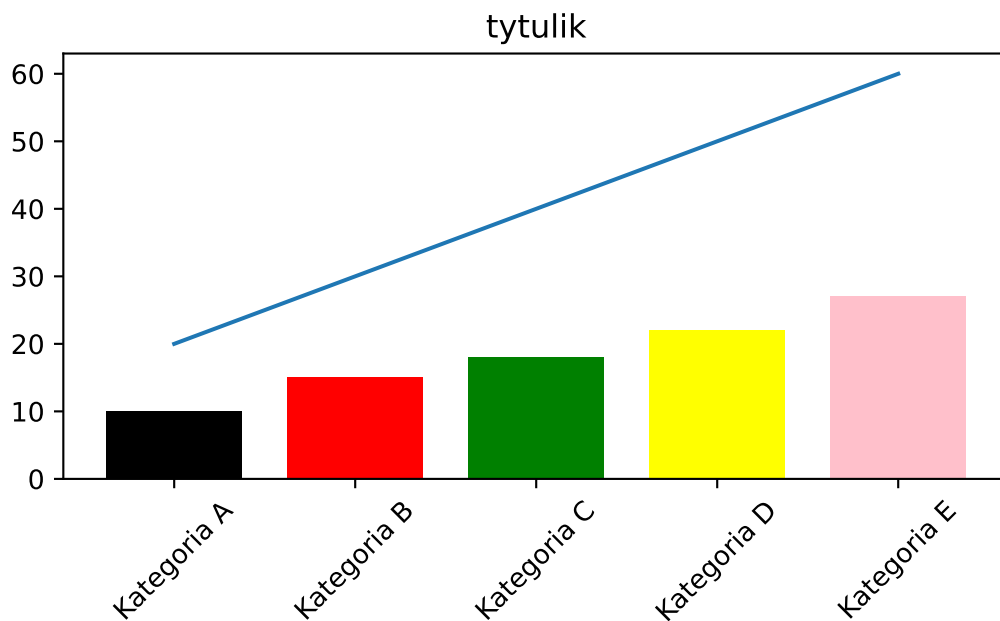
```
plt.ylabel('Wartości')

# Wyświetlenie wykresu
plt.show()
```



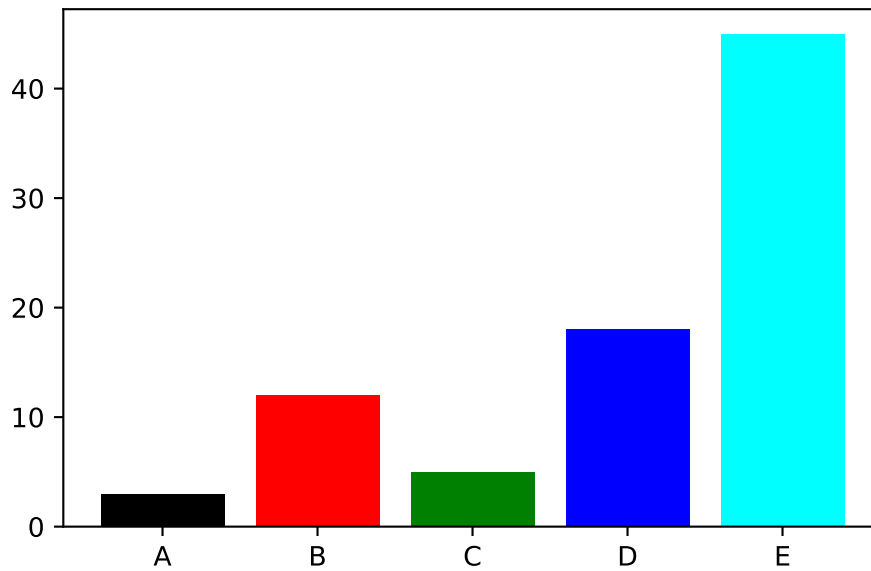
```
import numpy as np
import matplotlib.pyplot as plt

wys = [10, 15, 18, 22, 27]
x = np.arange(0, len(wys))
k = ["black", "red", "green", "yellow", "pink"]
plt.bar(x, wys, color=k, width=0.75)
etyk = ["Kategoria A", "Kategoria B", "Kategoria C", "Kategoria D", "Kategoria E"]
plt.xticks(x, etyk, rotation=45)
y2 = [20, 30, 40, 50, 60]
plt.plot(x, y2)
plt.title("tytulik")
plt.tight_layout()
plt.show()
```

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

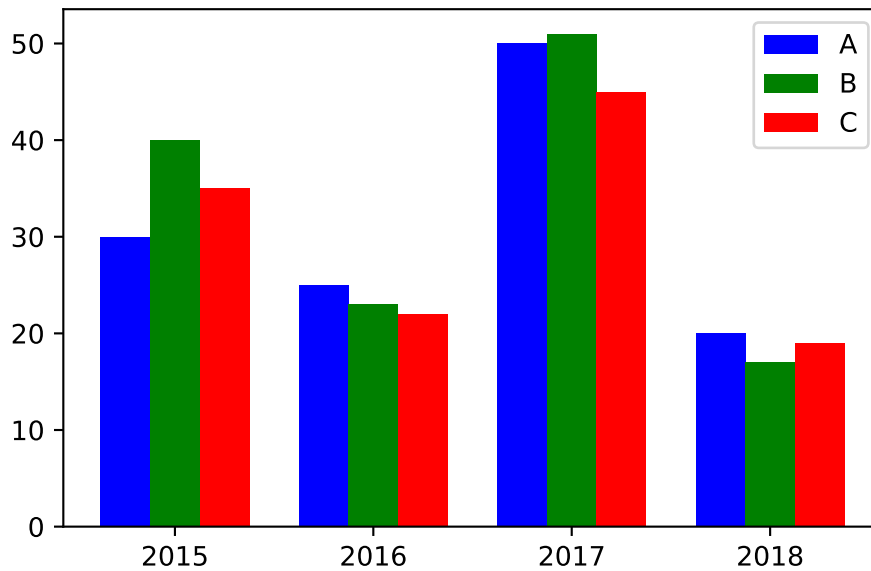
height = [3, 12, 5, 18, 45]
bars = ('A', 'B', 'C', 'D', 'E')
y_pos = np.arange(len(bars))
plt.bar(y_pos, height, color=['black', 'red', 'green', 'blue', 'cyan'])
plt.xticks(y_pos, bars)
plt.show()
```



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

data = [[30, 25, 50, 20],
        [40, 23, 51, 17],
        [35, 22, 45, 19]]
X = np.arange(4)

plt.bar(X + 0.00, data[0], color='b', width=0.25, label="A")
plt.bar(X + 0.25, data[1], color='g', width=0.25, label="B")
plt.bar(X + 0.50, data[2], color='r', width=0.25, label="C")
labelsbar = np.arange(2015, 2019)
plt.xticks(X + 0.25, labelsbar)
plt.legend()
plt.show()
```



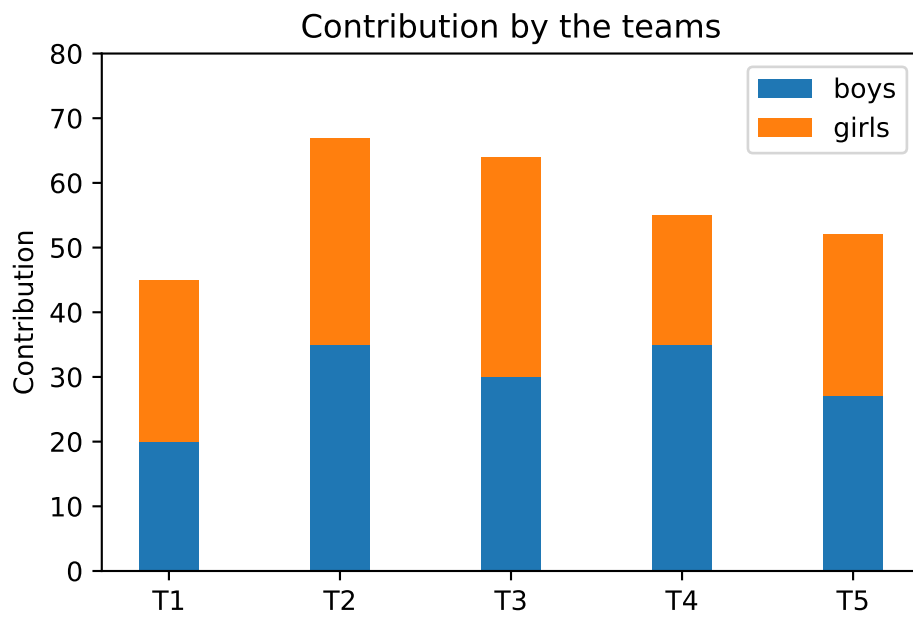
```
import numpy as np
import matplotlib.pyplot as plt

N = 5

boys = (20, 35, 30, 35, 27)
girls = (25, 32, 34, 20, 25)
ind = np.arange(N)
width = 0.35

plt.bar(ind, boys, width, label="boys")
plt.bar(ind, girls, width, bottom=boys, label="girls")

plt.ylabel('Contribution')
plt.title('Contribution by the teams')
plt.xticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend()
plt.show()
```



WYKRES SŁUPKOWY		
PIONOWY	→	POZIOMY
BAR	→	BARH
width	→	height
height	→	width
x	→	y
y	→	x
bottom	→	left
xticks	→	yticks
yticks	→	xticks

Funkcja `barh` służy do tworzenia wykresów słupkowych horyzontalnych (horizontal bar chart). Wykresy słupkowe horyzontalne są często stosowane, gdy chcemy porównać wartości różnych kategorii, a etykiety na osi X są długie lub są bardzo liczne.

Składnia funkcji to `plt.barh(y, width, height=0.8, left=None, align='center', data=None, **kwargs)`, gdzie:

- `y` - pozycje słupków na osi Y. Może to być sekwencja wartości numerycznych lub lista etykiet, które będą umieszczone na osi Y.
- `width` - szerokość słupków.
- `height` - wysokość słupków.

- `left` - położenie lewej krawędzi słupków. Domyślnie ustawione na `None`, co oznacza, że słupki zaczynają się od zera.
- `align` - sposób wyśrodkowania słupków wzdłuż osi Y. Domyślnie ustawione na `'center'`.
- `data` - obiekt `DataFrame`, który zawiera dane do wykresu.
- `**kwargs` - dodatkowe argumenty dotyczące formatowania wykresu, takie jak kolor, przezroczystość, etykiety osi, tytuł i legenda.

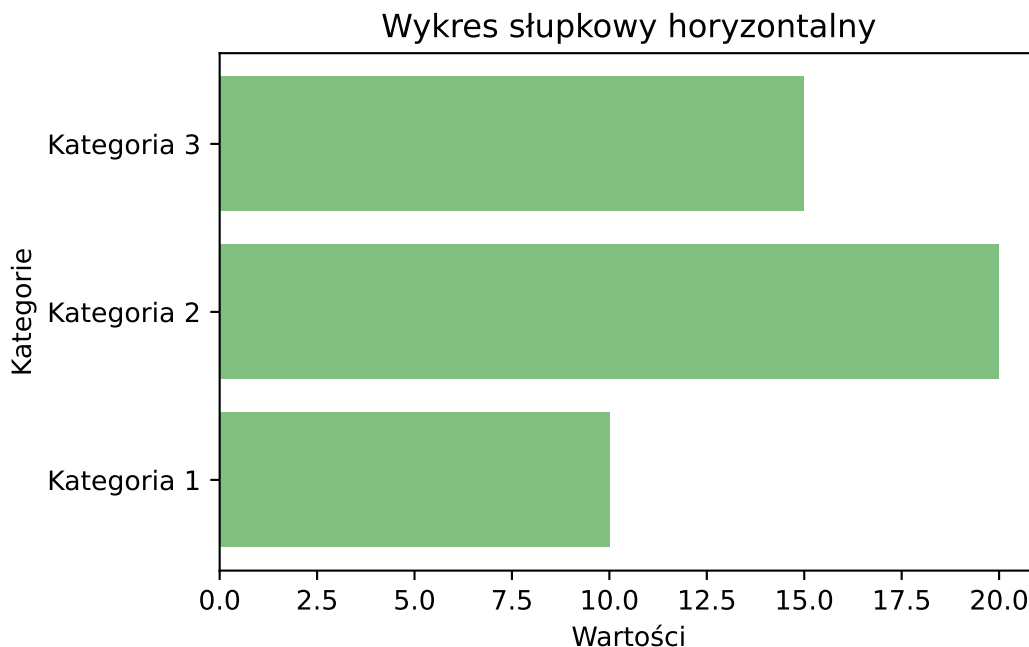
```
import matplotlib.pyplot as plt

# Dane
kategorie = ['Kategoria 1', 'Kategoria 2', 'Kategoria 3']
wartosci = [10, 20, 15]

# Tworzenie wykresu słupkowego horyzontalnego
plt.barh(kategorie, wartosci, color='green', alpha=0.5)

# Dodanie tytułu i etykiet osi
plt.title('Wykres słupkowy horyzontalny')
plt.xlabel('Wartości')
plt.ylabel('Kategorie')

# Wyświetlenie wykresu
plt.show()
```

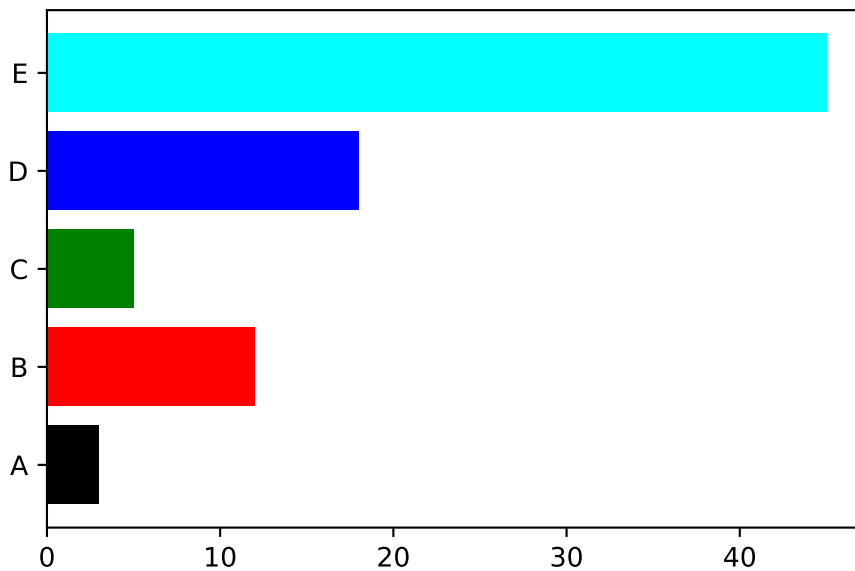


```

import numpy as np
import matplotlib.pyplot as plt

width = [3, 12, 5, 18, 45]
bars = ('A', 'B', 'C', 'D', 'E')
x_pos = np.arange(len(bars))
plt.barh(x_pos, width, color=['black', 'red', 'green', 'blue', 'cyan'])
plt.yticks(x_pos, bars)
plt.show()

```



```

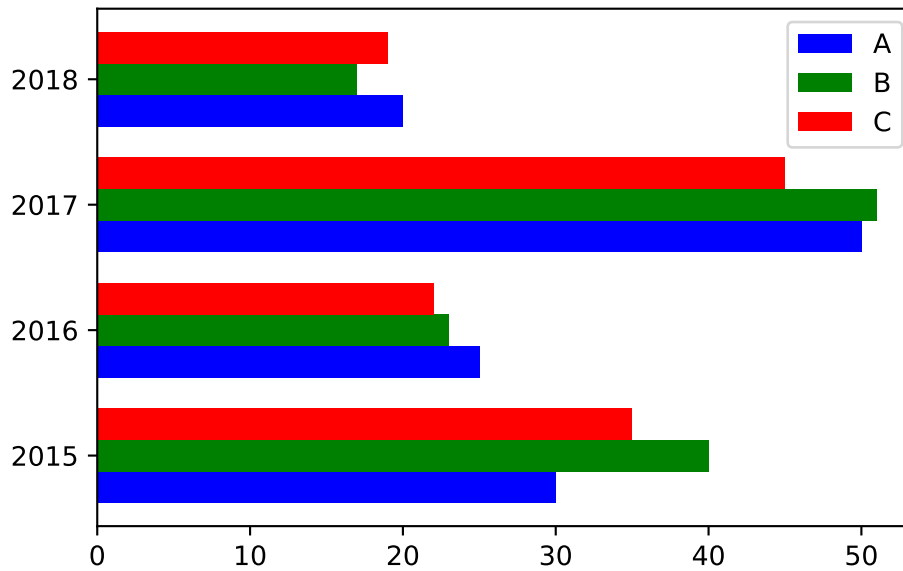
import numpy as np
import matplotlib.pyplot as plt

data = [[30, 25, 50, 20],
        [40, 23, 51, 17],
        [35, 22, 45, 19]]
Y = np.arange(4)

plt.barh(Y + 0.00, data[0], color='b', height=0.25, label="A")
plt.barh(Y + 0.25, data[1], color='g', height=0.25, label="B")
plt.barh(Y + 0.50, data[2], color='r', height=0.25, label="C")
labelsbar = np.arange(2015, 2019)
plt.yticks(Y + 0.25, labelsbar)
plt.legend()

```

```
plt.show()
```



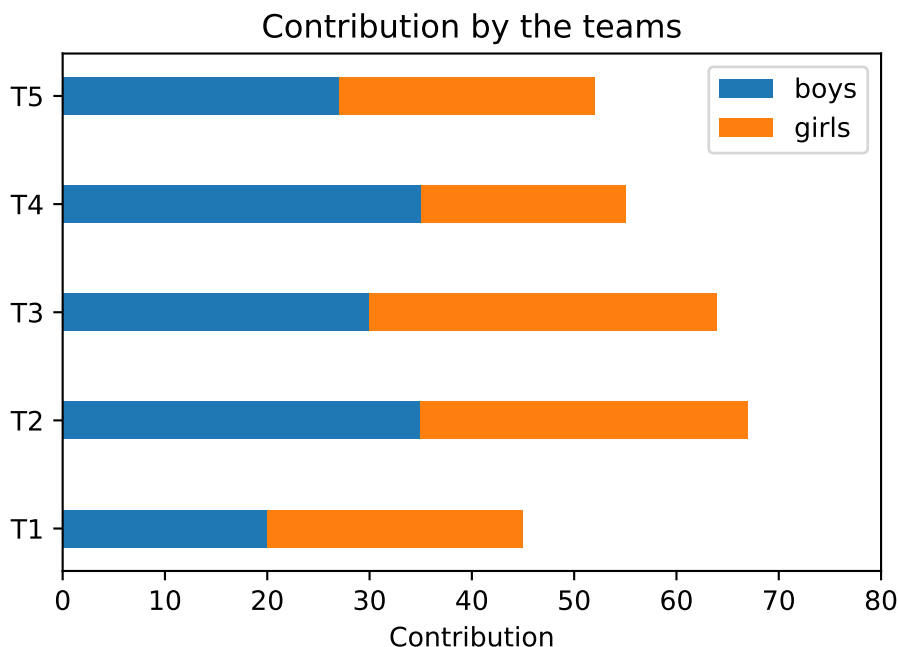
```
import numpy as np
import matplotlib.pyplot as plt

N = 5

boys = (20, 35, 30, 35, 27)
girls = (25, 32, 34, 20, 25)
ind = np.arange(N)
height = 0.35

plt.barh(ind, boys, height, label="boys")
plt.barh(ind, girls, height, left=boys, label="girls")

plt.xlabel('Contribution')
plt.title('Contribution by the teams')
plt.yticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))
plt.xticks(np.arange(0, 81, 10))
plt.legend()
plt.show()
```

6.20 Wykres pudełkowy

Wykres pudełkowy (inaczej box plot) jest stosowany do przedstawiania informacji o rozkładzie danych liczbowych oraz do identyfikacji wartości odstających. Jest szczególnie przydatny w przypadku analizy danych ciągłych, które mają różne wartości i rozkłady. Oto kilka typów danych, dla których wykres pudełkowy może być stosowany:

1. Porównanie grup: Wykres pudełkowy jest używany do porównywania rozkładu danych między różnymi grupami. Na przykład, można go użyć do porównania wyników testów uczniów z różnych szkół, wynagrodzeń w różnych sektorach czy wartości sprzedaży różnych produktów.
2. Identyfikacja wartości odstających: Wykres pudełkowy jest używany do identyfikacji wartości odstających (outlierów) w danych, które mogą wskazywać na błędy pomiarowe, nietypowe obserwacje lub wartości ekstremalne. Na przykład, może to być użyte do wykrywania anomalii w danych meteorologicznych, wartościach giełdowych czy danych medycznych.
3. Analiza rozkładu: Wykres pudełkowy pomaga zrozumieć rozkład danych, takich jak mediana, kwartyle, zakres wartości i potencjalne wartości odstające. Może to być użyte w analizie danych takich jak oceny, wzrost ludności, wartość akcji czy ceny nieruchomości.

4. Wizualizacja wielowymiarowych danych: Wykres pudełkowy może być używany do wizualizacji wielowymiarowych danych, przedstawiając rozkład wielu zmiennych na jednym wykresie. Na przykład, można porównać zmienne takie jak wiek, zarobki i wykształcenie w badaniu demograficznym.

Warto zauważyć, że wykres pudełkowy jest szczególnie przydatny, gdy chcemy zrozumieć rozkład danych, ale nie pokazuje on konkretnej liczby obserwacji ani wartości indywidualnych punktów danych. W takich przypadkach inne rodzaje wykresów, takie jak wykres punktowy, mogą być bardziej odpowiednie.

Wykres pudełkowy pokazuje pięć statystyk opisowych danych: minimum, pierwszy kwartył (Q1), medianę, trzeci kwartył (Q3) i maksimum.

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

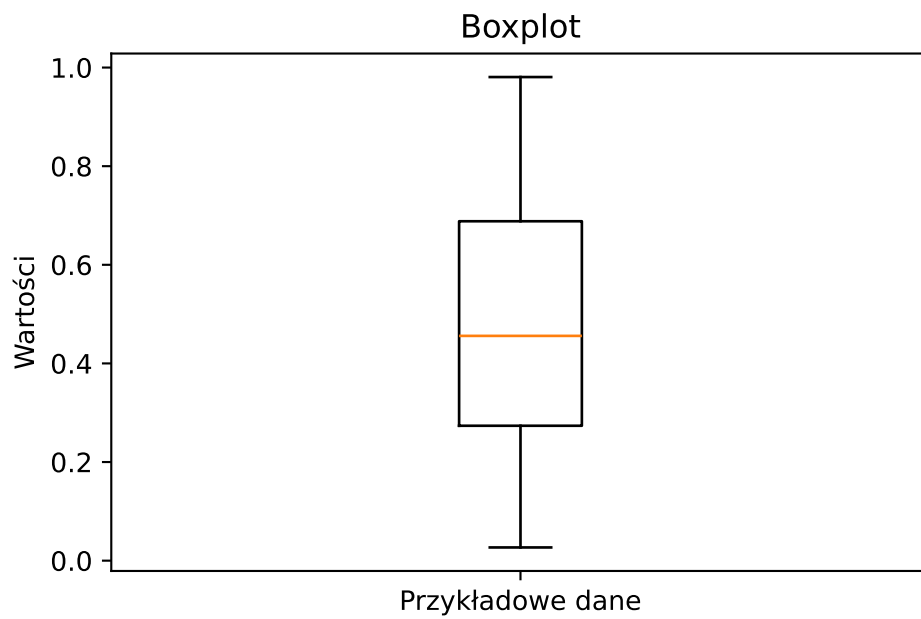
# Przykładowe dane
data = np.random.rand(100)

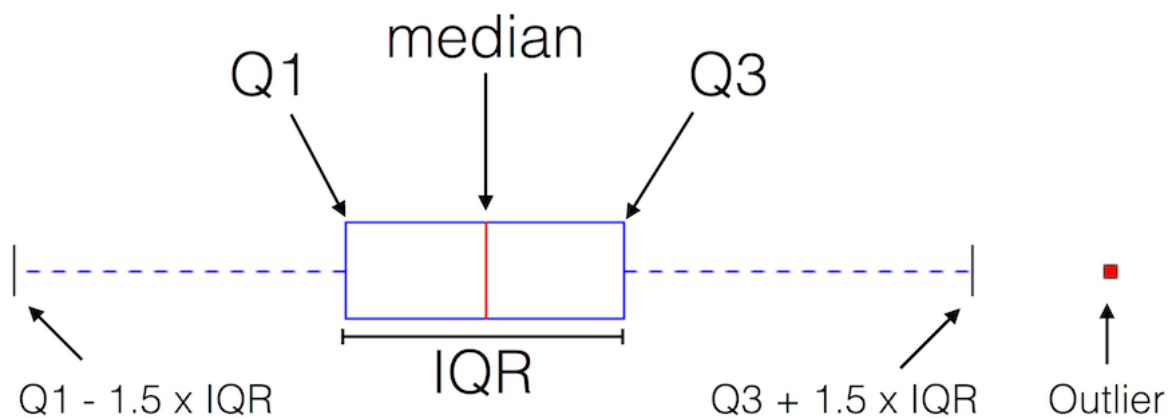
# Tworzenie wykresu
fig, ax = plt.subplots()

# Rysowanie boxplota
ax.boxplot(data)

# Dodanie opisów
ax.set_title('Boxplot')
ax.set_ylabel('Wartości')
ax.set_xticklabels(['Przykładowe dane'])

# Wyświetlanie wykresu
plt.show()
```





Q1: *Quartile 1*, or median of the *left* data subset after dividing the original data set into 2 subsets via the median (25% of the data points fall below this threshold)

Q3: *Quartile 3*, median of the *right* data subset (75% of the data points fall below this threshold)

IQR: *Interquartile-range*, $Q3 - Q1$

Outliers: Data points are considered to be outliers if
 $value < Q1 - 1.5 \times IQR$ or
 $value > Q3 + 1.5 \times IQR$



Sebastian Raschka, 2016

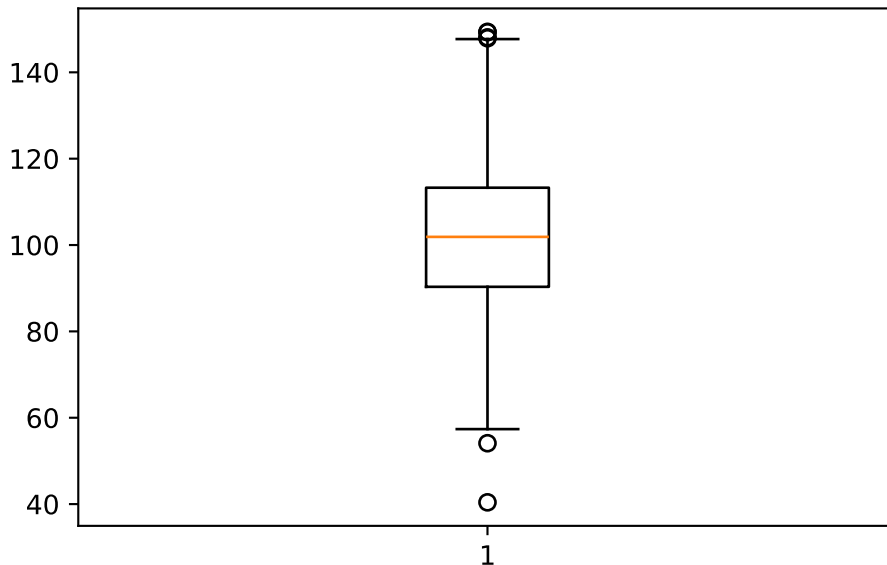
This work is licensed under a Creative Commons Attribution 4.0 International License

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

# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)

# Creating plot
plt.boxplot(data)
```

```
# show plot  
plt.show()
```



6.21 Histogram

Wykres histogramu jest stosowany do przedstawiania rozkładu danych liczbowych, zarówno ciągłych, jak i dyskretnych. Histogram pokazuje częstość występowania danych w określonych przedziałach (binach), co pozwala na analizę dystrybucji i identyfikację wzorców. Oto kilka typów danych, dla których histogram może być stosowany:

1. **Analiza rozkładu:** Histogram może być używany do analizy rozkładu danych, takich jak oceny, ceny, wartości akcji, wzrost ludności czy dane meteorologiczne. Pozwala to zrozumieć, jak dane są rozłożone, czy są skoncentrowane wokół pewnych wartości, czy mają długi ogon (tj. czy występują wartości odstające).
2. **Identyfikacja tendencji:** Histogram może pomóc w identyfikacji tendencji lub wzorców w danych. Na przykład, można użyć histogramu do identyfikacji sezonowych wzorców sprzedaży, zmian w wartościach giełdowych czy wzorców migracji ludności.
3. **Porównanie grup:** Histogram może być również używany do porównywania rozkładu danych między różnymi grupami. Na przykład, można go użyć do porównania wyników testów uczniów z różnych szkół, wynagrodzeń w różnych sektorach czy wartości sprzedaży różnych produktów.

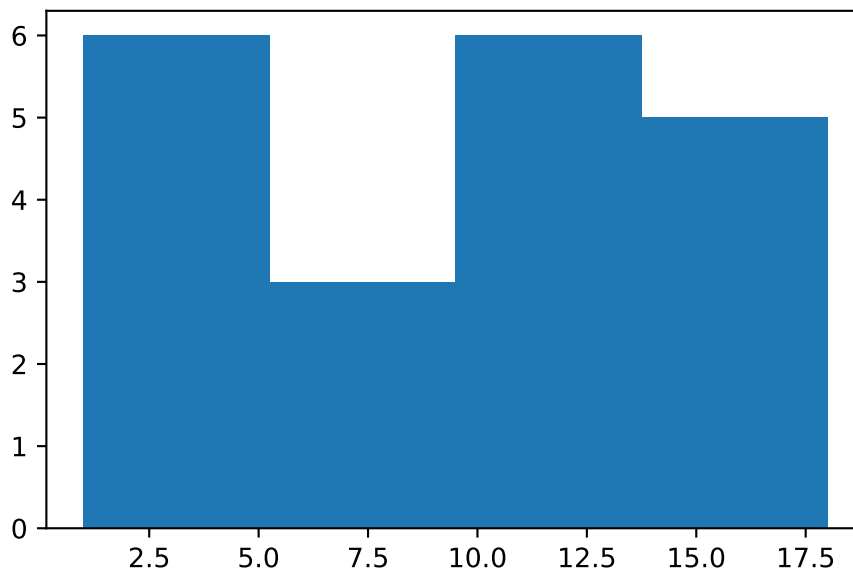
4. Szacowanie parametrów: Histogram może pomóc w szacowaniu parametrów rozkładu, takich jak średnia, mediana czy wariancja, co może być użyteczne w analizie statystycznej.

Warto zauważyć, że histogram jest odpowiedni dla danych liczbowych, ale nie jest przeznaczony do przedstawiania danych kategoryalnych. W takich przypadkach inne rodzaje wykresów, takie jak wykres słupkowy, mogą być bardziej odpowiednie.

```
import matplotlib.pyplot as plt

x = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
     10, 11, 11, 13, 13, 15, 16, 17, 18, 18]

plt.hist(x, bins=4)
plt.show()
```

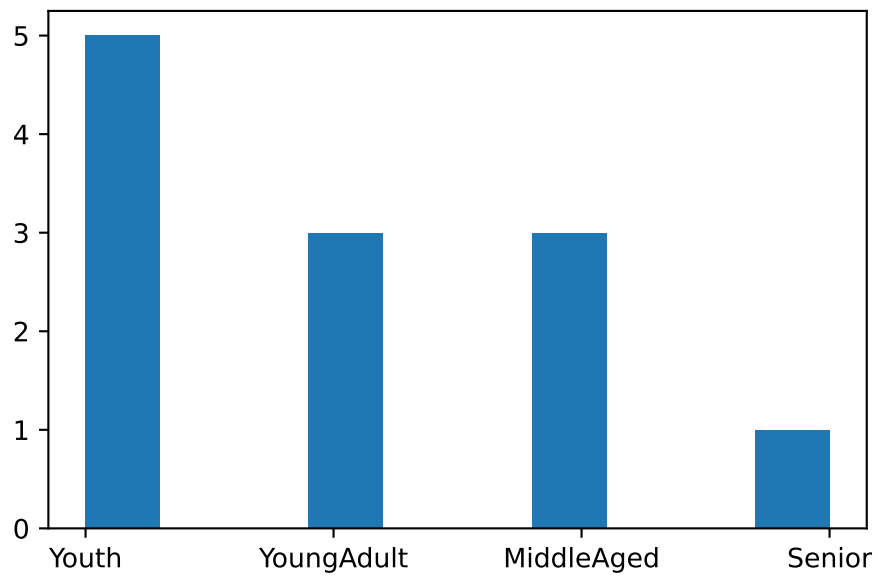


```
import pandas as pd
import matplotlib.pyplot as plt

ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats2 = pd.cut(ages, [18, 26, 36, 61, 100], right=False)
print(cats2)
group_names = ['Youth', 'YoungAdult',
               'MiddleAged', 'Senior']
```

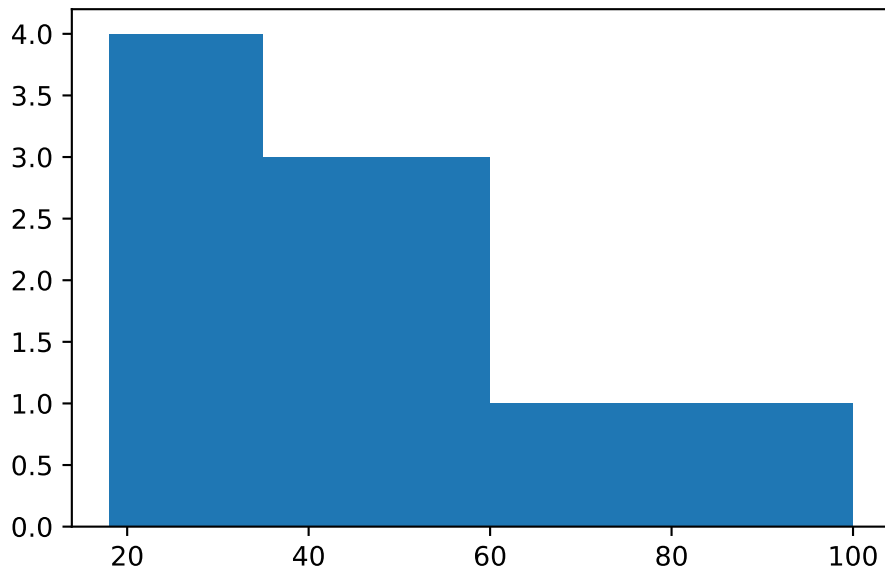
```
data = pd.cut(ages, bins, labels=group_names)
plt.hist(data)
plt.show()
```

```
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100), [36, 61), [36, 61), ...]]
Length: 12
Categories (4, interval[int64, left]): [[18, 26) < [26, 36) < [36, 61) < [61, 100)]
```



```
import matplotlib.pyplot as plt

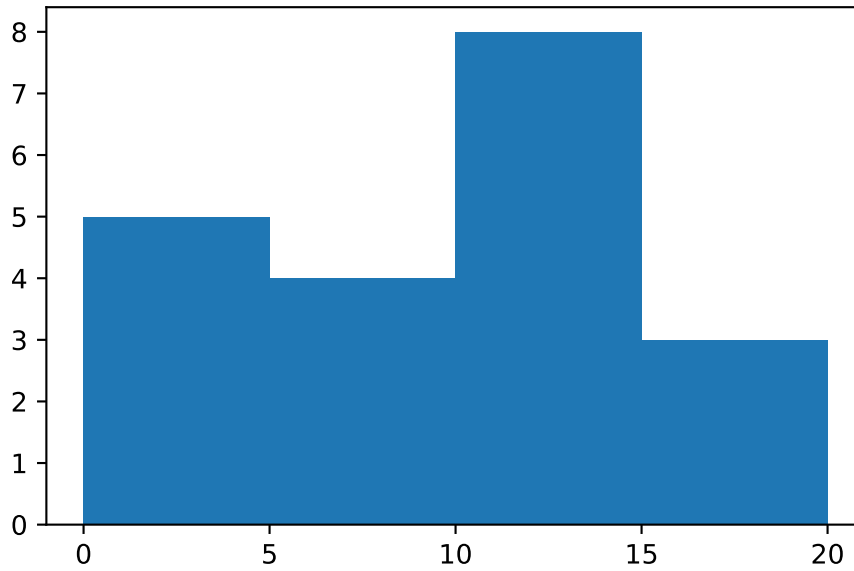
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
plt.hist(ages, bins=bins)
plt.show()
```



```
import matplotlib.pyplot as plt

x = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
     10, 11, 11, 13, 13, 15, 14, 12, 18, 18]

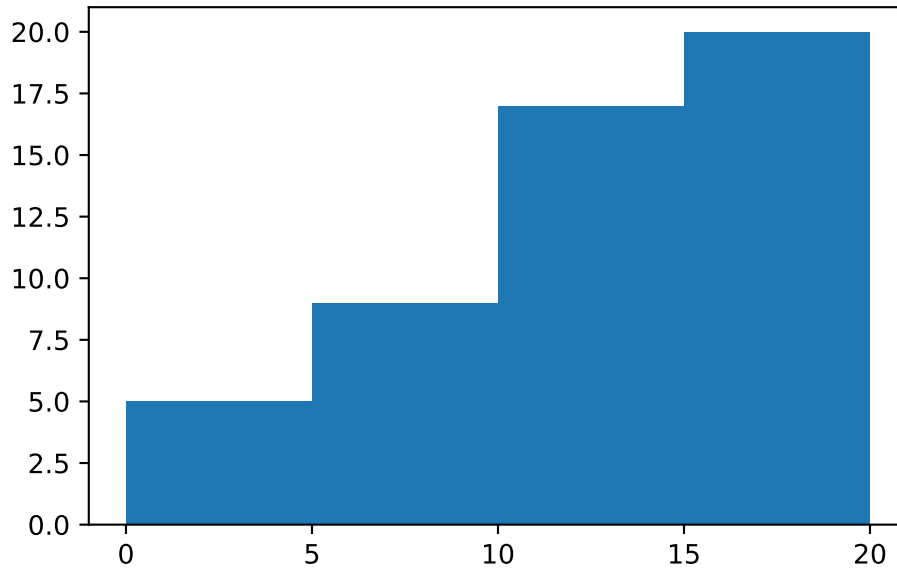
plt.hist(x, bins=[0, 5, 10, 15, 20])
plt.xticks([0, 5, 10, 15, 20])
plt.show()
```

```
import matplotlib.pyplot as plt

x = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
     10, 11, 11, 13, 13, 15, 14, 12, 18, 18]

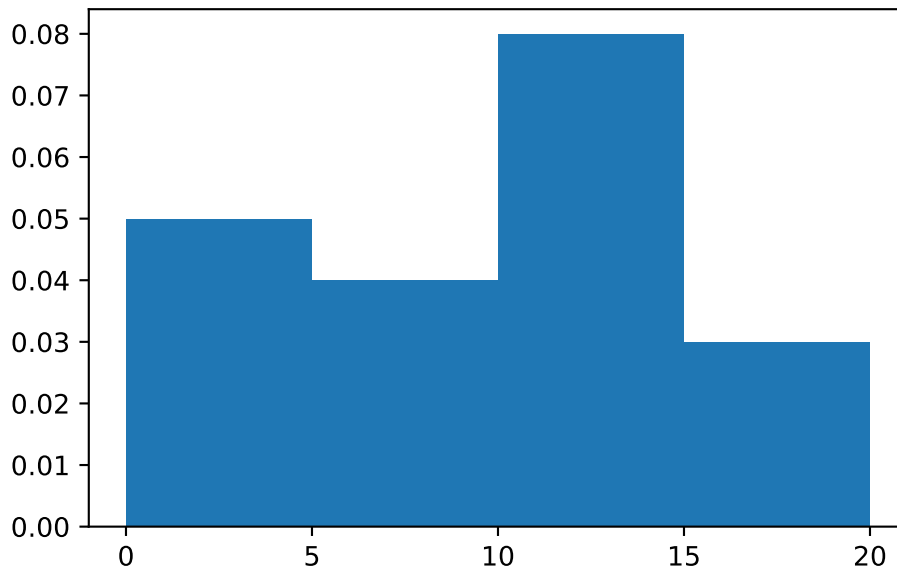
plt.hist(x, bins=[0, 5, 10, 15, 20], cumulative=True)
plt.xticks([0, 5, 10, 15, 20])
plt.show()
```



```
import matplotlib.pyplot as plt

x = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
     10, 11, 11, 13, 13, 15, 14, 12, 18, 18]

plt.hist(x, bins=[0, 5, 10, 15, 20], density=True)
plt.xticks([0, 5, 10, 15, 20])
plt.show()
```



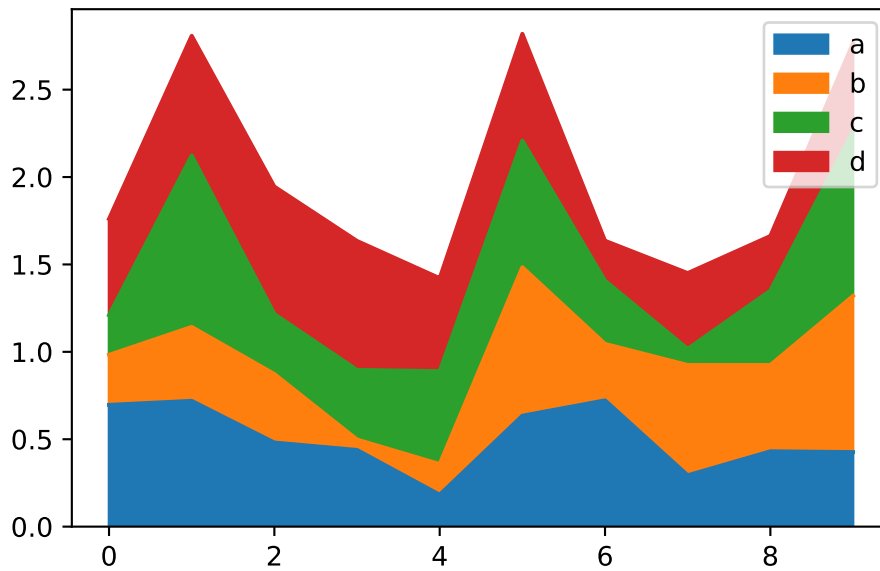
6.22 Wykres warstwowy

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html>

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

np.random.seed(123)

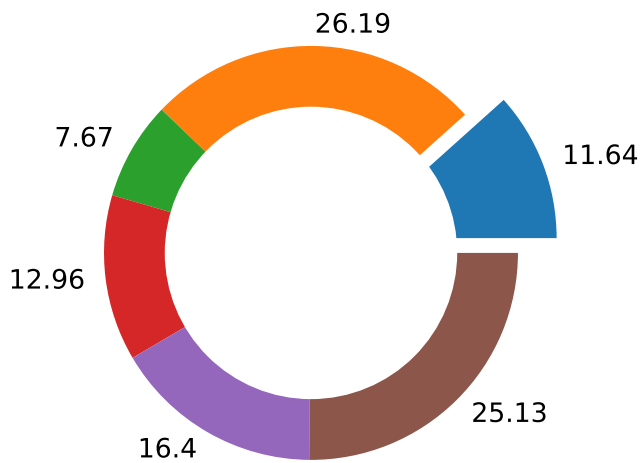
df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
df.plot.area()
plt.show()
```



6.23 Wykres pierścieniowy

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

np.random.seed(345)
data = np.random.randint(20, 100, 6)
total = sum(data)
data_per = data / total * 100
explode = (0.2, 0, 0, 0, 0, 0)
plt.pie(data_per, explode=explode, labels=[round(i, 2) for i in list(data_per)])
circle = plt.Circle((0, 0), 0.7, color='white')
p = plt.gcf()
p.gca().add_artist(circle)
plt.show()
```



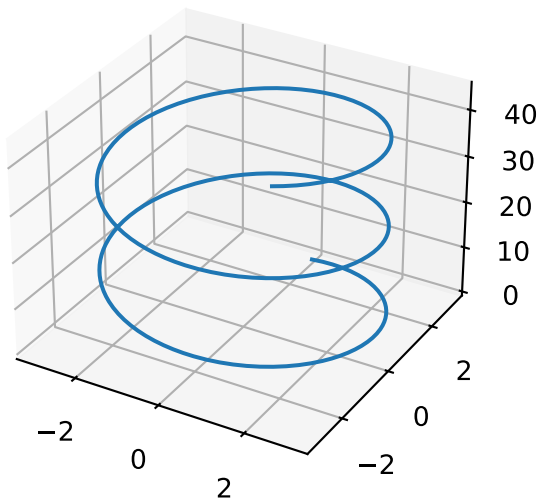
6.24 Wykresy w przestrzeni

6.24.1 Helisa

$$\begin{cases} x = a \cos(t) \\ y = a \sin(t) \\ z = at \end{cases}$$

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

fig = plt.figure()
ax = plt.axes(projection='3d')
t = np.linspace(0, 15, 1000)
a = 3
xline = a * np.sin(t)
yline = a * np.cos(t)
zline = a * t
ax.plot3D(xline, yline, zline)
plt.show()
```

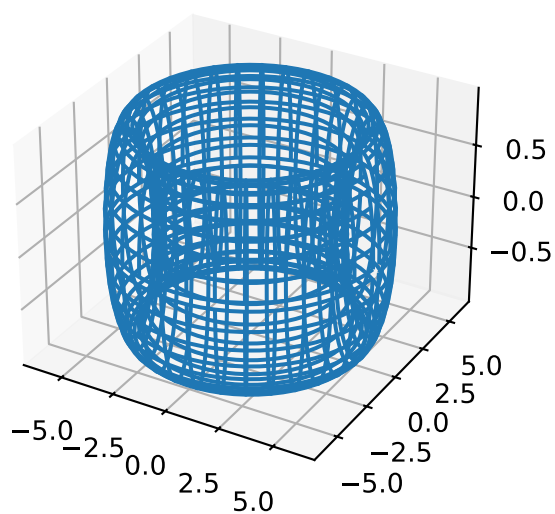


6.24.2 Torus

$$p(\alpha, \beta) = \left((R + r \cos \alpha) \cos \beta, (R + r \cos \alpha) \sin \beta, r \sin \alpha \right)$$

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

fig = plt.figure()
ax = plt.axes(projection='3d')
r = 1
R = 5
alpha = np.arange(0, 2 * np.pi, 0.1)
beta = np.arange(0, 2 * np.pi, 0.1)
alpha, beta = np.meshgrid(alpha, beta)
x = (R + r * np.cos(alpha)) * np.cos(beta)
y = (R + r * np.cos(alpha)) * np.sin(beta)
z = r * np.sin(alpha)
ax.plot_wireframe(x, y, z)
plt.show()
```



Źródło:

- <https://www.geeksforgeeks.org/bar-plot-in-matplotlib/>
- Dokumentacja <https://matplotlib.org/>
- <https://datatofish.com/plot-histogram-python/>
- <https://jakevdp.github.io/PythonDataScienceHandbook/04.12-three-dimensional-plotting.html>

7 Seaborn

<https://seaborn.pydata.org/>

```
import seaborn as sns
```

Biblioteka Seaborn, będąca nakładką na Matplotlib, ułatwia tworzenie “pięknych i czytelnych” wykresów statystycznych w języku Python.

1. Eksploracyjna analiza danych (EDA): Seaborn jest często używany do wizualizacji danych podczas eksploracyjnej analizy danych, co pomaga zrozumieć strukturę, trendy i relacje między zmiennymi w danych. Pozwala to na identyfikację wzorców, wykrywanie wartości odstających oraz ewentualnych problemów z danymi.
2. Wizualizacja danych kategoryalnych: Seaborn oferuje wiele rodzajów wykresów kategoryalnych, takich jak wykresy słupkowe, pudełkowe, skrzypcowe, czy punktowe. Dzięki temu łatwo porównać wartości między różnymi kategoriami, co pomaga lepiej zrozumieć dane.
3. Wizualizacja danych wielowymiarowych: Seaborn pozwala na tworzenie wykresów z wieloma zmiennymi jednocześnie. Funkcje takie jak `pairplot` i `heatmap` są użyteczne do analizy wielowymiarowych danych i identyfikacji korelacji między zmiennymi.
4. Wizualizacja rozkładów statystycznych: Seaborn umożliwia tworzenie histogramów, wykresów estymacji gęstości jądra (KDE) oraz dystrybuanty empirycznej. Dzięki tym narzędziom można zrozumieć rozkład danych, co jest niezbędne w analizie statystycznej i modelowaniu.
5. Wizualizacja regresji: Seaborn oferuje narzędzia do analizy regresji, takie jak `regplot` i `lmpplot`. Te funkcje pozwalają na wizualizację linii regresji, co ułatwia zrozumienie relacji między zmiennymi oraz ocenę dopasowania modeli.
6. Tworzenie siatek wykresów: Seaborn pozwala na tworzenie siatek wykresów za pomocą funkcji takich jak `FacetGrid` i `PairGrid`. Ułatwia to analizę i porównanie danych dla różnych podgrup, warstw czy kombinacji zmiennych.
7. Personalizacja wykresów: Seaborn umożliwia personalizację wykresów poprzez zmianę stylu, palet kolorów oraz innych parametrów wykresów. Dzięki temu można tworzyć atrakcyjne i profesjonalnie wyglądające wizualizacje danych.

8. Prezentacje i raporty: Seaborn może być używany do tworzenia wysokiej jakości wizualizacji danych w celach prezentacji, raportów czy publikacji naukowych. Estetyka i łatwość w personalizacji wykresów sprawiają, że wizualizacje stworzone w Seaborn są czytelne i angażujące dla odbiorców.
9. Analiza czasowa: Seaborn może być używany do analizy danych czasowych, takich jak szeregi czasowe czy dane sezonowe. Funkcje takie jak `lineplot` czy `relplot` pozwalają na wizualizację trendów oraz ewentualnych zmian w danych na przestrzeni czasu.
10. Porównanie modeli i algorytmów: Wizualizacje danych stworzone za pomocą Seaborn mogą być użyte do porównywania wyników różnych modeli uczenia maszynowego czy algorytmów. Można w ten sposób prezentować wyniki analizy, jakości klasyfikacji, estymacji czy prognozowania.
11. Nauczanie i edukacja: Seaborn jest często stosowany jako narzędzie do nauczania i edukacji, zwłaszcza w kontekście nauki statystyki, analizy danych i uczenia maszynowego. Wizualizacje danych ułatwiają zrozumienie złożonych koncepcji i relacji między zmiennymi.

7.1 Ładowanie wbudowanych danych

Funkcja `sns.load_dataset()` pozwala na ładowanie wbudowanych zestawów danych w bibliotece Seaborn. Wszystkie wbudowane zestawy danych są przechowywane jako ramki danych Pandas.

1. Ładowanie zestawu danych “iris”:

```
import seaborn as sns

iris_data = sns.load_dataset('iris')
print(iris_data.head())
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

2. Ładowanie zestawu danych “titanic”:

```
import seaborn as sns

titanic_data = sns.load_dataset('titanic')
print(titanic_data.head())
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	\
0	0	3	male	22.0	1	0	7.2500	S	Third	
1	1	1	female	38.0	1	0	71.2833	C	First	
2	1	3	female	26.0	0	0	7.9250	S	Third	
3	1	1	female	35.0	1	0	53.1000	S	First	
4	0	3	male	35.0	0	0	8.0500	S	Third	

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

3. Ładowanie zestawu danych “tips”:

```
import seaborn as sns

tips_data = sns.load_dataset('tips')
print(tips_data.head())
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

7.2 Wykres punktowy

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

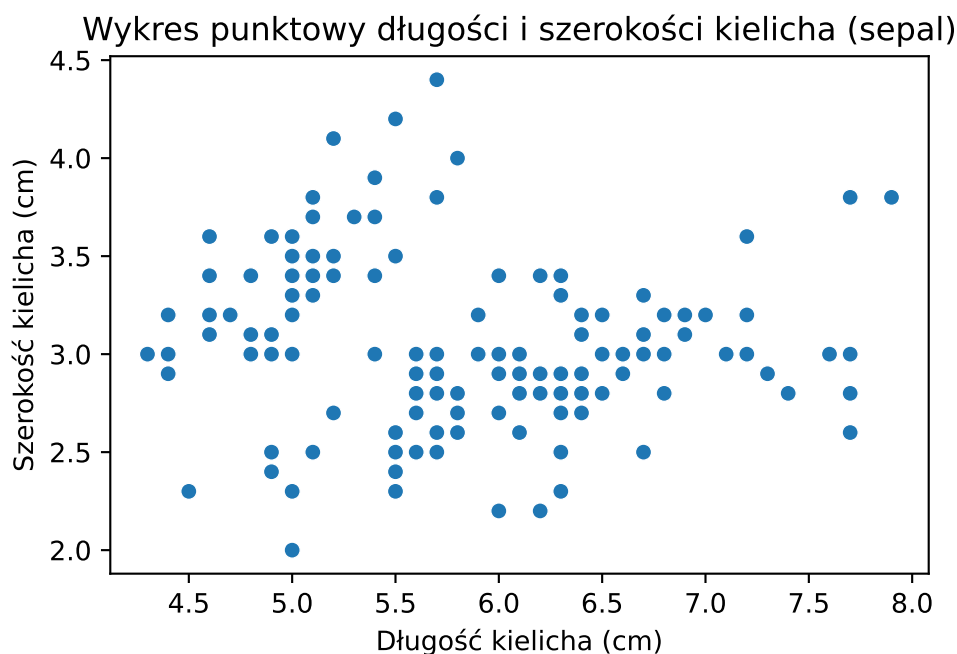
# Ładowanie danych
```

```
iris_data = sns.load_dataset('iris')

# Tworzenie wykresu punktowego
sns.scatterplot(x='sepal_length', y='sepal_width', data=iris_data)

# Dodanie tytułu i etykiet osi
plt.title('Wykres punktowy długości i szerokości kielicha (sepal)')
plt.xlabel('Długość kielicha (cm)')
plt.ylabel('Szerokość kielicha (cm)')

# Wyświetlenie wykresu
plt.show()
```



Funkcja `sns.scatterplot()` oferuje dodatkowe parametry, które pozwalają na modyfikację wykresu. Przykłady obejmują:

1. Parametr `hue`: pozwala na dodanie kolorowania punktów na podstawie wartości zmiennej kategoryjnej.

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

# Ładowanie danych
```

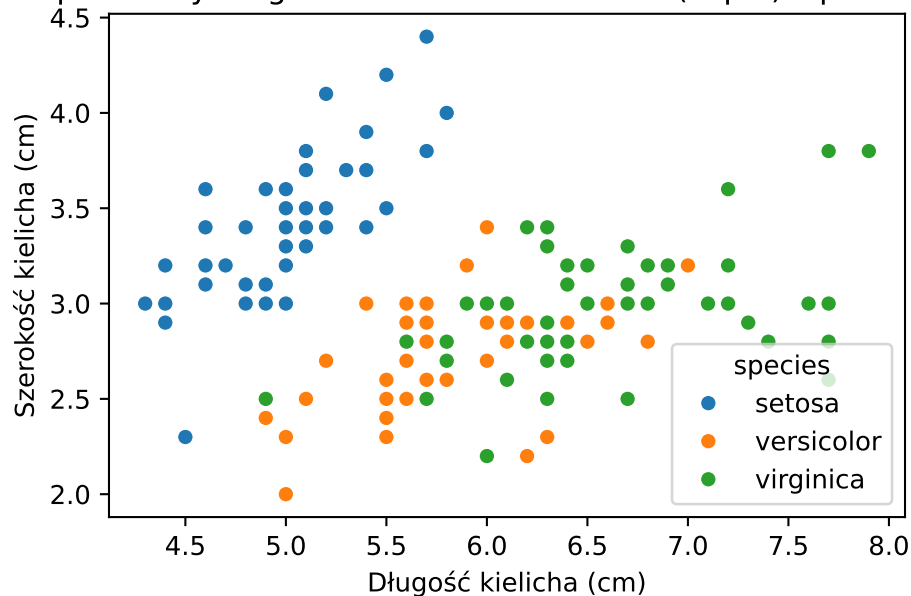
```
iris_data = sns.load_dataset('iris')

# Tworzenie wykresu punktowego z parametrem 'hue'
sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=iris_data)

# Dodanie tytułu i etykiet osi
plt.title('Wykres punktowy długości i szerokości kielicha (sepal) z parametrem hue')
plt.xlabel('Długość kielicha (cm)')
plt.ylabel('Szerokość kielicha (cm)')

# Wyświetlenie wykresu
plt.show()
```

Wykres punktowy długości i szerokości kielicha (sepal) z parametrem hue



2. Parametr `size`: pozwala na zmianę wielkości punktów na podstawie wartości zmiennej numerycznej.

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

# Ładowanie danych
iris_data = sns.load_dataset('iris')

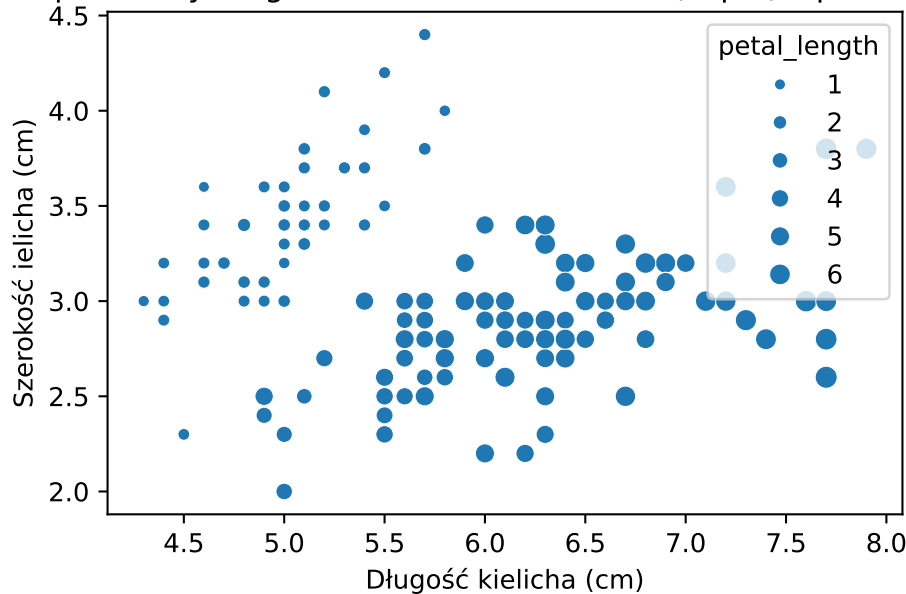
# Tworzenie wykresu punktowego z parametrem 'size'
```

```
sns.scatterplot(x='sepal_length', y='sepal_width', size='petal_length', data=iris_data)

# Dodanie tytułu i etykiet osi
plt.title('Wykres punktowy długości i szerokości kielicha (sepal) z parametrem size')
plt.xlabel('Długość kielicha (cm)')
plt.ylabel('Szerokość ielicha (cm)')

# Wyświetlenie wykresu
plt.show()
```

Wykres punktowy długości i szerokości kielicha (sepal) z parametrem size



3. Parametr **style**: pozwala na zmianę stylu punktów na podstawie wartości zmiennej kategoryjnej.

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

# Ładowanie danych
iris_data = sns.load_dataset('iris')

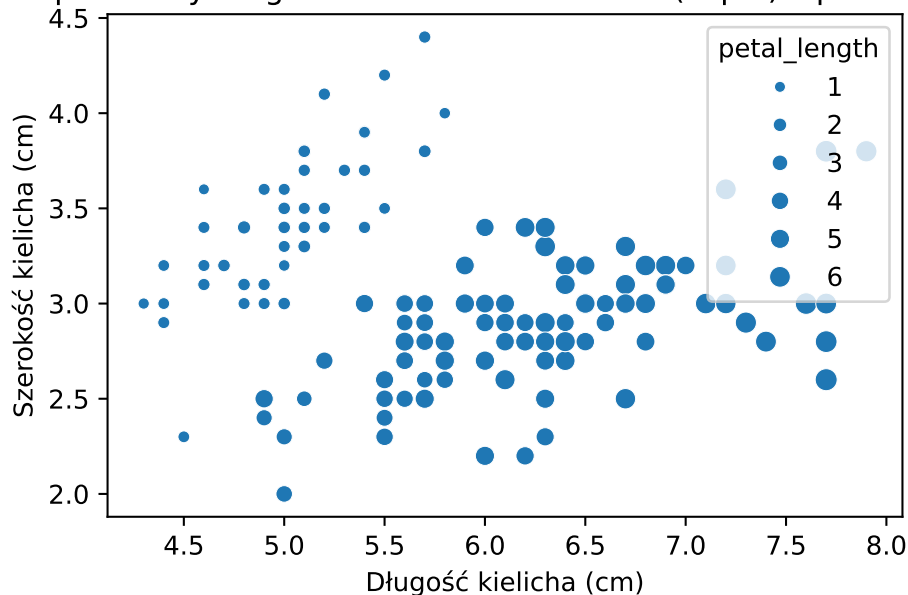
# Tworzenie wykresu punktowego z parametrem 'size'
sns.scatterplot(x='sepal_length', y='sepal_width', size='petal_length', data=iris_data)

# Dodanie tytułu i etykiet osi
```

```
plt.title('Wykres punktowy długości i szerokości kielicha (sepal) z parametrem size')
plt.xlabel('Długość kielicha (cm)')
plt.ylabel('Szerokość kielicha (cm)')

# Wyświetlenie wykresu
plt.show()
```

Wykres punktowy długości i szerokości kielicha (sepal) z parametrem size



7.3 Wykres liniowy

Funkcja `sns.lineplot()` w bibliotece Seaborn służy do tworzenia wykresów liniowych, które pozwalają na wizualizację związku między dwiema zmiennymi, zwłaszcza w przypadku danych czasowych. Wykres liniowy jest użyteczny do prezentacji trendów i zmian w danych na przestrzeni czasu lub innej zmiennej ciągłej.

Oto przykład użycia funkcji `sns.lineplot()`:

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

# Przykładowe dane
data = {
```

```

    'year': [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019],
    'value': [100, 110, 105, 120, 130, 140, 150, 170, 180, 200]
}

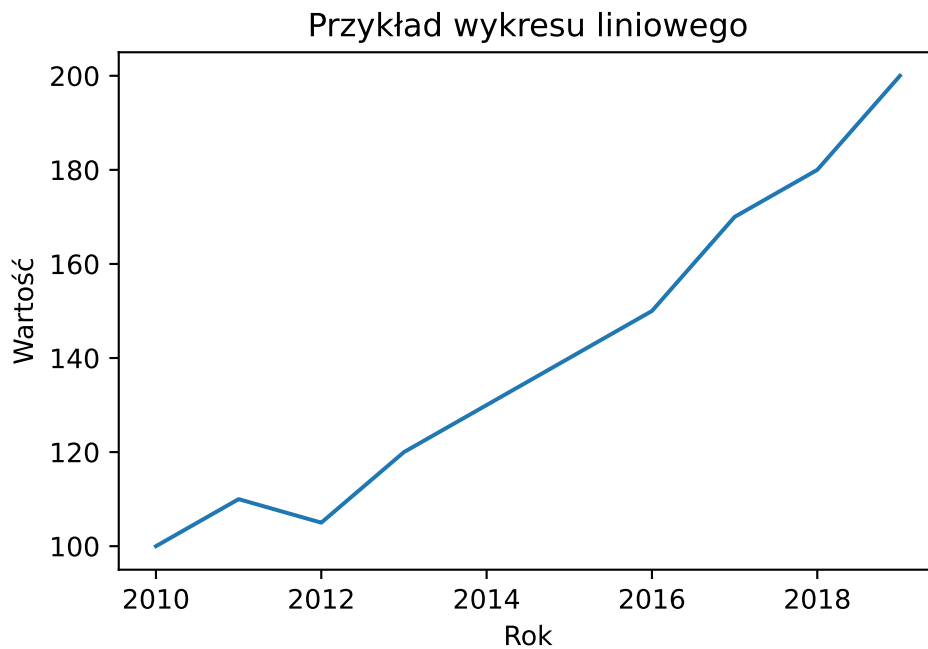
# Ładowanie danych jako ramki danych Pandas
import pandas as pd
df = pd.DataFrame(data)

# Tworzenie wykresu liniowego
sns.lineplot(x='year', y='value', data=df)

# Dodanie tytułu i etykiet osi
plt.title('Przykład wykresu liniowego')
plt.xlabel('Rok')
plt.ylabel('Wartość')

# Wyświetlenie wykresu
plt.show()

```



W powyższym przykładzie tworzymy wykres liniowy na podstawie ramki danych **df**. Parametry **x** i **y** określają zmienne, które mają być przedstawione na osiach wykresu. Parametr **data** wskazuje źródło danych, w tym przypadku ramkę danych Pandas.

Funkcja `sns.lineplot()` oferuje dodatkowe parametry, które pozwalają na modyfikację wykresu. Przykłady obejmują:

1. Parametr `hue`: pozwala na dodanie kolorowania linii na podstawie wartości zmiennej kategoryjnej.

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

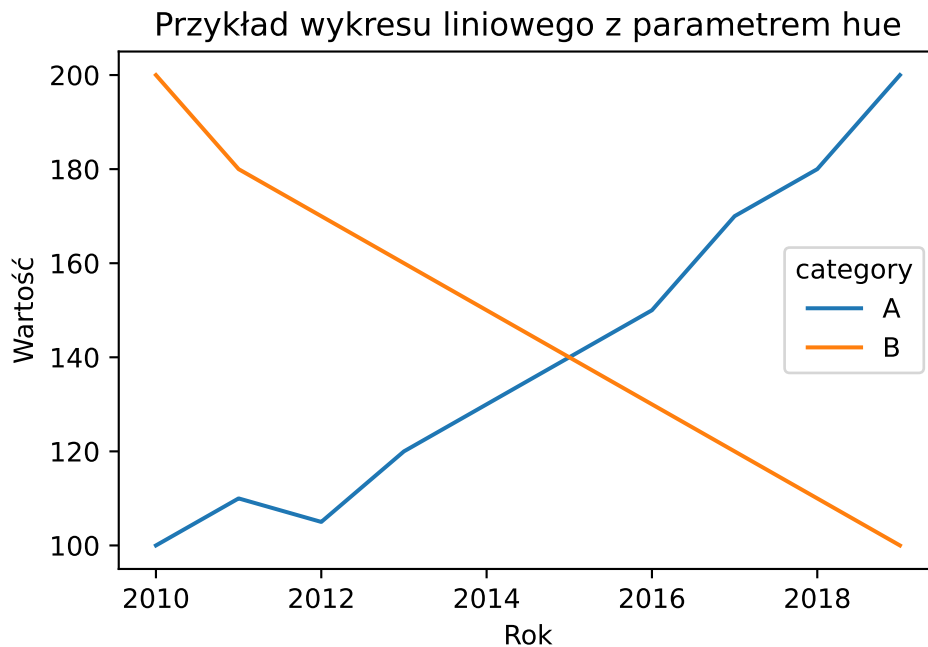
# Przykładowe dane
data = {
    'year': [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019] * 2,
    'value': [100, 110, 105, 120, 130, 140, 150, 170, 180, 200] +
             [200, 180, 170, 160, 150, 140, 130, 120, 110, 100],
    'category': ['A'] * 10 + ['B'] * 10
}

# Ładowanie danych jako ramki danych Pandas
df = pd.DataFrame(data)

# Tworzenie wykresu liniowego z parametrem 'hue'
sns.lineplot(x='year', y='value', hue='category', data=df)

# Dodanie tytułu i etykiet osi
plt.title('Przykład wykresu liniowego z parametrem hue')
plt.xlabel('Rok')
plt.ylabel('Wartość')

# Wyświetlenie wykresu
plt.show()
```

2. Parametr `style`: pozwala na zmianę stylu linii na podstawie wartości zmiennej kategoryjnej.

```
sns.lineplot(x='year', y='value', style='category', data=df)
```

3. Parametr `markers`: pozwala na dodanie znaczników do punktów danych.

```
sns.lineplot(x='year', y='value', markers=True, data=df)
```

4. Parametr `ci`: pozwala na dodanie przedziału ufności (confidence interval) dla wykresu, który może być użyteczny w przypadku wielokrotnych pomiarów dla tych samych wartości na osi X.

```
sns.lineplot(x='year', y='value', ci=95, data=df)
```

7.4 Style

Biblioteka Seaborn pozwala na dostosowywanie estetyki wykresów za pomocą palet kolorów i stylów, co pozwala na uzyskanie atrakcyjnych i czytelnych wizualizacji danych. Oto kilka przykładów, jak możemy dostosować estetykę wykresów w Seaborn:

1. Palety kolorów:

Palety kolorów pozwalają na zmianę kolorystyki wykresów. Seaborn oferuje kilka wbudowanych palet kolorów, które można łatwo zastosować do wykresów.

Przykład użycia palety kolorów 'coolwarm' w wykresie punktowym:

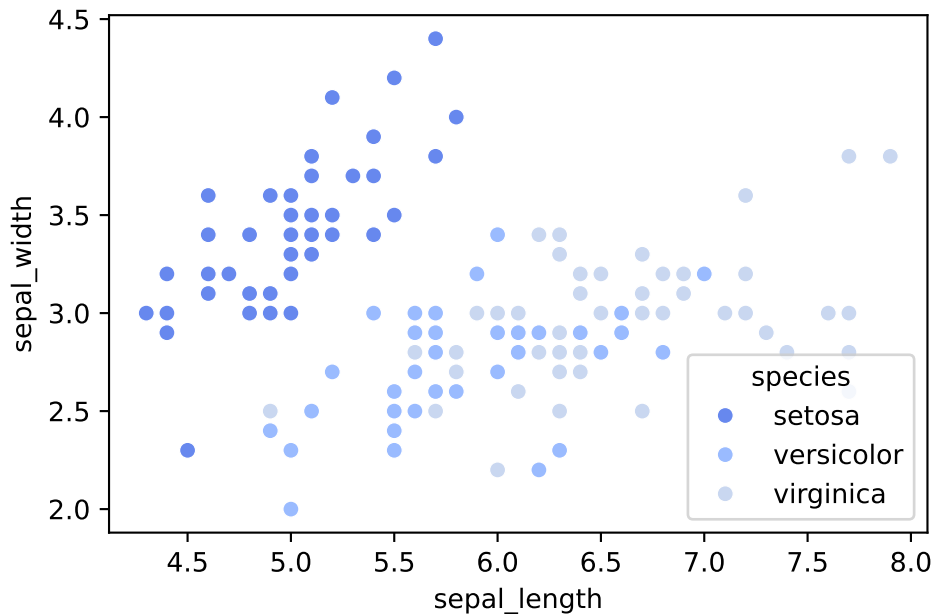
```
import seaborn as sns
import matplotlib.pyplot as plt

iris_data = sns.load_dataset('iris')

# Ustawienie palety kolorów 'coolwarm'
sns.set_palette('coolwarm')

sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=iris_data)

plt.show()
```



Aby zresetować paletę kolorów do domyślnej, użyj `sns.set_palette('deep')`.

2. Style wykresów:

Seaborn pozwala na zmianę stylu wykresów, co może wpłynąć na ogólny wygląd i czytelność wizualizacji. Biblioteka oferuje kilka wbudowanych stylów.

Przykład użycia stylu 'whitegrid' w wykresie punktowym:

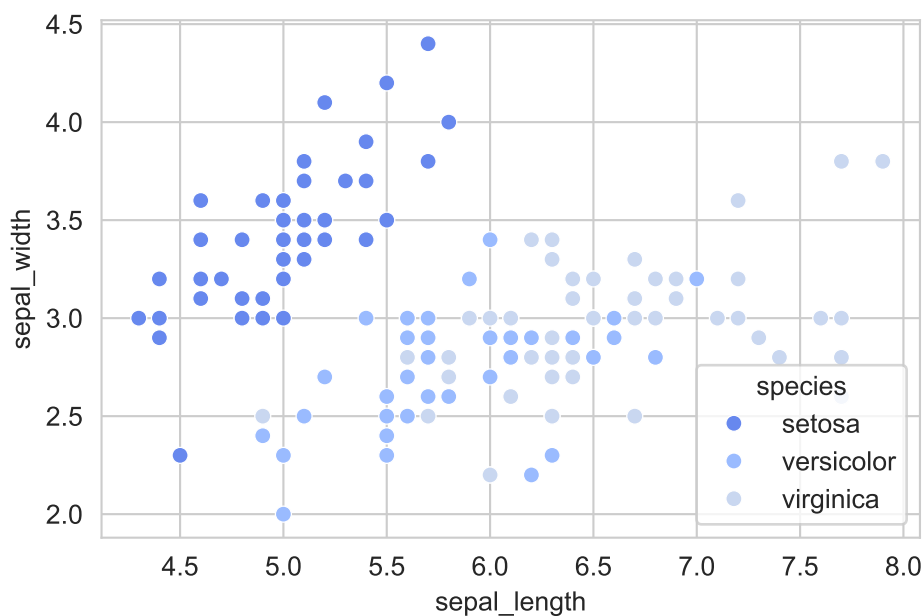
```
import seaborn as sns
import matplotlib.pyplot as plt

iris_data = sns.load_dataset('iris')

# Ustawienie stylu 'whitegrid'
sns.set_style('whitegrid')

sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=iris_data)

plt.show()
```



Inne dostępne style to: 'darkgrid', 'white', 'ticks' i 'dark'. Aby zresetować styl do domyślnego, użyj `sns.set_style('darkgrid')`.

3. Skalowanie elementów graficznych:

Seaborn pozwala na skalowanie elementów graficznych, co może być przydatne w przypadku różnych rozdzielczości ekranów lub potrzeby zmiany rozmiaru wykresów.

Przykład użycia kontekstu 'poster' w wykresie punktowym:

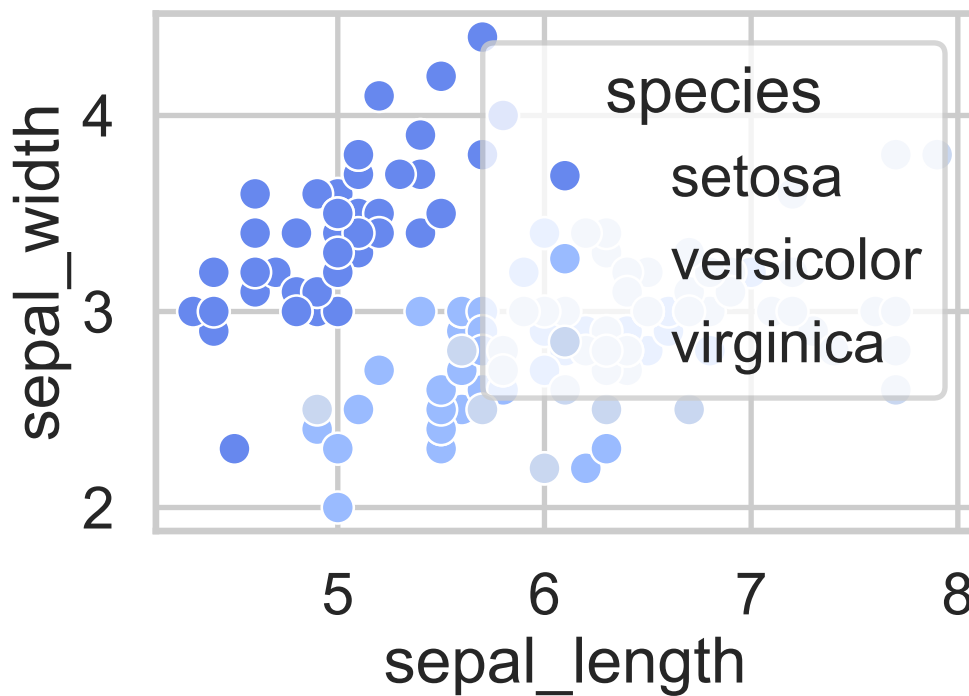
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
iris_data = sns.load_dataset('iris')

# Ustawienie kontekstu 'poster'
sns.set_context('poster')

sns.scatterplot(x='sepal_length', y='sepal_width', hue='species', data=iris_data)

plt.show()
```



Inne dostępne konteksty to: 'paper', 'notebook' i 'talk'. Aby zresetować kontekst do domyślnego, użyj `sns.set_context('notebook')`.

Kombinując te metody, można tworzyć różnorodne, atrakcyjne i czytelne wizualizacje danych.