



Analiza i wizualizacja danych

Piotr Jastrzębski

2025-05-06

Spis treści

1 Analiza i wizualizacja danych	8
I Wprowadzenie	9
2 Trochę teorii...	10
2.1 Test racjonalnego myślenia	10
2.2 Analiza danych	10
2.3 Wizualizacja danych	10
2.4 Analiza danych - podstawowe pojęcia	11
2.4.1 Współczesne znaczenia słowa “statystyka”:	11
2.4.2 “Masowość”	11
2.4.3 Podział statystyki	12
2.4.4 Zbiorowość/populacja	12
2.4.5 Jednostka statyczna	12
2.4.6 Cechy statystyczne	12
2.4.7 Skale	14
2.5 Rodzaje badań statystycznych	15
2.6 Etapy badania statystycznego	16
2.7 Analiza danych zastanych	16
2.8 Proces analizy danych	17
2.8.1 Zdefiniowanie wymagań	17
2.8.2 Gromadzenie danych	17
2.8.3 Przetwarzanie danych	18
2.8.4 Właściwa analiza danych	18
2.8.5 Raportowanie i dystrybucja wyników	18
2.9 Skąd brać dane?	18
2.10 Koncepcja “Tidy data”	19
2.10.1 Zasady “czystych danych”	19
2.10.2 Przykłady nieuporządkowanych danych	19
2.11 Parę rad na dobre prezentacje	20
2.11.1 Współczynnik kłamstwa	20
2.11.2 Współczynnik kłamstwa	20
2.12 Bibliografia	21

II NumPy	22
3 NumPy - start	23
3.1 Instalacja pakietu NumPy - opcja łatwiejsza “do przeklikania”	23
3.2 Instalacja pakietu NumPy - opcja terminala	24
3.3 Import biblioteki NumPy	24
3.4 Uruchamianie - tryb “Run” (wykonawczy)	25
3.5 Uruchamianie - tryb “Run in Python Console” (interaktywno-wykonawczy)	27
4 Lista a tablica	29
5 Atrybuty tablic ndarray	30
6 Typy danych	32
7 Tworzenie tablic	33
8 Indeksowanie, “krojenie”	42
9 Modyfikacja kształtu i rozmiaru	49
10 Broadcasting	56
11 Funkcje uniwersalne (ufunc)	62
11.1 Podstawowe operacje arytmetyczne	62
11.2 Funkcje trygonometryczne i pochodne	63
11.3 Funkcje wykładnicze i logarytmiczne	63
11.4 Funkcje zaokrąglające i wartości bezwzględne	64
11.5 Funkcje statystyczne i agregujące	64
12 Operacje na stringach	68
12.1 Tworzenie tablic z napisami	68
12.2 Podstawowe funkcje do modyfikacji tekstu	68
12.2.1 <code>numpy.strings.upper</code> i <code>numpy.strings.lower</code>	68
12.2.2 <code>numpy.strings.capitalize</code>	69
12.2.3 <code>numpy.strings.title</code>	69
12.3 Łączenie i rozdzielanie tekstów	70
12.3.1 <code>numpy.strings.add</code>	70
12.3.2 <code>numpy.strings.join</code>	70
12.3.3 <code>numpy.strings.split</code>	70
12.4 Wyszukiwanie i zamiana podciągów	71
12.4.1 <code>numpy.strings.find</code> i <code>numpy.strings.rfind</code>	71
12.4.2 <code>numpy.strings.replace</code>	71

12.5 Usuwanie zbędnych znaków	72
12.5.1 <code>numpy.strings.strip</code> , <code>numpy.strings.lstrip</code> i <code>numpy.strings.rstrip</code>	72
13 Alegbra liniowa	73
13.1 Iloczyn skalarny (dot product)	73
13.2 Mnożenie macierzowe	73
13.3 Mnożenie macierz-wektor	74
13.4 Rozwiązywanie układów równań liniowych	74
13.5 Wyznacznik macierzy	75
13.6 Wartości i wektory własne	75
13.7 Rozkład wartości osobliwych (SVD)	76
13.8 Norma macierzy/wektora	76
13.9 Macierz odwrotna	77
13.10 Funkcja <code>numpy.inner</code> - iloczyn wewnętrzny	78
13.11 Funkcja <code>numpy.outer</code> - iloczyn zewnętrzny	78
13.12 Funkcja <code>numpy.matmul</code> - mnożenie macierzowe	79
14 Filtrowanie zaawansowane	80
14.1 Funkcja <code>nonzero()</code>	80
14.2 Funkcja <code>where()</code>	81
14.3 Funkcje <code>indices()</code> i <code>ix_()</code>	82
14.3.1 <code>indices()</code>	82
14.3.2 <code>ix_()</code>	83
14.4 <code>ogrid</code> i operacje na siatkach	84
14.5 Funkcje <code>ravel_multi_index()</code> i <code>unravel_index()</code>	84
14.6 Indeksy diagonalne	85
14.7 3.1 Funkcja <code>take()</code>	86
14.8 Funkcja <code>choose()</code>	87
14.9 Funkcja <code>compress()</code>	87
14.10 Funkcje <code>diag()</code> i <code>diagonal()</code>	87
14.11 Funkcja <code>select()</code>	88
14.12 Funkcja <code>place()</code>	89
14.13 Funkcja <code>put()</code>	89
14.14 Funkcja <code>put_along_axis()</code>	89
14.15 Funkcja <code>putmask()</code>	90
14.16 Funkcja <code>fill_diagonal()</code>	90
15 Numpy - inne	92
15.1 Stałe	92
15.2 <code>numpy.inf</code>	93
15.3 <code>numpy.nan</code>	94
15.4 <code>numpy.newaxis</code>	94
15.5 Statystyka i agregacja	95

III Eksploracja danych	97
16 Etapy eksploracji danych	98
IV Pandas	99
17 Pandas - start	100
17.1 Import:	100
17.2 Podstawowe byty	100
18 Pandas - indeksowanie	106
19 Ładowanie danych	109
19.1 Obsługa plików csv	109
19.2 Obsługa plików z Excela	109
19.3 Obsługa sqlite3	110
20 Pandas - sortowanie	111
21 Pandas - szeregi czasowe	114
22 Pandas - dane tekstowe	118
22.1 Normalizacja	118
22.2 Operacje wektorowe na tekstach	120
23 Pandas - inne	123
23.1 Uzupełnianie braków	123
23.2 Obsługa brakujących danych	124
23.3 Usuwanie duplikatów	125
23.4 Zastępowanie wartościami	126
23.5 Dyskretyzacja i podział na koszyki	127
23.6 Wykrywanie i filtrowanie elementów odstających	129
23.7 Zmiana typu w kolumnie	130
23.8 Zmiana znaku kategoriach	132
23.9 Operacje manipulacyjne	133
V Analia struktury danych	155
24 Analiza struktury	156
24.1 Miary położenia	156
24.1.1 Średnia arytmetyczna	156
24.1.2 Dominanta	157
24.1.3 Mediana	158

24.1.4 Kwantyle	159
24.2 Miary zmienności	159
24.2.1 Rozstęp	160
24.2.2 Rozstęp międzykwartylowy	160
24.2.3 Odchylenie ćwiartkowe	160
24.2.4 Wariancja	161
24.2.5 Odchylenie standardowe	161
24.3 Miary asymetrii	162
24.4 Miary koncentracji	162
24.5 Przyśpieszanie działania:	163
VI Matplotlib	166
25 Matplotlib - start	167
26 Matplotlib - wykres liniowy	170
27 Matplotlib - dodatki cz.1	177
27.1 Parametry legendy	177
27.2 Style, kolory linii	179
28 Matplotlib - podwykresy	186
29 Matplotlib - zapis	192
30 Matplotlib - wykres punktowy	194
31 Matplotlib - wykres kołowy	206
31.1 Wykres pierścieniowy	216
32 Matplotlib - wykres słupkowy	219
33 Matplotlib - regresja	237
34 Matplotlib - kolory	243
35 Matplotlib - opcje wykresu	251
35.1 Argumenty <code>figure</code> w Matplotlib	251
35.2 Style	256
36 Matplotlib - dodatki cz.2.	258
36.1 Linie poziome i pionowe	258
36.2 Adnotacje (tekst) na wykresie	260
36.3 Etykiety osi	262

36.4 Etykiety podziałki osi	264
36.5 Siatka	266
37 Matplotlib - inne wykresy	270
37.1 Wykres kołowy	270
37.2 Wykres dwuosiowy	275
37.3 Wykres słupkowy	277
37.4 Wykres pudełkowy	290
37.5 Histogram	294
37.6 Wykres pierścieniowy	300
37.7 Wykresy w przestrzeni	301
37.7.1 Helisa	301
37.7.2 Torus	302
VII Zadania problemowe	304
38 Problem #1	305
39 Problem #2	308

1 Analiza i wizualizacja danych

Aktualna wersja dotyczy zajęć realizowanych w roku akademickim 2024/25.

Część I

Wprowadzenie

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 lili 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 pilka?

2.2 Analiza danych

Analiza danych to proces badania, czyszczenia, przekształcania i modelowania danych w celu odkrywania użytecznych informacji, formułowania wniosków i wspierania podejmowania decyzji. Jest to wieloetapowy proces, który obejmuje:

- Zbieranie danych z różnych źródeł
- Czyszczenie danych poprzez usuwanie błędów, braków i niespójności
- Eksplorację danych w celu zrozumienia ich struktury i cech charakterystycznych
- Przekształcanie danych do odpowiedniego formatu
- Stosowanie metod statystycznych i algorytmów uczenia maszynowego
- Interpretację wyników w kontekście konkretnego problemu biznesowego lub naukowego

Analiza danych znajduje zastosowanie w niemal każdej dziedzinie, od biznesu i finansów po nauki społeczne, medycynę i badania naukowe. Celem analizy danych jest przekształcenie surowych danych w wiedzę, która może być wykorzystana do podejmowania lepszych decyzji.

2.3 Wizualizacja danych

Wizualizacja danych to graficzna reprezentacja informacji i danych. Wykorzystuje elementy wizualne, takie jak wykresy, mapy i dashboardy, aby przedstawić relacje między danymi w sposób, który jest łatwy do zrozumienia i interpretacji. Dobra wizualizacja danych:

- Przedstawia złożone informacje w przystępny i intuicyjny sposób
- Ujawnia wzorce, trendy i odstępstwa, które mogą być trudne do zauważenia w surowych danych
- Wspiera proces analizy danych poprzez umożliwienie szybkiego przeglądania dużych zbiorów danych
- Ułatwia komunikację wyników analiz do różnych odbiorców, w tym osób nietechnicznych
- Pomaga opowiadać historie zawarte w danych (data storytelling)

Do najpopularniejszych typów wizualizacji danych należą wykresy słupkowe, liniowe, kołowe, mapy cieplne, drzewa hierarchiczne, chmury słów oraz interaktywne dashboardy. Wybór odpowiedniej formy wizualizacji zależy od typu danych, celu prezentacji oraz docelowej grupy odbiorców.

Wizualizacja danych jest kluczowym elementem procesu analizy danych, ponieważ pozwala na szybkie wyciąganie wniosków i podejmowanie decyzji na podstawie danych. Jest mostem między złożonymi danymi a ludzkim zrozumieniem.

2.4 Analiza danych - podstawowe pojęcia

2.4.1 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.

2.4.2 “Masowość”

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

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

2.4.3 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.

2.4.4 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.

2.4.5 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

2.4.6 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 (miejscie lub obszar) dotyczy badanie.
- przykład: studenci WMiI UWM w Olsztynie w roku akad. 2017/2018:
 - cecha rzeczowa: posiadanie legitymacji studenckiej,
 - cecha czasowa - studenci studiujejący 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 określona jednostką miary.
- cechy niemierzalne (jakościowe) - określone 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).

2.4.7 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
Budyzm	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 °C	Temperatura w °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

2.5 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.

2.6 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.

2.7 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.

2.8 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.

2.8.1 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 kategoryczne (tj. Etykiety tekstowe). Faza definiowania wymagań powinna dać odpowiedź na 2 zasadnicze pytania:

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

2.8.2 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.

2.8.3 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. Niezwykłe istotne jest też zwracanie uwagi na dane których wartości są powyżej lub poniżej ustalonych wcześniej progów (ekstrema).

2.8.4 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.

2.8.5 Raportowanie i dystrybucja wyników

Ta faza polega na ustalaniu w jakiej formie przekazywać wyniki. Analityk może rozważyć różne techniki wizualizacji danych, aby w sposób wyraźny 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.

2.9 Skąd brać dane?

Darmowa repozytoria danych:

- Bank danych lokalnych GUS - [link](#)

- Otwarte dane - [link](#)
- Bank Światowy - [link](#)

2.10 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. Date accessed: 25 oct. 2018. doi:<http://dx.doi.org/10.18637/jss.v059.i10>.

2.10.1 Zasady “czystych danych”

Idealne dane są zaprezentowane w tabeli:

Imię	Wiek	Wzrost	Kolor oczu
Adam	26	167	Brązowe
Sylwia	34	164	Piwne
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

2.10.2 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.

2.11 Parę rad na dobre prezentacje

Edward Tufte, prof z Yale

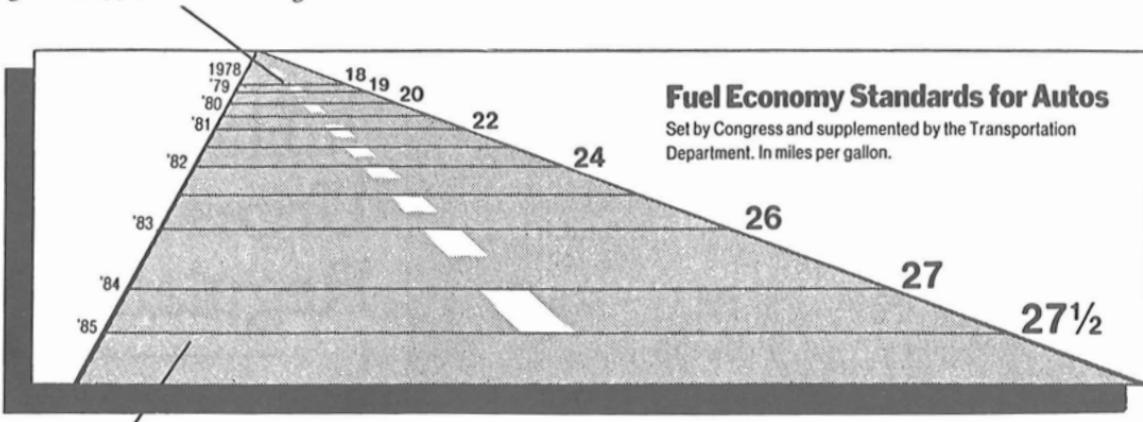
1. Prezentuj dane “na bogato”.
2. Nie ukrywaj danych, pokazuj prawdę.
3. Nie używaj wykresów śmieciowych.
4. Pokazuj zmienność danych, a nie projektuj jej.
5. Wykres ma posiadać jak najmniejszy współczynnik kłamstwa (lie-factor).
6. Powerpoint to зло!

2.11.1 Współczynnik kłamstwa

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

2.11.2 Współczynnik kłamstwa

This line, representing 18 miles per gallon in 1978, is 0.6 inches long.

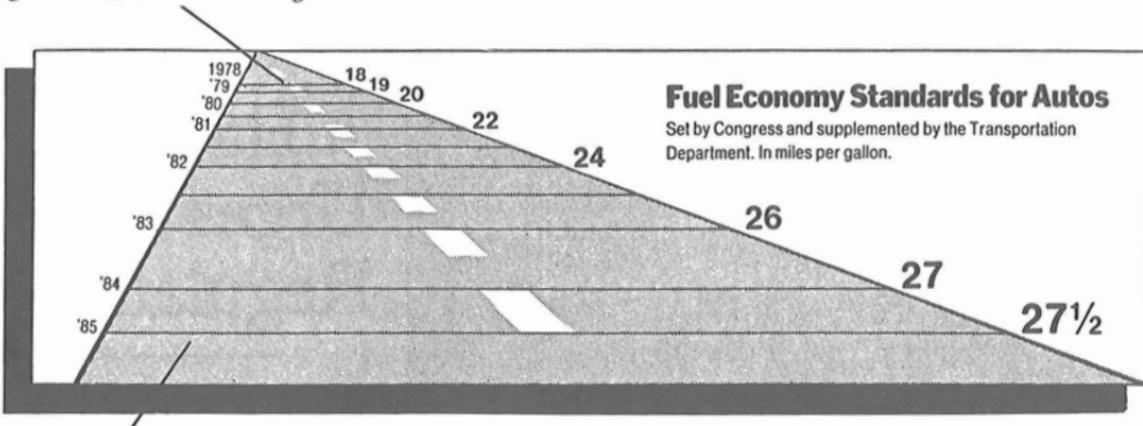


[Tufte, 1991] Edward Tufte, The Visual Display of Quantitative Information, Second Edition, Graphics Press, USA, 1991, p. 57 – 69.

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

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

This line, representing 18 miles per gallon in 1978, is 0.6 inches long.



This line, representing 27.5 miles per gallon in 1985, is 5.3 inches long.

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

2.12 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.

Część II

NumPy

3 NumPy - start

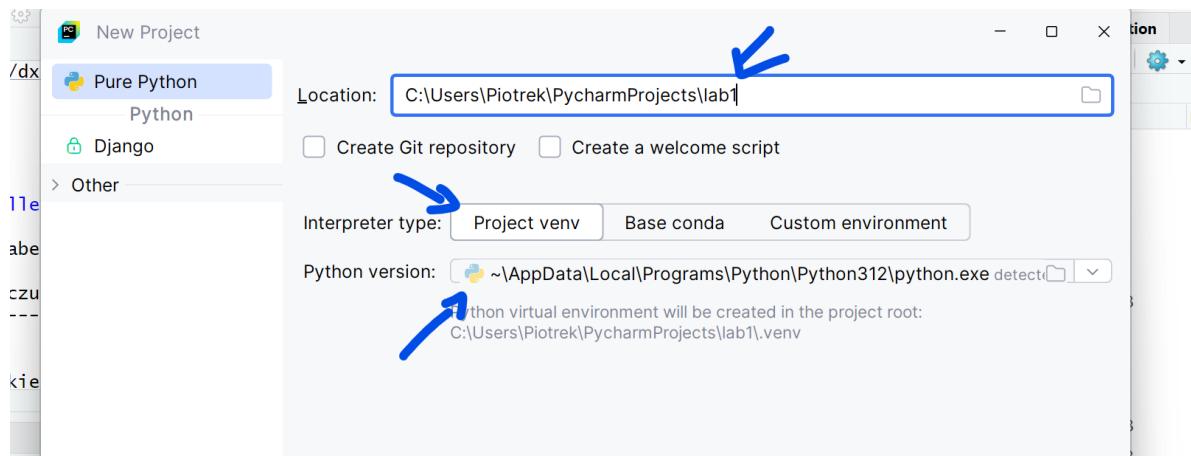
NumPy jest biblioteką Pythona służącą do obliczeń naukowych.

Zastosowania:

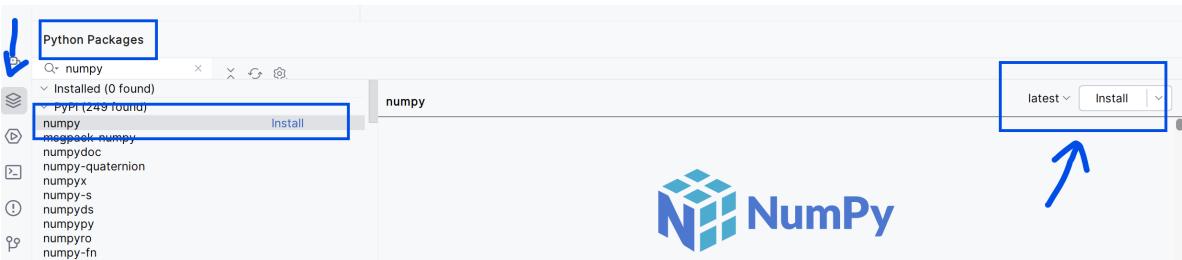
- algebra liniowa
- zaawansowane obliczenia matematyczne (numeryczne)
- całkowania
- rozwiązywanie równań
- ...

3.1 Instalacja pakietu NumPy - opcja łatwiejsza “do przeklikania”

- Tworzy projekt w PyCharm z venv - wersja 3.12.



- Za pomocą zakładki po lewej stronie na dole wyszukujemy pakiet i wybieramy instalację



3.2 Instalacja pakietu NumPy - opcja terminala

Komenda dla terminala:

```
python -m pip install numpy
```

```
python -m pip install numpy==2.2.0
```

3.3 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)
```

(1)

```

print("typ a:", type(a))                                     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.asmatrix('1 2; 3 4'))                      ⑦
print("f:", f)
g = np.array(np.asmatrix('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.

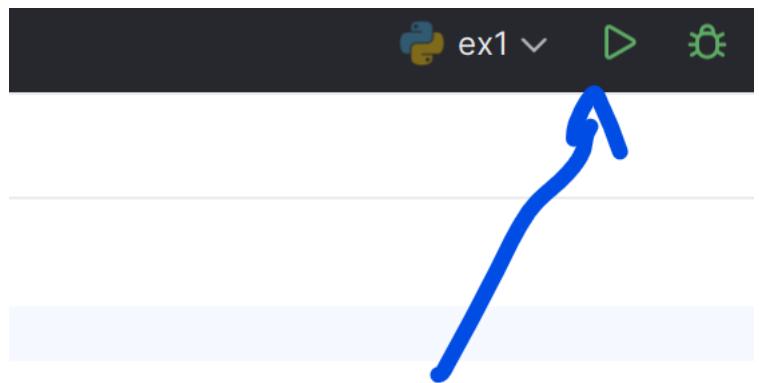
```

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'>

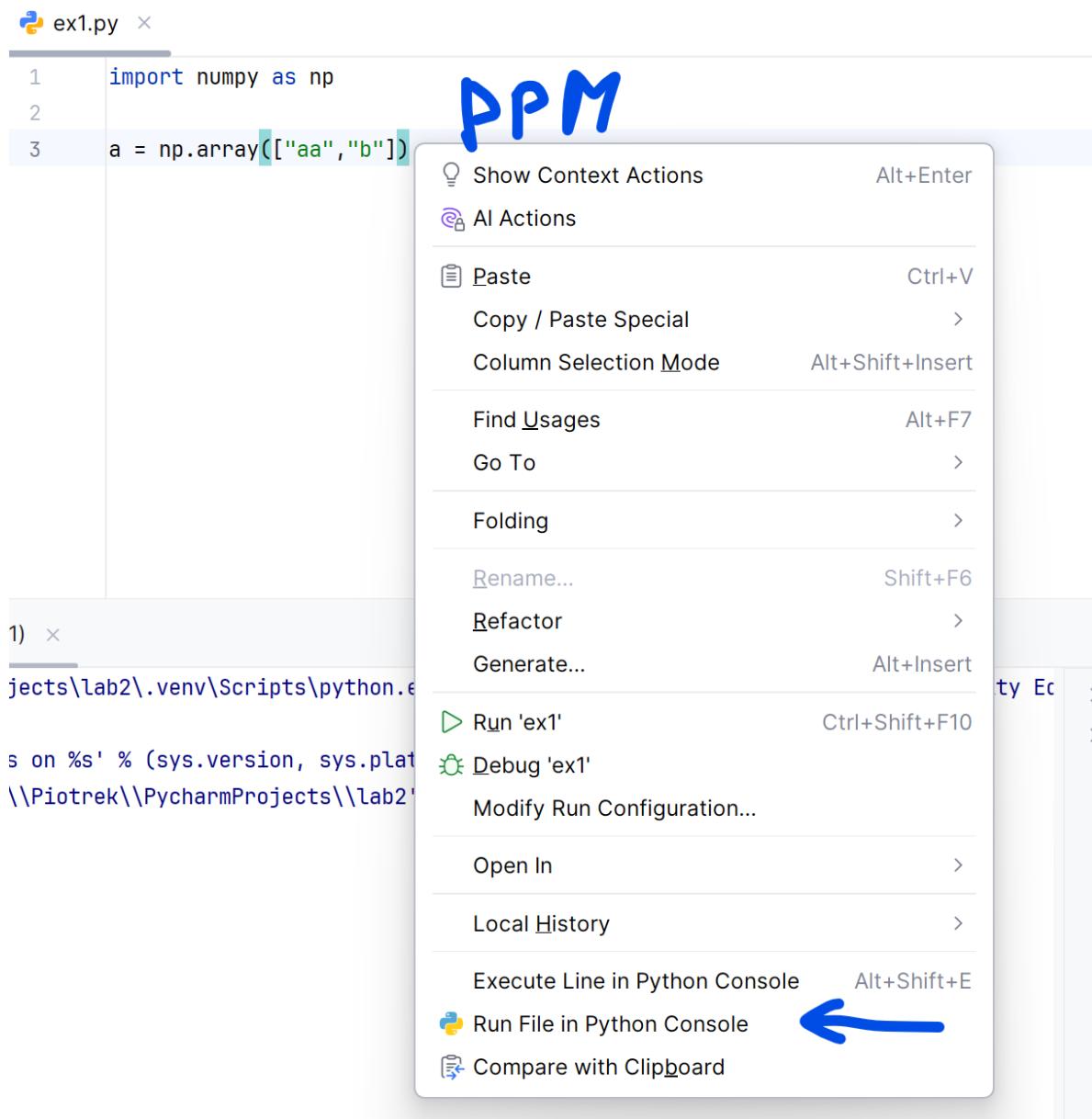
```

3.4 Uruchamianie - tryb “Run” (wykonawczy)

Run - zielona strzałka u góry.



3.5 Uruchamianie - tryb “Run in Python Console” (interaktywno-wykonawczy)



Ćwiczenie (ex1.py):

1. Stwórz proste tablice:

- $\begin{bmatrix} 1 & 2 & 7 \\ 6 & -3 & -3 \end{bmatrix}$
- $[6 \ 8 \ 9 \ -3]$
- $\begin{bmatrix} 4 \\ 3 \\ -3 \\ -7 \end{bmatrix}$
- $[bb \ cc \ ww \ 44]$

4 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))
```

```
--- 0.0022945404052734375 seconds ---
--- 0.037451744079589844 seconds ---
```

5 Atrybuty tablic ndarray

Atrybut	Opis
<code>shape</code>	krotka z informacją o liczbie elementów dla każdego z wymiarów
<code>size</code>	liczba elementów w tablicy (łączna)
<code>ndim</code>	liczba wymiarów tablicy
<code>nbytes</code>	liczba bajtów jaką tablica zajmuje w pamięci
<code>dtype</code>	typ danych

```
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: 40
 dtype: int64
```

```
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: 32
dtype: int64
```

NumPy nie wspiera postrzepionych tablic! Poniższy kod wygeneruje błąd:

```
import numpy as np

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

Ćwiczenia: (ex2.py)

Utwórz tablice numpy:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

$$C = \begin{bmatrix} 1.1 & 2.2 & 3.3 \\ 4.4 & 5.5 & 6.6 \end{bmatrix}$$

i sprawdź ich parametry.

6 Typy danych

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]]
```

7 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.]]]
(2, 3, 4)
```

`np.ones` - tworzy tablice 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.]]
```

```
[[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 dopuszczaamy 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]
```

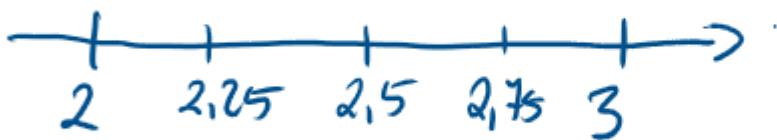
`np.linspace` - tablica wypełniona równomiernymi wartościami wg skali liniowej

```
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]
```

początek koniec liczba punktów
 np. `linspace(2.0, 3.0, num=5)`



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

`endpoint = False`

- wybrane ostatni punkt
(z przedej strony)

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

`dtype <-- ustala typ`

zwykłe używa się int
(wynik może mieć więcej miejsc ujemnych)

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)

```

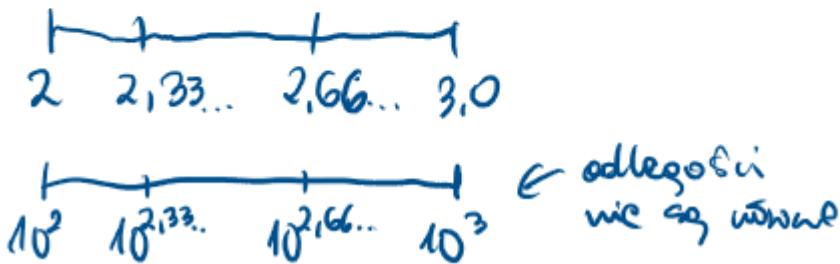
```

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

```

$\text{np.logspace}(2.0, 3.0, \text{num}=4)$

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 4607182418800017408 4611686018427387904]
```

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.]]
```

Ćwiczenia: (ex3.py)

- Utwórz jednowymiarową tablicę zawierającą liczby całkowite od 1 do 5 i przypisz ją do zmiennej A. Wynikowa tablica powinna mieć postać:

$$[1 \ 2 \ 3 \ 4 \ 5]$$

- Utwórz dwuwymiarową tablicę zawierającą elementy:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

i przypisz ją do zmiennej B.

- Utwórz tablicę zawierającą liczby od 0 do 9 (włącznie). Przypisz ją do zmiennej C. Oczekiwana postać:

$$[0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9]$$

- Utwórz tablicę zawierającą liczby od 10 do 30 z krokiem 5. Przypisz do D. Oczekiwana postać:

$$[10 \ 15 \ 20 \ 25 \ 30]$$

- Utwórz tablicę 5 wartości równomiernie rozłożonych pomiędzy 0 a 1. Przypisz do E. Przykładowa postać:

$$[0. \ 0.25 \ 0.5 \ 0.75 \ 1.]$$

- Utwórz dwuwymiarową tablicę o wymiarach 2x3 wypełnioną zerami. Przypisz do F. Oczekiwana postać:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Korzystając z np. `eye` utwórz macierz jednostkową 4x4. Przypisz do J. Oczekiwana postać:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

8 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])                                              ①
print("2:", a[-2])                                             ②
print("3:", a[3:6])                                            ③
print("4:", a[:])                                              ④
print("5:", a[0:-1])                                            ⑤
print("6:", a[: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:])                                              ①
print("2:", a[4:-1])                                             ②
print("3:", a[4:10:2])                                           ③
print("4:", a[::-1])                                             ④
print("5:", a[::-2])                                             ⑤
print("6:", a[::-2])                                             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 czwórką, 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, maski boolowskie)

```
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)
d = a[(a > 5) & (a%2 !=0)] # znak & odpowiada za AND
print(d)
e = a[(a > 5) | (a%2 !=0)] # znak | odpowiada za OR
print(e)
f = a[(a > 5) ^ (a%2 !=0)] # znak ^ odpowiada za XOR
print(f)
g = a[~(a > 0)]
print(g)

```

```

[ True  True False  True False  True  True  True False False  True
 True]
[ 2  5  4  8  9 11  8  1]
[ 9 11]
[ 5  -7   8   9  11 -23  -7   8   1]
[ 5  -7   8 -23  -7   8   1]
[ -2  -7 -23  -4  -7]

```

```

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]

```

Ćwiczenia: (ex4.py)

1. Rozważ jednowymiarową tablicę

$$A = [10 \ 20 \ 30 \ 40 \ 50].$$

Napisz polecenie , które zwróci trzeci element tablicy. Następnie spróbuj pobrać przedział od drugiego do czwartego elementu włącznie.

2. Dla tej samej tablicy

$$A = [10 \ 20 \ 30 \ 40 \ 50],$$

użyj “fancy indexing”, aby wybrać elementy o indeksach [0, 2, 4]. Spróbuj także wykorzystać negatywne indeksy, aby wybrać ostatni i przedostatni element w jednej operacji.

3. Rozważ dwuwymiarową tablicę

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Napisz polecenie, które zwróci drugi wiersz (jako tablicę jednowymiarową). Następnie pobierz cały pierwszy wiersz oraz dwie pierwsze kolumny.

4. Dla tablicy

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

użyj “fancy indexing”, aby wybrać elementy ($B_{1,1}, B_{0,2}, B_{2,0}$) za pomocą list indeksów w numpy. Otrzymaj wynik w postaci tablicy jednowymiarowej [5, 3, 7].

5. Rozważ tablicę

$$C = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{bmatrix}.$$

Napisz polecenie, które zwróci wszystkie elementy drugiego wiersza oprócz ostatniego. Następnie pobierz co drugi element z pierwszego wiersza.

6. Dla tablicy

$$C = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \end{bmatrix},$$

użyj “fancy indexing”, aby pobrać elementy pierwszego wiersza w kolejności [30, 10, 40] korzystając z tablicy indeksów np. [2, 0, 3]. Następnie zastosuj “fancy indexing” do drugiego wiersza, aby uzyskać [80, 50].

7. Rozważ jednowymiarową tablicę

$$D = [5 \ 10 \ 15 \ 20 \ 25 \ 30].$$

Za pomocą indeksowania wytnij ostatnie trzy elementy. Następnie pobierz wszystkie elementy o parzystych indeksach.

8. Dla tablicy

$$D = [5 \ 10 \ 15 \ 20 \ 25 \ 30],$$

użyj “fancy indexing” za pomocą maski boolowskiej (utwórz maskę wybierającą elementy większe niż 15) i otrzymaj odpowiednio przefiltrowaną tablicę. Następnie zastosuj tę maskę do pobrania konkretnych elementów.

9. Rozważ tablicę dwuwymiarową

$$E = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}.$$

Za pomocą indeksowania wybierz środkowy wiersz i wszystkie kolumny oprócz ostatniej. Następnie wybierz ostatni wiersz i ostatnią kolumnę.

10. Dla tablicy

$$E = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix},$$

użyj “fancy indexing”, aby w jednej operacji pobrać elementy ($E_{0,2}, E_{2,1}$) i ułożyć je w nowej tablicy. Spróbuj także stworzyć maskę boolowską wybierającą elementy większe niż 10 i pobrać wybrane wartości.

9 Modyfikacja kształtu i rozmiaru

```
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]]
```

Ćwiczenia: (ex5.py)

1. Rozważ tablicę jednowymiarową

$$A = [1 \ 2 \ 3 \ 4 \ 5 \ 6].$$

Przekształć ją tak, aby uzyskać tablicę dwuwymiarową o kształcie 2×3 .

2. Mając tablicę dwuwymiarową

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix},$$

uzyskaj jednowymiarowy “widok” jej elementów bez zmiany w danych źródłowych.

3. Rozważ tablicę

$$D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

Zmień jej orientację tak, aby wiersze stały się kolumnami, a kolumny wierszami.

4. Mając dwie tablice

$$E_1 = [1 \ 2 \ 3], \quad E_2 = [4 \ 5 \ 6],$$

połącz je w poziomie, tworząc jedną tablicę.

5. Dwie tablice

$$F_1 = [1 \ 2 \ 3], \quad F_2 = [4 \ 5 \ 6],$$

połącz w pionie, aby uzyskać tablicę o kształcie 2×3 .

6. Dla tablicy

$$G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix},$$

zmień jej rozmiar tak, aby stała się tablicą jednowymiarową o 4 elementach. Pozostałe elementy usuń.

7. Mając tablicę

$$H = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix},$$

usuń drugą kolumnę, otrzymując tablicę 3×2 .

8. Rozważ tablicę

$$I = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix},$$

zmień jej kształt tak, aby uzyskać tablicę 2×4 .

9. Mając tablicę

$$J = [1 \ 2 \ 3 \ 4],$$

przekształć ją w tablicę dwuwymiarową 2×2 , a następnie “spłaszcz” ją z powrotem do postaci jednowymiarowej.

10 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]]
```

The diagram shows three matrices. The first is a 3x2 matrix with values 1, 2, 5, 6, 9, 10. The second is a 3x2 matrix where every element is 4. The third is a 3x2 matrix with values 5, 6, 9, 10, 13, 14. An equals sign between the second and third matrices indicates they are the result of the addition operation.

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

```
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]]
```

$$\begin{array}{c}
 \text{a } 2 \times 2 \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 6 \\ \hline \end{array}
 \end{array}
 +
 \begin{array}{c}
 \text{b } 2 \times 1 \\
 \begin{array}{|c|c|} \hline 9 & 2 \\ \hline 9 & 2 \\ \hline \end{array}
 \end{array}
 =
 \begin{array}{c}
 \text{2x2} \\
 \begin{array}{|c|c|} \hline 10 & 4 \\ \hline 14 & 8 \\ \hline \end{array}
 \end{array}$$

$$\begin{array}{c}
 \text{a } 2 \times 2 \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5 & 6 \\ \hline \end{array}
 \end{array}
 +
 \begin{array}{c}
 \text{c } 2 \times 2 \\
 \begin{array}{|c|c|} \hline 4 & 4 \\ \hline -2 & -2 \\ \hline \end{array}
 \end{array}
 =
 \begin{array}{c}
 \text{2x2} \\
 \begin{array}{|c|c|} \hline 5 & 6 \\ \hline 3 & 4 \\ \hline \end{array}
 \end{array}$$

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]
    
```

$$\begin{bmatrix} 6 & -4 & 2 & 4 & 8 \\ -9 & 6 & -3 & -6 & -12 \end{bmatrix}$$

a

5	5	5	5	5
2	2	2	2	2
-3	-3	-3	-3	-3

b

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

1x5

=

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

3x5

Ćwiczenia: (ex6.py)

- Rozważ jednowymiarową tablicę

$$A = [1 \ 2 \ 3]$$

oraz skalar $k = 10$.

Wykonaj dodawanie, odejmowanie, mnożenie i dzielenie każdego elementu tablicy A przez k z wykorzystaniem broadcastingu.

- Dla dwóch tablic jednowymiarowych

$$B_1 = [1 \ 2 \ 3], \quad B_2 = [4 \ 5 \ 6],$$

wykonaj działanie $B_1 + B_2$, $B_1 - B_2$, $B_1 * B_2$ oraz B_1 / B_2 używając broadcastingu.

- Mając dwie tablice dwuwymiarowe:

$$C_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix},$$

dodaj je i odejmij od siebie, sprawdzając czy broadcasting zajdzie automatycznie.

- Rozważ tablicę dwuwymiarową

$$D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

oraz wektor

$$v = [10 \ 100 \ 1000].$$

Wykonaj mnożenie i dzielenie elementowe tablicy D przez v z wykorzystaniem broadcastingu.

5. Dla tablicy

$$E = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{bmatrix}$$

podnieś każdy element do kwadratu, a następnie podziel przez wektor

$$w = [2 \ 2 \ 2]$$

korzystając z broadcastingu.

6. Mając tablicę dwuwymiarową

$$F = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix},$$

oraz skalar $s = 2$, wykonaj $F * s$, a następnie F^s (podnieś każdy element do potęgi s) z zastosowaniem broadcastingu.

7. Rozważ tablicę

$$G = [10 \ 20 \ 30]$$

oraz kolumnową tablicę dwuwymiarową

$$h = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Dodaj do h tablicę G i zaobserwuj wynik broadcastingu.

8. Mając dwie tablice dwuwymiarowe o różnych wymiarach:

$$H_1 = [1 \ 2 \ 3], \quad H_2 = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix},$$

spróbuj je dodać i pomnożyć przez siebie, korzystając z broadcastingu.

9. Rozważ tablicę dwuwymiarową

$$J = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

oraz skalar $m = 5$.

Wykonaj kombinację działań: najpierw pomnóż J przez m , następnie odejmij m , a na końcu podziel wynik przez m – wszystko z wykorzystaniem broadcastingu.

11 Funkcje uniwersalne (ufunc)

Funkcje uniwersalne (tzw. *ufunc*) to jedne z najważniejszych narzędzi w NumPy. Są to funkcje działające element-po-elementie na tablicach, często implementowane w C, co zapewnia wysoką wydajność obliczeń. Dzięki *ufuncs* można w prosty i czytelny sposób wykonywać operacje arytmetyczne, trygonometryczne, statystyczne czy logiczne na całych tablicach bez konieczności pisania pętli w Pythonie.

11.1 Podstawowe operacje arytmetyczne

NumPy automatycznie przekształca operatory matematyczne w odpowiednie *ufunc*.

Na przykład:

- + odpowiada `np.add`
- - odpowiada `np.subtract`
- * odpowiada `np.multiply`
- / odpowiada `np.divide`
- ** odpowiada `np.power`

Przykład:

```
import numpy as np

A = np.array([1, 2, 3, 4])
B = np.array([10, 20, 30, 40])

# Operacje element-po-elemencie
sum_tab = np.add(A, B)      # to samo co A + B
diff_tab = np.subtract(B, A) # to samo co B - A
mul_tab = np.multiply(A, 2)  # to samo co A * 2
pow_tab = np.power(A, 3)    # to samo co A ** 3

print("Suma:", sum_tab)
print("Różnica:", diff_tab)
print("Mnożenie przez 2:", mul_tab)
print("Potęgowanie:", pow_tab)
```

```
Suma: [11 22 33 44]  
Różnica: [ 9 18 27 36]  
Mnożenie przez 2: [2 4 6 8]  
Potęgowanie: [ 1 8 27 64]
```

11.2 Funkcje trygonometryczne i pochodne

NumPy oferuje bogaty zestaw funkcji trygonometrycznych:

- `np.sin`, `np.cos`, `np.tan` – funkcje podstawowe,
- `np.arcsin`, `np.arccos`, `np.arctan` – odwrotne funkcje trygonometryczne,
- `np.sinh`, `np.cosh`, `np.tanh` – funkcje hiperboliczne.

Przykład:

```
import numpy as np  
  
x = np.linspace(0, np.pi, 5) # tablica [0, /4, /2, 3/4, ]  
sin_values = np.sin(x)  
cos_values = np.cos(x)  
  
print("Wartości sin(x):", sin_values)  
print("Wartości cos(x):", cos_values)  
  
Wartości sin(x): [0.0000000e+00 7.07106781e-01 1.0000000e+00 7.07106781e-01  
1.22464680e-16]  
Wartości cos(x): [ 1.0000000e+00 7.07106781e-01 6.12323400e-17 -7.07106781e-01  
-1.0000000e+00]
```

11.3 Funkcje wykładnicze i logarytmiczne

- `np.exp` – eksponenta,
- `np.log` – logarytm naturalny,
- `np.log10` – logarytm dziesiętny.

Przykład:

```

import numpy as np

A = np.array([1, np.e, np.e**2])
print("A:", A)
print("log(A):", np.log(A))
print("exp(A):", np.exp([0, 1, 2])) # exp(0)=1, exp(1)=e, exp(2)=e^2

```

```

A: [1.           2.71828183 7.3890561 ]
log(A): [0.  1.  2.]
exp(A): [1.           2.71828183 7.3890561 ]

```

11.4 Funkcje zaokrąglające i wartości bezwzględne

- `np.round` – zaokrąglą do najbliższej liczby,
- `np.floor` – podłoga,
- `np.ceil` – sufit,
- `np.trunc` – obcięcie do części całkowitej,
- `np.abs` – wartość bezwzględna.

Przykład:

```

import numpy as np

B = np.array([1.7, -2.5, 3.5, -4.1])
print("B:", B)
print("floor(B):", np.floor(B))
print("ceil(B):", np.ceil(B))
print("abs(B):", np.abs(B))

```

```

B: [ 1.7 -2.5  3.5 -4.1]
floor(B): [ 1. -3.  3. -5.]
ceil(B): [ 2. -2.  4. -4.]
abs(B): [1.7 2.5 3.5 4.1]

```

11.5 Funkcje statystyczne i agregujące

Choć wiele funkcji statystycznych dostępnych jest jako metody tablic (np. `A.mean()`, `A.std()`), istnieją też ufuncs działające element-po-elemencie lub akceptujące parametry osi:

- `np.minimum`, `np.maximum` – zwracają minimum/maksimum element-po-elementnie z dwóch tablic,
- `np.fmin`, `np.fmax` – podobne do wyżej wymienionych, ale ignorują wartości NaN,
- `np.sqrt` – pierwiastek kwadratowy,
- `np.square` – podniesienie do kwadratu.

Przykład:

```
import numpy as np

C1 = np.array([1, 4, 9, 16])
C2 = np.array([2, 2, 5, 20])

print("minimum elementów C1 i C2:", np.minimum(C1, C2))
print("maximum elementów C1 i C2:", np.maximum(C1, C2))
print("sqrt(C1):", np.sqrt(C1))
print("square(C2):", np.square(C2))

minimum elementów C1 i C2: [ 1  2  5 16]
maximum elementów C1 i C2: [ 2  4  9 20]
sqrt(C1): [1. 2. 3. 4.]
square(C2): [ 4   4  25 400]
```

Ćwiczenia: (ex7.py)

1. Mając tablice

$$A = [1 \ 4 \ 9 \ 16],$$

zastosuj funkcję uniwersalną, aby obliczyć pierwiastek kwadratowy każdego elementu.

2. Rozważ jednowymiarową tablicę

$$B = [-1 \ -2 \ 3 \ -4],$$

zastosuj funkcję uniwersalną, aby otrzymać wartości bezwzględne wszystkich elementów.

3. Dla tablicy

$$C = [0 \ \pi/2 \ \pi \ 3\pi/2],$$

oblicz wartość funkcji trygonometrycznej dla każdego elementu.

4. Mając tablicę

$$D = [1 \ e \ e^2],$$

zastosuj funkcję uniwersalną, aby obliczyć logarytm naturalny każdego elementu.

5. Dla tablicy dwuwymiarowej

$$E = \begin{bmatrix} 2 & 4 \\ 10 & 20 \end{bmatrix},$$

podziel każdy element przez skalar, a następnie podnieś uzyskane wartości do kwadratu.

6. Rozważ tablicę

$$F = [1 \ 2 \ 3],$$

podnieś każdy element do trzeciej potęgi, a następnie zastosuj funkcję uniwersalną, aby obliczyć eksponentę z otrzymanych wartości.

7. Mając tablicę

$$G = [-\pi \ -\pi/2 \ 0 \ \pi/2 \ \pi],$$

zastosuj odpowiednią funkcję uniwersalną, aby uzyskać cosinus każdego elementu.

8. Dla tablicy

$$H = [10 \ 100 \ 1000],$$

zastosuj funkcję uniwersalną, aby obliczyć logarytm dziesiętny każdego elementu.

9. Mając tablicę

$$I = [2 \ 8 \ 18 \ 32],$$

przekształć ją, stosując funkcję uniwersalną, tak aby każdy element był pierwiastkiem kwadratowym z wartości początkowej, a następnie pomnoż wyniki przez 2.

10. Rozważ tablicę

$$J = [-1 \ -4 \ -9 \ -16],$$

oblicz pierwiastek kwadratowy wartości bezwzględnych elementów tej tablicy, wykorzystując po kolej dwie różne funkcje uniwersalne.

12 Operacje na stringach

W NumPy poza dobrze znymi tablicami liczbowymi, istnieje również zestaw funkcji pozwalających na wektorowe operacje na ciągach znaków.

Ważne: Poniższe funkcje są zazwyczaj dostępne w module `numpy.char`. W dokumentacji znajdują się one w sekcji [String operations](#), jednak w tym materiale skupimy się na tym, jak można je wykorzystywać, zakładając interfejs z modułu `numpy.strings`. Jest to analogiczne do korzystania z `numpy.char`. Jest to nowsze podejście.

12.1 Tworzenie tablic z napisami

NumPy pozwala na przechowywanie tekstu w tablicach, np. tak:

```
import numpy as np

arr = np.array(["python", "NumPy", "data", "Science"])
print(arr)

['python' 'NumPy' 'data' 'Science']
```

12.2 Podstawowe funkcje do modyfikacji tekstu

Poniżej przedstawiono popularne funkcje do modyfikacji tekstu na tablicach stringów:

12.2.1 `numpy.strings.upper` i `numpy.strings.lower`

- `upper`: Zamiana wszystkich liter na wielkie.
- `lower`: Zamiana wszystkich liter na małe.

```
import numpy as np

arr = np.array(["python", "NumPy", "data", "Science"])

print(np.strings.upper(arr))
print(np.strings.lower(arr))
```

```
['PYTHON' 'NUMPY' 'DATA' 'SCIENCE']
['python' 'numpy' 'data' 'science']
```

12.2.2 numpy.strings.capitalize

Funkcja `capitalize` zamienia pierwszą literę wyrazu na wielką, a pozostałe na małe.

```
import numpy as np

arr = np.array(["python", "NumPy", "data", "Science"])
print(np.strings.capitalize(arr))
```

```
['Python' 'Numpy' 'Data' 'Science']
```

12.2.3 numpy.strings.title

Funkcja `title` sprawia, że każda część składowa tekstu (np. oddzielona spacją) zostaje zamieniona tak, by zaczynała się od wielkiej litery.

```
import numpy as np

arr2 = np.array(["python data science", "machine learning", "deep learning"])
print(np.strings.title(arr2))
```

```
['Python Data Science' 'Machine Learning' 'Deep Learning']
```

12.3 Łączenie i rozdzielanie tekstów

12.3.1 numpy.strings.add

Funkcja add łączy elementy tablic tekstowych, działając podobnie jak operator + na stringach, ale wektorowo.

```
import numpy as np

arr_a = np.array(["Hello", "Data"])
arr_b = np.array(["World", "Science"])

print(np.strings.add(arr_a, arr_b))

['HelloWorld' 'DataScience']
```

12.3.2 numpy.strings.join

Funkcja join pozwala na łączenie elementów tablicy przy użyciu wskazanego separatora.

```
import numpy as np

arr3 = np.array(["python", "numpy", "string"])
print(np.char.join("-", arr3))

['p-y-t-h-o-n' 'n-u-m-p-y' 's-t-r-i-n-g']
```

Uwaga: join wektoryzuje operację, traktując każdy element tablicy jako sekwencję znaków do połączenia separatorem.

12.3.3 numpy.strings.split

Pozwala na rozdzielanie stringów według podanego separatora. Zwraca tablicę zawierającą listy podłańcuchów.

```
import numpy as np

arr4 = np.array(["python-data-science", "machine-learning"])
print(np.char.split(arr4, sep="-"))
```

```
[list(['python', 'data', 'science']), list(['machine', 'learning'])]
```

12.4 Wyszukiwanie i zamiana podciągów

12.4.1 numpy.strings.find i numpy.strings.rfind

- **find**: Zwraca indeks pierwszego wystąpienia podłańcucha (lub -1, jeśli nie znaleziono).
- **rfind**: Zwraca indeks ostatniego wystąpienia podłańcucha (lub -1, jeśli nie znaleziono).

```
import numpy as np

arr5 = np.array(["python", "data", "numpy"])
print(np.strings.find(arr5, "a"))
```

```
[-1  1 -1]
```

12.4.2 numpy.strings.replace

`replace` zamienia wszystkie wystąpienia podłańcucha na nowy ciąg znaków.

```
import numpy as np

arr6 = np.array(["python", "pydata", "pypy"])
print(np.strings.replace(arr6, "py", "PY"))
```

```
['PYthon' 'PYdata' 'PYPY']
```

12.5 Usuwanie zbędnych znaków

12.5.1 `numpy.strings.strip`, `numpy.strings.lstrip` i `numpy.strings.rstrip`

- `strip`: Usuwa wskazane znaki z początku i końca.
- `lstrip`: Usuwa wskazane znaki z lewej strony (poczatku).
- `rstrip`: Usuwa wskazane znaki z prawej strony (konca).

```
import numpy as np

arr7 = np.array(["  python  ", "  numpy  "])
print(np.strings.strip(arr7))

['python' 'numpy']
```

Możemy również podać niestandardowe znaki do usunięcia:

```
import numpy as np

arr8 = np.array(["###data###", "***science***"])
print(np.strings.strip(arr8, "#*"))

['data' 'science']
```

13 Alegbra liniowa

13.1 Iloczyn skalarny (dot product)

Dla dwóch wektorów, `dot` oblicza ich iloczyn skalarny.

```
import numpy as np

# Iloczyn skalarny dwóch wektorów
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = np.dot(a, b) # 1*4 + 2*5 + 3*6
print(result) # Wynik: 32

# Alternatywny zapis za pomocą operatora @
result = a @ b
print(result) # Wynik: 32
```

32
32

13.2 Mnożenie macierzowe

Dla macierzy (tablic dwuwymiarowych), `dot` wykonuje standardowe mnożenie macierzowe.

```
import numpy as np
# Mnożenie macierzowe
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B)
print(C)
# Wynik:
# [[19 22]
#  [43 50]]
```

```
# To samo za pomocą operatora @  
C = A @ B  
print(C)
```

```
[[19 22]  
 [43 50]]  
[[19 22]  
 [43 50]]
```

13.3 Mnożenie macierz-wektor

Możemy również mnożyć macierz przez wektor:

```
import numpy as np  
# Mnożenie macierz-wektor  
A = np.array([[1, 2], [3, 4]])  
v = np.array([5, 6])  
result = np.dot(A, v)  
print(result) # Wynik: [17 39]
```

```
[17 39]
```

13.4 Rozwiązywanie układów równań liniowych

Funkcja `numpy.linalg.solve` rozwiązuje układy równań liniowych postaci $Ax = b$:

```
import numpy as np  
# Rozwiązywanie układu równań liniowych  
A = np.array([[3, 1], [1, 2]])  
b = np.array([9, 8])  
x = np.linalg.solve(A, b)  
print(x) # Wynik: [2. 3.]  
  
# Sprawdzenie rozwiązania  
np.dot(A, x) # Powinno być równe b
```

```
[2. 3.]
```

```
array([9., 8.])
```

13.5 Wyznacznik macierzy

Funkcja `numpy.linalg.det` oblicza wyznacznik macierzy:

```
import numpy as np
# Obliczanie wyznacznika
A = np.array([[1, 2], [3, 4]])
det_A = np.linalg.det(A)
print(det_A) # Wynik: -2.0
```

-2.0000000000000004

13.6 Wartości i wektory własne

Funkcja `numpy.linalg.eig` oblicza wartości i wektory własne macierzy:

```
import numpy as np
# Obliczanie wartości i wektorów własnych
A = np.array([[4, -2], [1, 1]])
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Wartości własne:", eigenvalues)
print("Wektory własne:")
print(eigenvectors)

# Sprawdzenie: A * v = lambda * v
for i in range(len(eigenvalues)):
    lambda_i = eigenvalues[i]
    v_i = eigenvectors[:, i]
    print(f" {i} = {lambda_i}")
    print("A * v =", np.dot(A, v_i))
    print(" * v =", lambda_i * v_i)
```

```
Wartości własne: [3.  2.]
Wektory własne:
[[0.89442719  0.70710678]
 [0.4472136   0.70710678]]
_0 = 3.0
A * v = [2.68328157 1.34164079]
 * v = [2.68328157 1.34164079]
_1 = 2.0
```

```
A * v = [1.41421356 1.41421356]
* v = [1.41421356 1.41421356]
```

13.7 Rozkład wartości osobliwych (SVD)

Rozkład SVD jest potężnym narzędziem w analizie danych:

```
import numpy as np
# Rozkład SVD
A = np.array([[1, 2], [3, 4], [5, 6]])
U, s, Vh = np.linalg.svd(A)
print("Macierz U:")
print(U)
print("Wartości osobliwe:", s)
print("Macierz V^H:")
print(Vh)

# Rekonstrukcja macierzy A
S = np.zeros((A.shape[0], A.shape[1]))
S[:len(s), :len(s)] = np.diag(s)
A_reconstructed = U @ S @ Vh
print("Rekonstruowana macierz A:")
print(A_reconstructed)
```

```
Macierz U:
[[-0.2298477  0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
Wartości osobliwe: [9.52551809 0.51430058]
Macierz V^H:
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
Rekonstruowana macierz A:
[[1. 2.]
 [3. 4.]
 [5. 6.]]
```

13.8 Norma macierzy/wektora

NumPy oferuje różne rodzaje norm:

```

import numpy as np
# Różne normy
v = np.array([3, 4])
print("Norma L1:", np.linalg.norm(v, 1)) # Norma L1: 7.0
print("Norma L2 (Euklidesowa):", np.linalg.norm(v)) # Norma L2: 5.0
print("Norma maksimum:", np.linalg.norm(v, np.inf)) # Norma maksimum: 4.0

A = np.array([[1, 2], [3, 4]])
print("Norma macierzowa Frobeniusa:", np.linalg.norm(A, 'fro')) # Norma Frobeniusa: 5.477...

```

```

Norma L1: 7.0
Norma L2 (Euklidesowa): 5.0
Norma maksimum: 4.0
Norma macierzowa Frobeniusa: 5.477225575051661

```

13.9 Macierz odwrotna

Funkcja `numpy.linalg.inv` oblicza macierz odwrotną:

```

import numpy as np
# Macierz odwrotna
A = np.array([[1, 2], [3, 4]])
A_inv = np.linalg.inv(A)
print("Macierz odwrotna:")
print(A_inv)

# Sprawdzenie: A * A^(-1) = I
print("A * A^(-1):")
print(np.dot(A, A_inv)) # Powinno być bliskie macierzy jednostkowej

```

```

Macierz odwrotna:
[[ -2.   1. ]
 [ 1.5 -0.5]]
A * A^(-1):
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]

```

13.10 Funkcja `numpy.inner` - iloczyn wewnętrzny

Funkcja `inner` oblicza iloczyn wewnętrzny dwóch tablic:

```
import numpy as np
# Iloczyn wewnętrzny
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = np.inner(a, b)
print(result) # 1*4 + 2*5 + 3*6 = 32

# Dla tablic 2D
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
result = np.inner(A, B)
print(result)
# Jest to równoważne wykonaniu iloczynu skalarnego wzdłuż ostatniego wymiaru
```

```
32
[[17 23]
 [39 53]]
```

13.11 Funkcja `numpy.outer` - iloczyn zewnętrzny

Funkcja `outer` oblicza iloczyn zewnętrzny dwóch wektorów:

```
import numpy as np
# Iloczyn zewnętrzny
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = np.outer(a, b)
print(result)
# Wynik:
# [[ 4  5  6]
#  [ 8 10 12]
#  [12 15 18]]
```

```
[[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
```

13.12 Funkcja `numpy.matmul` - mnożenie macierzowe

Funkcja `matmul` jest podobna do `dot`, ale ma nieco inne zachowanie dla tablic o wymiarach większych niż 2:

```
import numpy as np
# Porównanie dot i matmul
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

dot_result = np.dot(a, b)
matmul_result = np.matmul(a, b)

print("Wynik dot:")
print(dot_result)
print("Wynik matmul:")
print(matmul_result)
# Dla 2D są identyczne

# Ale dla tablic 3D i wyższych mogą się różnić
```

Wynik dot:

```
[[19 22]
 [43 50]]
```

Wynik matmul:

```
[[19 22]
 [43 50]]
```

14 Filtrowanie zaawansowane

14.1 Funkcja nonzero()

Zwraca indeksy elementów niezerowych w tablicy. Wynik jest zwracany jako krotka tablic, po jednej dla każdego wymiaru tablicy.

```
import numpy as np

arr = np.array([[3, 0, 0], [0, 4, 0], [5, 6, 0]])
indeksy = np.nonzero(arr)
print(indeksy) # (array([0, 1, 2, 2]), array([0, 1, 0, 1]))

# Wydobycie wartości niezerowych
wartosci = arr[indeksy]
print(wartosci) # [3 4 5 6]

# Alternatywnie można użyć:
indeksy_i_wartosci = np.argwhere(arr != 0)
print(indeksy_i_wartosci)
# [[0 0]
#  [1 1]
#  [2 0]
#  [2 1]]
```



```
(array([0, 1, 2, 2]), array([0, 1, 0, 1]))
[3 4 5 6]
[[0 0]
 [1 1]
 [2 0]
 [2 1]]
```

```

import numpy as np

arr = np.array([[3, 0, 0], [0, 4, 0], [5, 6, 0]])
indeksy = np.nonzero(arr)
print(indeksy) # (array([0, 1, 2, 2]), array([0, 1, 0, 1]))

```

	0	1	2
0	3	0	0
1	0	4	0
2	5	6	0

indeksy
wiersz

indeksy
kolumn

14.2 Funkcja where()

Zwraca elementy wybrane z x lub y w zależności od warunku. Jest to warunkowy selektor elementów.

```

import numpy as np

# Zastąp wartości niedodatnie przez 0
arr = np.array([1, -2, 3, -4, 5])
wynik = np.where(arr > 0, arr, 0)
print(wynik) # [1 0 3 0 5]

# Zastosowanie w tablicy 2D
arr_2d = np.array([[1, -2, 3], [-4, 5, -6]])
wynik_2d = np.where(arr_2d < 0, -1, arr_2d)
print(wynik_2d)
# [[ 1 -1  3]
#  [-1  5 -1]]

```

```
[1 0 3 0 5]
[[ 1 -1  3]
 [-1  5 -1]]
```

where(warunek, co jeśli prawda, co jeśli fałsz)

14.3 Funkcje indices() i ix_()

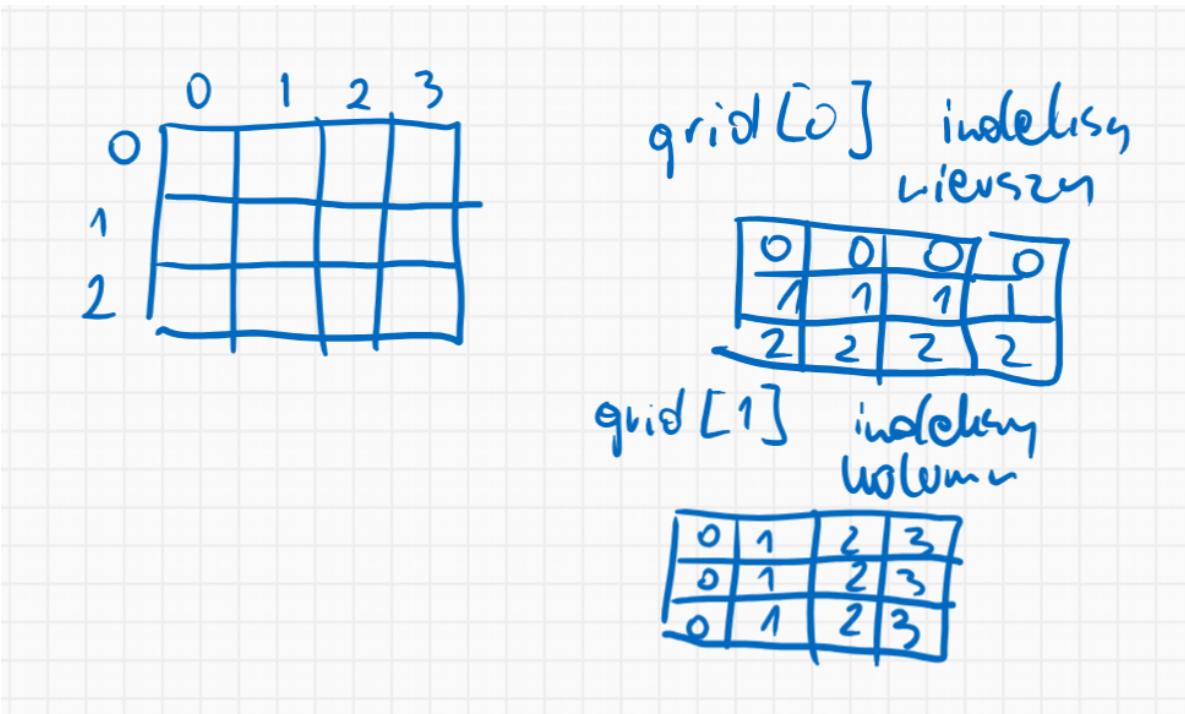
14.3.1 indices()

Tworzy tablicę reprezentującą indeksy siatki.

```
import numpy as np

# Tworzenie siatki indeksów 3x4
grid = np.indices((3, 4))
print(grid.shape) # (2, 3, 4)
print(grid[0]) # indeksy wierszy
# [[0 0 0 0]
#  [1 1 1 1]
#  [2 2 2 2]]
print(grid[1]) # indeksy kolumn
# [[0 1 2 3]
#  [0 1 2 3]
#  [0 1 2 3]]
```

```
(2, 3, 4)
[[0 0 0 0]
 [1 1 1 1]
 [2 2 2 2]]
[[0 1 2 3]
 [0 1 2 3]
 [0 1 2 3]]
```



14.3.2 ix_()

Konstruuje otwartą siatkę z wielu sekwencji, co jest przydatne do indeksowania wielowymiarowego.

```
import numpy as np
x = np.array([0, 1, 2])
y = np.array([3, 4, 5, 6])
indeksy = np.ix_(x, y)

# Tworzy indeksy dla wszystkich kombinacji (0,3), (0,4), ..., (2,6)
print(indeksy[0].shape, indeksy[1].shape) # (3, 1) (1, 4)

# Użycie do wybierania podtablicy
arr = np.arange(16).reshape(4, 4)
print(arr)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]
#  [12 13 14 15]]
```

```
podtablica = arr[np.ix_([0, 2, 3], [0, 2])]  
print(podtablica)  
# [[ 0  2]  
#  [ 8 10]  
#  [12 14]]
```

```
(3, 1) (1, 4)  
[[ 0  1  2  3]  
 [ 4  5  6  7]  
 [ 8  9 10 11]  
 [12 13 14 15]]  
[[ 0  2]  
 [ 8 10]  
 [12 14]]
```

14.4 ogrid i operacje na siatkach

ogrid pozwala na tworzenie otwartych siatek, co jest pamięciowo wydajniejsze niż pełne siatki.

```
import numpy as np  
  
# Siatka punktów w zakresie od -2 do 2 z krokiem 0.1  
x, y = np.ogrid[-2:2:0.1, -2:2:0.1]  
maska = x**2 + y**2 <= 1 # Okrąg o promieniu 1  
print(maska.shape) # (40, 40)
```

```
(40, 40)
```

14.5 Funkcje ravel_multi_index() i unravel_index()

Te funkcje konwertują między indeksami wielowymiarowymi a płaskimi.

```
import numpy as np  
  
# Konwersja indeksów wielowymiarowych na płaskie  
indeksy_wielo = np.array([[0, 0], [1, 1], [2, 1]])  
wymiary = (3, 3)
```

```

indeksy_plaskie = np.ravel_multi_index(indeksy_wielo.T, wymiary)
print(indeksy_plaskie) # [0 4 7]

# Konwersja indeksów płaskich na wielowymiarowe
indeksy_plaskie = np.array([0, 3, 8])
ksztalt = (3, 3)
indeksy_wielo = np.unravel_index(indeksy_plaskie, ksztalt)
print(indeksy_wielo) # (array([0, 1, 2]), array([0, 0, 2]))

```

```
[0 4 7]
(array([0, 1, 2]), array([0, 0, 2]))
```

14.6 Indeksy diagonalne

NumPy oferuje wiele funkcji do pracy z diagonalami macierzy.

```

import numpy as np
# Uzyskanie indeksów głównej przekątnej
n = 4
indeksy_diag = np.diag_indices(n)
print(indeksy_diag) # (array([0, 1, 2, 3]), array([0, 1, 2, 3]))

# Zastosowanie do ustawienia głównej przekątnej
arr = np.zeros((4, 4))
arr[indeksy_diag] = 1 # Ustawienie jedynek na głównej przekątnej
print(arr)
# [[1. 0. 0. 0.]
#  [0. 1. 0. 0.]
#  [0. 0. 1. 0.]
#  [0. 0. 0. 1.]]

# Uzyskanie indeksów z istniejącej tablicy
arr2 = np.ones((3, 3))
indeksy_diag2 = np.diag_indices_from(arr2)
print(indeksy_diag2)

(array([0, 1, 2, 3]), array([0, 1, 2, 3]))
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

```
[0. 0. 0. 1.]
(array([0, 1, 2]), array([0, 1, 2]))
```

14.7 3.1 Funkcja `take()`

Pobiera elementy z tablicy wzdłuż określonej osi na podstawie indeksów.

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])
indeksy = np.array([0, 2, 4])
wynik = np.take(arr, indeksy)
print(wynik) # [10 30 50]

# W tablicach wielowymiarowych możemy wybrać os
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
indeksy_wierszy = np.array([0, 2])
wynik_2d = np.take(arr_2d, indeksy_wierszy, axis=0)
print(wynik_2d)
# [[1 2 3]
#  [7 8 9]]
```

```
[10 30 50]
[[1 2 3]
 [7 8 9]]
```

```
##Funkcja take_along_axis()
```

Pobiera wartości z tablicy poprzez dopasowanie 1D indeksu i fragmentów danych. Jest bezpieczna dla duplikatów indeksów.

```
import numpy as np

arr = np.array([[10, 30, 20], [60, 40, 50]])
indeksy_kolejnosc = np.argsort(arr, axis=1)
wynik = np.take_along_axis(arr, indeksy_kolejnosc, axis=1)
print(wynik)
# [[10 20 30]
#  [40 50 60]]
```

```
[[10 20 30]
 [40 50 60]]
```

14.8 Funkcja choose()

Konstruuje tablicę wybierając elementy z listy tablic.

```
import numpy as np

opcje = [np.array([0, 1, 2, 3]),
         np.array([10, 11, 12, 13]),
         np.array([20, 21, 22, 23])]
indeksy = np.array([0, 2, 1, 0]) # Wybiera z której tablicy opcji wziąć element
wynik = np.choose(indeksy, opcje)
print(wynik) # [ 0 21 12 3]
```

[0 21 12 3]

14.9 Funkcja compress()

Zwraca wybrane elementy tablicy wzdłuż określonej osi.

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
maska = np.array([True, False, True])
wynik = np.compress(maska, arr, axis=1)
print(wynik)
# [[1 3]
#  [4 6]]
```

[[1 3]
[4 6]]

14.10 Funkcje diag() i diagonal()

Funkcje do pracy z przekątnymi.

```
import numpy as np

# Tworzenie tablicy diagonalnej
diag_arr = np.diag([1, 2, 3, 4])
```

```

print(diag_arr)
# [[1 0 0 0]
#  [0 2 0 0]
#  [0 0 3 0]
#  [0 0 0 4]]

# Pobieranie diagonali z tablicy
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
diag = np.diag(arr)
print(diag) # [1 5 9]

# Pobieranie przekątnej przesuniętej o 1
diag_offset = np.diagonal(arr, offset=1)
print(diag_offset) # [2 6]

```

```

[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
[1 5 9]
[2 6]

```

14.11 Funkcja select()

Zwraca tablicę zbudowaną z elementów z listy opcji, w zależności od warunków.

```

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
warunki = [arr < 3, arr < 6, arr < 9]
opcje = [100, 200, 300]
wynik = np.select(warunki, opcje, default=400)
print(wynik)
# [[100 100 200]
#  [200 200 300]
#  [300 300 400]]

```

```

[[100 100 200]
 [200 200 300]
 [300 300 400]]

```

14.12 Funkcja place()

Zmienia elementy tablicy na podstawie maski i podanych wartości.

```
import numpy as np

arr = np.arange(5)
maska = np.array([True, False, True, False, True])
np.place(arr, maska, [-1, -2, -3]) # Cyklicznie używa wartości [-1, -2, -3]
print(arr) # [-1  1 -2  3 -3]

[-1  1 -2  3 -3]
```

14.13 Funkcja put()

Zastępuje określone elementy tablicy podanymi wartościami.

```
import numpy as np

arr = np.arange(5)
indeksy = [0, 2, 4]
np.put(arr, indeksy, [10, 20, 30])
print(arr) # [10  1 20  3 30]

[10  1 20  3 30]
```

14.14 Funkcja put_along_axis()

Umieszcza wartości w tablicy docelowej, dopasowując 1D indeks i fragmenty danych wzdłuż określonej osi.

```
import numpy as np

arr = np.array([[10, 30, 20], [60, 40, 50]])
indeksy = np.argmax(arr, axis=1)
indeksy = np.expand_dims(indeksy, axis=1) # Przekształć do kształtu (2, 1)
np.put_along_axis(arr, indeksy, 99, axis=1)
print(arr)
# [[99 30 20]
#  [60 40 99]]
```

```
[[99 30 20]
 [60 99 50]]
```

14.15 Funkcja putmask()

Zmienia elementy tablicy na podstawie warunku i podanych wartości.

```
import numpy as np

arr = np.arange(5)
maska = np.array([True, False, True, False, True])
np.putmask(arr, maska, [-1, -2, -3]) # Cyklicznie używa wartości
print(arr) # [-1  1 -2  3 -3]
```

```
[-1  1 -3  3 -2]
```

14.16 Funkcja fill_diagonal()

Wypełnia główną przekątną tablicy podaną wartością.

```
import numpy as np

arr = np.zeros((4, 4))
np.fill_diagonal(arr, 5)
print(arr)
# [[5. 0. 0. 0.]
#  [0. 5. 0. 0.]
#  [0. 0. 5. 0.]
#  [0. 0. 0. 5.]]

arr_rect = np.zeros((4, 4, 4))
np.fill_diagonal(arr_rect, 9)
print(arr_rect[0]) # Wypełnia przekątną w każdym "plasterku" 3D tablicy
```

```
[[5. 0. 0. 0.]
 [0. 5. 0. 0.]
 [0. 0. 5. 0.]
 [0. 0. 0. 5.]]
[[9. 0. 0. 0.]]
```

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

15 Numpy - inne

15.1 Stałe

NumPy dostarcza kilka znanych stałych matematycznych, które mogą być przydatne w obliczeniach naukowych i inżynierskich. Wbudowane stałe takie jak liczba Pi czy podstawa logarytmu naturalnego e ułatwiają pisanie czytelnego i zwięzłego kodu.

1. `numpy.pi`

- Reprezentuje liczbę Pi () z dużą dokładnością.
- Pi to stosunek obwodu okręgu do jego średnicy.
- W przybliżeniu: 3.141592653589793

2. `numpy.e`

- Reprezentuje podstawę logarytmu naturalnego, e.
- e jest wykorzystywane w wielu dziedzinach, takich jak analiza matematyczna, probabilistyka, statystyka.
- W przybliżeniu: 2.718281828459045

3. `numpy.eulergamma`

- Reprezentuje stałą Eulera-Mascheroniego, zwykle oznaczaną jako (gamma).
- Pojawia się w analizie matematycznej, szczególnie w teorii liczb i badaniu szeregów harmonicznych.
- W przybliżeniu: 0.5772156649015329

```
import numpy as np

# Promień koła
r = 5.0

# Obwód koła: 2 * pi * r
obwod = 2 * np.pi * r
```

```

print("Obwód koła:", obwod)

# Pole koła: * r^2
pole = np.pi * r**2
print("Pole koła:", pole)

```

Obwód koła: 31.41592653589793
 Pole koła: 78.53981633974483

```

import numpy as np

# Przykładowy punkt x
x = 1.0

# Wartość funkcji e^x
exp_value = np.e**x
print("e^x dla x=1:", exp_value)

# Porównanie z funkcją np.exp
exp_compare = np.exp(x)
print("Porównanie z np.exp(1):", exp_compare)

```

e^x dla x=1: 2.718281828459045
 Porównanie z np.exp(1): 2.718281828459045

15.2 numpy.inf

- Opis: `np.inf` reprezentuje wartość nieskończoną (∞).
- Często pojawia się w obliczeniach, gdy wartość danego wyrażenia dąży do nieskończoności (np. dzielenie przez zero, pewne limity, itp.).
- Przykładowo, `1.0 / 0.0` zwróci ostrzeżenie i w konsekwencji może dać wartość `inf`.

```

import numpy as np

# Zastosowanie w tworzeniu masek logicznych
arr = np.array([1, 2, np.inf, 4, 5])
mask = np.isinf(arr)
print("Maska elementów o wartości inf:", mask)

```

```
Maska elementów o wartości inf: [False False  True False False]
```

15.3 numpy.nan

- **Opis:** np.nan oznacza “Not a Number” (NaN), czyli wartość nieokreślona lub nierepresentowalna w systemie liczbowym.
- Pojawia się, gdy wynik operacji numerycznej jest nieokreślony, np. $0.0/0.0$, inf - inf lub przy błędach wczytywania danych.
- Operacje arytmetyczne z nan zazwyczaj również zwracają nan.

```
import numpy as np

# Zamiana wartości nan w tablicy
data = np.array([1, 2, np.nan, 4, np.nan])
print("Oryginalne dane:", data)

# Wypełnienie wartości nan zerem
data_no_nan = np.nan_to_num(data, nan=0.0)
print("Dane bez nan:", data_no_nan)
```

```
Oryginalne dane: [ 1.  2. nan  4. nan]
Dane bez nan: [1.  2.  0.  4.  0.]
```

15.4 numpy.newaxis

- **Opis:** np.newaxis jest specjalną “stałą”/obiektem służącym do zmiany wymiarów tablic przez zwiększenie ich liczby wymiarów o 1.

```
import numpy as np

# Mamy tablicę 1D
vec = np.array([1, 2, 3, 4])
print("Oryginalna tablica:", vec, "Kształt:", vec.shape)

# Dodajemy nowy wymiar jako wymiar wierszy
vec_as_col = vec[:, np.newaxis]
print("Tablica jako kolumna:\n", vec_as_col, "Kształt:", vec_as_col.shape)
```

```

# Dodawanie wymiaru na początku
vec_as_row = vec[np.newaxis, :]
print("Tablica jako wiersz:\n", vec_as_row, "Kształt:", vec_as_row.shape)

# Kolejny przykład: dodanie wymiaru by z łatwością broadcastować operacje
a = np.array([10, 20, 30])
b = np.array([1, 2])
# Bez nowego wymiaru próba dodania a do b się nie powiedzie,
# bo kształty nie są kompatybilne.
# Z nowym wymiarem a ma kształt (3,1), a b (2,), co pozwala na broadcast
sum_matrix = a[:, np.newaxis] + b
print("Operacja z broadcast:\n", sum_matrix)

```

Oryginalna tablica: [1 2 3 4] Kształt: (4,)

Tablica jako kolumna:

```

[[1]
[2]
[3]
[4]] Kształt: (4, 1)

```

Tablica jako wiersz:

```

[[1 2 3 4]] Kształt: (1, 4)

```

Operacja z broadcast:

```

[[11 12]
[21 22]
[31 32]]

```

15.5 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.

Funkcja	Opis
np.any	Sprawdza czy co najmniej jeden z elementów jest różny od zera.

Część III

Eksploracja danych

16 Etapy eksploracji danych

- **Zbieranie danych:**

- Zebranie danych z różnych źródeł (bazy danych, pliki CSV, API, itd.).

- **Zrozumienie danych:**

- Analiza struktury danych, typów danych i ich znaczenia.
 - Eksploracja wstępnych zależności i trendów.

- **Czyszczenie danych:**

- Usuwanie braków, błędów i anomalii w danych.
 - Obsługa brakujących wartości i duplikatów.

- **Transformacja danych:**

- Normalizacja, standaryzacja, kodowanie zmiennych kategorycznych.
 - Tworzenie nowych zmiennych (cech).

- **Redukcja danych:**

- Selekcja istotnych cech lub zmniejszenie wymiarowości danych (np. PCA).

Część IV

Pandas

17 Pandas - start

Pandas jest biblioteką Pythona służącą do analizy i manipulowania danymi

17.1 Import:

```
import pandas as pd
```

17.2 Podstawowe typy

Seria - Series

a	3
b	-5
c	7
d	4

Index

Ramka danych - DataFrame

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

Columns

Index

```
import pandas as pd

s = pd.Series([3, -5, 7, 4])
print(s)
print("values")
print(s.to_numpy())
print(type(s.to_numpy()))
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'>
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)
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)
data2 = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print(data2)
```

```
   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_data = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print("Shape:", df_data.shape)
print("--")
print("Index:", df_data.index)
print("--")
print("columns:", df_data.columns)
print("--")
df_data.info()
```

```

print("---")
print(df_data.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

Ćwiczenia: (ex8.py)

- Napisz kod, który utworzy serię z następującej listy liczb: [10, 20, 30, 40, 50]. Wyświetl serię w formacie tabelarycznym:

Index	Value
0	10
1	20
2	30
3	40
4	50

- Utwórz serię, gdzie kluczami będą miesiące ('Jan', 'Feb', 'Mar'), a wartościami odpowiednie temperatury: [0, 3, 5]. Wyświetl w formacie tabelarycznym:

Month	Temperature
Jan	0
Feb	3
Mar	5

3. Stwórz pustą ramkę danych z kolumnami **Product**, **Price**, **Quantity**, a następnie wypełnij ją danymi:

Product	Price	Quantity
Apple	1.2	10
Banana	0.5	20
Orange	0.8	15

18 Pandas - indeksowanie

```
import pandas as pd

data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],
        'Population': [11190846, 1303171035, 207847528]}
data2 = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print(data2.iloc[[0], [0]])
print("--")
print(data2.loc[[0], ['Country']])
print("--")
print(data2.loc[2])
print("--")
print(data2.loc[:, 'Capital'])
print("--")
print(data2.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]}
data2 = pd.DataFrame(data, columns=['Country', 'Population', 'Capital'])
print(data2['Population'])
print("--")
print(data2[data2['Population'] > 1200000000])
print("--")
```

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

Ćwiczenia: (ex9.py)

Poćwicz indeksowanie na poniższej ramce (nie muszą być wszystkie wiersze):

- **Kolumny kategoryczne:**

- Region – region sprzedaży
- Product – rodzaj produktu
- Sales_Channel – kanał sprzedaży (online, sklep stacjonarny, hurt)

- **Kolumny liczbowe:**

- Units_Sold – liczba sprzedanych jednostek
- Revenue – przychód w tysiącach GBP
- Profit – zysk w tysiącach GBP

Region	Product	Sales_Channel	Units_Sold	Revenue	Profit
North	Electronics	Online	120	60.5	15.2
South	Furniture	Retail	80	45.0	12.0
East	Clothing	Online	200	35.0	8.5
West	Electronics	Wholesale	150	70.0	20.5
North	Furniture	Retail	90	50.5	13.2
South	Clothing	Online	300	55.0	10.0
East	Electronics	Retail	110	62.0	16.0
West	Furniture	Online	70	30.0	7.5
North	Clothing	Wholesale	250	40.0	9.0
South	Electronics	Retail	130	75.0	22.0

19 Ładowanie danych

19.1 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 tysięczny
- `decimal='.'` - separator dziesiętny

Zapis `pandas.DataFrame.to_csv`

Dokumentacja: [link](#)

19.2 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 tysięczny

- `decimal='.'` - separator dziesiętny

Ćwiczenie: (ex10.py)

Poćwicz ładowanie danych z plików

<https://github.com/pjastr/AIWD-files>

19.3 Obsługa sql lite3

```
import pandas as pd
from sqlite3 import connect

conn = connect('sales_data2.db')
data = pd.read_sql("SELECT * FROM sales_data", con=conn)
```

20 Pandas - sortowanie

```
import pandas as pd

# Przykładowa ramka danych
data = pd.DataFrame({
    'Name': ['Alice', 'Tom', 'Charlie'],
    'Age': [25, 42, 35],
    'Salary': [50000, 60000, 70000]
})

# Sortowanie po kolumnie 'Age'
s1 = data.sort_values(by='Age')
print(s1)
# Sortowanie w odwrotnej kolejności
s2 = data.sort_values(by='Salary', ascending=False)
print(s2)
# Sortowanie według 'Age' rosnąco, a następnie 'Salary' malejąco
s3 = data.sort_values(by=['Age', 'Salary'], ascending=[True, False])
print(s3)
```

```
Name  Age  Salary
0   Alice  25  50000
2  Charlie  35  70000
1     Tom  42  60000
      Name  Age  Salary
2  Charlie  35  70000
1     Tom  42  60000
0   Alice  25  50000
      Name  Age  Salary
0   Alice  25  50000
2  Charlie  35  70000
1     Tom  42  60000
```

```

import pandas as pd

# Przykładowa ramka danych
data = pd.DataFrame({
    'Name': ['Alice', 'Tom', 'Charlie'],
    'Age': [25, 41, 35],
    'Salary': [50000, 60000, 70000]
})

# Sortowanie inplace (zamiana istniejącej zmiennej) - obecnie niezalecane
data.sort_values(by='Age', inplace=True)
print(data)

```

	Name	Age	Salary
0	Alice	25	50000
2	Charlie	35	70000
1	Tom	41	60000

```

import pandas as pd

df2 = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'Dave'],
    'Age': [25, 30, None, 35],
    'Salary': [50000, None, 70000, 60000]
})

# Sortowanie z NaN na końcu
s2 = df2.sort_values(by='Age', na_position='last')
print(s2)
# Sortowanie z NaN na początku
s3 = df2.sort_values(by='Age', na_position='first')
print(s3)

```

	Name	Age	Salary
0	Alice	25.0	50000.0
1	Bob	30.0	NaN
3	Dave	35.0	60000.0
2	Charlie	NaN	70000.0

	Name	Age	Salary
2	Charlie	NaN	70000.0
0	Alice	25.0	50000.0

```
1      Bob  30.0      NaN
3     Dave  35.0  60000.0
```

```
import pandas as pd

df3 = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'Dave'],
    'Age': [25, 30, None, 35],
    'Salary': [50000, None, 70000, 60000]
})

s3 = df3.sort_values(by='Name', key=lambda x: x.str.len())
print(s3)
```

```
Name    Age    Salary
1      Bob  30.0      NaN
3     Dave  35.0  60000.0
0    Alice  25.0  50000.0
2  Charlie   NaN  70000.0
```

Ćwiczenie: (exsort.py)

Załaduj poniższe pliki i posortuj wg wybranych samodzielnie kryteriów:

- date_sale.csv
- wynagrodzenia21.csv

21 Pandas - szeregi czasowe

Zamiana stringu na format `datetime` (dato-czasowy)

```
import pandas as pd

data = {'date': ['2023-01-01', '2023-01-02', '2023-01-03'], 'value': [10, 15, 20]}
data_frame = pd.DataFrame(data)
data_frame['date'] = pd.to_datetime(data_frame['date'])
print(data)

{'date': ['2023-01-01', '2023-01-02', '2023-01-03'], 'value': [10, 15, 20]}
```

Argument `errors` w funkcji `pd.to_datetime` kontroluje, jak funkcja ma się zachować, gdy napotka nieprawidłowe dane podczas próby konwersji wartości na obiekty `datetime`. Możliwe wartości dla `errors` to:

1. '`raise`' (domyślnie): Rzuca wyjątek, jeśli napotka nieprawidłowy format danych.
2. '`coerceNaT` (Not a Time).
3. '`ignore`

Kod do wklejenia do środowiska:

```
import pandas as pd

data = {'date': ['2023-01-01', 'invalid', '2023-01-03'], 'value': [10, 15, 20]}
data_frame = pd.DataFrame(data)
data_frame['date'] = pd.to_datetime(data_frame['date'], errors='ignore')
```

Argument `format` w funkcji `pandas.to_datetime` pozwala określić dokładny format daty i czasu, który ma zostać użyty do parsowania wartości wejściowych. Jest to przydatne, gdy dane wejściowe mają stały, specyficzny format, co może przyspieszyć przetwarzanie i zmniejszyć ryzyko błędnej interpretacji dat.

```

import pandas as pd

# Przykładowe dane wejściowe z różnymi formatami
data1 = ['01-01-2025', '15-03-2025', '30-12-2025'] # Format: DD-MM-YYYY
data2 = ['2025/01/01', '2025/03/15', '2025/12/30'] # Format: YYYY/MM/DD

# Konwersja z określonym formatem (DD-MM-YYYY)
df1 = pd.DataFrame(data1)
df1[0] = pd.to_datetime(df1[0], format='%d-%m-%Y')
print("Konwersja z formatem '%d-%m-%Y':")
print(df1)

# Konwersja z określonym formatem (YYYY/MM/DD)
df2 = pd.DataFrame(data2)
df2[0] = pd.to_datetime(df2[0], format='%Y/%m/%d')
print("\nKonwersja z formatem '%Y/%m/%d':")
print(df2)

```

Konwersja z formatem '%d-%m-%Y':

```

0
0 2025-01-01
1 2025-03-15
2 2025-12-30

```

Konwersja z formatem '%Y/%m/%d':

```

0
0 2025-01-01
1 2025-03-15
2 2025-12-30

```

Kod formatu	Opis	Przykład
%Y	Rok w formacie 4-cyfrowym	2025
%y	Rok w formacie 2-cyfrowym	25
%m	Miesiąc (cyfry, 2-cyfrowe)	01 (styczeń), 12 (grudzień)
%d	Dzień miesiąca (2-cyfrowe)	01, 15, 31
%B	Pełna nazwa miesiąca	January, December
%b	Skrót nazwy miesiąca	Jan, Dec
%A	Pełna nazwa dnia tygodnia	Monday, Sunday
%a	Skrót nazwy dnia tygodnia	Mon, Sun
%H	Godzina w formacie 24-godzinnym	00, 12, 23

Kod formatu	Opis	Przykład
%I	Godzina w formacie 12-godzinnym	01, 11
%p	AM/PM	AM, PM
%M	Minuty	00, 30, 59
%S	Sekundy	00, 30, 59

Polskie nazwy miesięcy w mianowniku lub skrócie:

```
import locale
import pandas as pd

locale.setlocale(locale.LC_ALL, 'PL')
# locale.setlocale(locale.LC_TIME, 'pl_PL.UTF-8') # Na systemach Linux/Mac
# locale.setlocale(locale.LC_TIME, 'Polish_Poland.1250') # Na Windows

data = ['10 styczeń 2025', '15 grudzień 2025', '5 marzec 2025']
data_frame = pd.DataFrame(data)
data_frame[0] = pd.to_datetime(data_frame[0], format='%d %B %Y')
print(data_frame)
data2 = ['10 sty 2025', '15 gru 2025', '5 mar 2025']
df2 = pd.DataFrame(data2)
df2[0] = pd.to_datetime(df2[0], format='%d %b %Y')
print(df2)
```

```
0
0 2025-01-10
1 2025-12-15
2 2025-03-05
0
0 2025-01-10
1 2025-12-15
2 2025-03-05
```

Ćwiczenie: (extime.py)

Załaduj poniższe pliki i przekształć kolumnę z datą:

- date_sale.csv
- date_temp.csv

Wskazówka:

```
import pandas as pd

data = pd.read_csv("date_sale.csv", parse_dates=["Sale_Date"], date_format="%d-%m-%Y")
```

22 Pandas - dane tekstowe

22.1 Normalizacja

Normalizacja danych tekstowych polega na przekształceniu tekstu w jednolity i porównywalny format. W Pandas można to osiągnąć poprzez zastosowanie różnych operacji na kolumnach zawierających dane tekstowe.

Stare podejście (na piechotę, pełna kontrola):

```
import pandas as pd

# Przykładowa ramka danych
data = pd.DataFrame({
    'Text': ['Hello World ', 'Pandas Library43', ' Data Science ']
})

# Usunięcie białych znaków
data['Text'] = data['Text'].str.strip()
print(data)

# Konwersja do małych liter
data['Text'] = data['Text'].str.lower()
print(data)

# Konwersja do wielkich liter
data['Text'] = data['Text'].str.upper()
print(data)

# Usunięcie znaków specjalnych
data['Text'] = data['Text'].str.replace(r'\w\s+', '', regex=True)
print(data)

# Usunięcie liczb
data['Text'] = data['Text'].str.replace(r'\d+', '', regex=True)
print(data)

# Usunięcie duplikatów
data = data.drop_duplicates(subset='Text')
print(data)
```

```
        Text
0      Hello World
1 Pandas Library43
2 Data Science
        Text
0      hello world
1 pandas library43
2 data science
        Text
0      HELLO WORLD
1 PANDAS LIBRARY43
2 DATA SCIENCE
        Text
0      HELLO WORLD
1 PANDAS LIBRARY43
2 DATA SCIENCE
        Text
0      HELLO WORLD
1 PANDAS LIBRARY
2 DATA SCIENCE
        Text
0      HELLO WORLD
1 PANDAS LIBRARY
2 DATA SCIENCE
```

Nowsza wersja (wygodna, ale w detalach trudna)

```
import pandas as pd

# Utworzenie przykładowej serii z różnymi formami zapisu tego samego tekstu
s = pd.Series(['café', 'cafe\u0301', 'caf '])

# Normalizacja do jednolitej formy
normalized = s.str.normalize('NFC')

# Sprawdzenie czy wszystkie wartości są teraz identyczne
print(normalized.unique()) # Powinno zwrócić 1
```

1

Inne opcje: ‘NFC’, ‘NFKC’, ‘NFD’, ‘NFKD’

22.2 Operacje wektorowe na tekstach

Oto tabela w języku Markdown wyjaśniająca funkcje z `pandas.Series.str` i ich zastosowanie:

Funkcja	Opis
<code>len()</code>	Zwraca długość każdego ciągu znaków w serii.
<code>lower()</code>	Konwertuje wszystkie znaki na małe litery.
<code>translate()</code>	Zastępuje znaki według podanej mapy translacji.
<code>islower()</code>	Sprawdza, czy wszystkie znaki w ciągu są małymi literami.
<code>ljust()</code>	Justuje tekst w lewo, wypełniając go określonym znakiem do zadanej szerokości.
<code>upper()</code>	Konwertuje wszystkie znaki na wielkie litery.
<code>startswith()</code>	Sprawdza, czy ciąg znaków zaczyna się od podanego prefiksu.
<code>isupper()</code>	Sprawdza, czy wszystkie znaki w ciągu są wielkimi literami.
<code>rjust()</code>	Justuje tekst w prawo, wypełniając go określonym znakiem do zadanej szerokości.
<code>find()</code>	Zwraca indeks pierwszego wystąpienia podcięgu; zwraca -1, jeśli podciąg nie istnieje.
<code>endswith()</code>	Sprawdza, czy ciąg znaków kończy się podanym sufiksem.
<code>isnumeric()</code>	Sprawdza, czy ciąg zawiera tylko znaki numeryczne.
<code>center()</code>	Centruje tekst, wypełniając go określonym znakiem do zadanej szerokości.
<code>rfind()</code>	Zwraca indeks ostatniego wystąpienia podcięgu; zwraca -1, jeśli podciąg nie istnieje.
<code>isalnum()</code>	Sprawdza, czy ciąg zawiera tylko litery i cyfry.
<code>isdecimal()</code>	Sprawdza, czy ciąg zawiera tylko znaki dziesiętne.
<code>zfill()</code>	Wypełnia ciąg zerami z lewej strony, aby osiągnąć określoną długość.
<code>index()</code>	Zwraca indeks pierwszego wystąpienia podcięgu; zgłasza wyjątek, jeśli podciąg nie istnieje.
<code>isalpha()</code>	Sprawdza, czy ciąg zawiera tylko litery.
<code>split()</code>	Dzieli ciąg na listę podcięgów na podstawie separatora (domyślnie spacja).
<code>strip()</code>	Usuwa białe znaki (lub inne wskazane znaki) z obu stron ciągu.
<code>rindex()</code>	Zwraca indeks ostatniego wystąpienia podcięgu; zgłasza wyjątek, jeśli podciąg nie istnieje.
<code>isdigit()</code>	Sprawdza, czy ciąg zawiera tylko cyfry.
<code>rsplit()</code>	Dzieli ciąg od prawej strony na listę podcięgów na podstawie separatora (domyślnie spacja).
<code>rstrip()</code>	Usuwa białe znaki (lub inne wskazane znaki) z prawej strony ciągu.
<code>capitalize()</code>	Zmienia pierwszą literę na wielką, a resztę na małe.
<code>isspace()</code>	Sprawdza, czy ciąg zawiera tylko białe znaki.

Funkcja	Opis
<code>partition()</code>	Dzieli ciąg na trzy części: przed separator, separator i po separatorze.
<code>lstrip()</code>	Usuwa białe znaki (lub inne wskazane znaki) z lewej strony ciągu.
<code>swapcase()</code>	Zmienia wielkość liter na przeciwną (małe na wielkie i odwrotnie).
<code>istitle()</code>	Sprawdza, czy ciąg jest sformatowany jako tytuł (pierwsze litery wyrazów są wielkie).
<code>rpartition()</code>	Dzieli ciąg na trzy części od prawej strony: przed separator, separator i po separatorze.

Zwykle operacje wektorowe są szybsze:

```
import time
import pandas as pd

# Tworzenie przykładowej ramki danych z 2000000 wierszami
data = {'Text': ['Pandas is awesome'] * 2000000}
data2 = pd.DataFrame(data)

# Funkcja, która konwertuje tekst na małe litery (przykładowa operacja)
def to_lower(text):
    return text.lower()

# 1. Operacja wektorowa
start_vectorized = time.time()
data2['Vectorized'] = data2['Text'].str.lower()
end_vectorized = time.time()

# 2. Operacja z list comprehension
start_comprehension = time.time()
data2['Comprehension'] = [to_lower(text) for text in data2['Text']]
end_comprehension = time.time()

# Czasy wykonania
vectorized_time = end_vectorized - start_vectorized
comprehension_time = end_comprehension - start_comprehension

# Wynik
print(vectorized_time, comprehension_time)
```

0.15144801139831543 0.295424222946167

23 Pandas - inne

23.1 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
```

23.2 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.isna())
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

23.3 Usuwanie duplikatów

```
import pandas as pd

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

23.4 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
```

23.5 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]
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))
```

```
[(0.74, 0.98], (0.028, 0.27], (0.028, 0.27], (0.74, 0.98], (0.27, 0.5], ..., (0.028, 0.27],
Length: 20
Categories (4, interval[float64, right]): [(0.028, 0.27] < (0.27, 0.5] < (0.5, 0.74] < (0.74,
```

```

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())

```

```

[(0.0317, 0.638], (0.0317, 0.638], (-0.687, 0.0317], (-0.687, 0.0317], (0.638, 3.057], ...,
Length: 1000
Categories (4, interval[float64, right]): [(-2.8689999999999998, -0.687] < (-0.687, 0.0317] ...
(-2.8689999999999998, -0.687]      250
(-0.687, 0.0317]                      250
(0.0317, 0.638]                       250
(0.638, 3.057]                        250
Name: count, dtype: int64

```

23.6 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	0.015201	0.010712	0.022034	-0.048752
std	1.023092	1.006975	0.992200	0.995020
min	-3.607188	-3.608614	-3.058460	-3.161708
25%	-0.691654	-0.697235	-0.655643	-0.728871
50%	0.019670	0.036924	0.003099	-0.026097

```

75%      0.717831    0.700505    0.700538    0.556587
max      3.005379    3.276989    2.847965    2.864033
---
841   -3.05846
Name: 2, dtype: float64
---
          0         1         2         3
55  -3.607188  1.961095 -0.751534  0.934036
93   3.005379 -0.076705  1.325572 -0.816526
101  0.320489 -0.173302  0.314575 -3.100732
246 -3.563883  1.469810  0.132427  0.408923
289  0.526651 -3.608614 -1.020767  0.570112
492  1.163101  1.149153  0.869409 -3.161708
514  1.815652 -0.015901  0.696526 -3.044633
616 -3.323272 -1.067982  0.335108  0.440992
841 -1.696792 -2.073041 -3.058460 -0.009462
853  0.998971  3.276989  1.452056  0.550910

```

23.7 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']
}
data2 = pd.DataFrame(data)

# Wyświetlenie oryginalnej ramki danych
print("Oryginalna ramka danych:")
print(data2)

# Zmiana typu danych kolumny 'A' na int
data2['A'] = pd.Series(data2['A'], dtype=int)

# Zmiana typu danych kolumny 'B' na float
data2['B'] = pd.Series(data2['B'], dtype=float)

```

```
# Wyświetlenie ramki danych po zmianie typów
print("\nRamka danych po zmianie typów:")
print(data2)
```

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']
}
data2 = pd.DataFrame(data)

# Wyświetlenie oryginalnej ramki danych
print("Oryginalna ramka danych:")
print(data2)

# Zmiana typu danych kolumny 'A' na int
data2['A'] = data2['A'].astype(int)

# Zmiana typu danych kolumny 'B' na float
data2['B'] = data2['B'].astype(float)
```

```
# Wyświetlenie ramki danych po zmianie typów
print("\nRamka danych po zmianie typów:")
print(data2)
```

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

23.8 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']
}
data2 = pd.DataFrame(data)

# Wyświetlenie oryginalnej ramki danych
print("Oryginalna ramka danych:")
print(data2)

# Zmiana małych liter na duże w kolumnie 'A'
```

```

data2['A'] = data2['A'].str.upper()

# Zastąpienie kropki przecinkiem w kolumnie 'B'
data2['B'] = data2['B'].str.replace('. ', ',')

# Wyświetlenie ramki danych po modyfikacji
print("\nRamka danych po modyfikacji:")
print(data2)

```

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

23.9 Operacje manipulacyjne

Ściągawka 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 wzdluz wspolnej 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, sort=False,
                suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)

```

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 posortowany 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

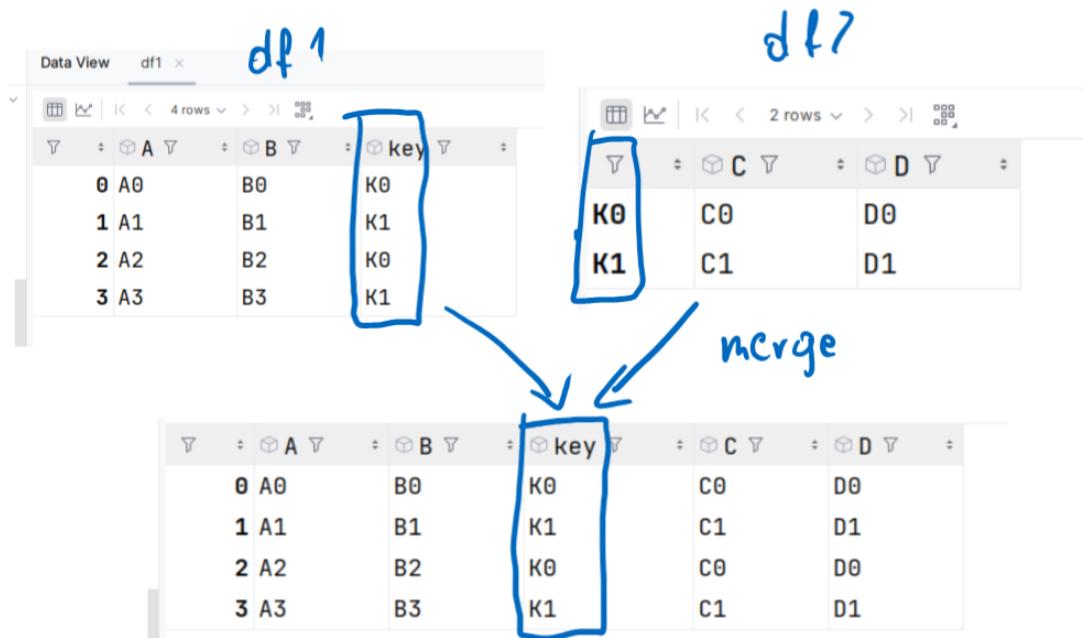
df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2', 'A3'],
    'B': ['B0', 'B1', 'B2', 'B3'],
    'key': ['K0', 'K1', 'K0', 'K1']
})

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']
})

```

```

        '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

	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

df1

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

df2

Y	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K4	C2	D2
3	K5	C3	D3

merge
inner

Y	key	A	B	C	D	_merge
0	K0	A0	B0	C0	D0	both
1	K1	A1	B1	C1	D1	both

df1

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

df2

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K4	C2	D2
3	K5	C3	D3

merge
outer

	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

Nan - break

df1

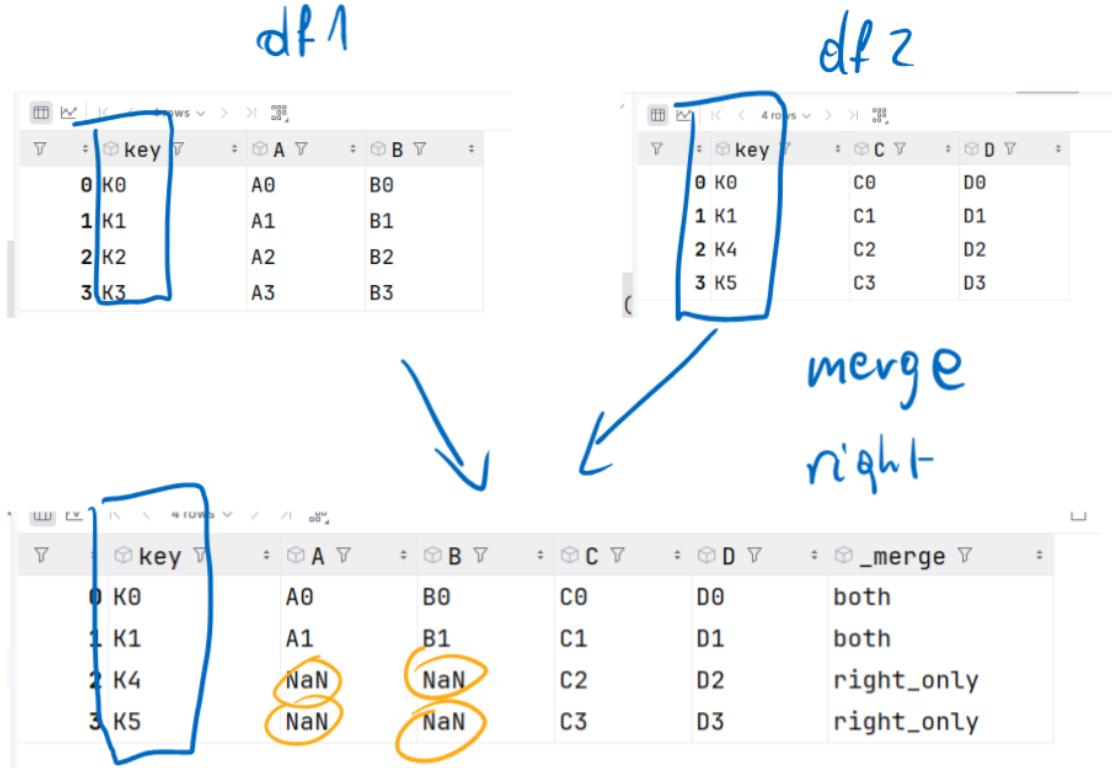
	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

df2

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K4	C2	D2
3	K5	C3	D3

merge
left

	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



- 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. Domyslnie 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

df1

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

df2

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

join

	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 wzduż określonej osi.

Podstawowe użycie tej metody wygląda następująco:

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

Gdzie:

- `objs`: sekwencja ramek danych, które chcesz połączyć.
- `axis`: os, wzduż 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
```

The screenshot shows three DataFrames in a Jupyter Notebook:

- df1**: A DataFrame with columns A and B, containing rows 0 to 5.
- df2**: A DataFrame with columns A and B, containing rows 0 to 2.
- concatenated_df**: A DataFrame resulting from concatenating df1 and df2 along the index, with `ignore_index=True`.

Annotations with arrows point from the text `pd.concat(objs=[df1, df2], ignore_index=True)` to the concatenated DataFrame and to the individual DataFrames above it.

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

	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
df2	0	NaN	NaN	C0	D0
	1	NaN	NaN	C1	D1
	2	NaN	NaN	C2	D2

	A	B
0	A0	B0
1	A1	B1
2	A2	B2

	C	D
0	C0	D0
1	C1	D1
2	C2	D2

```
pd.concat( objs: [df1, df2], axis=1)
```

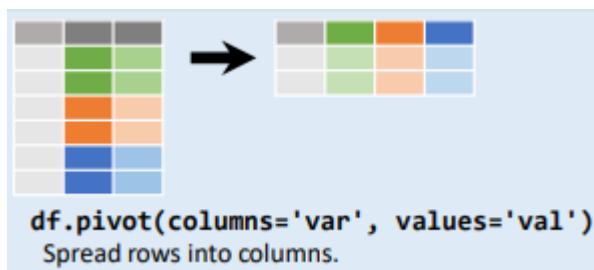
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2

The figure consists of three screenshots of a Jupyter Notebook interface. The top-left screenshot shows a DataFrame with columns A and B, and rows 0, 1, 2, containing values A0, A1, A2, B0, B1, B2 respectively. The top-right screenshot shows a DataFrame with columns C and D, and rows 0, 1, 2, containing values C0, C1, C2, D0, D1, D2 respectively. The bottom screenshot shows the result of concatenating these DataFrames with keys 'df1' and 'df2'. It has two levels of headers: 'df1' and 'df2' at the top, and A, B, C, D across the columns. The first six rows (0, 1, 2 for df1, and 0, 1, 2 for df2) have values A0-A2, B0-B2, NaN-NaN, and C0-C2, D0-D2 respectively. Handwritten pink text below the middle screenshot reads "podwójny index (multiindex)".

	A	B	C	D
df1	A0 A1 A2	B0 B1 B2	NaN NaN NaN	NaN NaN NaN
df2	NaN NaN NaN	NaN NaN NaN	C0 C1 C2	D0 D1 D2
0	A0 A1 A2	B0 B1 B2	NaN NaN NaN	NaN NaN NaN
1			C0 C1 C2	D0 D1 D2
2				

- 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
```

The image shows two DataFrames in a Jupyter Notebook interface. The top DataFrame has columns 'foo', 'bar', 'baz', and 'zoo'. The bottom DataFrame is the result of pivoting the top one, with columns 'A', 'B', and 'C'.

```

df = pd.DataFrame({
    'foo': ['one', 'one', 'one', 'two', 'two', 'two'],
    'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
    'baz': ['x', 'y', 'z', 'q', 'w', 't']
})
df.pivot(index='foo', columns='bar', values='baz')

```

	A	B	C
one	1	2	3
two	4	5	6

- `wide_to_long`

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

Metoda `wide_to_long` jest używana do przekształcenia danych z szerokiego formatu (gdzie każda kolumna zawiera wiele zmiennych) do długiego formatu (gdzie każda kolumna zawiera jedną zmienną z wieloma pomiarami). Jest to przydatne, gdy mamy dane, które są rozłożone w wielu kolumnach z powtarzającymi się lub sekwencyjnymi nazwami, i chcemy przekształcić te dane w sposób, który ułatwia analizę i wizualizację.

Wyjaśnienie parametrów `wide_to_long`

- **stubnames:** Lista początkowych części nazw kolumn, które mają zostać przekształcone.
- **i:** Nazwa kolumny lub lista kolumn, które identyfikują poszczególne wiersze. W naszym przykładzie jest to `id`, które unikalnie identyfikuje osobę.
- **j:** Nazwa nowej kolumny, w której będą przechowywane różne poziomy zmiennych (w naszym przypadku `rok`).
- **sep:** Opcjonalny separator (domyślnie "").

```

import pandas as pd

# Przykładowe dane
data = {
    'id': ['A', 'B', 'C'],
    'height_2020': [180, 175, 165],
    'weight_2020': [70, 76, 65],
    'height_2021': [181, 176, 166],
    'weight_2021': [71, 77, 66]
}

data2 = pd.DataFrame(data)

# Przekształcenie do formatu długiego
df_long = pd.wide_to_long(data2, stubnames=['height', 'weight'], i='id', j='year', sep='_')
df_long = df_long.reset_index()

print(df_long)

```

	id	year	height	weight
0	A	2020	180	70
1	B	2020	175	76
2	C	2020	165	65
3	A	2021	181	71
4	B	2021	176	77
5	C	2021	166	66

```

df_wide = pd.DataFrame({
    'id': ['A', 'B', 'C'],
    'height_2020': [180, 175, 165],
    'weight_2020': [70, 76, 65],
    'height_2021': [181, 176, 166],
    'weight_2021': [71, 77, 66]
})

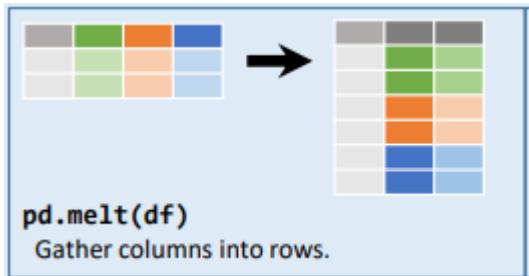
df_long = pd.wide_to_long(df_wide, stubnames=['height', 'weight'], i='id', j='year', sep='_')
df_long.reset_index()

```

	<code>id</code>	<code>height_2020</code>	<code>weight_2020</code>	<code>height_2021</code>	<code>weight_2021</code>
0	A	180	70	181	71
1	B	175	76	176	77
2	C	165	65	166	66

	<code>id</code>	<code>year</code>	<code>height</code>	<code>weight</code>
0	A	2020	180	70
1	B	2020	175	76
2	C	2020	165	65
3	A	2021	181	71
4	B	2021	176	77
5	C	2021	166	66

- `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=0)
```

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

data = 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(data)
melted_df = data.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

The screenshot shows a Jupyter Notebook interface with two DataFrames and a code cell.

DataFrame 1:

	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

DataFrame 2:

	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

Code Cell:

```
df.melt(id_vars=['A', 'B'], value_vars=['C', 'D'], var_name='My_Var', value_name='My_Val')
```

An arrow points from the original DataFrame to the resulting DataFrame, indicating the transformation.

Część V

Analia struktury danych

24 Analiza struktury

Miary statystyczne

- to charakterystyki liczbowe pozwalające opisać właściwości rozkładu badanej cechy.
- inne nazwy:
 - parametry - dane analizowane z całej populacji,
 - statystyki próby - dane analizowane z próby.

Klasyfikacja miar statystycznych:

- miary położenia (miary poziomu, miary przeciętne) - pozwalają określić gdzie w zbiorze wartości znajdują się dane pochodzące z obserwacji,
- miary zróżnicowania (miary rozproszenia, zmienności, rozrzutu, dyspersji) - pozwalają określić zróżnicowanie jednostek,
- miary asymetrii (miary skośności) - pozwalają określić asymetrię (czy większość jednostek ma wartości większe lub mniejsze względem przeciętnego poziomu),
- miary koncentracji - pozwalają określić skupienie wartości względem średniej.

24.1 Miary położenia

- średnie klasyczne:
 - średnia arytmetyczna,
- średnie pozycyjne i kwantyle:
 - dominanta/moda,
 - mediana (kwartyl drugi),
 - kwantyle (kwartyle, decyle, percentile).

24.1.1 Średnia arytmetyczna

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Interpretacja średniej arytmetycznej polega na rozumieniu jej jako reprezentacji “środkowego” lub “typowego” poziomu cechy badanej zbiorowości. Średnia daje ogólne wyobrażenie o danych, ale może być myląca w przypadku obecności skrajnych wartości (outlierów), które mogą znacząco wpływać na jej wartość. Przydatna jest w wielu dziedzinach, od ekonomii po nauki społeczne, jako sposób na podsumowanie danych i porównanie różnych grup lub zestawów danych. Warto pamiętać, że średnia nie zawsze jest najlepszym wyborem dla skośnych rozkładów i może nie odzwierciedlać adekwatnie rozkładu danych, zwłaszcza w obecności skrajnych wartości.

```
import pandas as pd

# Przykładowe dane jako seria Pandas
dane = pd.Series([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

# Obliczanie średniej
srednia = dane.mean()

print(f"Średnia: {srednia}")
```

Średnia: 55.0

24.1.2 Dominanta

- symbol: Do
- inaczej wartość modalna, moda.
- dla cechy skokowej jest to wartość cechy występująca najczęściej.
- dla cechy ciągły to wartość cechy, wokół której oscyluje najwięcej pomiarów (argument, dla którego gęstość prawdopodobieństwa przyjmuje wartość największą)

```
import pandas as pd

# Przykładowe dane dla zmiennej skokowej
dane_skokowe = pd.Series([1, 2, 2, 3, 3, 3, 4, 4, 4, 4])

# Obliczanie mody
moda_skokowa = dane_skokowe.mode()

print(f"Moda dla zmiennej skokowej: {moda_skokowa.tolist()}")
```

Moda dla zmiennej skokowej: [4]

Uwagi:

- nie zawsze można ją określić dokładnie.
- wyznaczenie dominanty ma sens kiedy rozkład jest jednomodalny (jednostki mają jeden punkt skupienia), liczenie jej dla rozkładów wielomodalnych jest błędem.
- nie jest wrażliwa na skrajne wartości jak średnia arytmetyczna.
- w przypadku rozkładu symetrycznego dominantą równa się średniej

24.1.3 Mediana

- symbol: M_e
- można ją wyznaczyć dla cech wyrażonych w dowolnej skali poza skalą nominalną.
- wartość cechy jaką ma jednostka w środku uporządkowanego ciągu obserwacji.
- dla nieparzystej liczby obserwacji: wartość dla pozycji $\frac{n+1}{2}$
- dla parzystej liczby obserwacji:
 - wyznaczamy wartości dla pozycji $\frac{n}{2}$ oraz $\frac{n}{2} + 1$
 - liczymy średnią wartości

Uwaga: częstym błędem jest mylenie wartości cechy z jej pozycją.

Kod

```
import pandas as pd

# Przykładowe dane w DataFrame
df = pd.DataFrame({
    'Kolumna1': [10, 20, 30, 40, 50],
    'Kolumna2': [15, 25, 35, 45, 55]
})

# Obliczanie mediany dla każdej kolumny
mediany = df.median()

print("Medianą dla każdej kolumny:")
print(mediany)

Medianą dla każdej kolumny:
Kolumna1    30.0
Kolumna2    35.0
dtype: float64
```

Interpretacja mediany:

- przynajmniej połowa jednostek jest mniejsza lub równa medianie.
- mediana jest nieczuła na wartości ekstremalne.

24.1.4 Kwantyle

- wartości cechy, które dzielą zbiorowość na określone części pod względem liczby jednostek.
- najczęściej używane:
 - kwartyle - dzielą zbiorowość na 4 równe części (kwartyl drugi to mediana)
 - decyle - dzielą zbiorowość na 10 równych części
 - percentile (centyle) - dzielą zbiorowość na 100 równych części.

Kwartyle

- symbole: Q_1, Q_2, Q_3

```
import pandas as pd

# Przykładowe dane jako seria Pandas
dane = pd.Series([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

# Obliczanie kwartylów
kwartyl_1 = dane.quantile(0.25) # Pierwszy kwartyl (Q1)
mediana = dane.quantile(0.50) # Mediana (Q2)
kwartyl_3 = dane.quantile(0.75) # Trzeci kwartyl (Q3)

print(f"Pierwszy kwartyl (Q1): {kwartyl_1}")
print(f"Mediana (Q2): {mediana}")
print(f"Trzeci kwartyl (Q3): {kwartyl_3}")
```

Pierwszy kwartyl (Q1): 32.5
 Mediana (Q2): 55.0
 Trzeci kwartyl (Q3): 77.5

24.2 Miary zmienności

- podział:
 - miary klasyczne - na podstawie wszystkich obserwacji,
 - * wariancja,

- * odchylenie standardowe,
- miary pozycyjne - na podstawie wartości cechy zajmujących określone pozycje,
 - * rozstęp
 - * rozstęp międzykwartylowy,

24.2.1 Rozstęp

- symbol $R = \max - \min$
- inaczej empiryczny obszar zmienności, amplituda wahań.
- różnica między wartością maksymalną a wartością minimalną.

```
import pandas as pd

# Przykładowe dane jako seria Pandas
dane = pd.Series([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

# Obliczanie maksimum i minimum
maksimum = dane.max()
minimum = dane.min()

# Obliczanie różnicy między maksimum a minimum
roznica = maksimum - minimum

print(f"Maksimum: {maksimum}")
print(f"Minimum: {minimum}")
print(f"Różnica między maksimum a minimum: {roznica}")
```

Maksimum: 100
 Minimum: 10
 Różnica między maksimum a minimum: 90

24.2.2 Rozstęp międzykwartylowy

- symbol $R_Q = Q_3 - Q_1$
- różnica pomiędzy kwartylem trzecim a kwartylem pierwszym.
- mierzy zakres 50% środkowych jednostek.

24.2.3 Odchylenie ćwiartkowe

- wzór: $Q = \frac{Q_3 - Q_1}{2}$
- połowa rozstępu międzykwartylowego.

24.2.4 Wariancja

Wariancja informuje o tym, jak duże jest zróżnicowanie wyników w danym zbiorze wartości cechy. Inaczej mówiąc, czy wyniki są bardziej skoncentrowane wokół średniej, czy są małe różnice pomiędzy średnią a poszczególnymi wynikami czy może rozproszenie wyników jest duże, duża jest różnica poszczególnych wyników od średniej.

- wzór: $s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}$ (populacja) lub $s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$ (próba)

24.2.5 Odchylenie standardowe

Odchylenie standardowe mierzy zróżnicowanie o mianie zgodnym z mianem badanej cechy, daje przeciętne różnice poszczególnych wartości cechy od średniej arytmetycznej.

- wzór: $s = \sqrt{s^2}$

```
import pandas as pd

# Przykładowe dane jako seria Pandas
dane = pd.Series([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

# Obliczanie wariancji
wariancja = dane.var() # Uwaga: Pandas domyślnie dzieli przez (n-1), co odpowiada wariancji

# Obliczanie odchylenia standardowego
odchylenie_standardowe = dane.std() # Domyślnie liczone z próby (n-1)

print(f"Wariancja: {wariancja}")
print(f"Odchylenie standardowe: {odchylenie_standardowe}")
```

```
Wariancja: 916.6666666666666
Odchylenie standardowe: 30.276503540974915
```

W praktyce często zakłada się, że dane mają rozkład normalny. Założenie to nigdy nie jest całkowicie spełnione. Rozkład normalny ma bowiem niezerową gęstość prawdopodobieństwa dla każdej wartości ze zbioru liczb rzeczywistych, a w realnym świecie wartości zmiennych losowych są zawsze ograniczone, na przykład nie istnieją ludzie o ujemnym wzroście. Bardzo często jednak założenie to jest spełnione z wystarczająco dobrym przybliżeniem. Im lepiej jest ono uzasadnione, tym bliższe prawdy mogą być poniższe stwierdzenia:

- 68% wartości cechy leży w odległości $\leq s$ od wartości oczekiwanej

- 95,5% wartości cechy leży w odległości $\leq 2s$ od wartości oczekiwanej
- 99,7% wartości cechy leży w odległości $\leq 3s$ od wartości oczekiwanej.

https://en.wikipedia.org/wiki/68%25-90%25-99.7_%25_rule

24.3 Miary asymetrii

Jak rozpoznać typ rozkładu?

- rozkład symetryczny

$$\bar{x} = Me = Do$$

- rozkład o asymetrii prawostronnej

$$\bar{x} > Me > Do$$

- rozkład o asymetrii lewostronnej

$$\bar{x} < Me < Do$$

- klasyczny współczynnik asymetrii

- wzór:

$$A_s = \frac{m_3}{(s^2)^{\frac{3}{2}}}$$

24.4 Miary koncentracji

Współczynnik kurtozy

- inaczej współczynnik koncentracji, współczynnik spłaszczenia.
- wzór:

$$K = \frac{m_4}{s^4} \quad m_4 = \frac{\sum_{i=1}^n (x_i - \bar{x})^4}{n}$$

Interpretacja kurtozy

- $K = 3$ - rozkład ma taką koncentrację jak rozkład normalny
- $K > 3$ - koncentracja silniejsza
- $K < 3$ - koncentracja słabsza

Czasem bada się współczynnik ekcesu $K' = K - 3$.

Wysoka kurtoza oznacza większą liczbę wartości ekstremalnych (skrajnych), natomiast niska kurtoza wskazuje na rozkład bardziej płaski niż normalny.

```
import pandas as pd

# Przykładowe dane w serii
dane = pd.Series([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

# Obliczanie skośności
skosnosc = dane.skew()

# Obliczanie kurtozy
kurtoza = dane.kurt()

print(f"Skośność: {skosnosc}")
print(f"Kurtoza: {kurtoza}")
```

```
Skośność: 0.0
Kurtoza: -1.2000000000000002
```

24.5 Przyśpieszanie działania:

```
import pandas as pd
import numpy as np

# Tworzymy ramkę danych
data = {
    "Produkt": ["A", "B", "A", "C", "B", "A", "C", "B", "A", "C"],
    "Sprzedaż": [200, 150, 250, 300, 200, 220, 310, 180, 240, 290],
    "Koszt": [120, 100, 140, 180, 110, 130, 190, 105, 125, 170],
    "Marża": [80, 50, 110, 120, 90, 90, 120, 75, 115, 120],
}

df = pd.DataFrame(data)
print(df)
```

Ramka danych:

Produkt	Sprzedaż	Koszt	Marża
A	200	120	80
B	150	100	50
A	250	140	110
C	300	180	120
B	200	110	90
A	220	130	90
C	310	190	120
B	180	105	75
A	240	125	115
C	290	170	120

Średnia dla wybranych dwóch kolumn

```
# Obliczamy średnią dla kolumn "Sprzedaż" i "Koszt"
mean_values = df[["Sprzedaż", "Koszt"]].mean()
print(mean_values)
```

Funkcja `describe()`

```
# Generujemy opisowe statystyki dla danych liczbowych
description = df.describe()
print(description)
```

Działanie funkcji `agg`

Funkcja `agg` pozwala na zastosowanie różnych miar dla wielu kolumn.

```
# Zastosowanie funkcji agregujących
agg_results = df[["Sprzedaż", "Koszt"]].agg(["mean", "sum", "max"])
print(agg_results)
```

Działanie `groupby`

Grupowanie danych według kolumny kategorycznej (np. “Produkt”).

```
# Grupowanie i obliczanie średniej
grouped = df.groupby("Produkt")[["Sprzedaż", "Koszt"]].mean()
print(grouped)
```

Opcja `numeric_only=True`

Opcja `numeric_only=True` pozwala analizować tylko kolumny liczbowe, pomijając kategoryczne.

```
# Obliczanie sumy tylko dla kolumn liczbowych
numeric_sum = df.sum(numeric_only=True)
print(numeric_sum)
```

Ćwiczenia: (ex11.py - ex20.py)

Sprawdź, dla których plików wygodnie jest liczenie odpowiednich charakterystyk.

<https://github.com/pjastr/AIWD-files>

Część VI

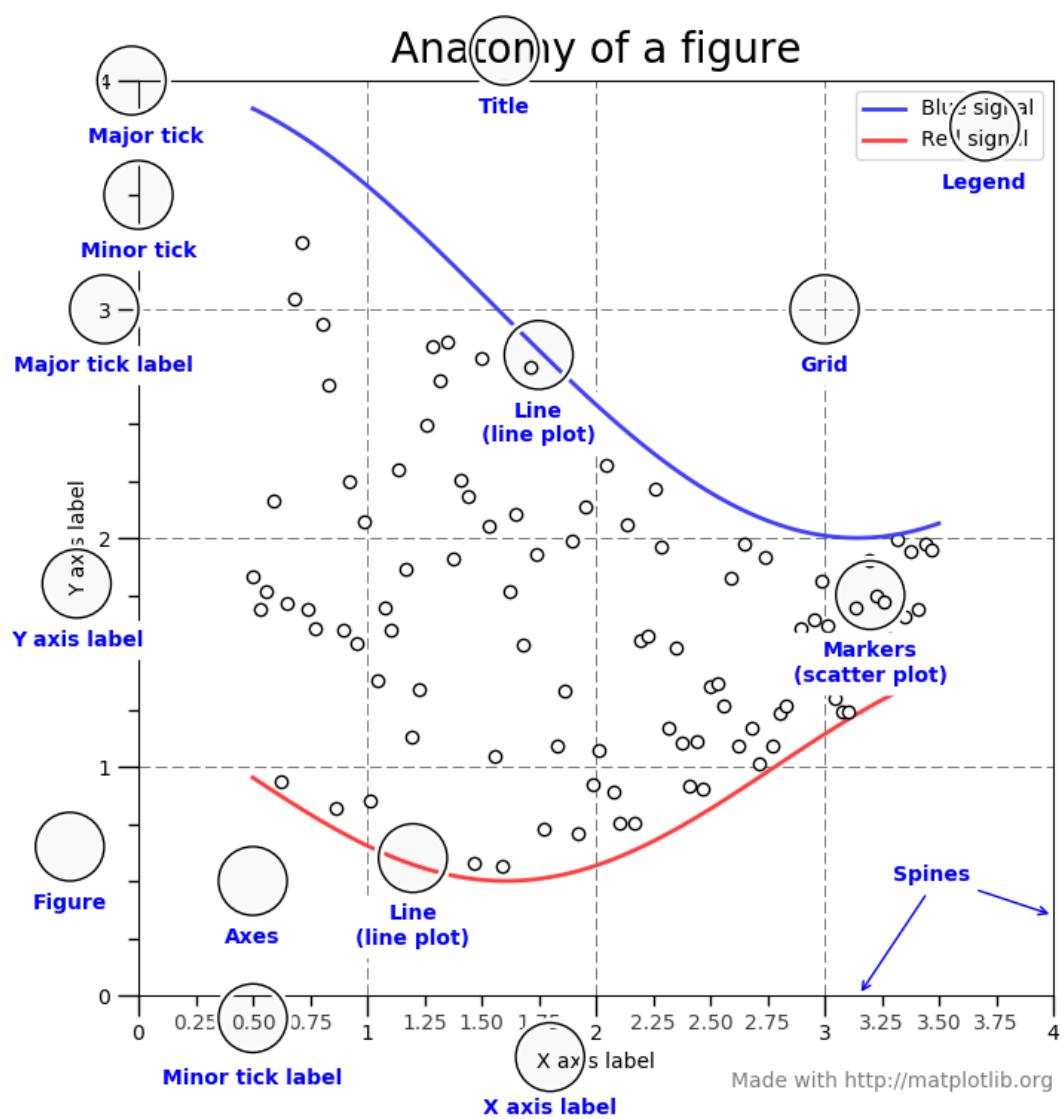
Matplotlib

25 Matplotlib - start

Ladowanie biblioteki:

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

<https://matplotlib.org/>



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>

JDEA :

- 1) IMPORT BIBLIOTEK
- 2) „WYODRZĘBNIENIE” WELTORÓW NA WSPOŁRZĘDNE WYKRESU
- 3) „OPCJE WYKRESU”
- 4) „FUNKCJE RYSUJĄCE”
- 5) „DODANKI”
- 6) ZAPIS DO PLIKU
- 7) „SHOW ()”

26 Matplotlib - wykres liniowy

Wykres liniowy to jedno z najpotężniejszych narzędzi wizualizacji danych, które pozwala nam dostrzec zależności i trendy, które mogłyby pozostać niezauważone w surowych danych. Poniżej przedstawiam szczegółowe objaśnienie, kiedy i dlaczego warto sięgnąć po ten typ wykresu.

Wykres liniowy najlepiej sprawdza się, gdy chcemy przedstawić **zmiany wartości w czasie** lub w **funkcji innej zmiennej ciągłej**. Jego siła tkwi w zdolności do ukazywania ciągłych relacji między punktami danych, co pozwala na łatwe śledzenie trendów i wzorców.

1. Analiza zmian w czasie

Wykres liniowy doskonale obrazuje, jak dane zmieniają się w kolejnych jednostkach czasu. Jest nieoceniony przy prezentacji: - Trendów gospodarczych (wzrost PKB, inflacja, stopy bezrobocia) - Zmian na rynkach finansowych (kursy walut, ceny akcji, stopy procentowe) - Danych klimatycznych i pogodowych (zmiany temperatury, opady, poziom zanieczyszczeń) - Wskaźników zdrowotnych (tętno, poziom cukru we krwi, ciśnienie)

2. Ukazywanie zależności między zmiennymi

Gdy chcemy zbadać, jak jedna zmienna wpływa na drugą, wykres liniowy pozwala na intuicyjne przedstawienie tych relacji: - Związek między poziomem wykształcenia a średnimi zarobkami - Korelacja między wiekiem a określonymi umiejętnościami - Zależność między nakładami na reklamę a wynikami sprzedaży

3. Porównywanie wielu trendów jednocześnie

Wykres liniowy umożliwia efektywne zestawienie kilku serii danych na jednym wykresie: - Analiza sprzedaży różnych produktów w czasie - Porównanie wyników różnych regionów, zespołów lub krajów - Zestawienie faktycznych wyników z planowanymi celami - Porównanie różnych wskaźników ekonomicznych w tym samym okresie

4. Analiza korelacji i przyczynowości

Linie na wykresie pomagają dostrzec, jak zmiany jednej zmiennej mogą wpływać na inne: - Badanie wpływu cen paliwa na sprzedaż różnych typów pojazdów - Analiza związku między temperaturą otoczenia a zużyciem energii - Ocena wpływu kampanii marketingowych na świadomość marki

5. Identyfikacja anomalii i punktów zwrotnych

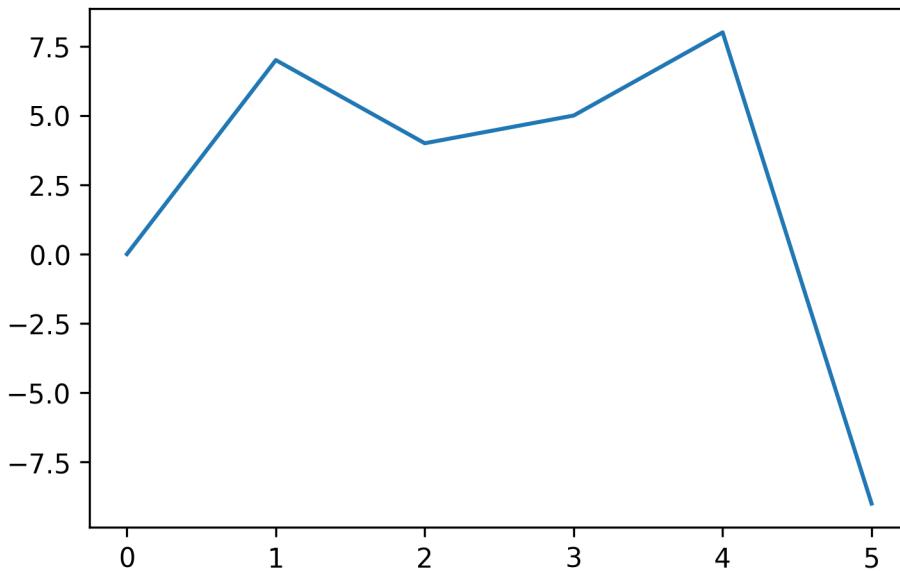
Wykres liniowy pozwala szybko zauważać wartości odstające oraz momenty znaczących zmian:
- Wykrywanie nietypowych wzorców w danych finansowych - Identyfikacja punktów przełomowych w trendach społecznych - Rozpoznawanie sezonowych wahań w danych

Choć wykresy liniowe są najodpowiedniejsze dla danych ciągłych, mogą być również wykorzystywane do prezentacji danych dyskretnych, o ile istnieje logiczny związek między kolejnymi punktami. Na przykład, miesięczne wyniki sprzedaży to dane dyskretne, ale ich przedstawienie w formie linii pomoże dostrzec trend roczny.

Należy jednak pamiętać, że łączenie punktów linią sugeruje ciągłość między nimi. Jeśli nie ma logicznej ciągłości między punktami danych (np. przy porównywaniu niezwiązań ze sobą kategorii), lepszym wyborem będzie wykres słupkowy lub punktowy.

```
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

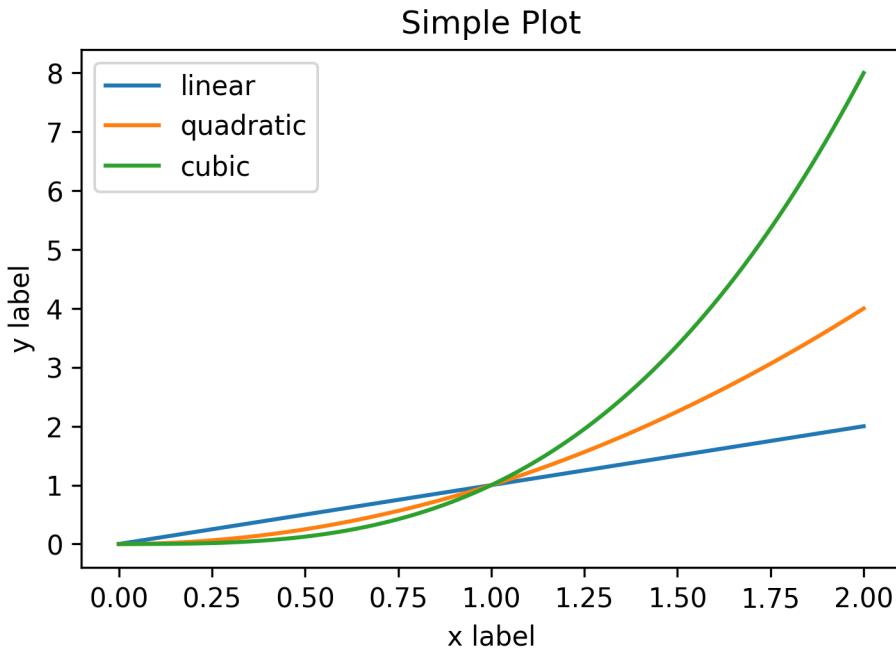
x = np.linspace(0, 2, 100) ①
plt.plot(x, x, label='linear') ②
plt.plot(x, x ** 2, label='quadratic') ③
```

```

plt.plot(x, x ** 3, label='cubic')          ④
plt.xlabel('x label')                      ⑤
plt.ylabel('y label')                      ⑥
plt.title("Simple Plot")                  ⑦
plt.legend()                            ⑧
plt.show()                             ⑨

```

- 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 sześciennego ($y = x^3$) 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.



Wersja obiektowa:

```

import matplotlib.pyplot as plt
import numpy as np

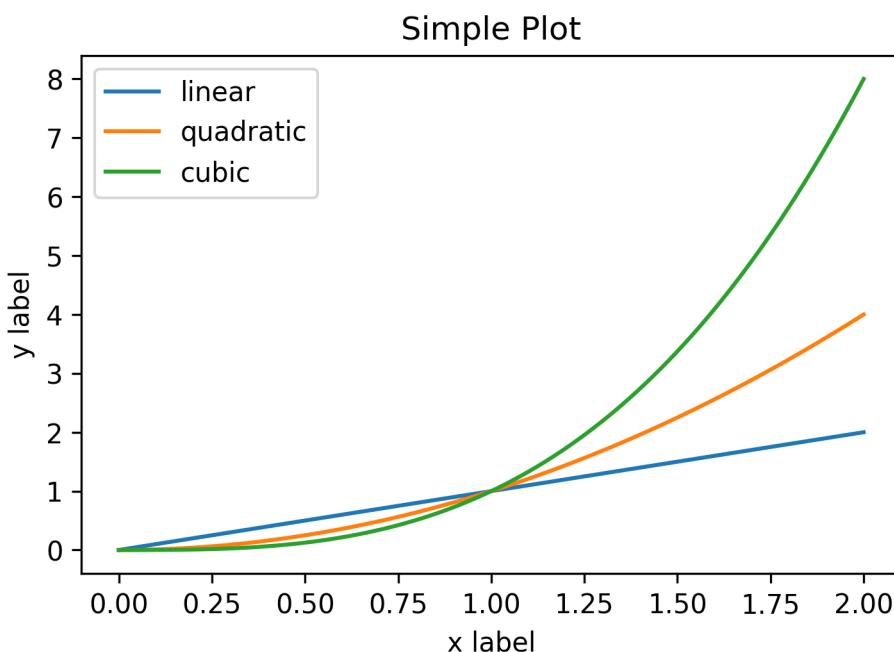
fig, ax = plt.subplots()
x = np.linspace(0, 2, 100)

ax.plot(x, x, label='linear')
ax.plot(x, x ** 2, label='quadratic')
ax.plot(x, x ** 3, label='cubic')

ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")

ax.legend()
plt.show()

```



Wersja z datami:

```

import matplotlib.pyplot as plt
import numpy as np

```

```

daty = np.arange('2024-01', '2025-01', dtype='datetime64')

index = np.arange(len(daty))

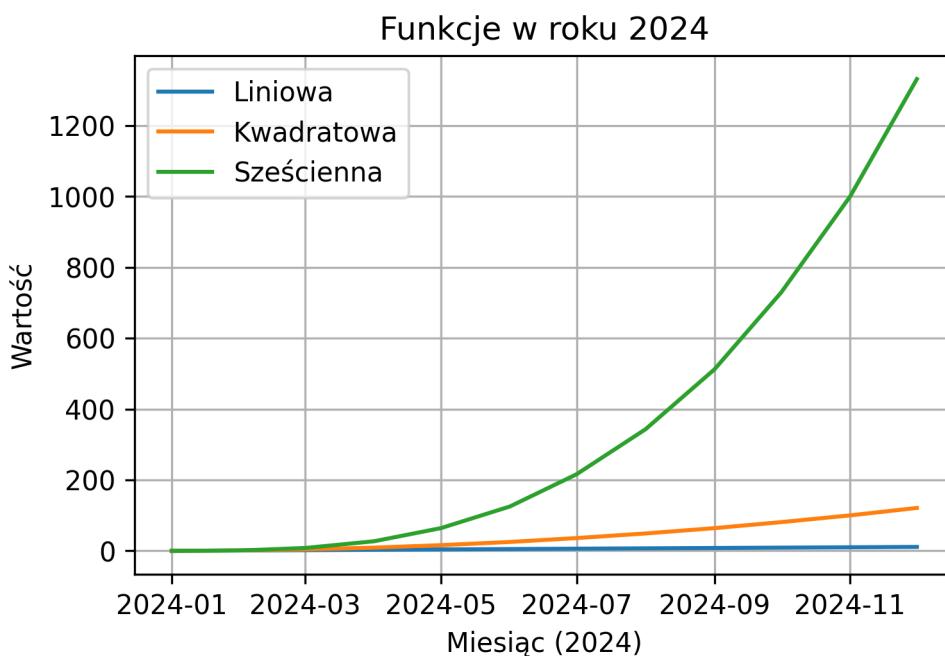
y_1 = index
y_2 = index ** 2
y_3 = index ** 3

plt.plot(daty, y_1, label='Liniowa')
plt.plot(daty, y_2, label='Kwadratowa')
plt.plot(daty, y_3, label='Sześcienna')

plt.xlabel('Miesiąc (2024)')
plt.ylabel('Wartość')
plt.title("Funkcje w roku 2024")
plt.legend()
plt.grid(True)

plt.show()

```



```

import matplotlib.pyplot as plt
import numpy as np

```

```

# Tworzenie danych
daty = np.arange('2024-01', '2025-01', dtype='datetime64')
index = np.arange(len(daty))
y_1 = index
y_2 = index**2
y_3 = index**3

# Tworzenie figury i osi w podejściu obiektowym
fig, ax = plt.subplots()

# Rysowanie linii na osiach
ax.plot(daty, y_1, label='Liniowa')
ax.plot(daty, y_2, label='Kwadratowa')
ax.plot(daty, y_3, label='Sześcienna')

# Dodawanie etykiet i tytułu
ax.set_xlabel('Miesiąc (2024)')
ax.set_ylabel('Wartość')
ax.set_title('Funkcje w roku 2024')

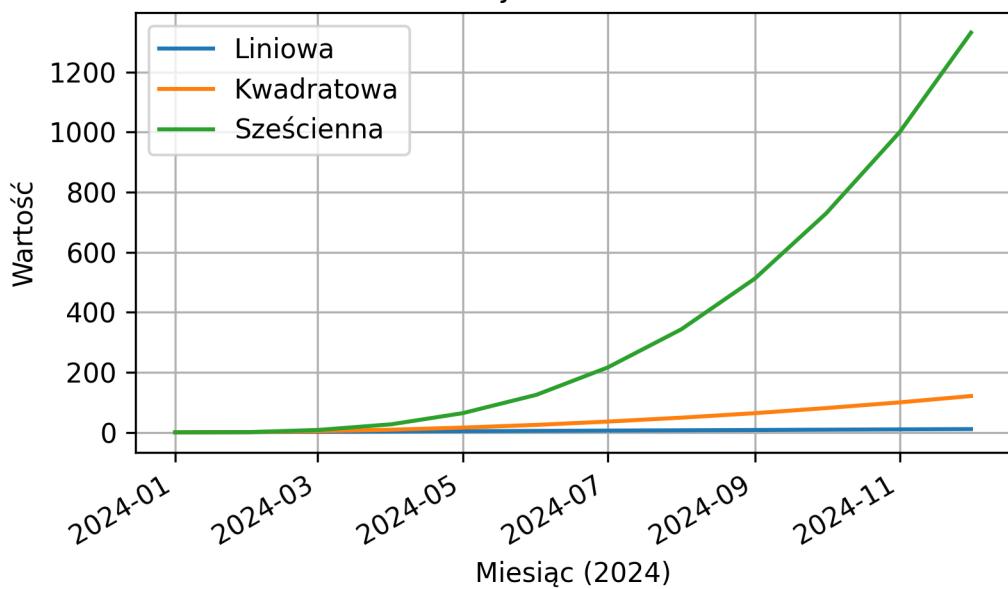
# Dodawanie legendy i siatki
ax.legend()
ax.grid(True)

# Dostosowanie wyglądu - opcjonalne ulepszenia
fig.autofmt_xdate() # Automatyczne formatowanie dat na osi X dla lepszej czytelności
plt.tight_layout() # Automatyczne dostosowanie rozmiaru wykresu

# Wyświetlenie wykresu
plt.show()

```

Funkcje w roku 2024



27 Matplotlib - dodatki cz.1

27.1 Parametry legendy

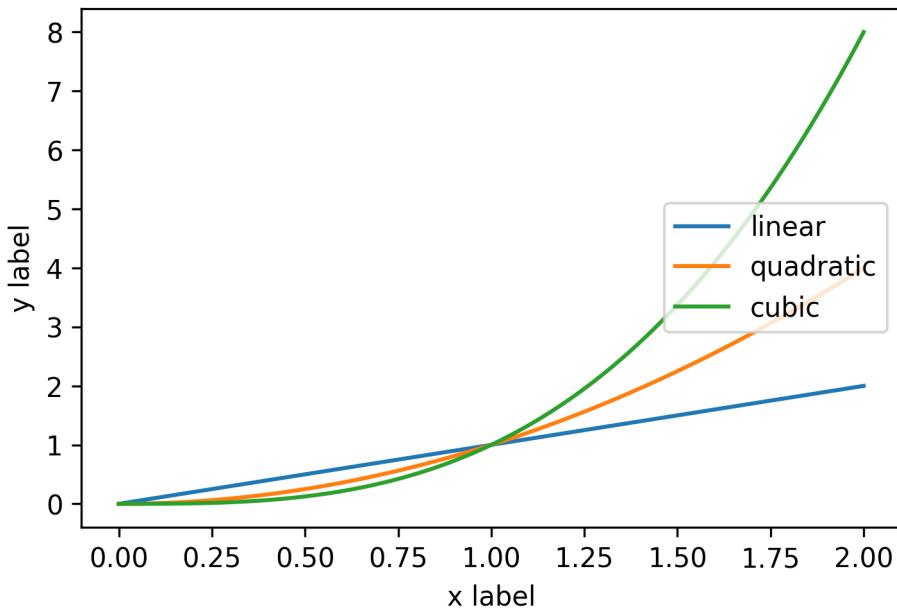
Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

**Parametry lokalizacji
legendy**

```
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(loc = 5)
plt.show()
```

Simple Plot



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

# Tworzymy obiekt Figure i osie (Axes)
fig = plt.figure()
ax = fig.add_subplot(111) # 111 oznacza: 1 wiersz, 1 kolumna, pierwszy wykres

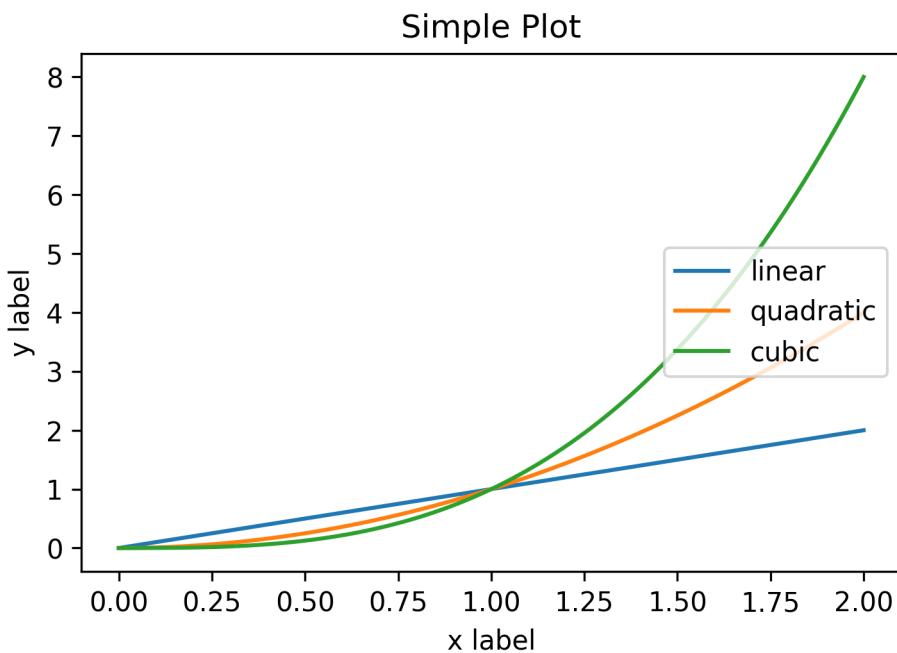
# Generujemy dane
x = np.linspace(0, 2, 100)

# Rysujemy wykresy na osi
ax.plot(x, x, label='linear')
ax.plot(x, x**2, label='quadratic')
ax.plot(x, x**3, label='cubic')

# Dodajemy etykiety i tytuł
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")

# Dodajemy legendę
ax.legend(loc=5) # loc=5 oznacza położenie "right"
```

```
# Wyświetlamy wykres  
plt.show()
```

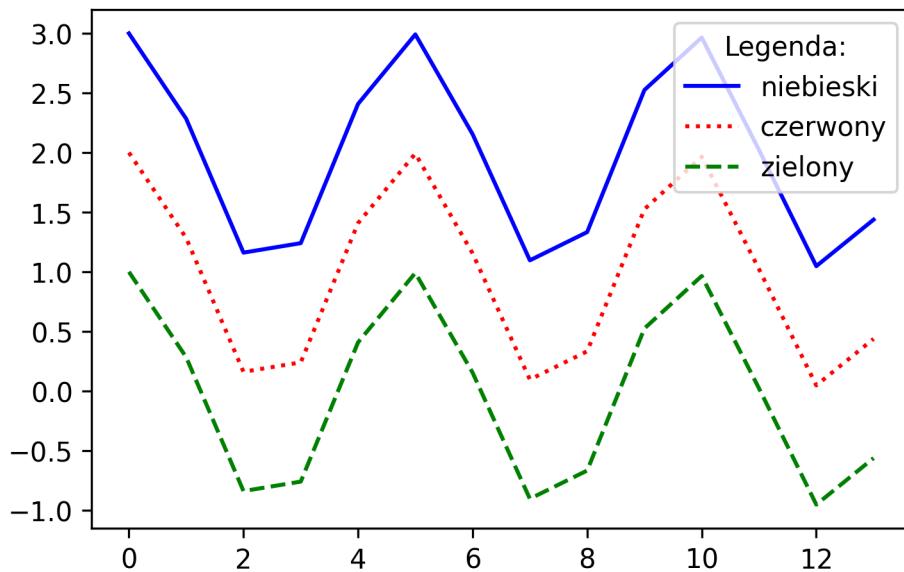


27.2 Style, kolory linii

```
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, przemnozonej 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 góre. 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 góre. 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

<code>'solid'</code>	
<code>'dotted'</code>	
<code>'dashed'</code>	
<code>'dashdot'</code>	

Parametrized linestyles

<code>loosely dotted (0, (1, 10))</code>	
<code>dotted (0, (1, 1))</code>	
<code>densely dotted (0, (1, 1))</code>	
<code>loosely dashed (0, (5, 10))</code>	
<code>dashed (0, (5, 5))</code>	
<code>densely dashed (0, (5, 1))</code>	
<code>loosely dashdotted (0, (3, 10, 1, 10))</code>	
<code>dashdotted (0, (3, 5, 1, 5))</code>	
<code>densely dashdotted (0, (3, 1, 1, 1))</code>	
<code>dashdotdotted (0, (3, 5, 1, 5, 1, 5))</code>	
<code>loosely dashdotdotted (0, (3, 10, 1, 10, 1, 10))</code>	
<code>densely dashdotdotted (0, (3, 1, 1, 1, 1, 1))</code>	

Linestyle	Description
<code>'-' OR 'solid'</code>	solid line
<code>'--' OR 'dashed'</code>	dashed line
<code>'-.-' OR 'dashdot'</code>	dash-dotted line
<code>:</code> OR <code>'dotted'</code>	dotted line
<code>'None'</code> or <code>' '</code> or <code>''</code>	draw nothing

Uproszczzone

(offset, (on_off_seq))

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

↑
line
↑
czerwony
↑
zielony
↑
niebieski

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

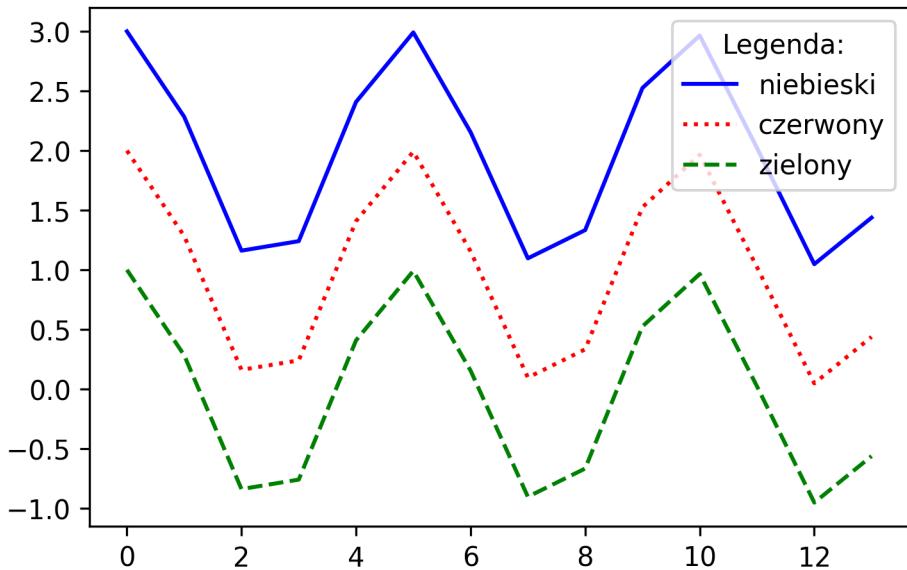
fig = plt.figure()
ax = fig.add_subplot(111)

x = np.arange(14)
y = np.cos(5 * x)

ax.plot(x, y + 2, 'blue', linestyle="--", label="niebieski")
ax.plot(x, y + 1, 'red', linestyle=":", label="czerwony")
ax.plot(x, y, 'green', linestyle="--", label="zielony")

ax.legend(title='Legenda:')

plt.show()
```



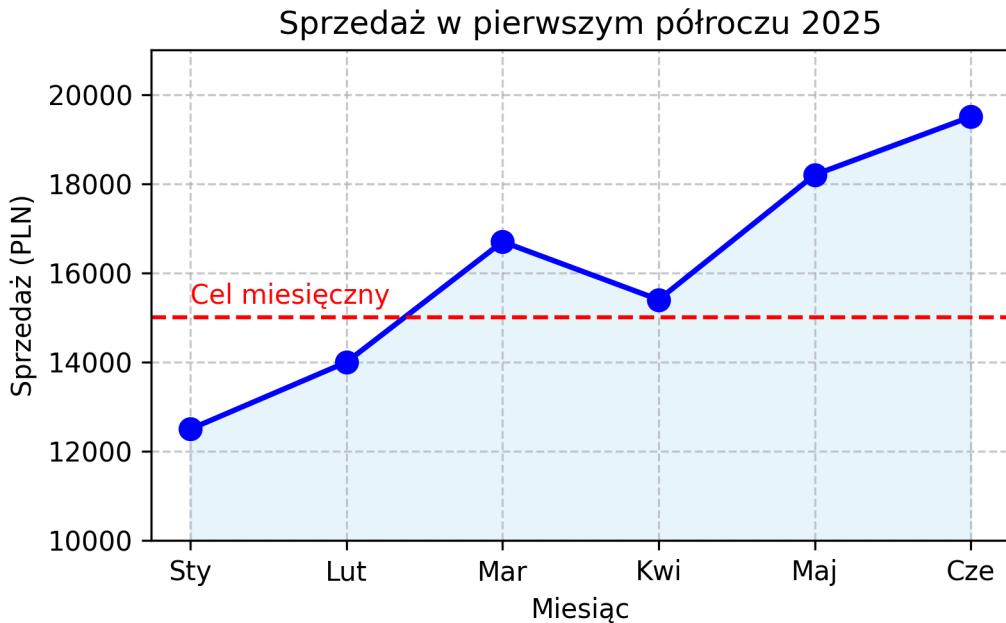
```

import matplotlib.pyplot as plt
import numpy as np

miesiace = ['Sty', 'Lut', 'Mar', 'Kwi', 'Maj', 'Cze']
sprzedaz = [12500, 14000, 16700, 15400, 18200, 19500]

plt.plot(miesiace, sprzedaz, 'bo-', linewidth=2, markersize=8)
plt.grid(True, linestyle='--', alpha=0.7)
plt.title('Sprzedaż w pierwszym półroczu 2025')
plt.xlabel('Miesiąc')
plt.ylabel('Sprzedaż (PLN)')
plt.ylim([10000, 21000])
plt.fill_between(miesiace, sprzedaz, 10000, alpha=0.2, color='skyblue')
plt.axhline(y=15000, color='red', linestyle='--')
plt.text(0, 15300, 'Cel miesięczny', color='red')
plt.tight_layout()
plt.show()

```



```

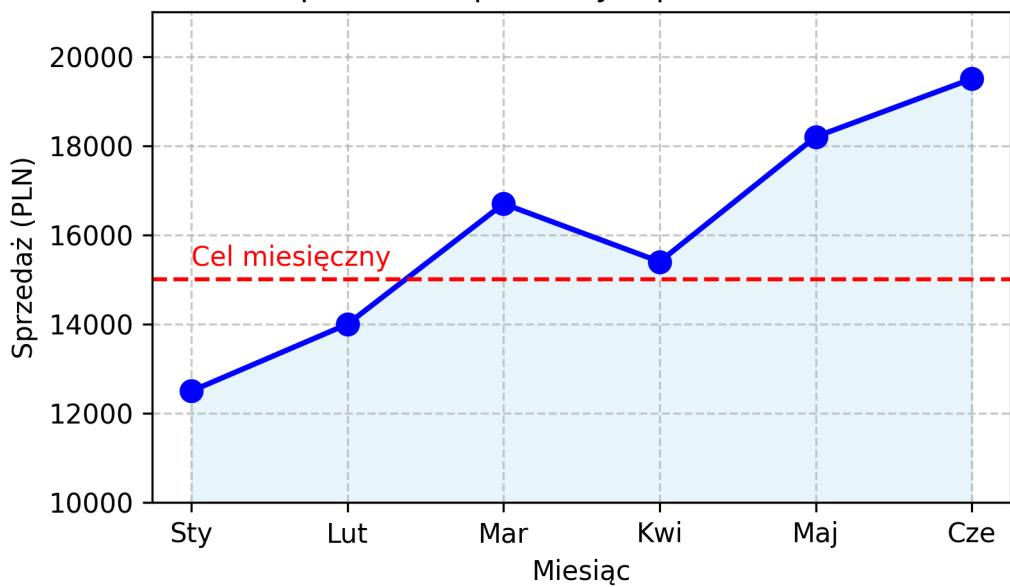
import matplotlib.pyplot as plt
import numpy as np

miesiace = ['Sty', 'Lut', 'Mar', 'Kwi', 'Maj', 'Cze']
sprzedaz = [12500, 14000, 16700, 15400, 18200, 19500]

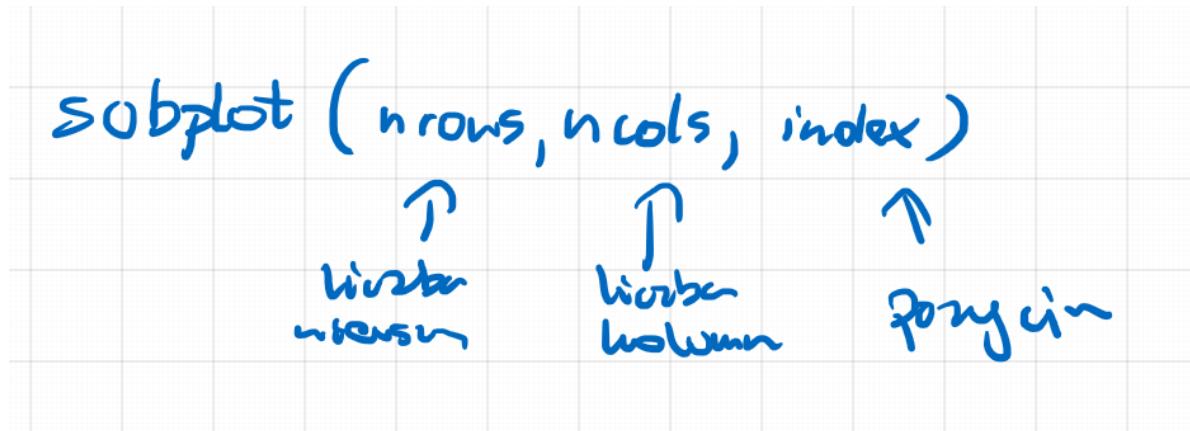
fig, ax = plt.subplots()
ax.plot(miesiace, sprzedaz, 'bo-', linewidth=2, markersize=8)
ax.grid(True, linestyle='--', alpha=0.7)
ax.set_title('Sprzedaż w pierwszym półroczu 2025')
ax.set_xlabel('Miesiąc')
ax.set_ylabel('Sprzedaż (PLN)')
ax.set_ylim([10000, 21000])
ax.fill_between(miesiace, sprzedaz, 10000, alpha=0.2, color='skyblue')
ax.axhline(y=15000, color='red', linestyle='--')
ax.text(0, 15300, 'Cel miesięczny', color='red')
plt.tight_layout()
plt.show()

```

Sprzedaż w pierwszym półroczu 2025



28 Matplotlib - 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)')

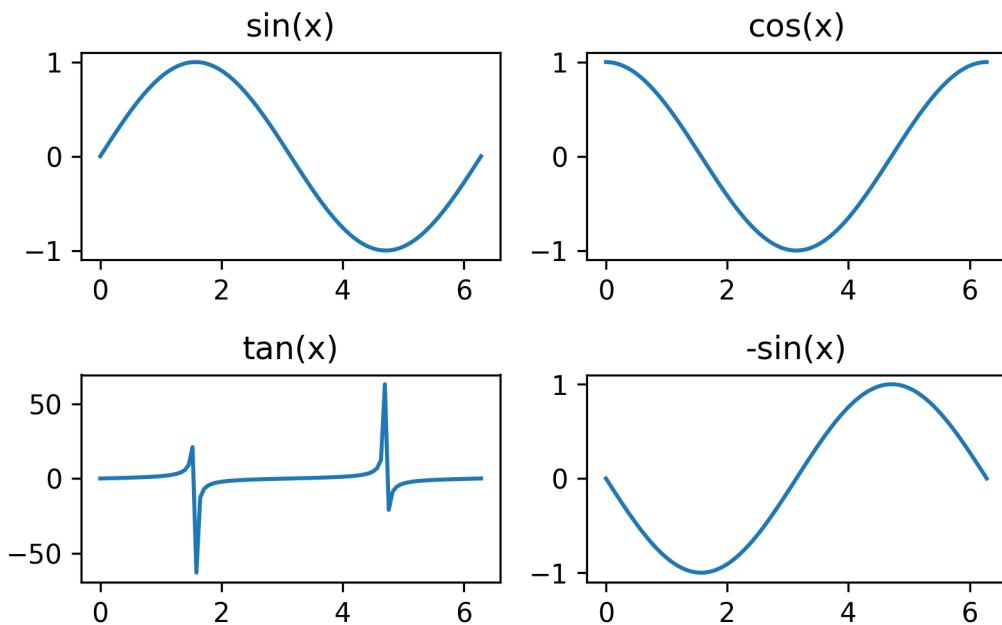
# Trzeci wykres (w lewym dolnym rogu)
plt.subplot(2, 2, 3)
plt.plot(x, np.tan(x))
plt.title('tan(x)')

# 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()

```

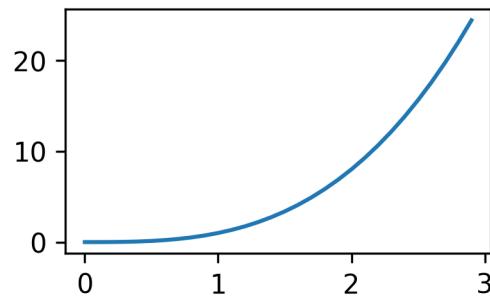
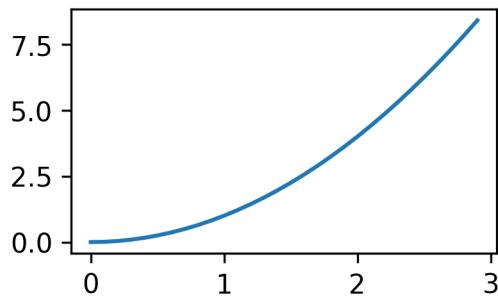
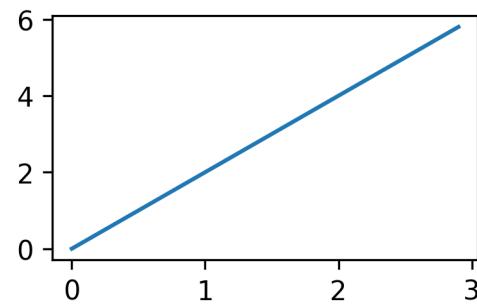
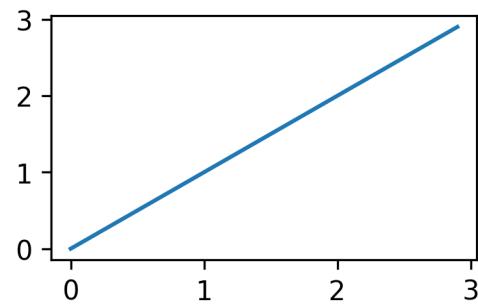


```

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()

```



```

import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 3, 0.1)

fig, axes = plt.subplots(2, 2)
axes[0, 0].plot(x, x)

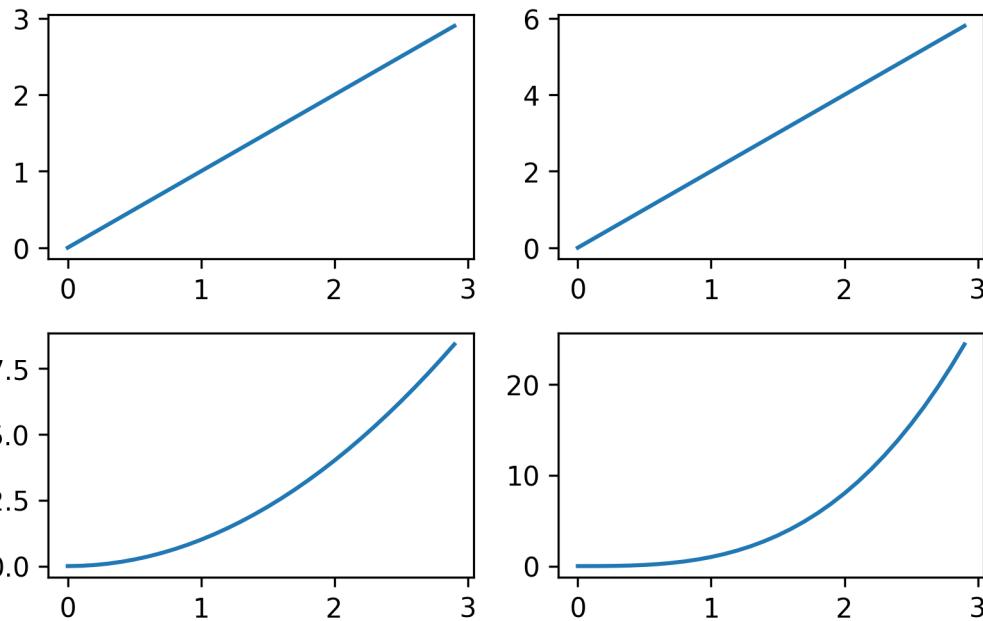
```

```

axes[0, 1].plot(x, x * 2)
axes[1, 0].plot(x, x * x)
axes[1, 1].plot(x, x ** 3)

fig.tight_layout()
plt.show()

```



```

import numpy as np
import matplotlib.pyplot as plt

# Przygotowanie danych do wizualizacji
x = np.linspace(0, 10, 100) # Tworzymy 100 punktów w zakresie od 0 do 10
y1 = np.sin(x) # Funkcja sinus
y2 = np.cos(x) # Funkcja cosinus
y3 = np.exp(-0.2*x) * np.sin(x) # Funkcja tłumionego sinusa
y4 = x**2 / 20 # Funkcja kwadratowa

# Tworzenie figury i podwykresów (2 rzędy, 2 kolumny)
# W podejściu proceduralnym używamy pyplot bezpośrednio
plt.figure(figsize=(12, 8)) # Ustawiamy rozmiar całej figury (szerokość, wysokość w calach)

# Pierwszy podwykres (lewy górnny)
plt.subplot(2, 2, 1) # 2 rzędy, 2 kolumny, pozycja 1

```

```

plt.plot(x, y1, 'r-', linewidth=2) # Czerwona linia
plt.title('Funkcja sinus')
plt.xlabel('X')
plt.ylabel('sin(x)')
plt.grid(True) # Dodajemy siatkę
plt.axhline(y=0, color='k', linestyle='--', alpha=0.3) # Linia pozioma na y=0

# Drugi podwykres (prawy górnny)
plt.subplot(2, 2, 2) # 2 rzędy, 2 kolumny, pozycja 2
plt.plot(x, y2, 'b-', linewidth=2) # Niebieska linia
plt.title('Funkcja cosinus')
plt.xlabel('X')
plt.ylabel('cos(x)')
plt.grid(True)
plt.axhline(y=0, color='k', linestyle='--', alpha=0.3)

# Trzeci podwykres (lewy dolny)
plt.subplot(2, 2, 3) # 2 rzędy, 2 kolumny, pozycja 3
plt.plot(x, y3, 'g-', linewidth=2) # Zielona linia
plt.title('Tłumiony sinus')
plt.xlabel('X')
plt.ylabel('exp(-0.2x) * sin(x)')
plt.grid(True)
plt.axhline(y=0, color='k', linestyle='--', alpha=0.3)

# Czwarty podwykres (prawy dolny)
plt.subplot(2, 2, 4) # 2 rzędy, 2 kolumny, pozycja 4
plt.plot(x, y4, 'orange', linewidth=2, linestyle='--', marker='o', markevery=10, markersize=10)
plt.title('Funkcja kwadratowa')
plt.xlabel('X')
plt.ylabel('x2/20')
plt.grid(True)

# Dodanie adnotacji na ostatnim wykresie
plt.annotate('Punkt (5, 1.25)', xy=(5, 1.25), xytext=(6, 1.8),
             arrowprops=dict(facecolor='black', shrink=0.05, width=1.5))

# Dostosowanie układu wykresów - zapobiega nakładaniu się
plt.tight_layout()

# Dodanie ogólnego tytułu dla całej figury
plt.suptitle('Przykład wykresów z podwykresami - podejście proceduralne', fontsize=16)

```

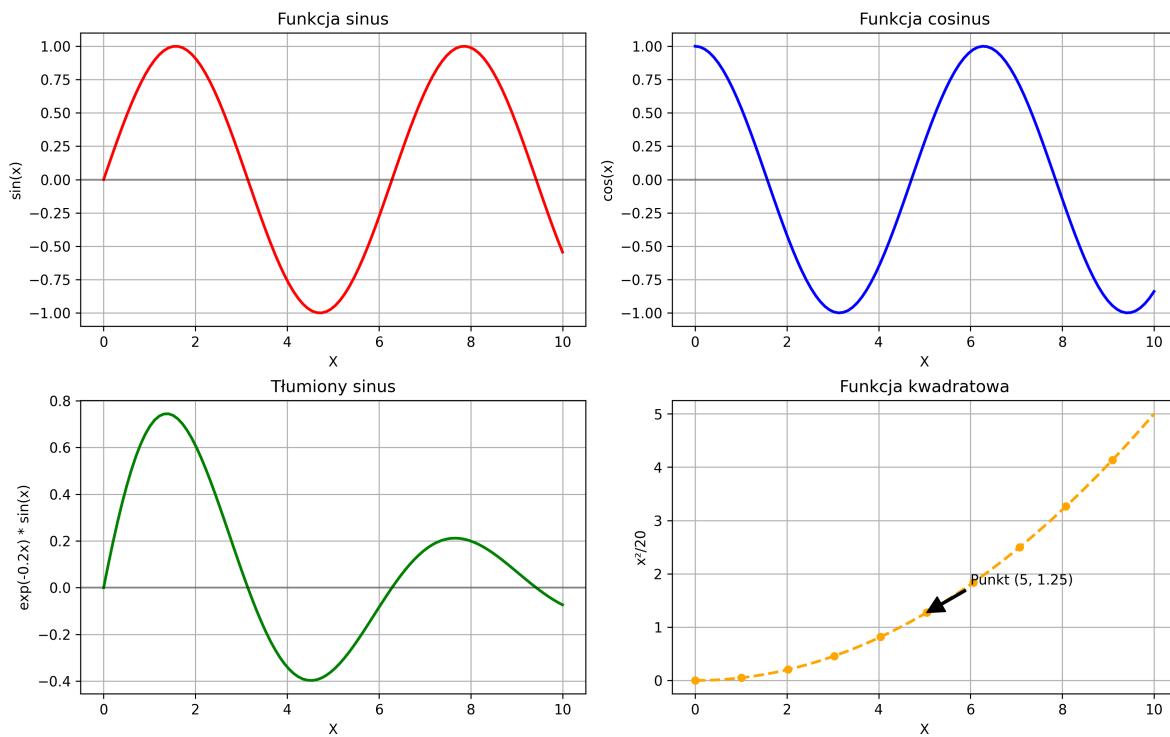
```

plt.subplots_adjust(top=0.9) # Dodanie miejsca na główny tytuł

# Wyświetlenie wykresu
plt.show()

```

Przykład wykresów z podwykresami - podejście proceduralne



29 Matplotlib - zapis

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

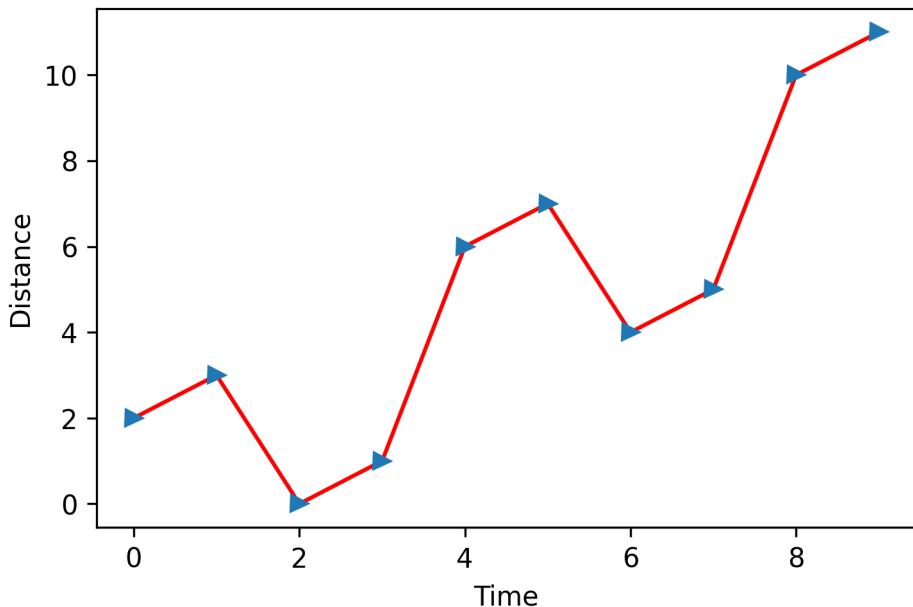
x = np.arange(0, 10)
y = x ** 2
# Labeling the Axes and Title
plt.title("Graph Drawing")
plt.xlabel("Time")
plt.ylabel("Distance")

# 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')
```

Graph Drawing



ZAPIS DO PLIKU:

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

↑
wzór pliku
(jedna strona)

↑
rozszczepienie
(dwie strony)

WAŻNE!

→ DO ZAPISU JPC POTRZĘBNA
BIBLIOTEKA PILLOW

→ SAVEFIG POWINIEN BYĆ UWIDOWANY
PRZED SHOW!

30 Matplotlib - wykres 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 długiego publicznego.

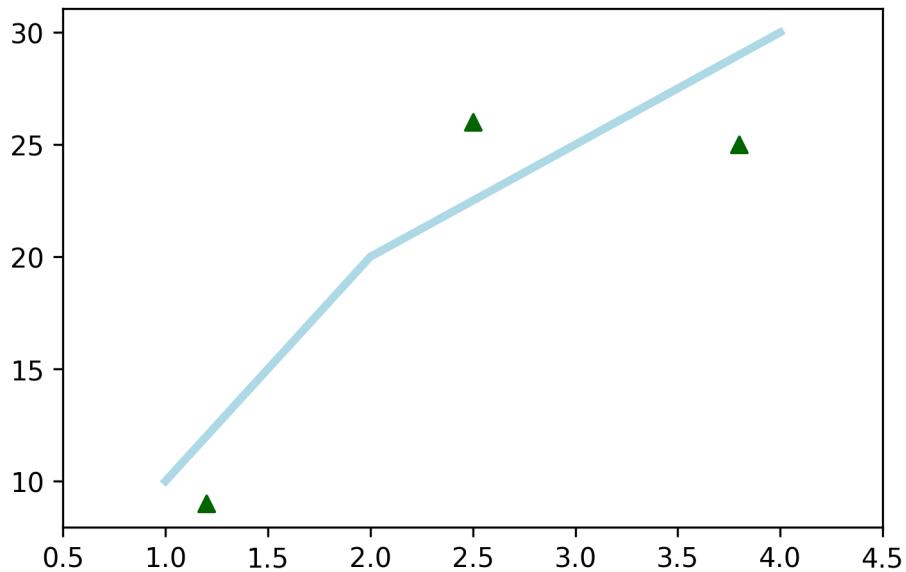
Wykresy punktowe są szczególnie przydatne, gdy mamy do czynienia z danymi o różnym charakterze (ciągłe lub dyskretne) oraz gdy chcemy zbadać korelację, grupy, wzorce lub wartości odstające.

```
import matplotlib.pyplot as plt

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(block=True)
```

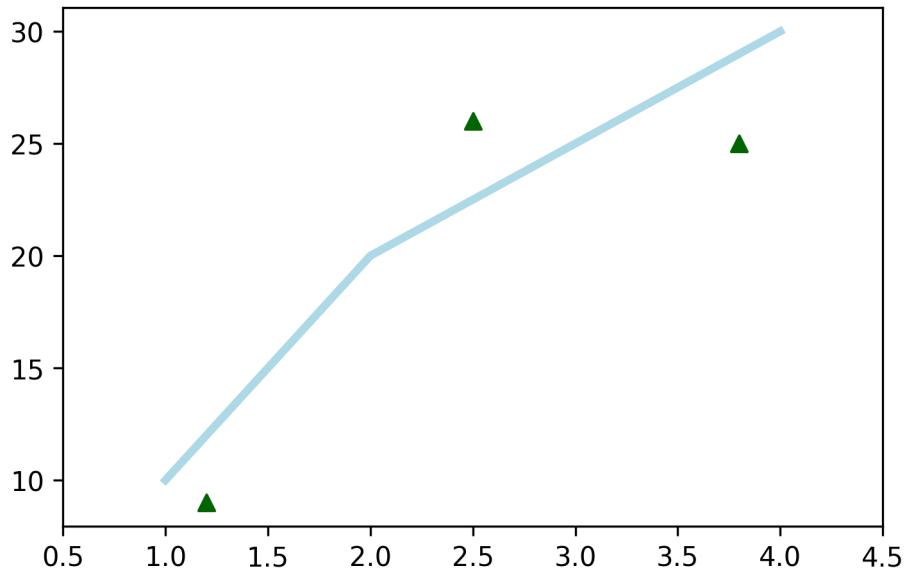
- ① `plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3)` - Tworzy wykres liniowy z podanymi współrzędnymi punktów (1, 10), (2, 20), (3, 25) i (4, 30). Kolor linii to jasnoniebieski (lightblue), a jej grubość wynosi 3.
- ② `plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', 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 góre (^).

③ plt.xlim(0.5, 4.5) - Ustala zakres wartości na osi X, zaczynając od 0.5 do 4.5.



```
import matplotlib.pyplot as plt

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

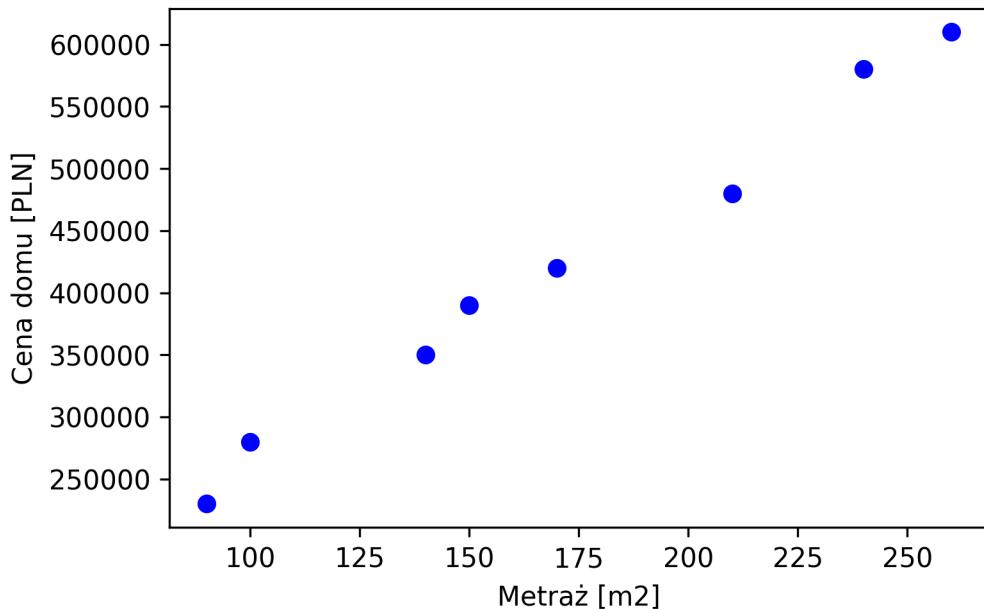


```
import matplotlib.pyplot as plt

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')           ①
plt.xlabel('Metraż [m2]')
plt.ylabel('Cena domu [PLN]')
plt.title('Związek między metrażem a ceną domu')
plt.show(block=True)
```

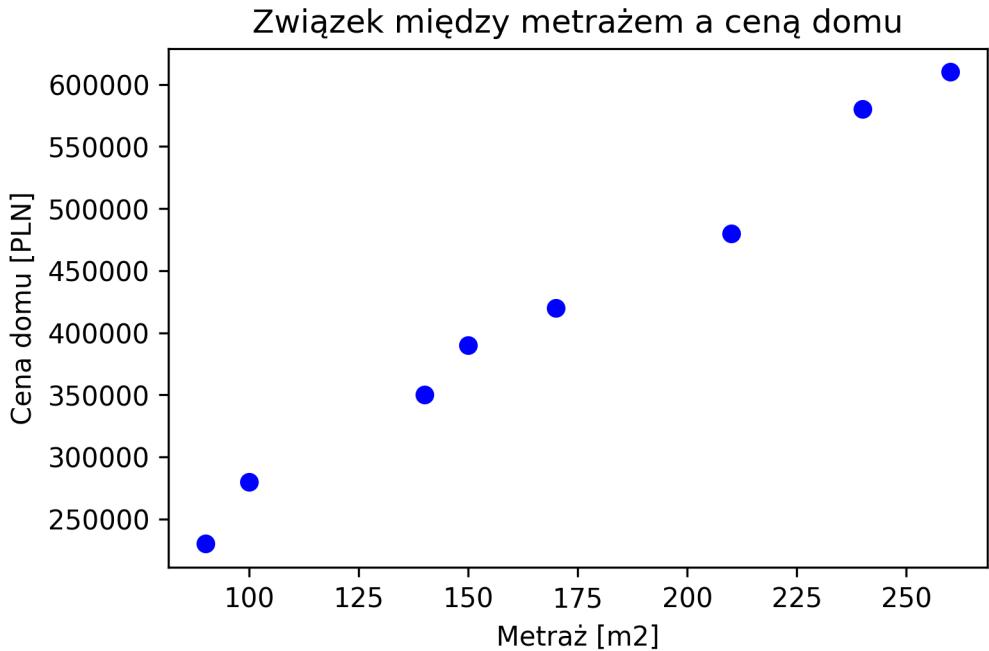
① `plt.scatter(square_meters, house_prices, color='blue', marker='o'):` tworzy wykres punktowy (scatter plot) z metrażem domów na osi X (`square_meters`) i cenami domów na osi Y (`house_prices`). Punkty są koloru niebieskiego (`color='blue'`) i mają kształt kółka (`marker='o'`).

Związek między metrażem a ceną domu



```
import matplotlib.pyplot as plt

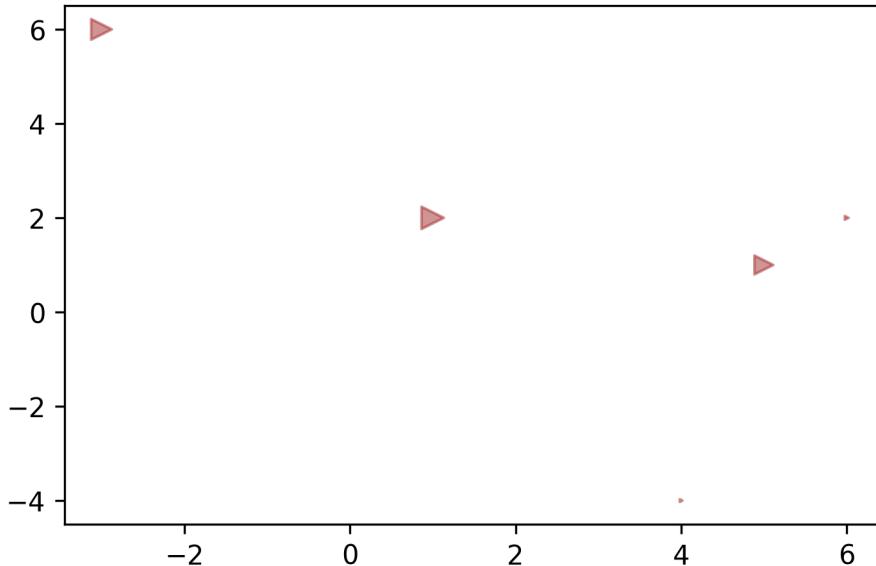
fig, ax = plt.subplots()
house_prices = [230000, 350000, 480000, 280000, 420000, 610000, 390000, 580000]
square_meters = [90, 140, 210, 100, 170, 260, 150, 240]
ax.scatter(square_meters, house_prices, color='blue', marker='o')
ax.set_xlabel('Metraż [m2]')
ax.set_ylabel('Cena domu [PLN]')
ax.set_title('Związek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



```
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) ①
plt.show(block=True)
```

- 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 pandas as pd
import matplotlib.pyplot as plt

data = {
    'product_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'price': [19.99, 29.99, 14.99, 49.99, 9.99, 39.99, 24.99, 34.99, 44.99, 15.99],
    'units_sold': [150, 85, 200, 50, 300, 75, 120, 95, 60, 180],
    'rating': [4.5, 4.2, 4.8, 3.9, 4.6, 4.1, 4.3, 4.0, 3.8, 4.7],
    'discount': [0.1, 0.05, 0.15, 0.0, 0.2, 0.1, 0.05, 0.08, 0.0, 0.12]
}

df = pd.DataFrame(data)
x_values = df['price']
y_values = df['units_sold']
colors = df['rating']
sizes = df['discount'] * 1000 + 50

plt.scatter(x_values, y_values, s=sizes, c=colors, alpha=0.7, cmap='viridis') ⑦

plt.colorbar(label='Ocena produktu') ⑧

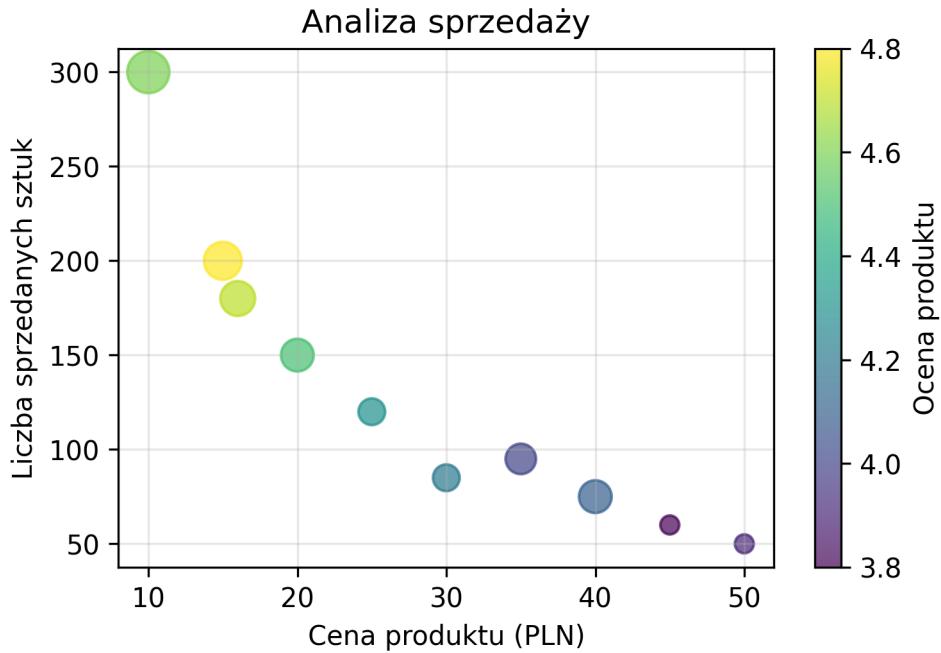
plt.xlabel('Cena produktu (PLN)') ⑨
plt.ylabel('Liczba sprzedanych sztuk') ⑩
plt.title('Analiza sprzedaży') ⑪

```

```
plt.grid(True, alpha=0.3) (12)
```

```
plt.show() (13)
```

- ① `data = {...}`: tworzy słownik zawierający dane o produktach, gdzie każdy klucz odpowiada nazwie kolumny, a wartości to listy zawierające dane dla 10 produktów (id produktu, cena, liczba sprzedanych sztuk, ocena i rabat).
- ② `df = pd.DataFrame(data)`: konwertuje słownik `data` na DataFrame biblioteki pandas, co tworzy uporządkowaną tabelę danych z nazwanymi kolumnami.
- ③ `x_values = df['price']`: wyodrębnia kolumnę z cenami produktów i przypisuje ją do zmiennej `x_values`, która będzie użyta jako współrzędne X na wykresie.
- ④ `y_values = df['units_sold']`: wyodrębnia kolumnę z liczbą sprzedanych sztuk i przypisuje ją do zmiennej `y_values`, która będzie użyta jako współrzędne Y na wykresie.
- ⑤ `colors = df['rating']`: wyodrębnia kolumnę z ocenami produktów i przypisuje ją do zmiennej `colors`, która posłuży do nadania kolorów punktom na wykresie.
- ⑥ `sizes = df['discount'] * 1000 + 50`: tworzy zmienną `sizes` określającą rozmiar punktów przez przemnożenie wartości rabatu przez 1000 i dodanie 50, co zapewnia widoczność wszystkich punktów (minimalna wielkość 50 jednostek).
- ⑦ `plt.scatter(x_values, y_values, s=sizes, c=colors, alpha=0.7, cmap='viridis')`: tworzy wykres punktowy (scatter plot) używając cen jako współrzędnych X, sprzedaży jako Y, wielkości punktów określonych przez rabaty, kolorów opartych na ocenach, z przezroczystością 0.7 i paletą kolorów ‘viridis’.
- ⑧ `plt.colorbar(label='Ocena produktu')`: dodaje pasek kolorów (colorbar) do wykresu, który pokazuje skalę ocen produktów i ich odpowiadające kolory.
- ⑨ `plt.xlabel('Cena produktu (PLN)')`: ustawia etykietę osi X, opisującą że pokazuje cenę produktu w złotych.
- ⑩ `plt.ylabel('Liczba sprzedanych sztuk')`: ustawia etykietę osi Y, opisującą że pokazuje liczbę sprzedanych sztuk produktu.
- ⑪ `plt.title('Analiza sprzedaży')`: nadaje wykresowi tytuł “Analiza sprzedaży”.
- ⑫ `plt.grid(True, alpha=0.3)`: włącza siatkę na wykresie z przezroczystością 0.3, co poprawia czytelność danych.
- ⑬ `plt.show()`: wyświetla przygotowany wykres.



```

import pandas as pd
import matplotlib.pyplot as plt

data = {
    'product_id': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'price': [19.99, 29.99, 14.99, 49.99, 9.99, 39.99, 24.99, 34.99, 44.99, 15.99],
    'units_sold': [150, 85, 200, 50, 300, 75, 120, 95, 60, 180],
    'rating': [4.5, 4.2, 4.8, 3.9, 4.6, 4.1, 4.3, 4.0, 3.8, 4.7],
    'discount': [0.1, 0.05, 0.15, 0.0, 0.2, 0.1, 0.05, 0.08, 0.0, 0.12]
}

df = pd.DataFrame(data)
x_values = df['price']
y_values = df['units_sold']
colors = df['rating']
sizes = df['discount'] * 1000 + 50

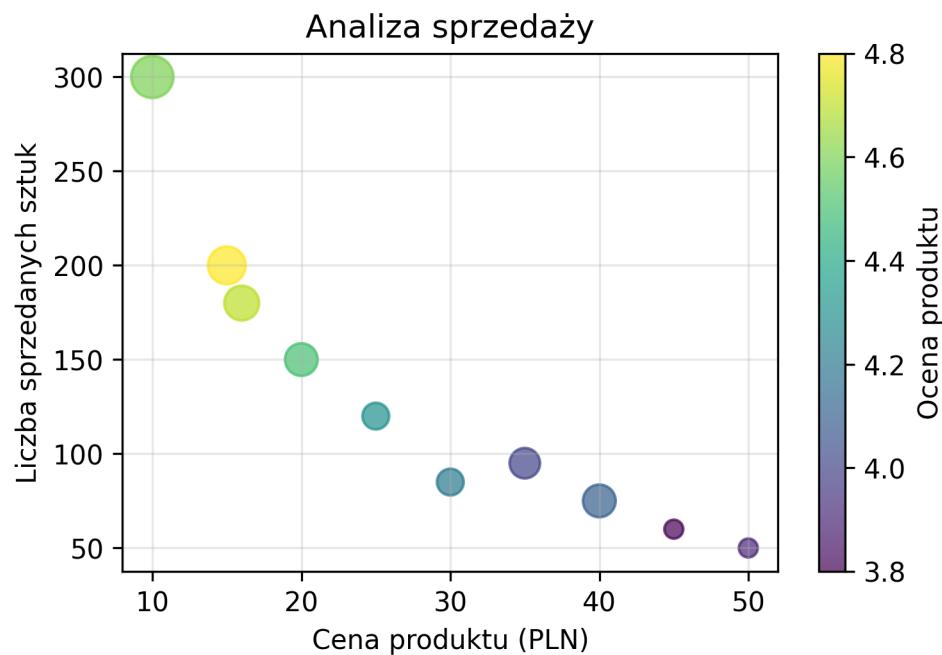
fig, ax = plt.subplots()
scatter = ax.scatter(x_values, y_values, s=sizes, c=colors, alpha=0.7, cmap='viridis')
cbar = fig.colorbar(scatter, ax=ax)
cbar.set_label('Ocena produktu')

ax.set_xlabel('Cena produktu (PLN)')

```

```
ax.set_ylabel('Liczba sprzedanych sztuk')
ax.set_title('Analiza sprzedaży')
ax.grid(True, alpha=0.3)

plt.show()
```



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

marker	symbol	description
"."	●	point
·	pixel	
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	⌄	tri_down
"2"	⌂	tri_up
"3"	⌄	tri_left
"4"	⌅	tri_right
"8"	●	octagon

"s"		square
"p"		pentagon
"P"		plus (filled)
"*"		star
"h"		hexagon1
"H"		hexagon2
"+"		plus
"x"		x
"X"		x (filled)
"D"		diamond
"d"		thin_diamond
" "		vline
"_"		hline

0	(TICKLEFT)	-	tickleft
1	(TICKRIGHT)	-	tickright
2	(TICKUP)		tickup
3	(TICKDOWN)		tickdown
4	(CARETLEFT)	◀	caretleft
5	(CARETRIGHT)	▶	caretright
6	(CARETUP)	▲	caretup
7	(CARETDOWN)	▼	caretdown
8	(CARETLEFTBASE)	◀	caretleft (centered at base)
9	(CARETRIGHTBASE)	▶	caretright (centered at base)
10	(CARETUPBASE)	▲	caretup (centered at base)
11	(CARETDOWNBASE)	▼	caretdown (centered at base)
"none"	or	"None"	nothing
" "	or	".."	nothing
"\$...\$"	f		Render the string using mathtext. E.g. "\$f\$" for marker showing the letter f.

31 Matplotlib - 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.).

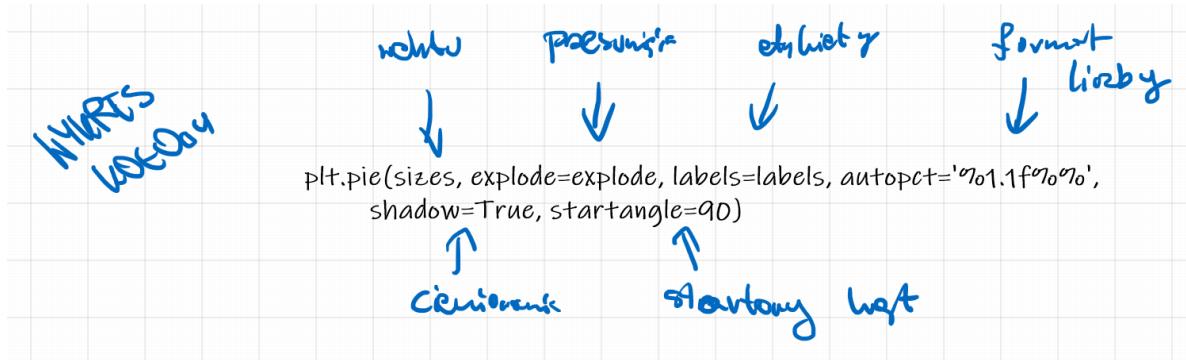
Chociaż wykresy kołowe mają swoje zastosowania, są również krytykowane za ograniczoną precyzję w ocenie proporcji. Dlatego często zaleca się stosowanie innych rodzajów wykresów, takich jak słupkowe (bar chart) czy stosunkowe (stacked bar chart), które mogą być bardziej przejrzyste i precyzyjne w porównywaniu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

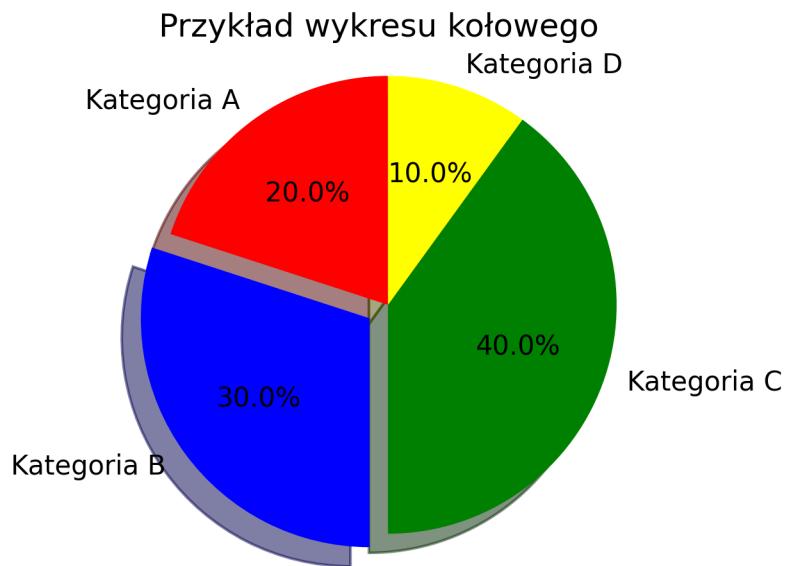
# Dane
sizes = [20, 30, 40, 10]
labels = ['Kategoria A', 'Kategoria B', 'Kategoria C', 'Kategoria D']
colors = ['red', 'blue', 'green', 'yellow']
explode = (0, 0.1, 0, 0) # Wyróżnienie segmentu Kategoria B

# Tworzenie wykresu kołowego
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%.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()
    
```

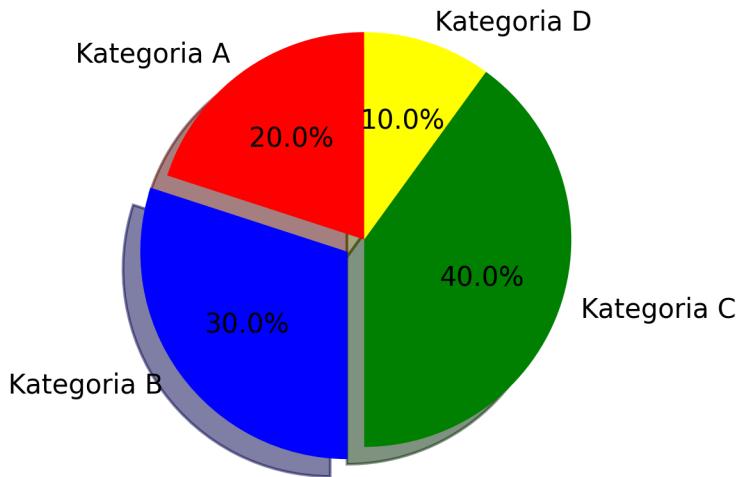


```
import matplotlib.pyplot as plt

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)

fig, ax = plt.subplots()
ax.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%.1f%%', shadow=True,
       ax.set_title('Przykład wykresu kołowego')
       ax.set_aspect('equal')
       plt.show()
```

Przykład wykresu kołowego

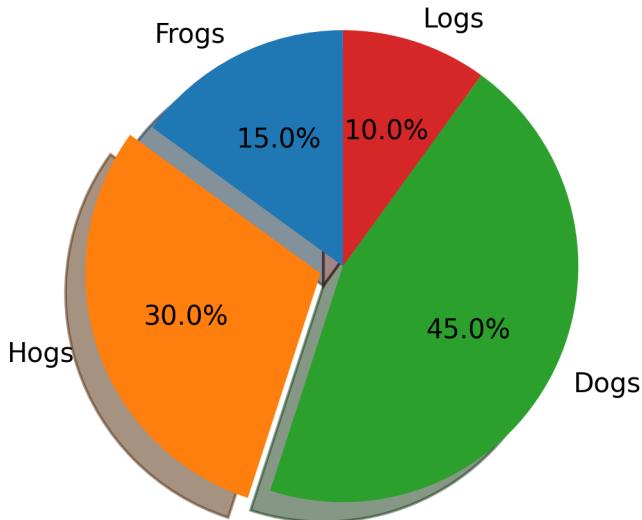


```
import matplotlib.pyplot as plt

# 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')

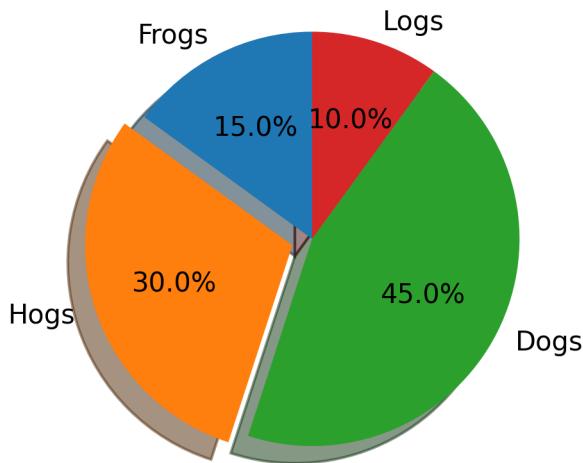
plt.show()
```



```
import matplotlib.pyplot as plt

labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
explode = [0, 0.1, 0, 0]

fig, ax = plt.subplots()
ax.pie(sizes, explode=explode, labels=labels, autopct='%.1f%%', shadow=True, startangle=90)
ax.set_aspect('equal')
plt.show()
```



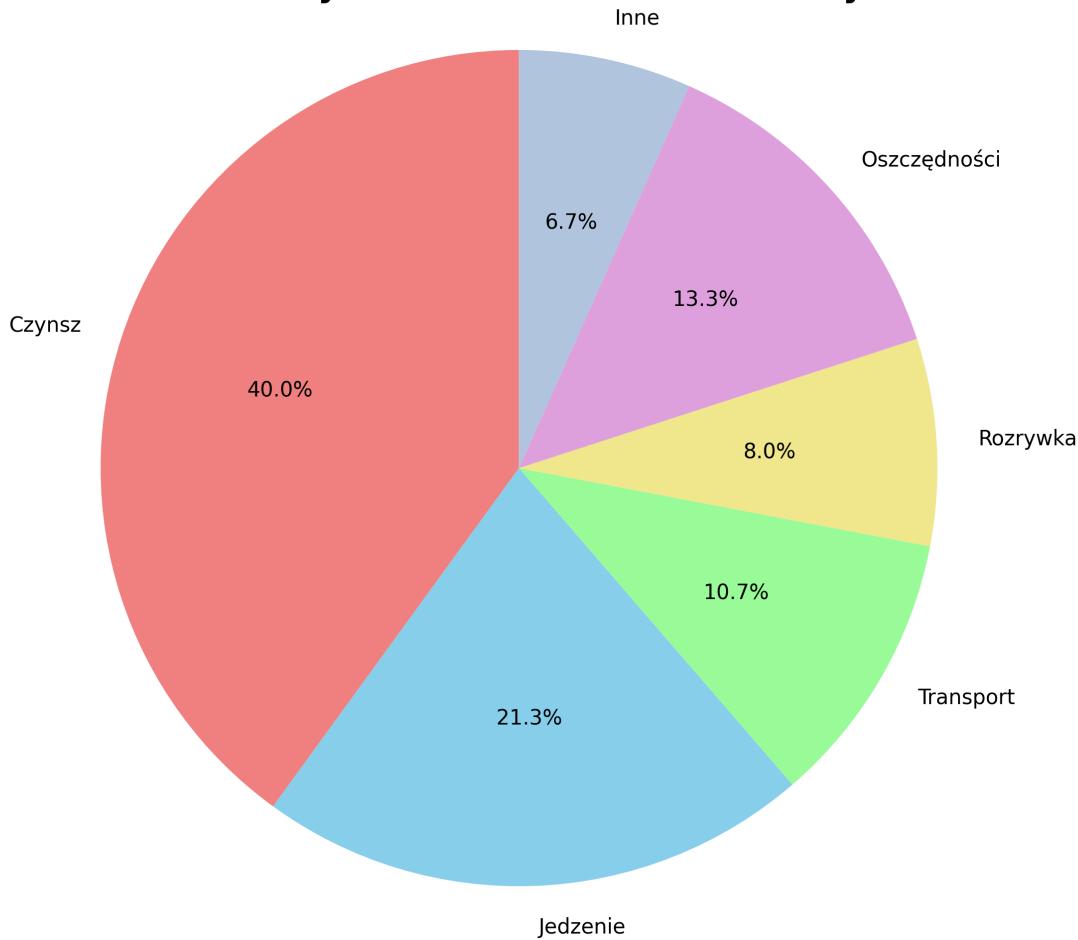
```

import matplotlib.pyplot as plt          ①
kategorie = ['Czysz', 'Jedzenie', 'Transport', 'Rozrywka', 'Oszczędności', 'Inne'] ②
wydatki = [1500, 800, 400, 300, 500, 250]           ③
kolory = ['lightcoral', 'skyblue', 'palegreen', 'khaki', 'plum', 'lightsteelblue'] ④
plt.figure(figsize=(10, 8))            ⑤
plt.pie(wydatki,
        labels=kategorie,
        colors=kolory,
        autopct='%.1f%%',
        startangle=90)                  ⑥
plt.title('Podział wydatków w budżecie domowym', fontsize=16, fontweight='bold') ⑦
plt.axis('equal')                     ⑧
plt.show()                           ⑨

```

- ① `import matplotlib.pyplot as plt`: importuje bibliotekę matplotlib.pyplot pod skróconą nazwą plt, co pozwala na wygodne użycie funkcji do tworzenia wykresów.
- ② `kategorie = ['Czysz', 'Jedzenie', 'Transport', 'Rozrywka', 'Oszczędności', 'Inne']`: tworzy listę stringów przedstawiających kategorie wydatków budżetu domowego.
- ③ `wydatki = [1500, 800, 400, 300, 500, 250]`: tworzy listę liczb reprezentujących kwoty wydatków (w złotówkach) dla każdej odpowiadającej kategorii.
- ④ `kolory = ['lightcoral', 'skyblue', 'palegreen', 'khaki', 'plum', 'lightsteelblue']`: tworzy listę nazw kolorów, które będą użyte dla każdego wycinka wykresu kołowego.
- ⑤ `plt.figure(figsize=(10, 8))`: tworzy nową figurę (obszar rysowania) o wymiarach 10 cali szerokości na 8 cali wysokości.
- ⑥ `plt.pie(wydatki, labels=kategorie, colors=kolory, autopct='%.1f%%', startangle=90)`: rysuje wykres kołowy z wartościami z listy `wydatki`, etykietami z listy `kategorie`, kolorami z listy `kolory`. Parametr `autopct='%.1f%%'` formatuje wyświetlane wartości procentowe z dokładnością do jednego miejsca po przecinku i dodaje znak procentu. Parametr `startangle=90` określa, że wykres rozpocznie się od kąta 90 stopni (góra).
- ⑦ `plt.title('Podział wydatków w budżecie domowym', fontsize=16, fontweight='bold')`: dodaje tytuł do wykresu z rozmiarem czcionki 16 i pogrubionym tekstem.
- ⑧ `plt.axis('equal')`: ustawia równe proporcje osi X i Y, co zapewnia, że wykres kołowy będzie idealnie okrągły, a nie eliptyczny.
- ⑨ `plt.show()`: wyświetla stworzony wykres w oknie graficznym.

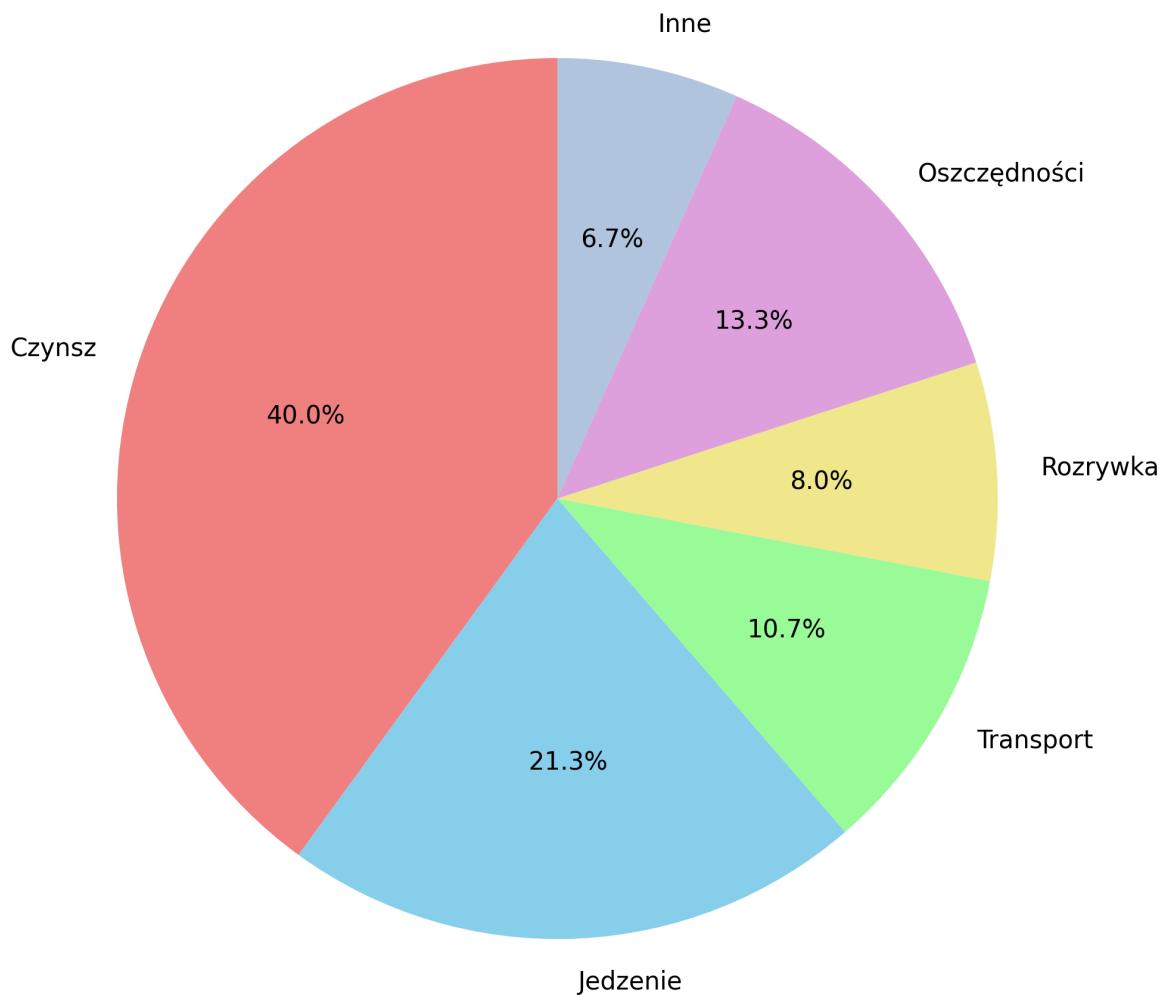
Podział wydatków w budżecie domowym



```
import matplotlib.pyplot as plt
kategorie = ['Czyszcz', 'Jedzenie', 'Transport', 'Rozrywka', 'Oszczędności', 'Inne']
wydatki = [1500, 800, 400, 300, 500, 250]
kolory = ['lightcoral', 'skyblue', 'palegreen', 'khaki', 'plum', 'lightsteelblue']

fig, ax = plt.subplots(figsize=(10, 8))
ax.pie(wydatki,
       labels=kategorie,
       colors=kolory,
       autopct='%.1f%%',
       startangle=90)
ax.set_title('Podział wydatków w budżecie domowym', fontsize=16, fontweight='bold')
ax.set_aspect('equal')
plt.show()
```

Podział wydatków w budżecie domowym



Zamiana na mapę kolorów:

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm # Dodajemy import modułu cm (color maps)

kategorie = ['Czyszcz', 'Jedzenie', 'Transport', 'Rozrywka', 'Oszczędności', 'Inne']
wydatki = [1500, 800, 400, 300, 500, 250]

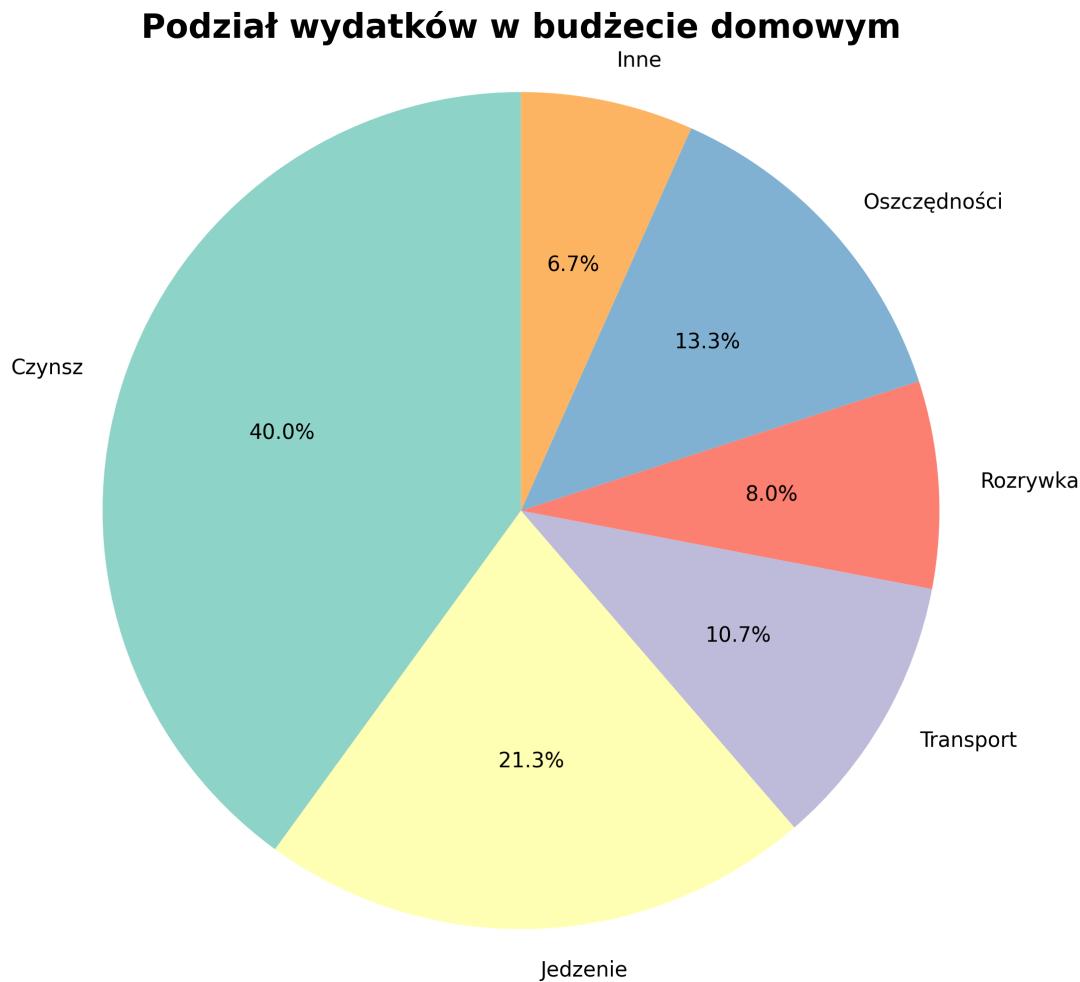
# Używamy jakościowej mapy kolorów 'Set3'
```

```

cmap = plt.colormaps['Set3']
kolory = [cmap(i) for i in range(len(kategorie))]

plt.figure(figsize=(10, 8))
plt.pie(wydatki, labels=kategorie, colors=kolory, autopct='%.1f%%', startangle=90)
plt.title('Podział wydatków w budżecie domowym', fontsize=16, fontweight='bold')
plt.axis('equal')
plt.show()

```



```

import matplotlib.pyplot as plt
import matplotlib.cm as cm

kategorie = ['Czyszcz', 'Jedzenie', 'Transport', 'Rozrywka', 'Oszczędności', 'Inne']

```

```

wydatki = [1500, 800, 400, 300, 500, 250]

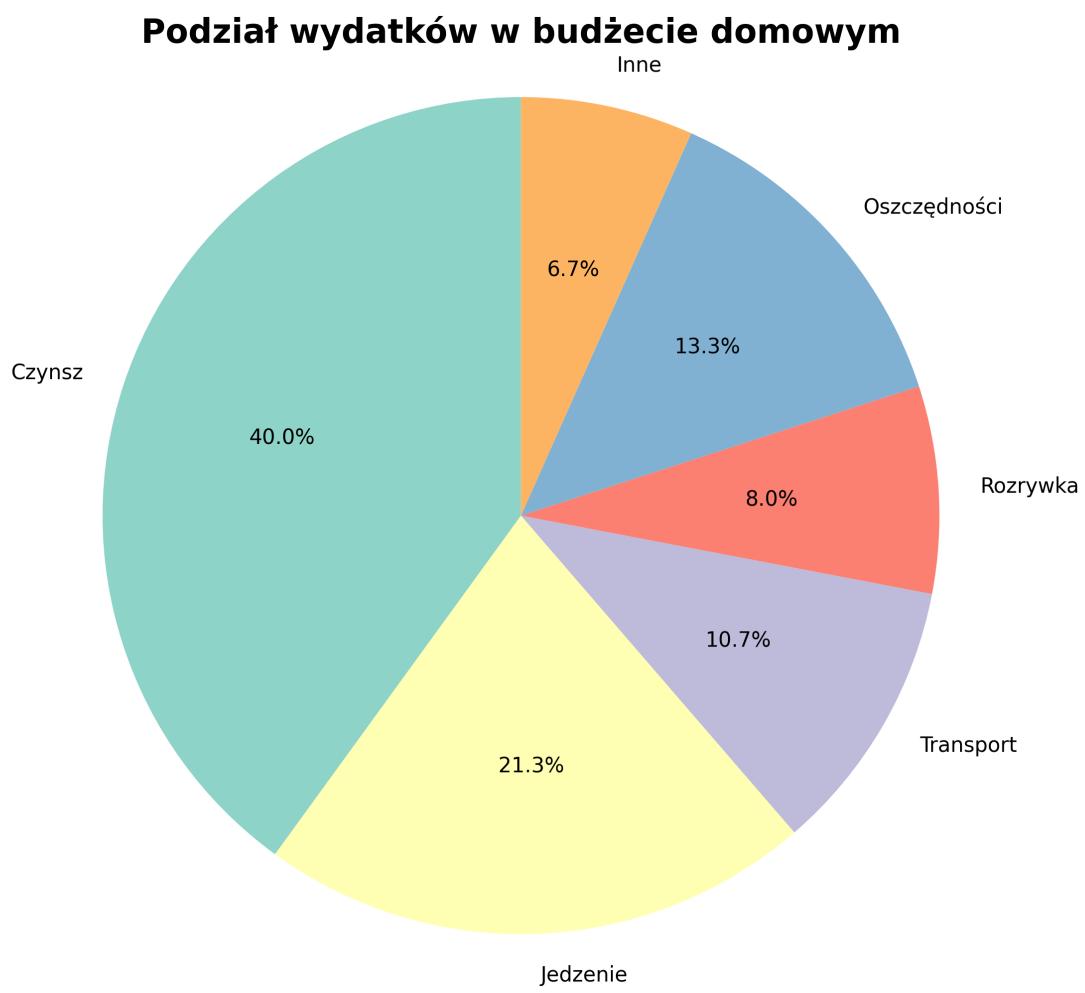
cmap = plt.colormaps['Set3']
kolory = [cmap(i) for i in range(len(kategorie))]

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

ax.pie(wydatki, labels=kategorie, colors=kolory, autopct='%.1f%%', startangle=90)
ax.set_title('Podział wydatków w budżecie domowym', fontsize=16, fontweight='bold')
ax.axis('equal')

plt.show()

```

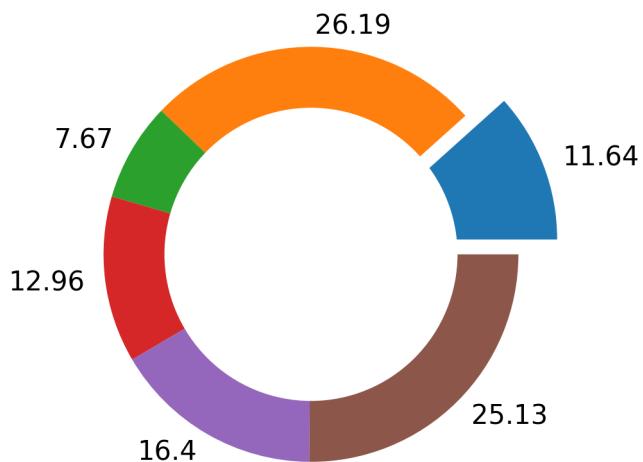


31.1 Wykres pierścieniowy

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

np.random.seed(345)                                     ①
data = np.random.randint(20, 100, 6)                     ②
total = sum(data)
data_per = data / total * 100                           ④
explode = (0.2, 0, 0, 0, 0, 0)                         ⑤
plt.pie(data_per, explode=explode, labels=[round(i, 2) for i in list(data_per)]) ⑥
circle = plt.Circle((0, 0), 0.7, color='white')          ⑦
p = plt.gcf()
p.gca().add_artist(circle)
plt.show()                                              ⑩
```

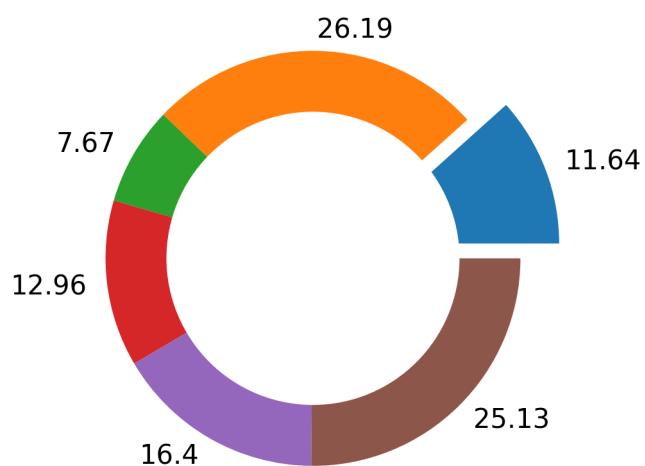
- ① `np.random.seed(345)`: ustawia ziarno generatora liczb losowych na wartość 345, co zapewnia powtarzalność wyników.
- ② `data = np.random.randint(20, 100, 6)`: generuje tablicę z 6 losowymi liczbami całkowitymi w zakresie od 20 do 99 (włącznie).
- ③ `total = sum(data)`: oblicza sumę wszystkich wygenerowanych liczb.
- ④ `data_per = data / total * 100`: oblicza wartości procentowe każdej liczby względem sumy całkowitej.
- ⑤ `explode = (0.2, 0, 0, 0, 0, 0)`: tworzy krotkę określającą wysunięcie wycinka dla każdego elementu (pierwszy wycinek będzie wysunięty o 0.2).
- ⑥ `plt.pie(data_per, explode=explode, labels=[round(i, 2) for i in list(data_per)])`: tworzy wykres kołowy z wartościami procentowymi, z określonym wysunięciem i etykietami zaokrąglonymi do 2 miejsc po przecinku.
- ⑦ `circle = plt.Circle((0, 0), 0.7, color='white')`: tworzy białe koło o środku w punkcie (0, 0) i promieniu 0.7.
- ⑧ `p = plt.gcf()`: pobiera aktualne obiekty figury (get current figure).
- ⑨ `p.gca().add_artist(circle)`: dodaje utworzone białe koło do aktualnych osi wykresu, tworząc efekt “donut chart”.
- ⑩ `plt.show()`: wyświetla wykres.



```
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)

fig, ax = plt.subplots()
ax.pie(data_per, explode=explode, labels=[round(i, 2) for i in list(data_per)])
circle = plt.Circle((0, 0), 0.7, color='white')
ax.add_artist(circle)
plt.show()
```



32 Matplotlib - wykres słupkowy

Wykres słupkowy jest stosowany do przedstawiania danych kategorialnych 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 szeregow czasowych: Wykres słupkowy może być również używany do przedstawiania danych szeregow czasowych w przypadku, gdy zmiany występują w regularnych odstępach czasu, np. roczna sprzedaż, miesięczne opady czy tygodniowe przychody.

Warto zauważyc, 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 wzduż 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          ①

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

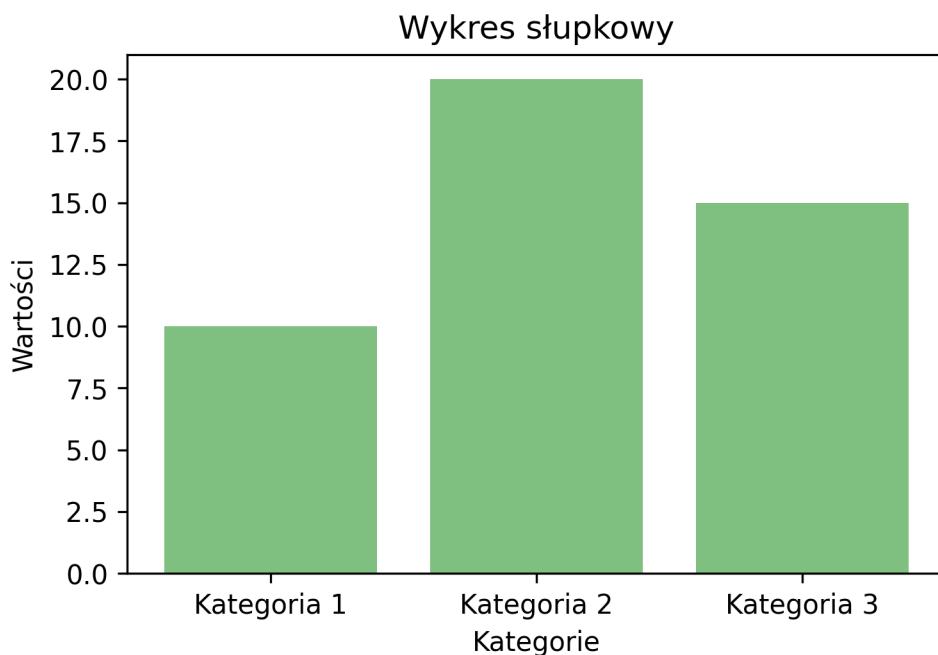
plt.bar(kategorie, wartosci, color='green', alpha=0.5)        ④

plt.title('Wykres słupkowy')           ⑤
plt.xlabel('Kategorie')                ⑥
plt.ylabel('Wartości')                 ⑦

plt.show()                           ⑧

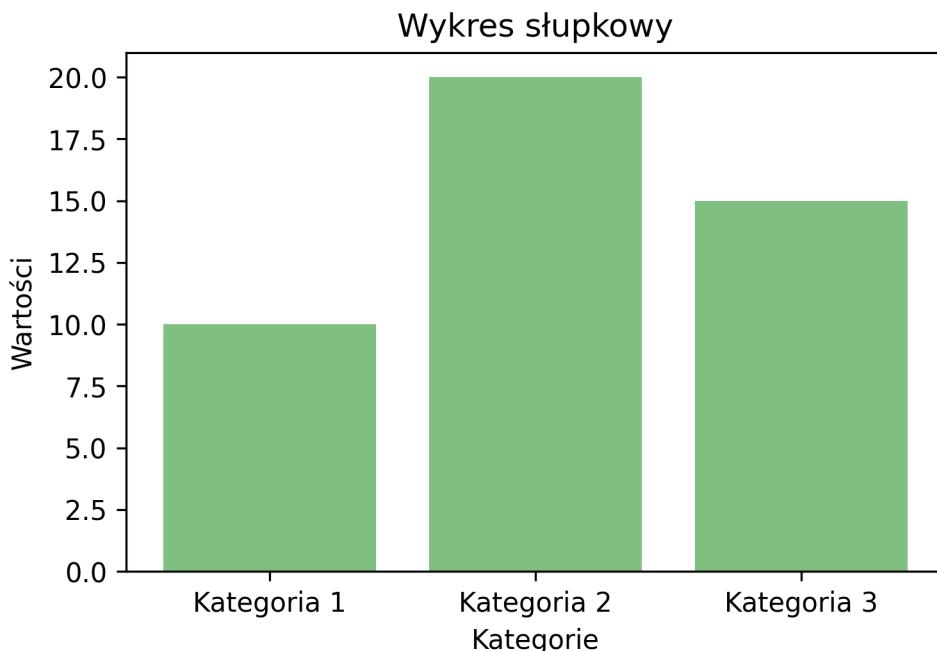
```

- ① `import matplotlib.pyplot as plt`: importuje bibliotekę matplotlib.pyplot jako `plt`, która jest potrzebna do tworzenia wykresów.
- ② `kategorie = ['Kategoria 1', 'Kategoria 2', 'Kategoria 3']`: tworzy listę `kategorie` zawierającą nazwy trzech kategorii, które będą użyte na osi X wykresu słupkowego.
- ③ `wartosci = [10, 20, 15]`: tworzy listę `wartosci` zawierającą wartości liczbowe odpowiadające każdej kategorii - te wartości określają wysokość poszczególnych słupków.
- ④ `plt.bar(kategorie, wartosci, color='green', alpha=0.5)`: tworzy wykres słupkowy, gdzie:
- ⑤ `plt.title('Wykres słupkowy')`: nadaje tytuł wykresu "Wykres słupkowy".
- ⑥ `plt.xlabel('Kategorie')`: dodaje etykietę osi X z tekstem "Kategorie".
- ⑦ `plt.ylabel('Wartości')`: dodaje etykietę osi Y z tekstem "Wartości".
- ⑧ `plt.show()`: wyświetla przygotowany wykres.



```
import matplotlib.pyplot as plt

kategorie = ['Kategoria 1', 'Kategoria 2', 'Kategoria 3']
wartosci = [10, 20, 15]
fig, ax = plt.subplots()
ax.bar(kategorie, wartosci, color='green', alpha=0.5)
ax.set_title('Wykres słupkowy')
ax.set_xlabel('Kategorie')
ax.set_ylabel('Wartości')
plt.show()
```



```

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

dane_sprzedazy = {
    'produkt': ['Smartfon', 'Tablet', 'Laptop', 'Słuchawki', 'Smartwatch'],
    'sprzedane_sztuki': [156, 89, 234, 312, 178],
    'kategoria': ['telefony', 'tablety', 'komputery', 'akcesoria', 'zegarek']
}

df = pd.DataFrame(dane_sprzedazy)

y_pos = np.arange(len(df))

kolory = {'telefony': 'blue', 'tablety': 'red', 'komputery': 'green',
          'akcesoria': 'purple', 'zegarek': 'orange'}

colors_list = [kolory[kat] for kat in df['kategoria']]

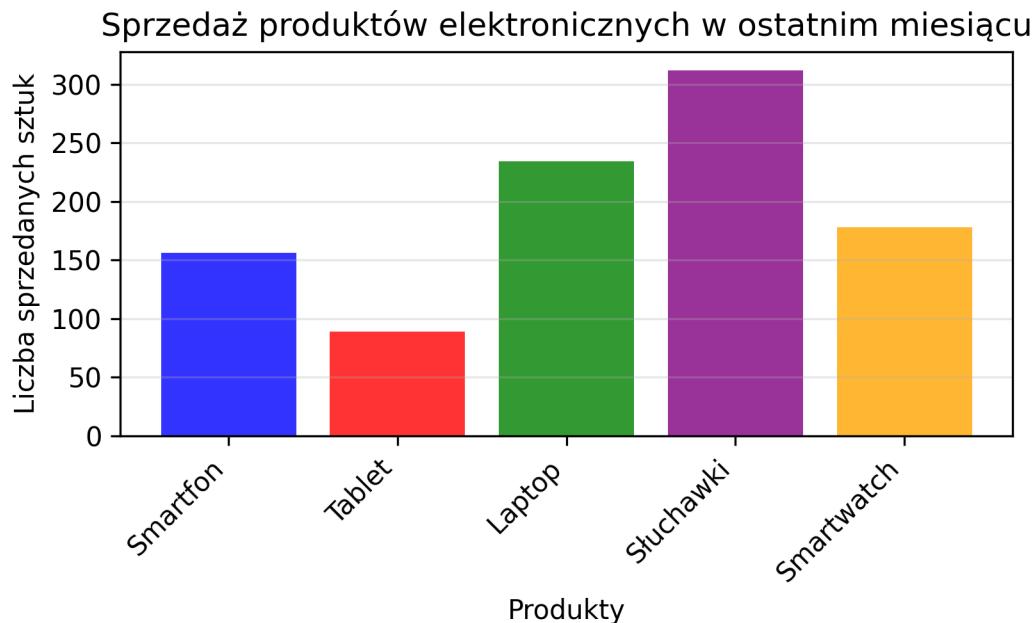
plt.bar(y_pos, df['sprzedane_sztuki'], color=colors_list, alpha=0.8)
plt.xticks(y_pos, df['produkt'], rotation=45, ha='right')
plt.xlabel('Produkty')
plt.ylabel('Liczba sprzedanych sztuk')

```

```

plt.title('Sprzedaż produktów elektronicznych w ostatnim miesiącu')
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

```



```

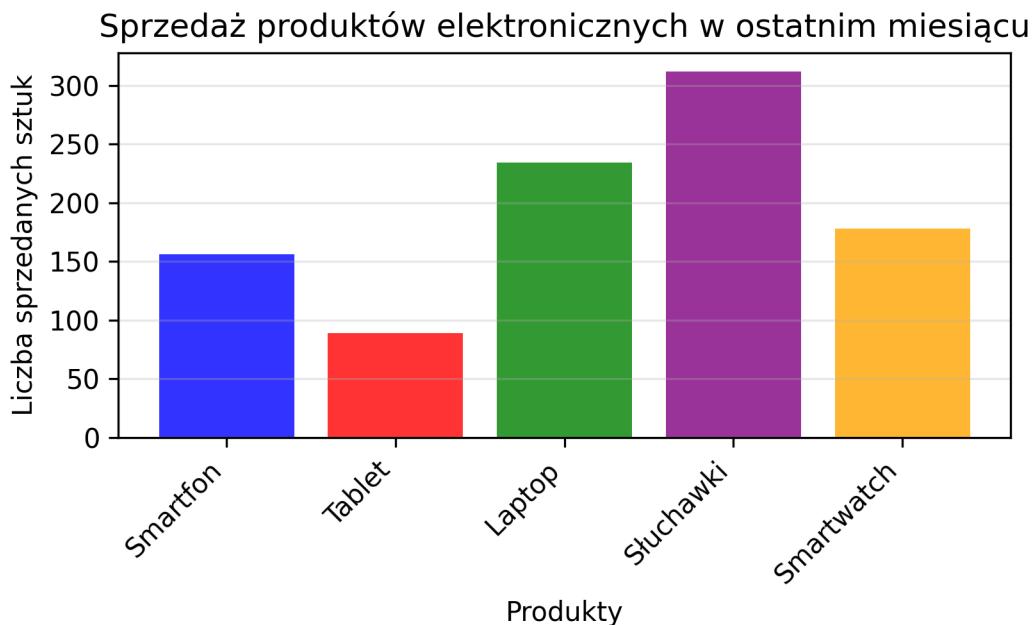
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dane_sprzedazy = {
    'produkt': ['Smartfon', 'Tablet', 'Laptop', 'Słuchawki', 'Smartwatch'],
    'sprzedane_sztuki': [156, 89, 234, 312, 178],
    'kategoria': ['telefony', 'tablety', 'komputery', 'akcesoria', 'zegarek']
}
df = pd.DataFrame(dane_sprzedazy)
y_pos = np.arange(len(df))
kolory = {'telefony': 'blue', 'tablety': 'red', 'komputery': 'green',
          'akcesoria': 'purple', 'zegarek': 'orange'}
colors_list = [kolory[kat] for kat in df['kategoria']]
fig, ax = plt.subplots()
ax.bar(y_pos, df['sprzedane_sztuki'], color=colors_list, alpha=0.8)
ax.set_xticks(y_pos)
ax.set_xticklabels(df['produkt'], rotation=45, ha='right')
ax.set_xlabel('Produkty')

```

```

ax.set_ylabel('Liczba sprzedanych sztuk')
ax.set_title('Sprzedaż produktów elektronicznych w ostatnim miesiącu')
ax.grid(axis='y', alpha=0.3)
fig.tight_layout()
plt.show()

```



```

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

df = pd.DataFrame({
    'Kwartał': ['Q1', 'Q2', 'Q3', 'Q4'],
    'Telefony': [30, 25, 50, 20],
    'Laptopy': [40, 23, 51, 17],
    'Tablety': [35, 22, 45, 19]
})

telefony = df['Telefony']
laptopy = df['Laptopy']
tablety = df['Tablety']
kwartaly = df['Kwartał']

X = np.arange(len(kwartaly))
plt.bar(X - 0.25, telefony, color='dodgerblue', width=0.25, label='Telefony') ⑩

```

```

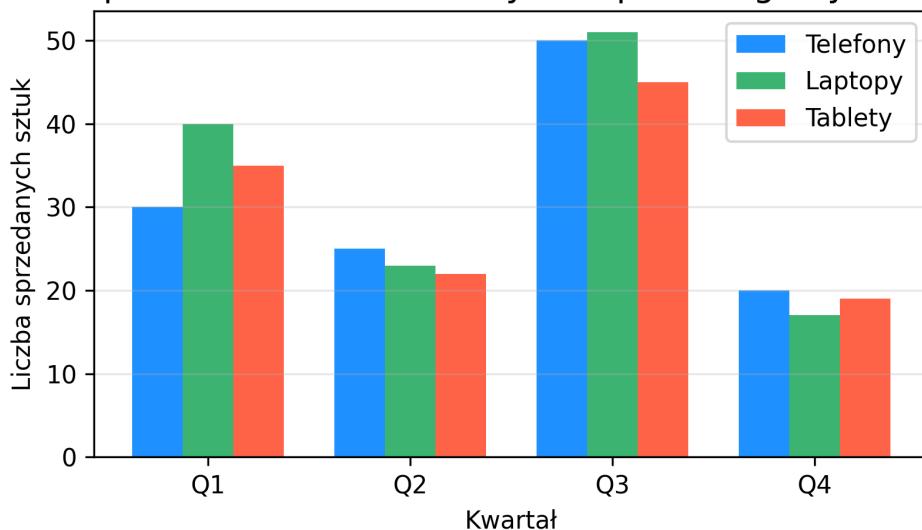
plt.bar(X, laptopy, color='mediumseagreen', width=0.25, label='Laptopy')      (11)
plt.bar(X + 0.25, tablety, color='tomato', width=0.25, label='Tablety')        (12)
plt.xticks(X, kwartaly)                                                       (13)
plt.legend()                                                                (14)
plt.title('Sprzedaż produktów elektronicznych w poszczególnych kwartałach', fontsize=14) (15)
plt.ylabel('Liczba sprzedanych sztuk')                                         (16)
plt.xlabel('Kwartał')                                                       (17)
plt.grid(axis='y', alpha=0.3)                                                 (18)
plt.tight_layout()                                                       (19)
plt.show()                                                               (20)

```

- ① `import numpy as np`: importuje bibliotekę NumPy do operacji numerycznych i przypisuje jej alias `np`.
- ② `import matplotlib.pyplot as plt`: importuje moduł pyplot z biblioteki Matplotlib do tworzenia wykresów i przypisuje mu alias `plt`.
- ③ `import pandas as pd`: importuje bibliotekę Pandas do pracy z danymi i przypisuje jej alias `pd`.
- ④ `df = pd.DataFrame({...})`: tworzy DataFrame z danymi o sprzedaży produktów elektronicznych w poszczególnych kwartałach.
- ⑤ `telefony = df['Telefony']`: wyodrębnia kolumnę 'Telefony' z DataFrame i przypisuje ją do zmiennej `telefony`.
- ⑥ `laptopy = df['Laptopy']`: wyodrębnia kolumnę 'Laptopy' z DataFrame i przypisuje ją do zmiennej `laptopy`.
- ⑦ `tablety = df['Tablety']`: wyodrębnia kolumnę 'Tablety' z DataFrame i przypisuje ją do zmiennej `tablety`.
- ⑧ `kwartaly = df['Kwartał']`: wyodrębnia kolumnę 'Kwartał' z DataFrame i przypisuje ją do zmiennej `kwartaly`.
- ⑨ `X = np.arange(len(kwartaly))`: tworzy tablicę indeksów od 0 do długości listy kwartałów (0, 1, 2, 3), która posłuży do pozycjonowania słupków.
- ⑩ `plt.bar(X - 0.25, telefony, color='dodgerblue', width=0.25, label='Telefony')`: rysuje słupki dla telefonów z przesunięciem w lewo (-0.25), kolorem niebieskim i szerokością 0.25.
- ⑪ `plt.bar(X, laptopy, color='mediumseagreen', width=0.25, label='Laptopy')`: rysuje słupki dla laptopów w środku (bez przesunięcia), kolorem zielonym i szerokością 0.25.
- ⑫ `plt.bar(X + 0.25, tablety, color='tomato', width=0.25, label='Tablety')`: rysuje słupki dla tabletów z przesunięciem w prawo (+0.25), kolorem czerwonym i szerokością 0.25.
- ⑬ `plt.xticks(X, kwartaly)`: ustawia etykiety osi X na nazwy kwartałów (Q1, Q2, Q3, Q4) w pozycjach określonych przez tablicę X.
- ⑭ `plt.legend()`: dodaje legendę do wykresu, która pokazuje kolory i nazwy poszczególnych kategorii produktów.

- ⑯ plt.title('Sprzedaż produktów elektronicznych w poszczególnych kwartałach', fontsize=14): ustawia tytuł wykresu z rozmiarem czcionki 14.
- ⑰ plt.ylabel('Liczba sprzedanych sztuk'): dodaje etykietę osi Y opisującą, co reprezentują wartości na osi pionowej.
- ⑱ plt.xlabel('Kwartał'): dodaje etykietę osi X opisującą, co reprezentują wartości na osi poziomej.
- ⑲ plt.grid(axis='y', alpha=0.3): dodaje poziome linie siatki (tylko na osi Y) z przeroczystością 0.3.
- ⑳ plt.tight_layout(): automatycznie dostosowuje marginesy wykresu, aby wszystkie elementy były dobrze widoczne i nie zachodziły na siebie.
- ㉑ plt.show(): wyświetla wykres

Sprzedaż produktów elektronicznych w poszczególnych kwartałach



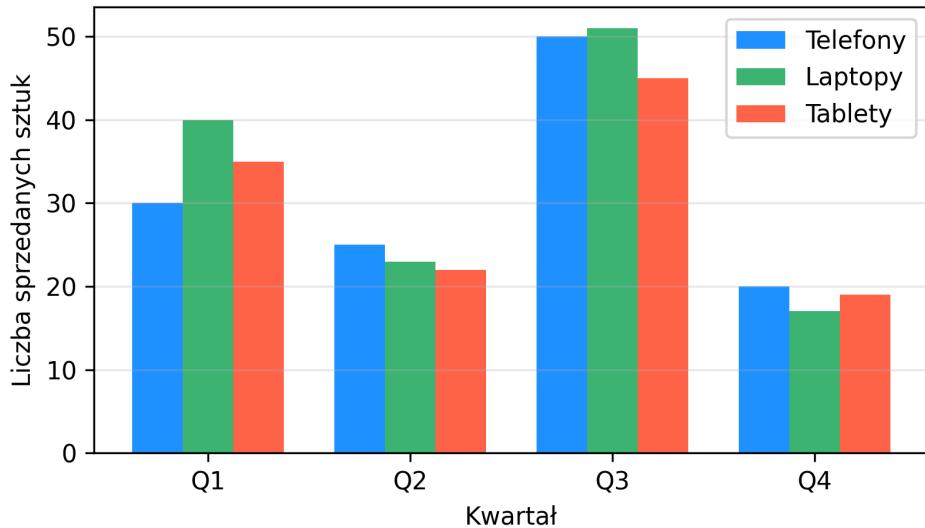
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
df = pd.DataFrame({
    'Kwartał': ['Q1', 'Q2', 'Q3', 'Q4'],
    'Telefony': [30, 25, 50, 20],
    'Laptopy': [40, 23, 51, 17],
    'Tablety': [35, 22, 45, 19]
})
telefony = df['Telefony']
laptopy = df['Laptopy']
tablety = df['Tablety']
kwartaly = df['Kwartał']
```

```

X = np.arange(len(kwartaly))
fig, ax = plt.subplots()
ax.bar(X - 0.25, telefony, color='dodgerblue', width=0.25, label='Telefony')
ax.bar(X, laptopy, color='mediumseagreen', width=0.25, label='Laptopy')
ax.bar(X + 0.25, tablety, color='tomato', width=0.25, label='Tablety')
ax.set_xticks(X)
ax.set_xticklabels(kwartaly)
ax.legend()
ax.set_title('Sprzedaż produktów elektronicznych w poszczególnych kwartałach', fontsize=14)
ax.set_ylabel('Liczba sprzedanych sztuk')
ax.set_xlabel('Kwartał')
ax.grid(axis='y', alpha=0.3)
fig.tight_layout()
plt.show()

```

Sprzedaż produktów elektronicznych w poszczególnych kwartałach



```

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

```

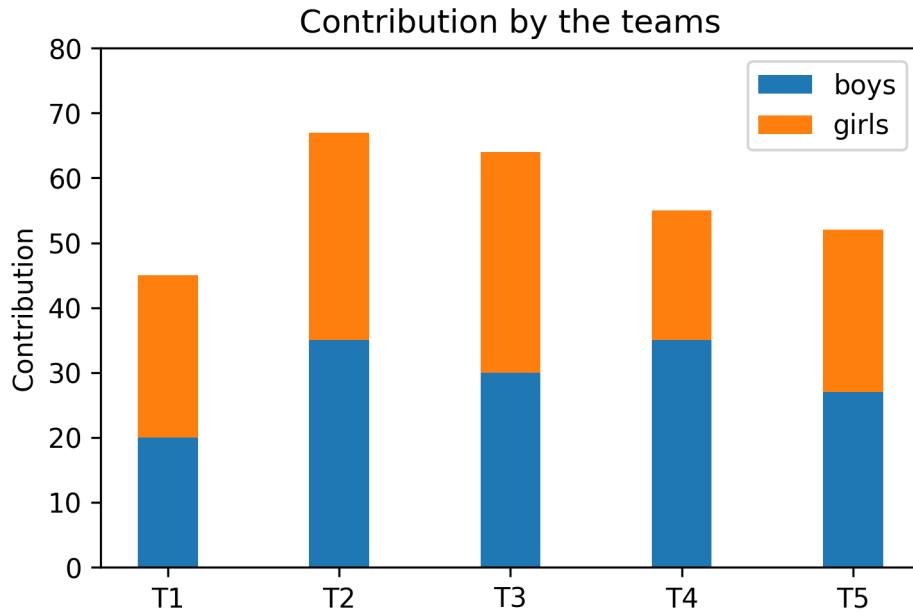
```

plt.bar(ind, boys, width, label="boys")                                     ⑧
plt.bar(ind, girls, width, bottom=boys, label="girls")                       ⑨

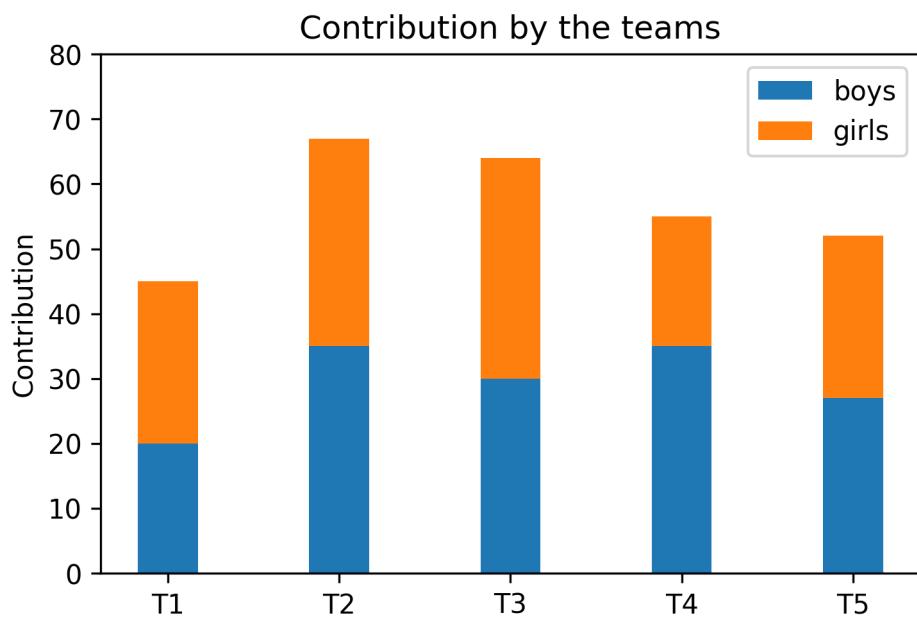
plt.ylabel('Contribution')                                                 ⑩
plt.title('Contribution by the teams')                                      ⑪
plt.xticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))                           ⑫
plt.yticks(np.arange(0, 81, 10))                                             ⑬
plt.legend()                                                               ⑭
plt.show()                                                                ⑮

```

- ① `import numpy as np`: importuje bibliotekę NumPy do operacji numerycznych i przypisuje jej alias `np`.
- ② `import matplotlib.pyplot as plt`: importuje moduł pyplot z biblioteki Matplotlib do tworzenia wykresów i przypisuje mu alias `plt`.
- ③ `N = 5`: definiuje stałą `N` równą 5, która reprezentuje liczbę zespołów/grup danych.
- ④ `boys = (20, 35, 30, 35, 27)`: tworzy krotkę zawierającą wartości wkładu chłopców dla każdego z 5 zespołów.
- ⑤ `girls = (25, 32, 34, 20, 25)`: tworzy krotkę zawierającą wartości wkładu dziewcząt dla każdego z 5 zespołów.
- ⑥ `ind = np.arange(N)`: tworzy tablicę indeksów [0, 1, 2, 3, 4], która określa pozycje słupków na osi X.
- ⑦ `width = 0.35`: ustawia szerokość słupków na 0.35 jednostki.
- ⑧ `plt.bar(ind, boys, width, label="boys")`: rysuje słupki dla chłopców na pozycjach określonych przez `ind`, z wysokościami z `boys`, szerokością `width` i etyktą "boys".
- ⑨ `plt.bar(ind, girls, width, bottom=boys, label="girls")`: rysuje słupki dla dziewcząt na tych samych pozycjach, ale używając parametru `bottom=boys`, który sprawia, że słupki dziewcząt są umieszczone ponad słupkami chłopców, tworząc wykres słupkowy skumulowany.
- ⑩ `plt.ylabel('Contribution')`: dodaje etykietę osi Y opisującą, że wartości reprezentują wkład.
- ⑪ `plt.title('Contribution by the teams')`: nadaje tytuł wykresu "Contribution by the teams".
- ⑫ `plt.xticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))`: ustawia etykiety osi X na nazwy zespołów (T1, T2, T3, T4, T5) w pozycjach określonych przez tablicę `ind`.
- ⑬ `plt.yticks(np.arange(0, 81, 10))`: ustawia znaczniki na osi Y co 10 jednostek, od 0 do 80.
- ⑭ `plt.legend()`: dodaje legendę do wykresu, która pokazuje kolory i etykiety dla chłopców i dziewcząt.
- ⑮ `plt.show()`: wyświetla wykres



```
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
fig, ax = plt.subplots()
ax.bar(ind, boys, width, label="boys")
ax.bar(ind, girls, width, bottom=boys, label="girls")
ax.set_ylabel('Contribution')
ax.set_title('Contribution by the teams')
ax.set_xticks(ind)
ax.set_xticklabels(['T1', 'T2', 'T3', 'T4', 'T5'])
ax.set_yticks(np.arange(0, 81, 10))
ax.legend()
plt.show()
```



WYKRES STUPKOWY

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.

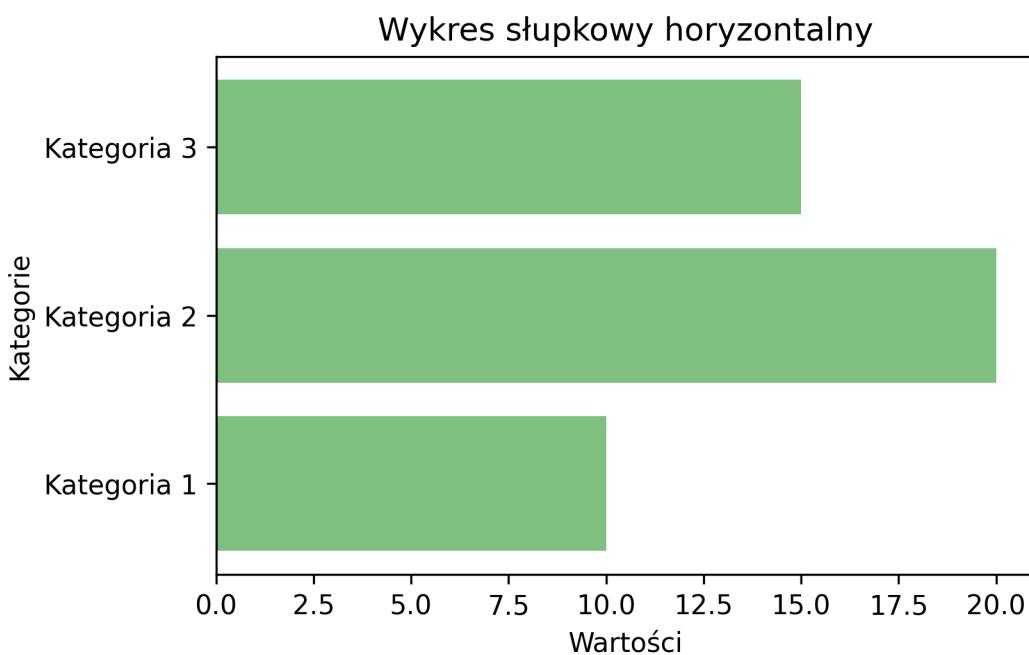
```
import matplotlib.pyplot as plt

# 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')

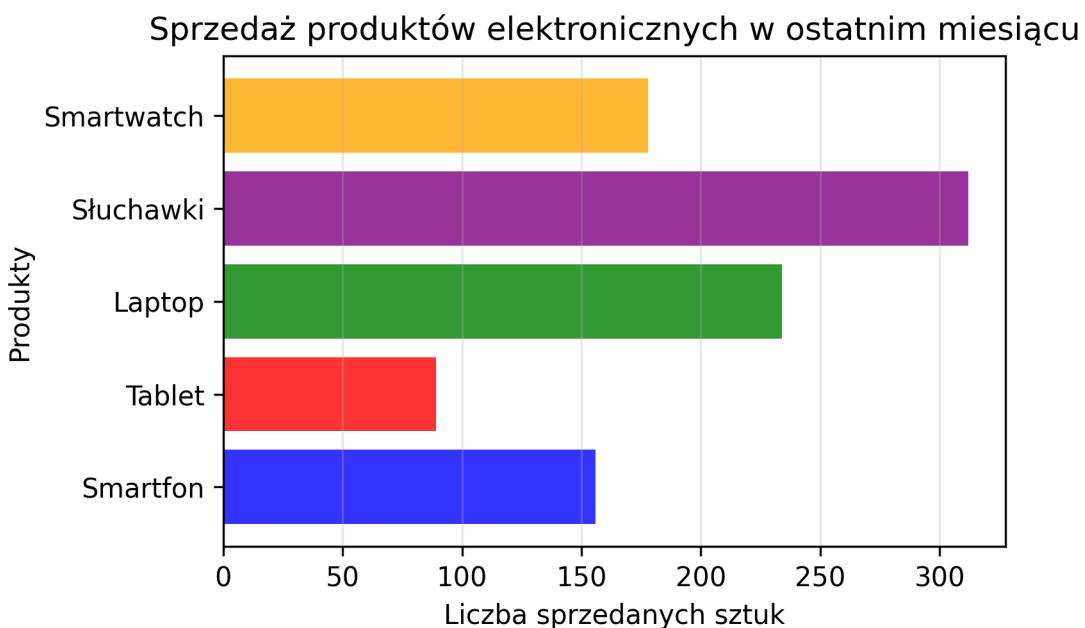
# Wyświetlenie wykresu
plt.show()
```



```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dane_sprzedazy = {
    'produkt': ['Smartfon', 'Tablet', 'Laptop', 'Słuchawki', 'Smartwatch'],
    'sprzedane_sztuki': [156, 89, 234, 312, 178],
    'kategoria': ['telefony', 'tablety', 'komputery', 'akcesoria', 'zegarek']
}
df = pd.DataFrame(dane_sprzedazy)
y_pos = np.arange(len(df))
kolory = {'telefony': 'blue', 'tablety': 'red', 'komputery': 'green',
          'akcesoria': 'purple', 'zegarek': 'orange'}
colors_list = [kolory[kat] for kat in df['kategoria']]
plt.barh(y_pos, df['sprzedane_sztuki'], color=colors_list, alpha=0.8)
plt.yticks(y_pos, df['produkt'])
plt.ylabel('Produkty')
plt.xlabel('Liczba sprzedanych sztuk')
plt.title('Sprzedaż produktów elektronicznych w ostatnim miesiącu')
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.show()

```



```

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

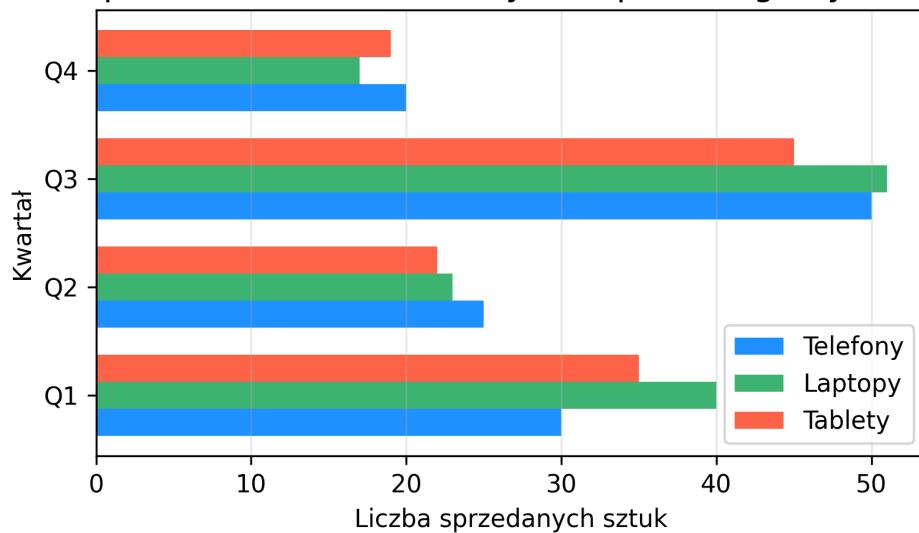
df = pd.DataFrame({
    'Kwartał': ['Q1', 'Q2', 'Q3', 'Q4'],
    'Telefony': [30, 25, 50, 20],
    'Laptopy': [40, 23, 51, 17],
    'Tablety': [35, 22, 45, 19]
})

telefony = df['Telefony']
laptopy = df['Laptopy']
tablety = df['Tablety']
kwartaly = df['Kwartał']
Y = np.arange(len(kwartaly))

plt.barh(Y, telefony, color='dodgerblue', height=0.25, label='Telefony')
plt.barh(Y + 0.25, laptopy, color='mediumseagreen', height=0.25, label='Laptopy')
plt.barh(Y + 0.50, tablety, color='tomato', height=0.25, label='Tablety')
plt.yticks(Y + 0.25, kwartaly)
plt.legend()
plt.title('Sprzedaż produktów elektronicznych w poszczególnych kwartałach', fontsize=14)
plt.xlabel('Liczba sprzedanych sztuk')
plt.ylabel('Kwartał')
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.show()

```

Sprzedaż produktów elektronicznych w poszczególnych kwartałach



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

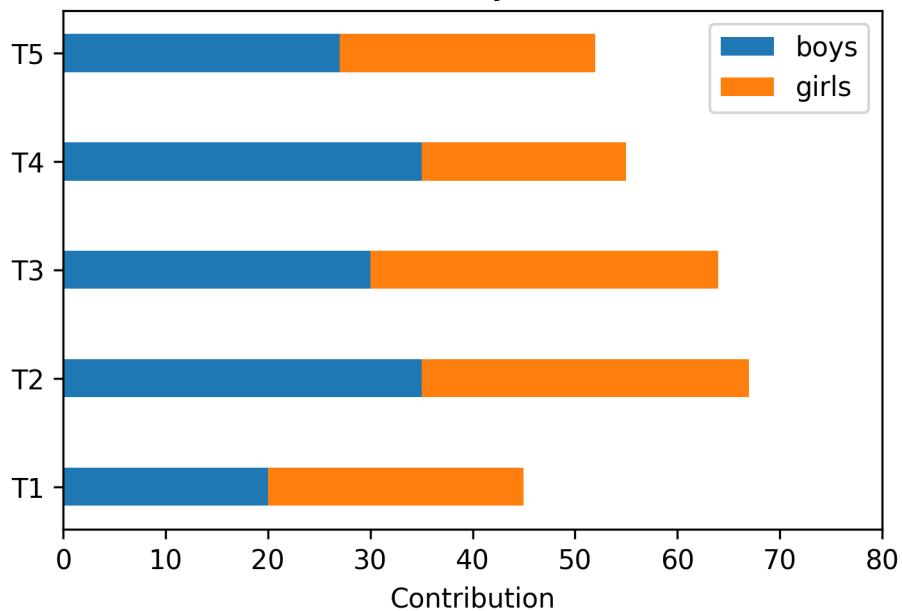
N = 5

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()
```

Contribution by the teams



33 Matplotlib - regresja

W poniższej części omówiony jest przykład działania regresji wielomianowej. Inne rodzaje regresji można zmienić wybierając inną komendę do jej generowania.

Regresja - metoda statystyczna pozwalająca na badanie związku pomiędzy wielkościami danych. Celem regresji wielowymiarowej jest ilościowe ujęcie związków pomiędzy wieloma zmiennymi niezależnymi (objaśniającymi, czynnikami, predyktorami) a zmienną zależną (kryterialną, objaśnianą, odpowiedzią).

Przykłady regresji wielowymiarowej:

- Wytrzymałość betonu zależy od składników użytych przy jego produkcji. Pytanie: W jakiej proporcji stosować te składniki, by wytrzymałość była największa?
- Cena mieszkania zależy od.... Pytanie: jak udział poszczególnych elementów wpływa na to, aby cena rynkowa była najwyższa?
- Udzielenie kredytu zależy od Pytanie: jak udział poszczególnych elementów wpływa na decyzję o przyznaniu lub nie kredytu? *czy to na pewno regresja?*

W ujęcie “naukowym”, badania statystyczne mają w ogólności wyjaśniać zależności pomiędzy różnymi cechami badanej populacji.

Cele badań w analizie regresji:

- Scharakteryzowanie relacji (między innymi jej zasięgu, kierunku i siły).
- Określenie modelu matematycznego, który w najbardziej wiarygodny sposób oddaje zachowanie się odpowiedzi (innymi słowy, znalezienie odpowiedniej funkcji, która może być później wykorzystana do predykcji).
- Określenie, które ze zmiennych objaśniających są ważne w analizie współzależności i uszeregowanie tych zmiennych ze względu na siłę wpływu na zmienną objaśnianą.
- Porównywanie różnych modeli dla jednej zmiennej objaśnianej, tzn. porównanie modeli, które składają się z różnych zestawów zmiennych objaśniających.
- Określenie interakcji zmiennych objaśniających oraz (przy dwukierunkowej zależności) określenie zależności zmiennych objaśniających od zmniejszej objaśnianej.
- Oszacowanie punktowe wartości współczynników regresji (kierunek i siła współzależności oraz istotność statystyczna parametrów wprowadzonych do modelu).

Uzyskiwane wyniki należy zweryfikować pod kątem następujących kryteriów:

- Określenie logicznego związku pomiędzy zmiennymi, tzn. sprawdzenie czy uzyskane wyniki nie kolidują z naturą zjawiska.
- Sprawdzenie czy przyczyna poprzedza w czasie skutek.
- Analiza siły związku pomiędzy zmiennymi np. wysoka korelacja między zmiennymi, które w rzeczywistości nie oddziałują na siebie.
- Sprawdzenie czy otrzymany model sprawdza się w rzeczywistości.
- Spójności wyników.
- Określenie zgodności wyników z wiedzą teoretyczną oraz doświadczalną.
- Rozpatrzenie możliwości otrzymania badanego skutku, jako przejawu działania różnych przyczyn oraz możliwości wystąpienia kilku skutków jednej przyczyny.

Najczęściej stosowane funkcje w analizie regresji:

- funkcja liniowa $f(x) = ax + b$,
- funkcja wielomianowa, np. kwadratowa $f(x) = ax^2 + bx + c$,
- funkcja logarytmiczna $f(x) = \ln x$,
- funkcja eksponencjalna $f(x) = e^{-x}$,
- funkcja logistyczna $f(x) = \frac{1}{1+e^{-x}}$.

Wybór metody współzależności wielu zmiennych:

Metoda analizy	Zmienna objaśniana	Zmienne objaśniające
Regresja wieloraka	ciągła	ciągłe (dopuszcza się także dyskretnie)
Analiza wariancji	ciągła	jakościowe
Analiza kowariancji	ciągła	jakościowe (symboliczne) i ciągłe
Regresja Poissona	dyskretna	różne typy
Regresja logistyczna	dwuwartościowa	różne typy

Zastosowania:

- Ekonometria: Regresja wielowymiarowa może być używana do analizowania wpływu różnych czynników na wzrost gospodarczy, takich jak inwestycje, konsumpcja, inflacja czy poziom zatrudnienia.
- Medycyna: W badaniach medycznych regresja wielowymiarowa może pomóc w identyfikacji czynników wpływających na rozwój chorób, takich jak wiek, dieta, styl życia czy obciążenie genetyczne.

- Marketing: Regresja wielowymiarowa może być stosowana do analizy wpływu różnych cech produktów na sprzedaż, np. cen, reklam, rodzaju opakowania, czy też konkurencji.
- Finanse: Regresja wielowymiarowa może być używana do analizowania wpływu różnych czynników na zwrot z inwestycji, takich jak ryzyko, stopy procentowe, wzrost gospodarczy czy polityka fiskalna.
- Inżynieria: W inżynierii regresja wielowymiarowa może pomóc w analizie wpływu różnych parametrów na wydajność maszyn, takich jak temperatura, ciśnienie czy prędkość.
- Nauki społeczne: W naukach społecznych regresja wielowymiarowa może być stosowana do analizy wpływu różnych czynników na wyniki edukacyjne uczniów, takich jak poziom wykształcenia rodziców, dochody czy środowisko kulturowe.

Rozważamy wpływ zbioru k zmiennych X_1, \dots, X_k na zmienną Y . Należy wprowadzić do modelu jak największą liczbę zmiennych niezależnych oraz powinny się w nim znaleźć zmienne silnie skorelowane ze zmienną zależną i jednocześnie jak najsłabiej skorelowane między sobą.

Liniowy model regresji wielowymiarowej:

$$Y = \beta_0 + \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + \dots + \beta_k \cdot X_k + \varepsilon.$$

β_i - współczynniki regresji (parametry modelu) opisujące wpływ i -tej zmiennej. ε - składnik losowy.

```
import numpy as np

# Przykładowe dane
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([2.1, 2.9, 4.2, 6.1, 8.1, 9.9])

# Dopasowanie linii prostoliniowej (stopień 1) z pełnymi wynikami
coefs, stats = np.polynomial.polynomial.polyfit(x, y, deg=1, full=True)

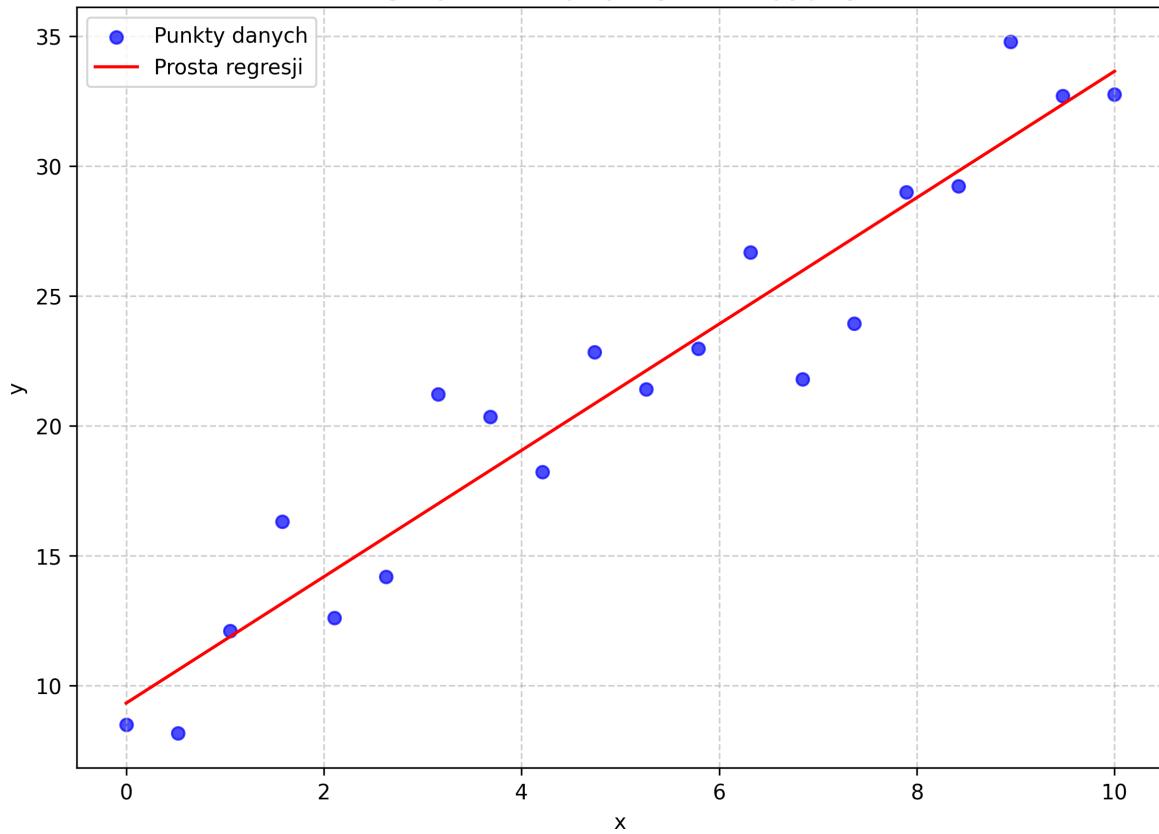
# Wyświetlenie wyników
print("Współczynniki dopasowania:", coefs)
print("\nInformacje diagnostyczne:")
print("Reszty:", stats[0]) # Suma kwadratów reszt
print("Ranga macierzy układu:", stats[1])
print("Wartości osobliwe macierzy układu:", stats[2])
print("Przewidywany błąd dopasowania:", stats[3])
```

Współczynniki dopasowania: [1.51428571 1.61428571]

```
Informacje diagnostyczne:  
Reszty: [0.87142857]  
Ranga macierzy układu: 2  
Wartości osobliwe macierzy układu: [1.35119311 0.41746518]  
Przewidywany błąd dopasowania: 1.3322676295501878e-15
```

```
import numpy as np  
import matplotlib.pyplot as plt  
  
# Przykładowe dane  
np.random.seed(42) # Ustawienie ziarna losowego dla powtarzalności wyników  
x = np.linspace(0, 10, 20) # 20 równomiernie rozłożonych punktów między 0 a 10  
y = 3 * x + 7 + np.random.normal(0, 3, len(x)) # Dane y z losowym szumem  
  
# Dopasowanie linii prostoliniowej (regresja liniowa)  
coefs = np.polynomial.polynomial.polyfit(x, y, deg=1) # Współczynniki regresji (y = a + b*x)  
  
# Wyznaczenie linii regresji  
x_fit = np.linspace(min(x), max(x), 100) # Dodatkowe punkty dla gładkiej linii regresji  
y_fit = coefs[0] + coefs[1] * x_fit  
  
# Tworzenie wykresu  
plt.figure(figsize=(8, 6))  
plt.scatter(x, y, color='blue', label='Punkty danych', alpha=0.7) # Wykres punktowy  
plt.plot(x_fit, y_fit, color='red', label=f'Prosta regresji') # Prosta regresji  
  
# Dostosowanie wyglądu wykresu  
plt.title('Regresja liniowa przy użyciu numpy.polyfit')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.legend()  
plt.grid(True, linestyle='--', alpha=0.6)  
plt.tight_layout()  
  
# Wyświetlenie wykresu  
plt.show()
```

Regresja liniowa przy użyciu numpy.polyfit



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

# Przykładowe dane
x = np.array([-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]) # Punkty x
y = np.array([100, 88, 36, 16, 7, -3, 4, 16, 36, 52, 100]) # Punkty y z losowym szumem (wizualne)

# Dopasowanie krzywej stopnia drugiego
coefs = np.polynomial.polynomial.polyfit(x, y, deg=2) # Współczynniki regresji (y = a + b*x + c*x^2)

# Wyznaczenie krzywej regresji
x_fit = np.linspace(min(x), max(x), 100) # Dodatkowe punkty dla gładkiej krzywej regresji
y_fit = coefs[0] + coefs[1] * x_fit + coefs[2] * x_fit**2

# Tworzenie wykresu
plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='blue', label='Punkty danych', alpha=0.7) # Wykres punktowy
```

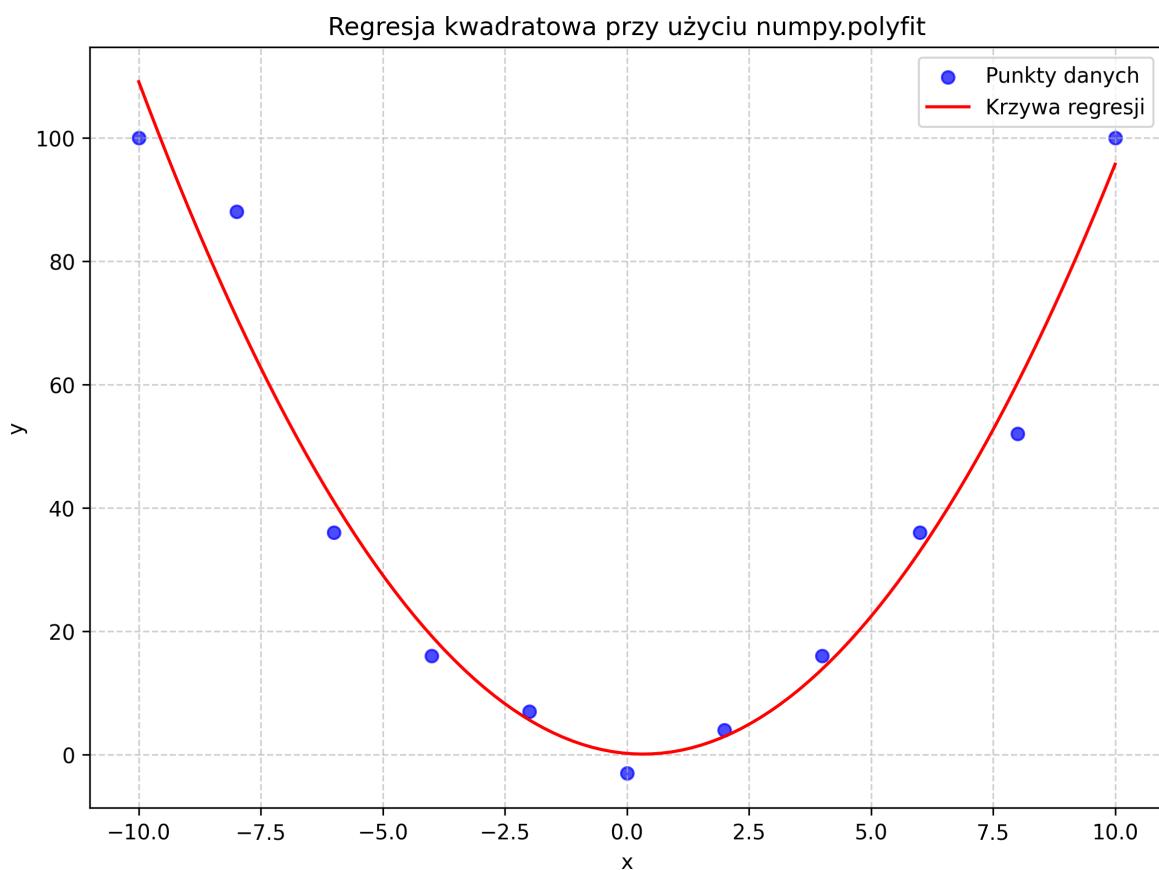
```

plt.plot(x_fit, y_fit, color='red', label=f'Krzywa regresji') # Krzywa regresji

# Dostosowanie wyglądu wykresu
plt.title('Regresja kwadratowa przy użyciu numpy.polyfit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()

# Wyświetlenie wykresu
plt.show()

```



34 Matplotlib - kolory

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



b



g



r



c



m



y



k



w



tab:blue



tab:orange



tab:green



tab:red



tab:purple



tab:brown



tab:pink



tab:gray



tab:olive



tab:cyan



```

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('Ós X')
plt.ylabel('Ós Y')

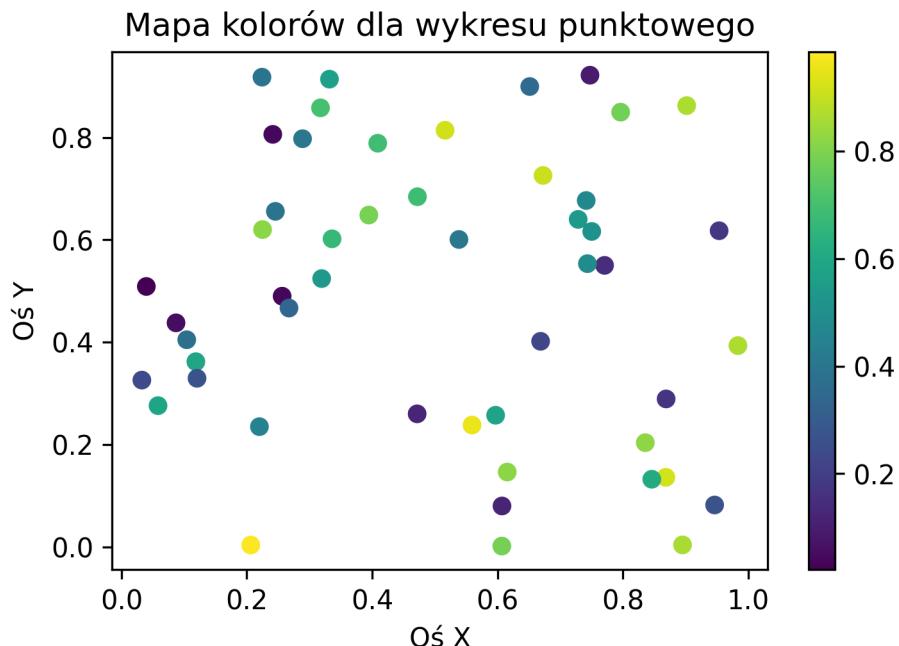
```

①

②

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

- 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 wartościami numerycznymi na mapie kolorów.

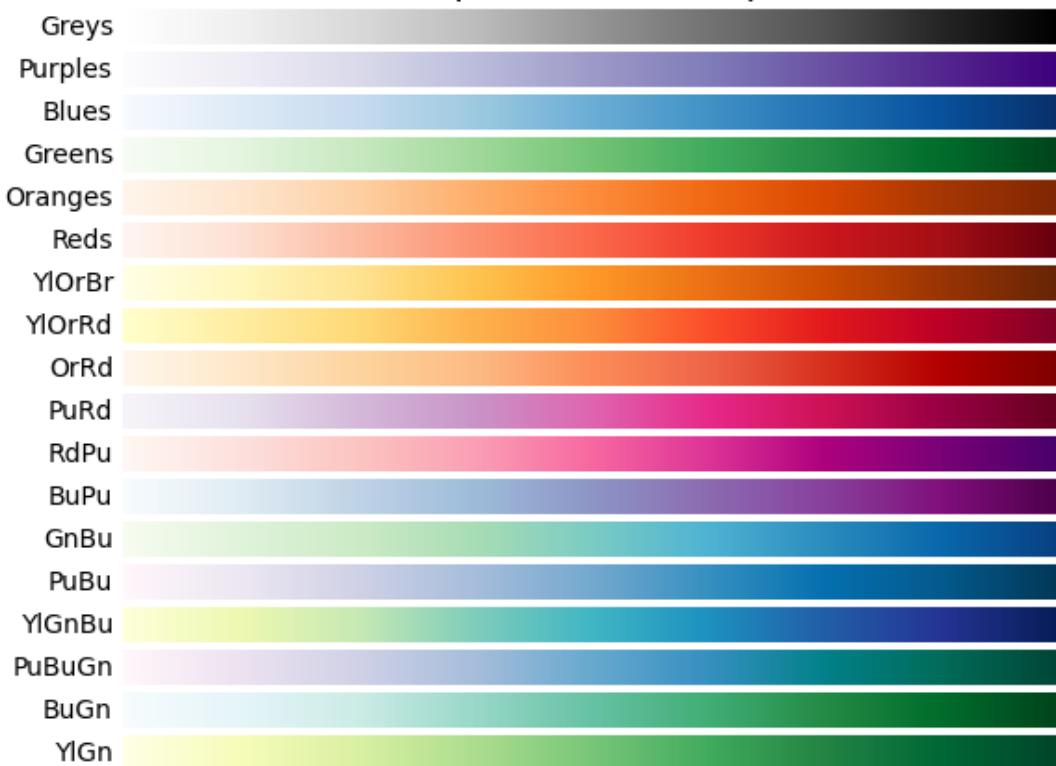


Mapy kolorów

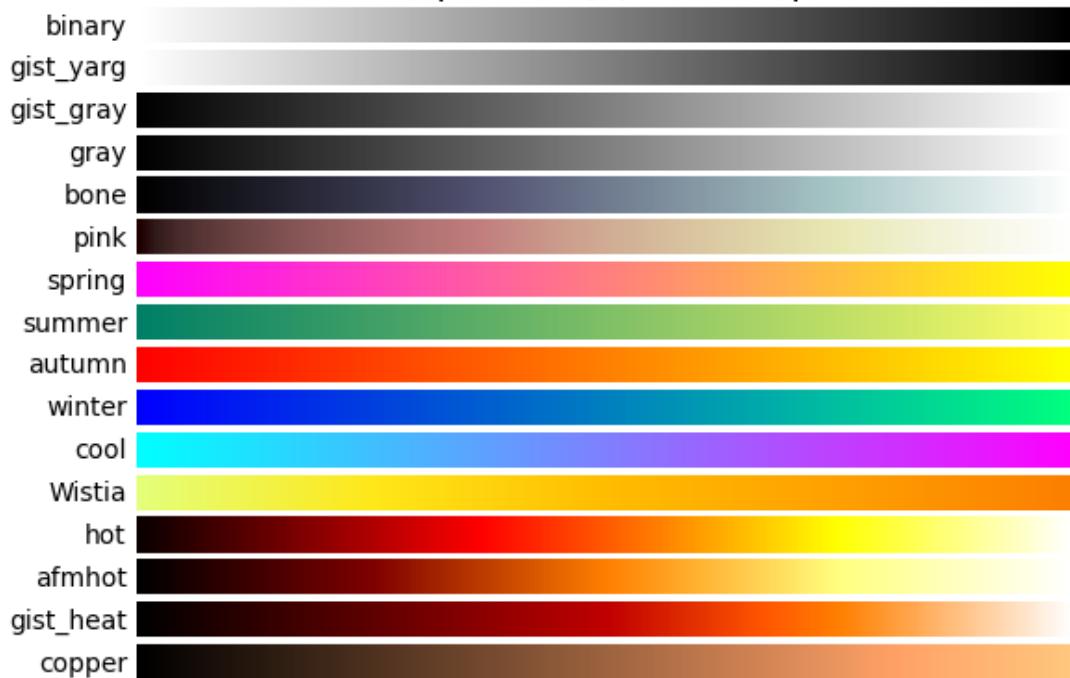
Lista wbudowanych map kolorów: <https://matplotlib.org/stable/tutorials/colors/colormaps.html>



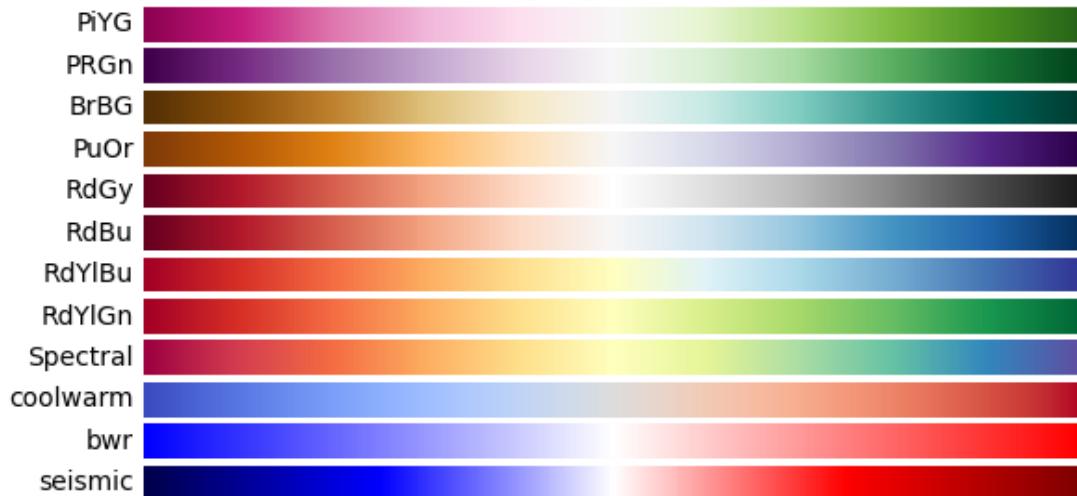
Sequential colormaps



Sequential (2) colormaps



Diverging colormaps

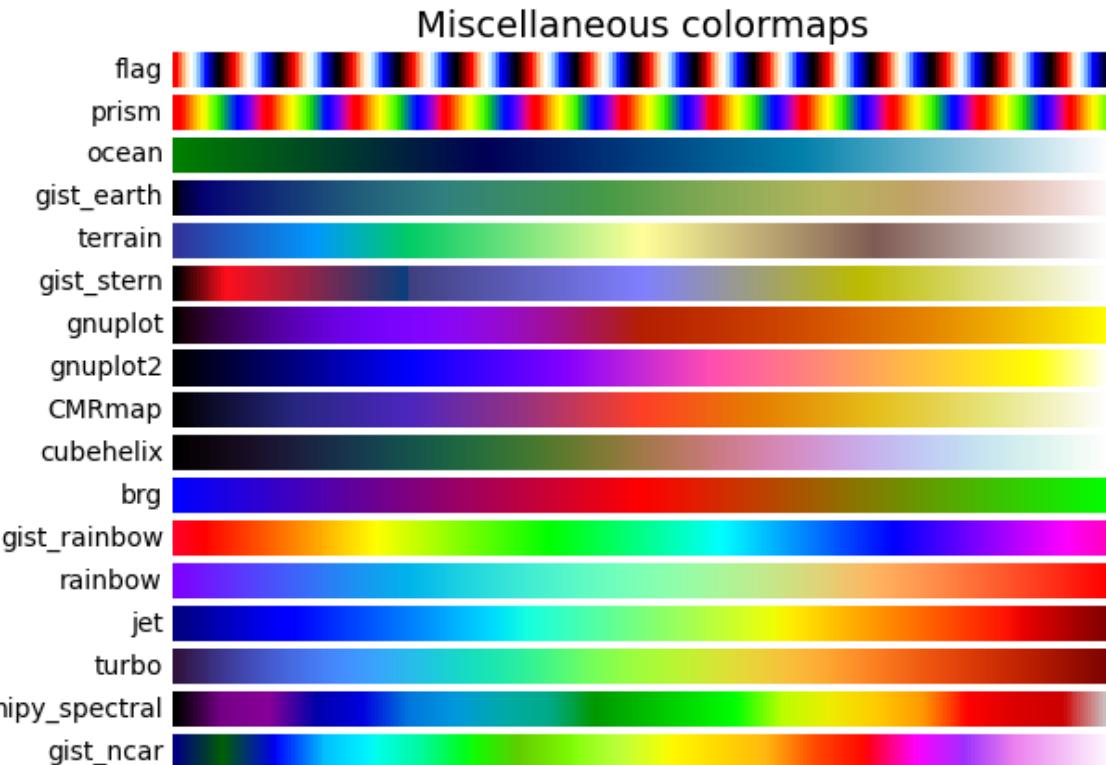


Cyclic colormaps



Qualitative 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

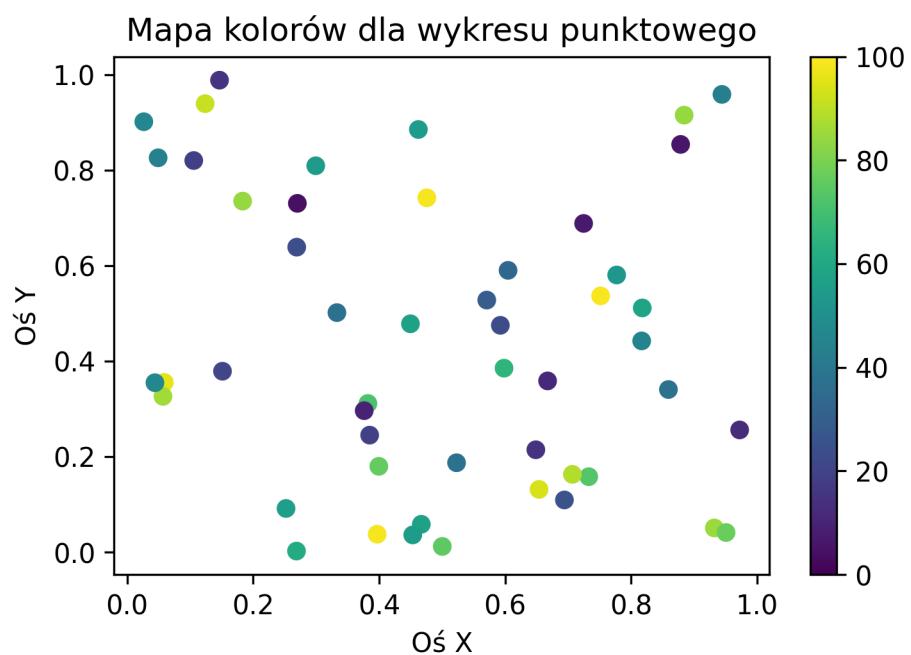
# Tworzenie wykresu punktowego z mapą kolorów
plt.scatter(x, y, c=z, cmap=cmap, norm=norm)
plt.colorbar()

# Dodanie etykiet osi

```

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

# Wyświetlenie wykresu
plt.show(block=True)
```



35 Matplotlib - opcje wykresu

35.1 Argumenty figure w Matplotlib

W Matplotlib kluczową rolę w zarządzaniu właściwościami wykresu pełni obiekt **figure**, który reprezentuje “plotno”, na którym rysowane są wszystkie elementy graficzne. Funkcje **plt.figure()** oraz **plt.subplots()** pozwalają dostosować różne aspekty figury.

```
plt.figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=None, t
```

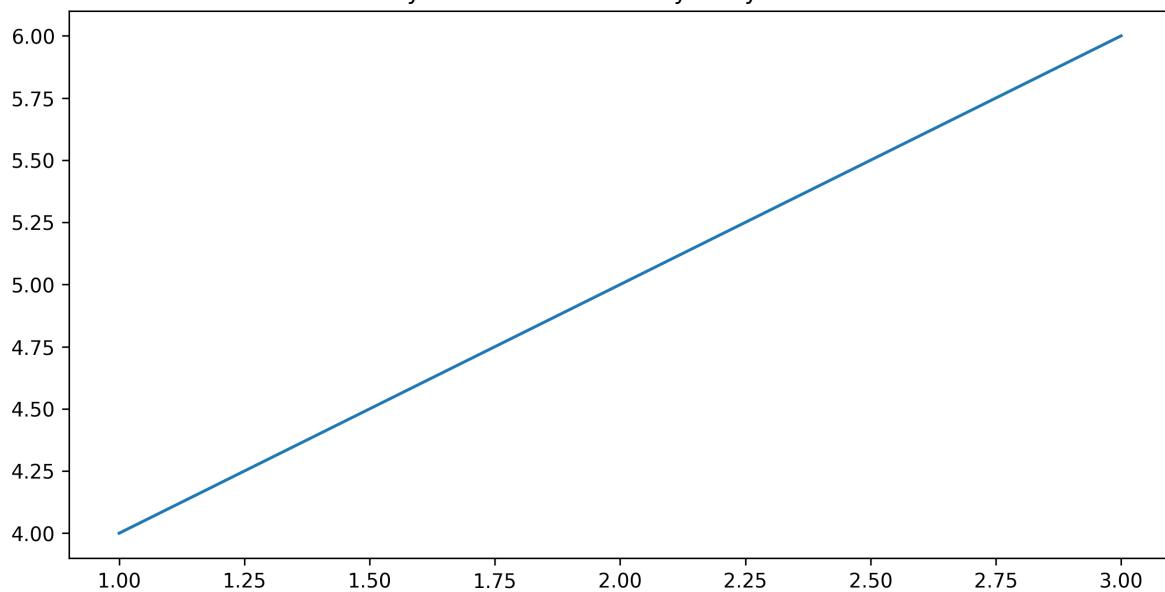
Argumenty:

Argument	Opis	Domyślana wartość
num	Numer identyfikacyjny lub nazwa figury (może być liczbą lub ciągiem znaków).	None (automatyczne)
figsize	Rozmiar figury w calach, podany jako krotka (szerokość, wysokość).	(6.4, 4.8)
dpi	Rozdzielcość figury w punktach na cal (dots per inch).	100
facecolor	Kolor tła figury (całe plotno).	white
edgecolor	Kolor ramki figury.	white
frameon	Czy figura powinna mieć ramkę (True lub False).	True
tight_layout	Automatyczne dopasowanie elementów na figurze w celu uniknięcia nakładania się (True lub False).	False

```
import matplotlib.pyplot as plt

# Tworzenie figury o wymiarach 10x5 cali
plt.figure(figsize=(10, 5))
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Wykres z niestandardowymi wymiarami")
plt.show(block=True)
```

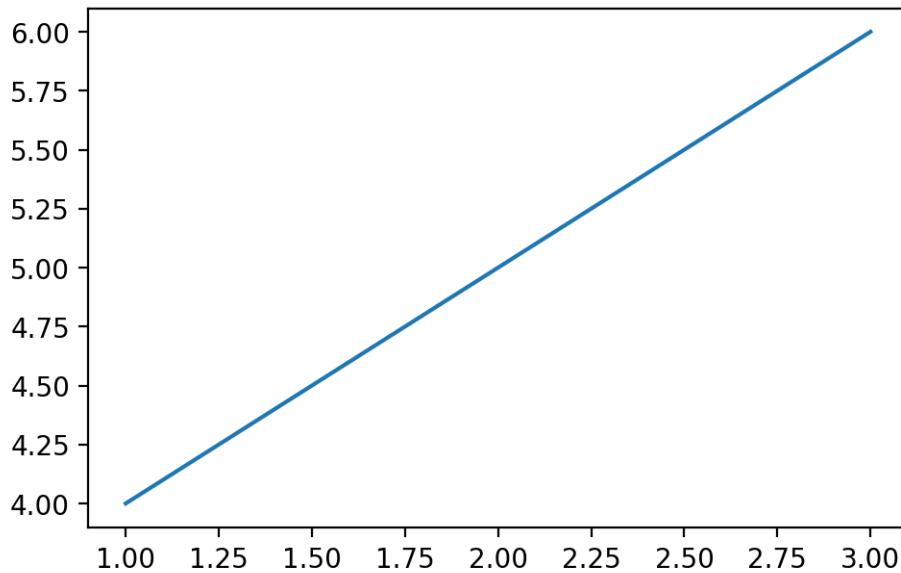
Wykres z niestandardowymi wymiarami



```
import matplotlib.pyplot as plt

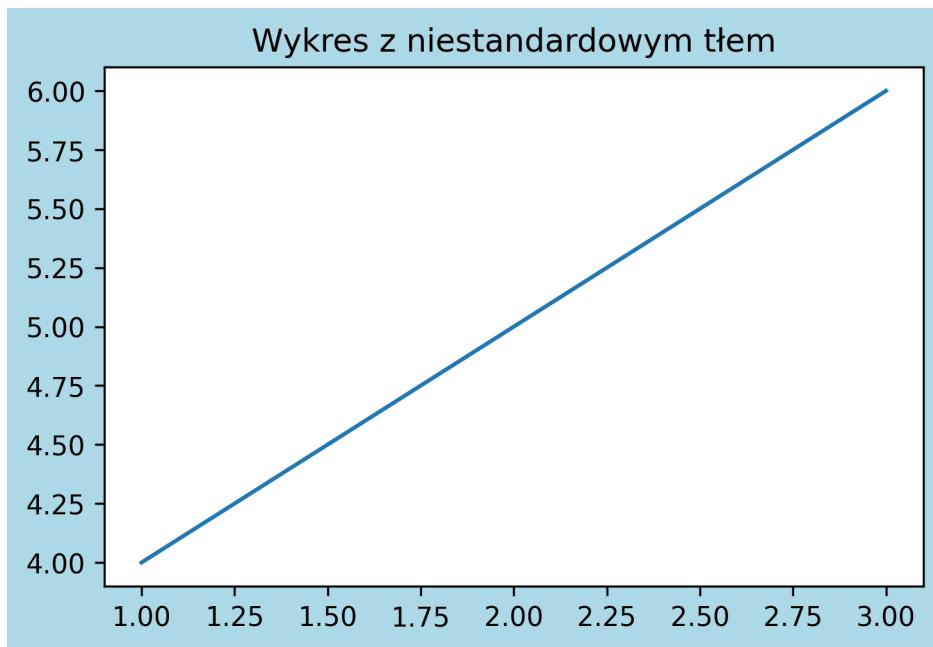
plt.figure(dpi=200) # Wyższa rozdzielcość
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Wykres z wyższą rozdzielcością")
plt.show(block=True)
```

Wykres z wyższą rozdzielczością



```
import matplotlib.pyplot as plt

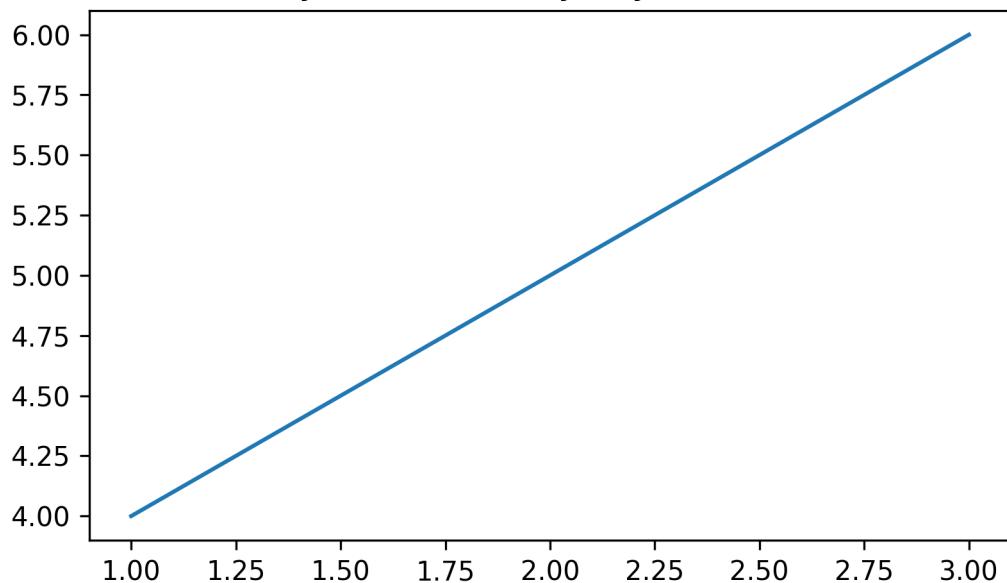
plt.figure(facecolor='lightblue', edgecolor='gray') # Kolor tła i ramki
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Wykres z niestandardowym tłem")
plt.show(block=True)
```



```
import matplotlib.pyplot as plt

plt.figure(tight_layout=True) # Automatyczne dopasowanie
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Wykres z automatycznym układem")
plt.show(block=True)
```

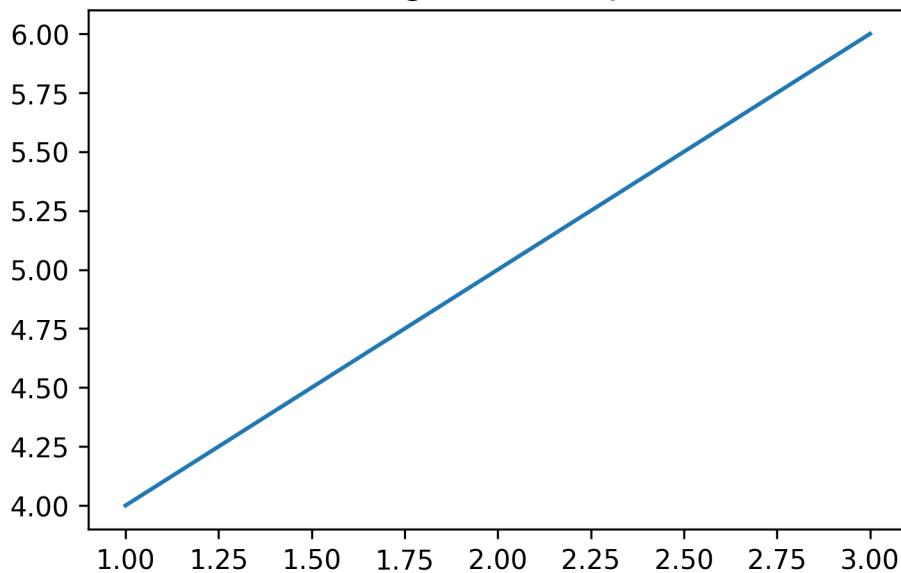
Wykres z automatycznym układem



```
import matplotlib.pyplot as plt

plt.figure(num="Moja Figura") # Nazwa figury
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Figura z nazwą")
plt.show(block=True)
plt.show()
```

Figura z nazwą



35.2 Style

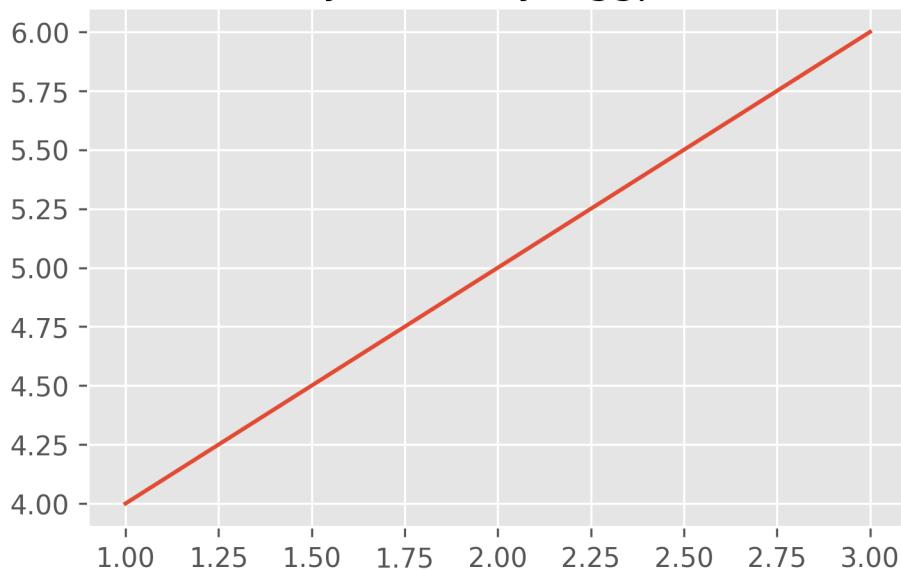
matplotlib oferuje wbudowane style, które pozwalają na szybkie i łatwe dostosowanie wyglądu wykresów. Za pomocą funkcji `plt.style.use` można załadować gotowy styl, dzięki czemu wykresy zyskają spójny i estetyczny wygląd bez potrzeby ręcznego ustawiania wszystkich parametrów.

```
plt.style.use(style)
```

```
import matplotlib.pyplot as plt

# Użycie stylu 'ggplot'
plt.style.use('ggplot')
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Wykres w stylu ggplot")
plt.show(block=True)
```

Wykres w stylu ggplot



Lista stylów:

```
print(plt.style.available)
```

Rozpiska: https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

36 Matplotlib - dodatki cz.2.

36.1 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:

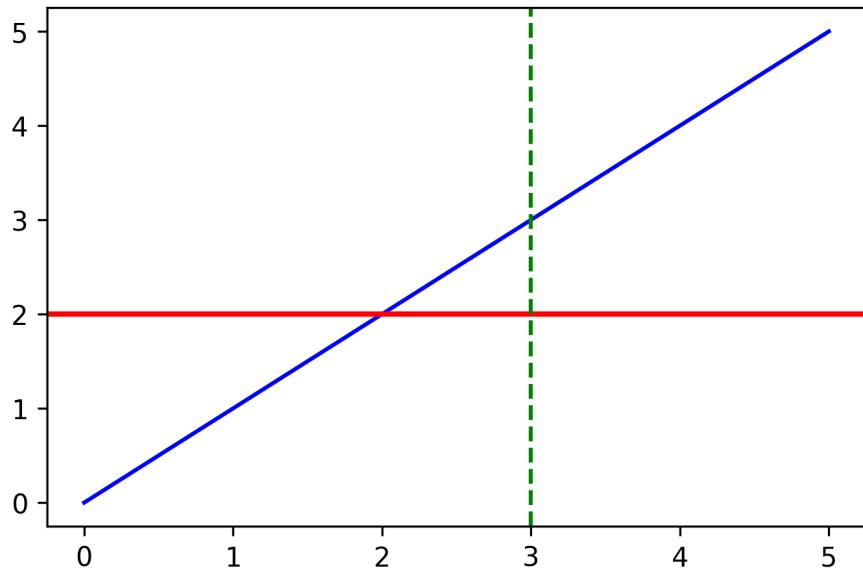
- `y` - wartość na osi Y, przez którą przechodzi linia (domyślnie 0)
- `xmin, xmax` - wartości z zakresu 0-1 określające początek i koniec linii na osi X (domyślnie 0 i 1)
- `**kwargs` - dodatkowe argumenty, takie jak `color`, `linewidth` czy `linestyle`, służące do kontrolowania wyglądu linii

`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

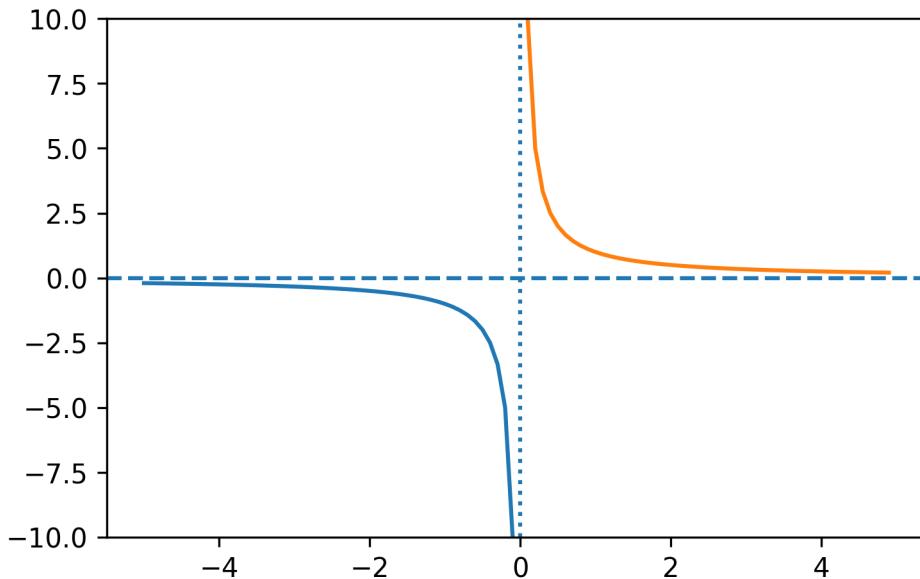
plt.plot([0, 5], [0, 5], color='blue')
plt.axhline(2, color='red', linewidth=2) # Horyzontalna linia przechodząca przez Y=2
plt.axvline(3, color='green', linestyle='--') # Pionowa linia przechodząca przez X=3, styl -
plt.show(block=True)
```



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(block=True)
```



36.2 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')

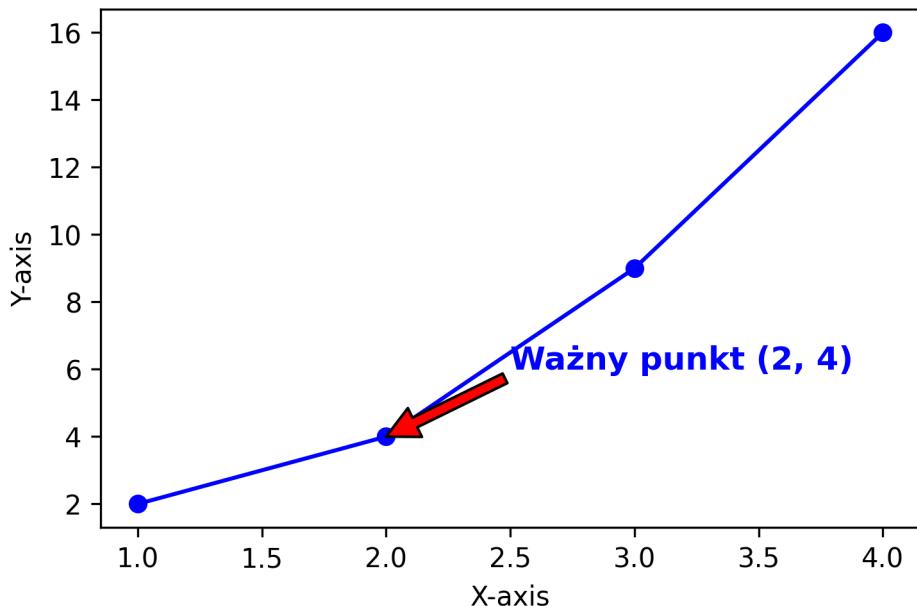
plt.annotate('Ważny punkt (2, 4)',
```

```

        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(block=True)

```



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),
            xytext=(2.5, 6), arrowprops=dict(facecolor='red'),
            fontsize=12, color='blue', fontweight='bold')

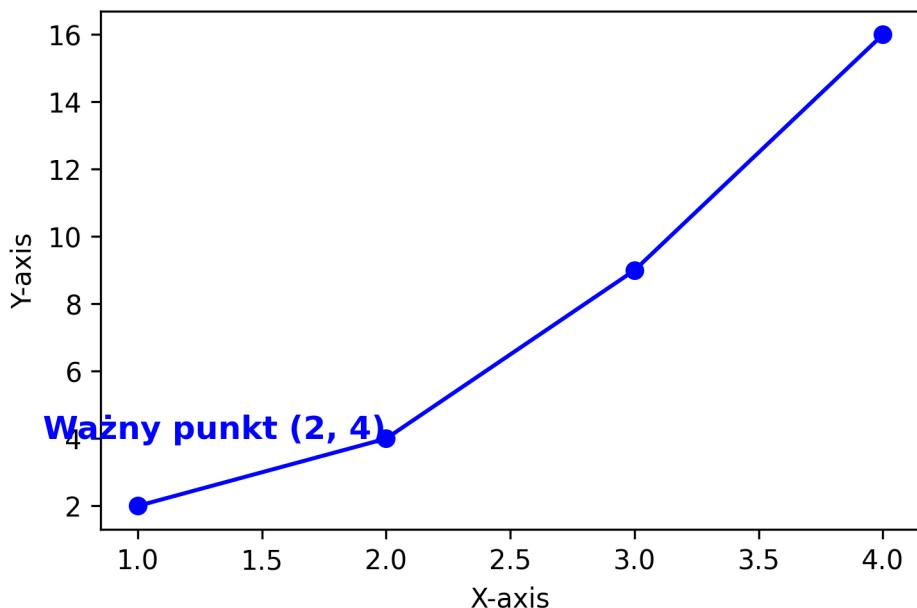
```

```

        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(block=True)

```



36.3 Etykiety osi

Funkcje `xlabel` i `ylabel` służą do dodawania etykiety 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

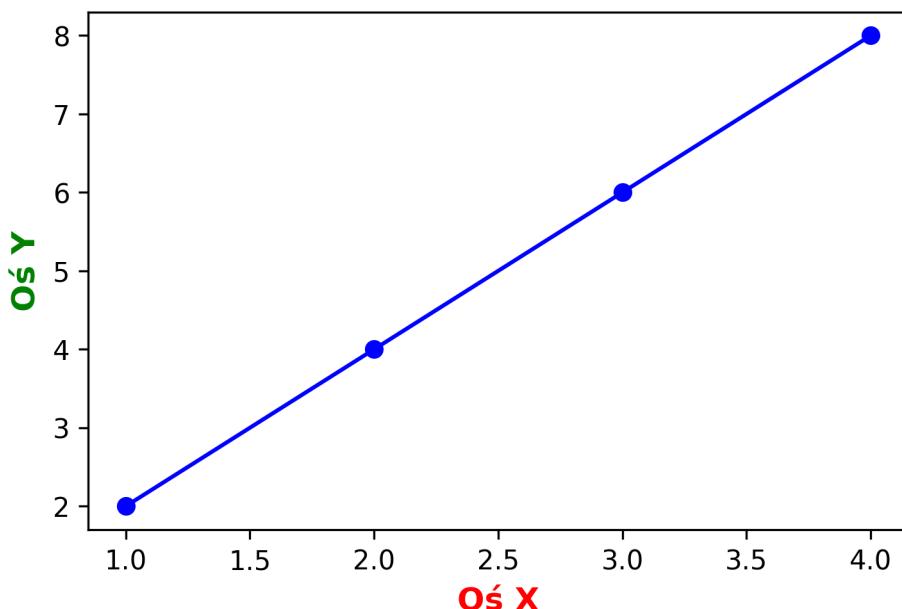
x = [1, 2, 3, 4]
y = [2, 4, 6, 8]

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(block=True)

```



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)

```

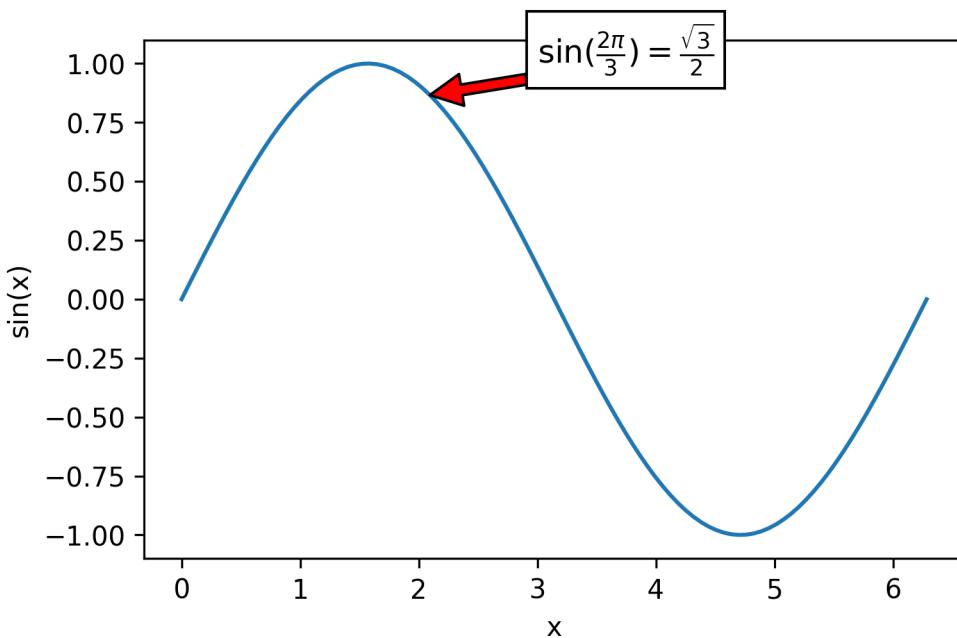
```

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(block=True)

```



36.4 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)

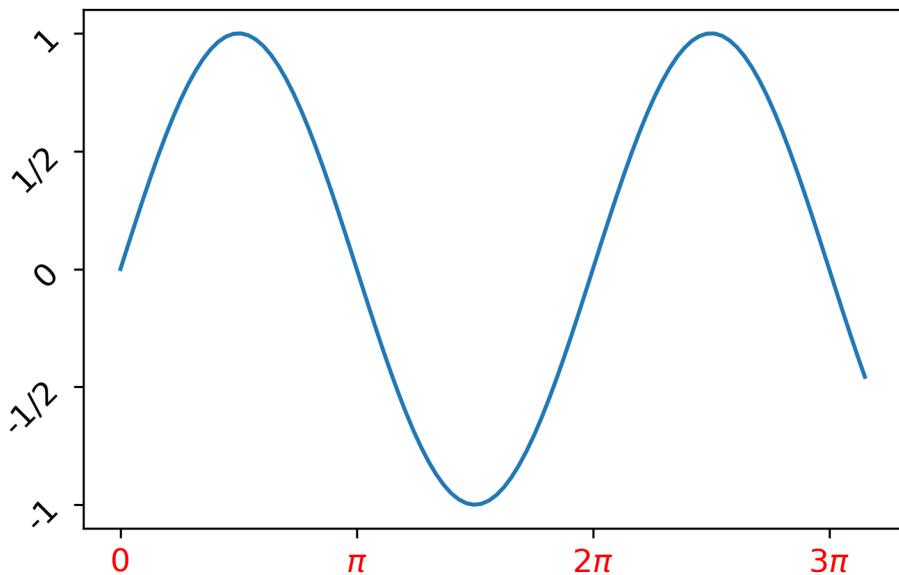
plt.plot(x, y)

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(block=True)
```

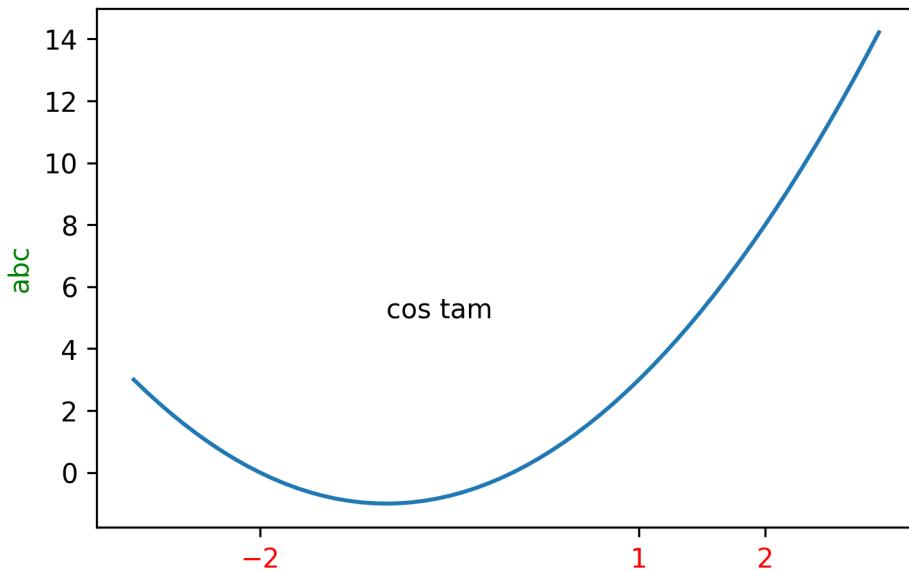


```

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(block=True)

```



36.5 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. Domyslnie 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)

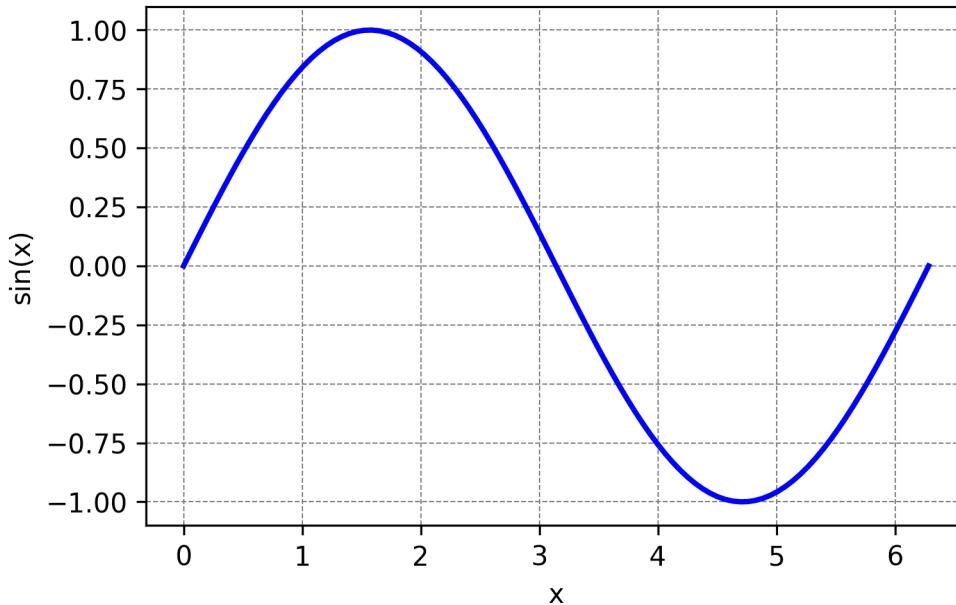
# Tworzenie wykresu
plt.plot(x, y, color='blue', linewidth=2)

# 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)')

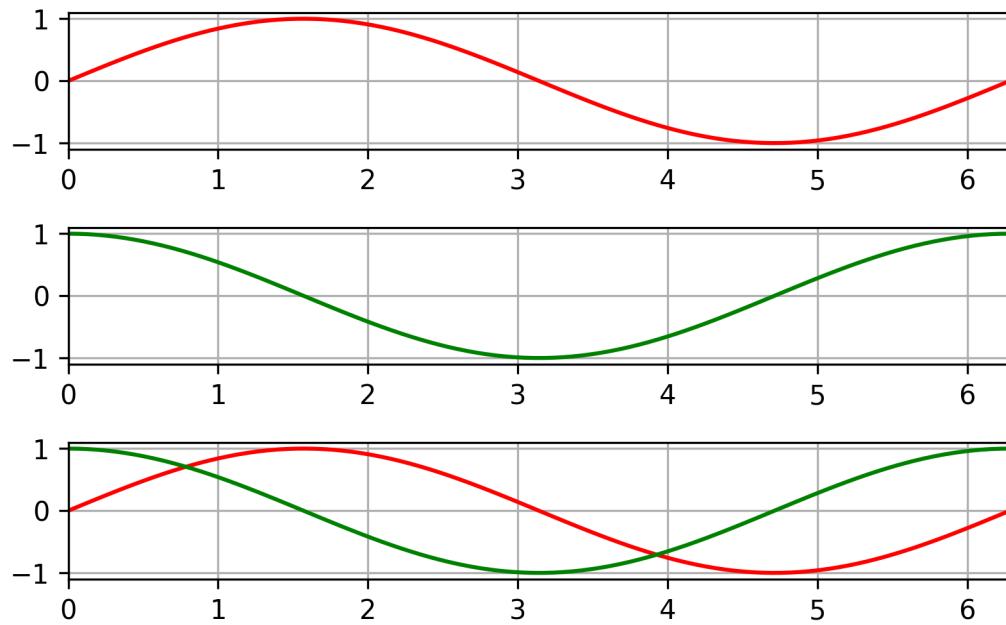
# Wyświetlenie wykresu
plt.show(block=True)
```

Wykres funkcji $\sin(x)$



```
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(block=True)
```



37 Matplotlib - inne wykresy

37.1 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.).

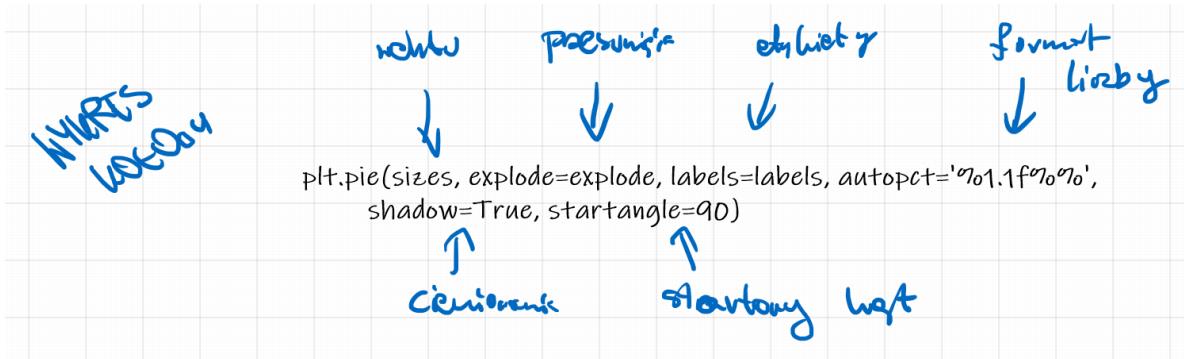
Chociaż wykresy kołowe mają swoje zastosowania, są również krytykowane za ograniczoną precyzję w ocenie proporcji. Dlatego często zaleca się stosowanie innych rodzajów wykresów, takich jak słupkowe (bar chart) czy stosunkowe (stacked bar chart), które mogą być bardziej przejrzyste i precyzyjne w porównywaniu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

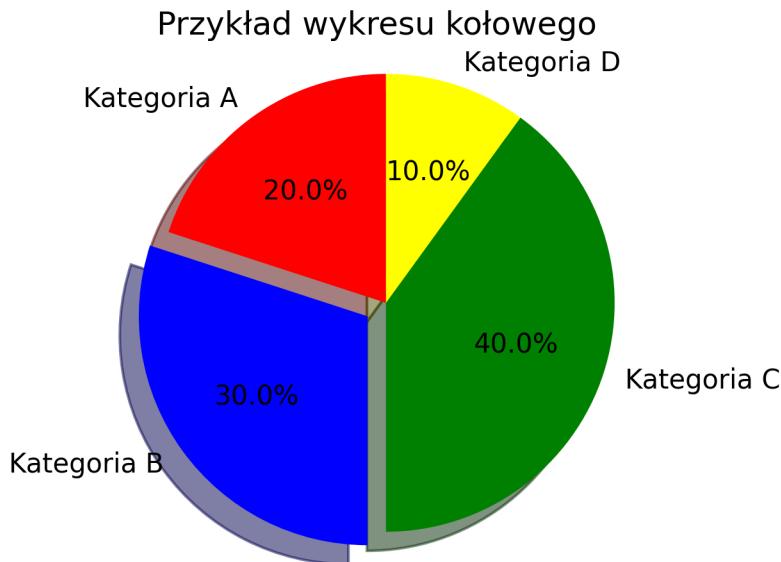
# Tworzenie wykresu kołowego
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%.1f%%', shadow=True)

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

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

plt.show(block=True)

```

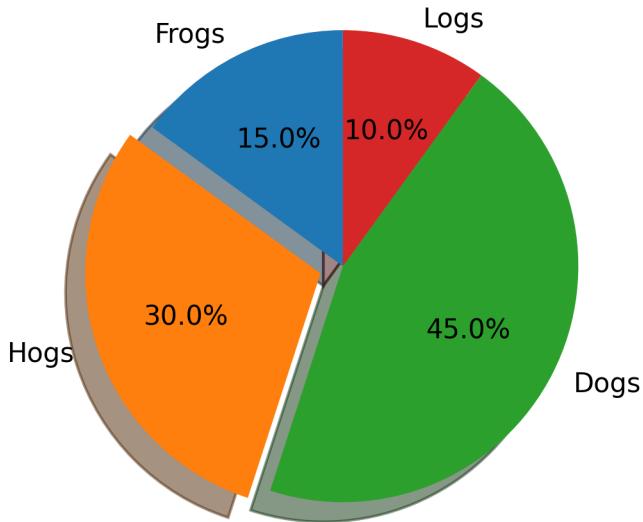


```
import matplotlib.pyplot as plt

# 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='%.1f%%',
         shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show(block=True)
```

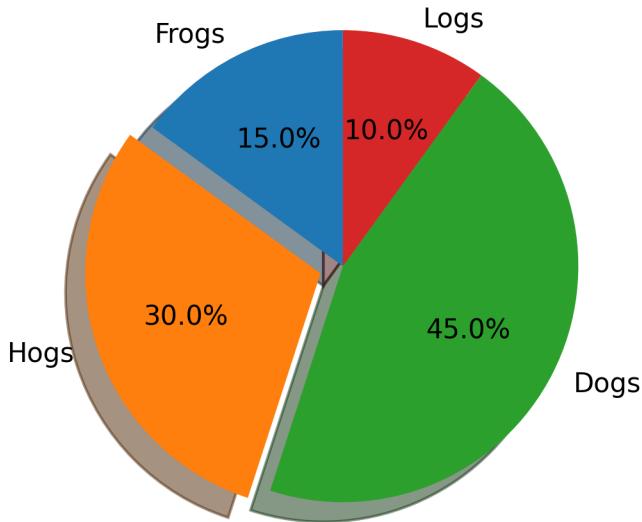


```
import matplotlib.pyplot as plt

# 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='%.1f%%',
        shadow=True, startangle=90)
plt.axis('equal')

plt.show(block=True)
```



```
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)]

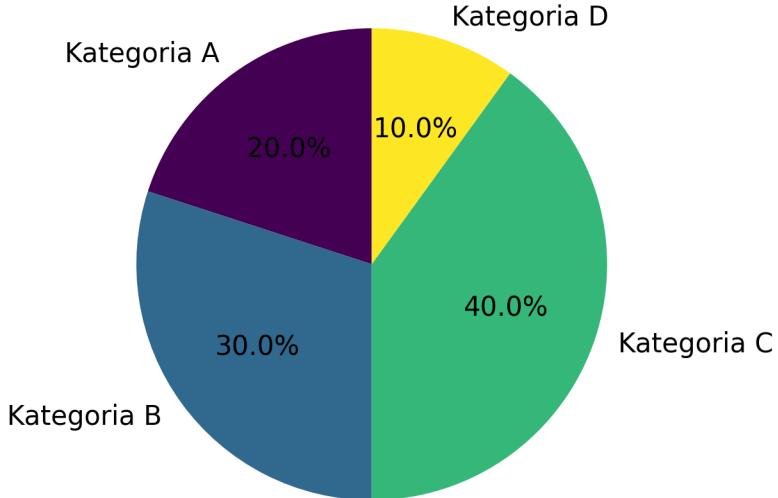
# Tworzenie wykresu kołowego
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)

# 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(block=True)
```

Przykład wykresu kołowego z mapą kolorów



37.2 Wykres dwuosiowy

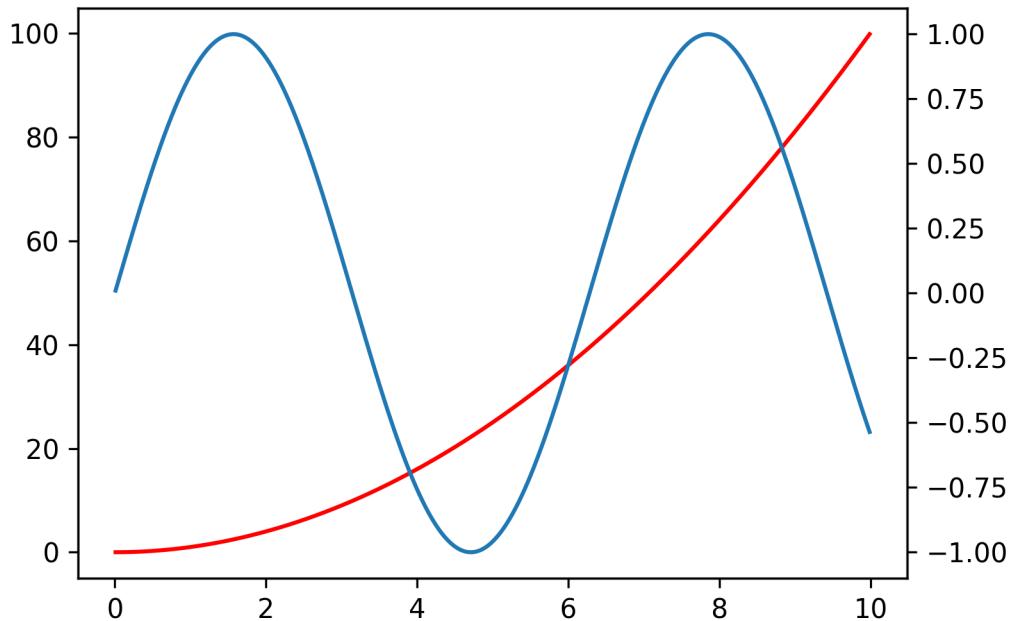
Funkcja `twinx` w bibliotece Matplotlib pozwala na utworzenie drugiej osi Y, która będzie współdzielić oś X z pierwszą osią Y. Dzięki temu, można w prosty sposób przedstawić dwie serie danych, które są mierzne w różnych jednostkach, ale mają wspólną zmienną niezależną.

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 osie 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(block=True)
```



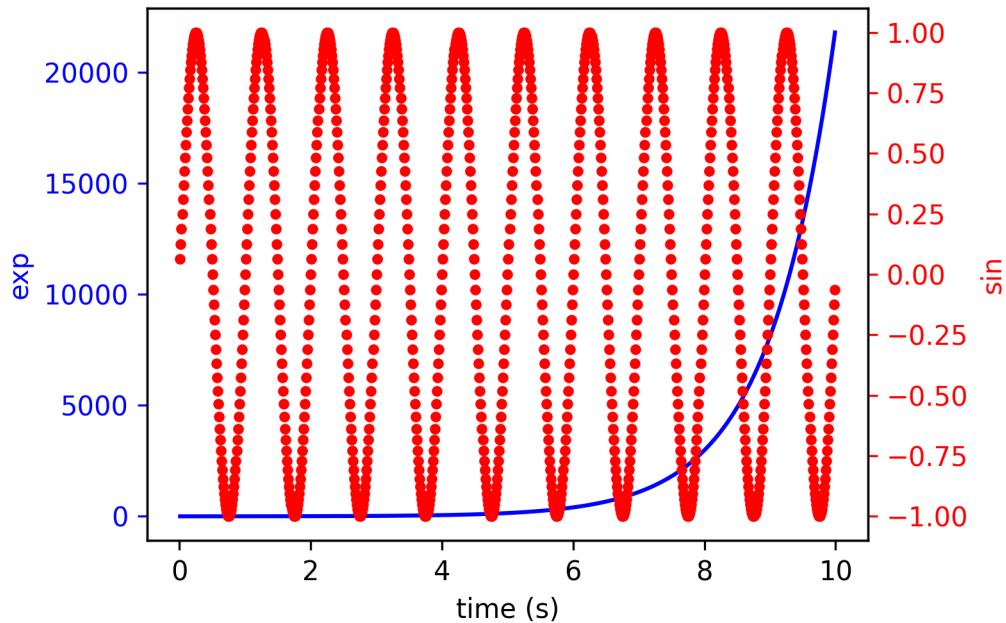
```
import numpy as np
import matplotlib.pyplot as plt

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(block=True)
```



37.3 Wykres słupkowy

Wykres słupkowy jest stosowany do przedstawiania danych kategorialnych 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 legende.

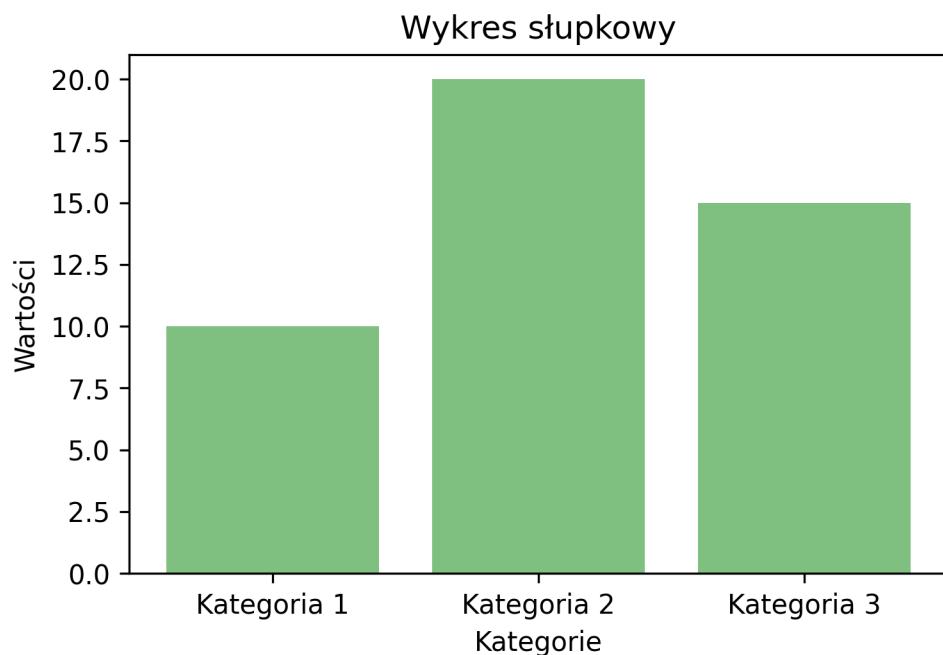
```
import matplotlib.pyplot as plt

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

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

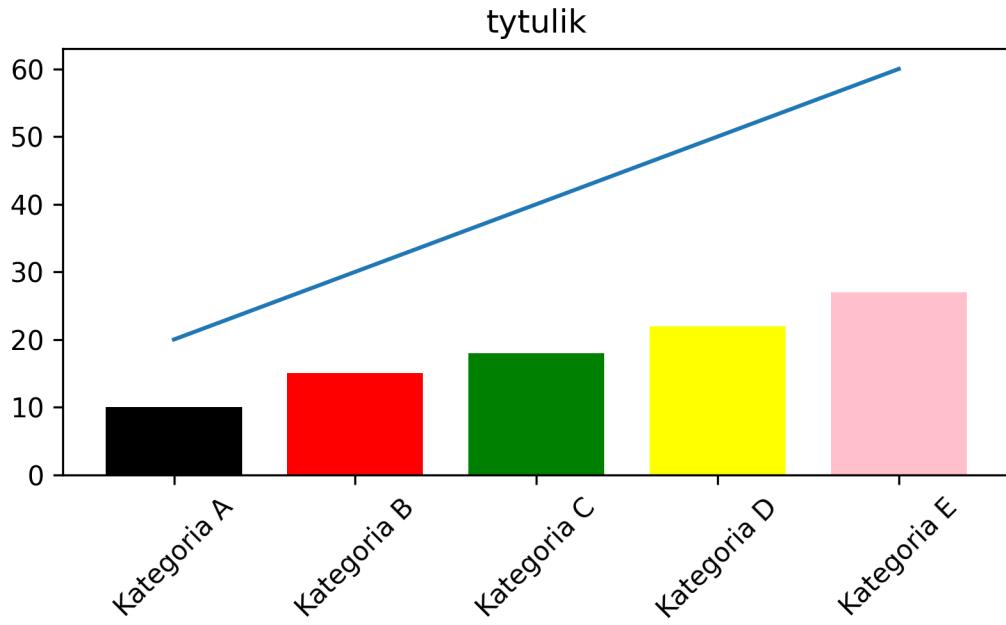
# Dodanie tytułu i etykiet osi
plt.title('Wykres słupkowy')
plt.xlabel('Kategorie')
plt.ylabel('Wartości')

# Wyświetlenie wykresu
plt.show(block=True)
```



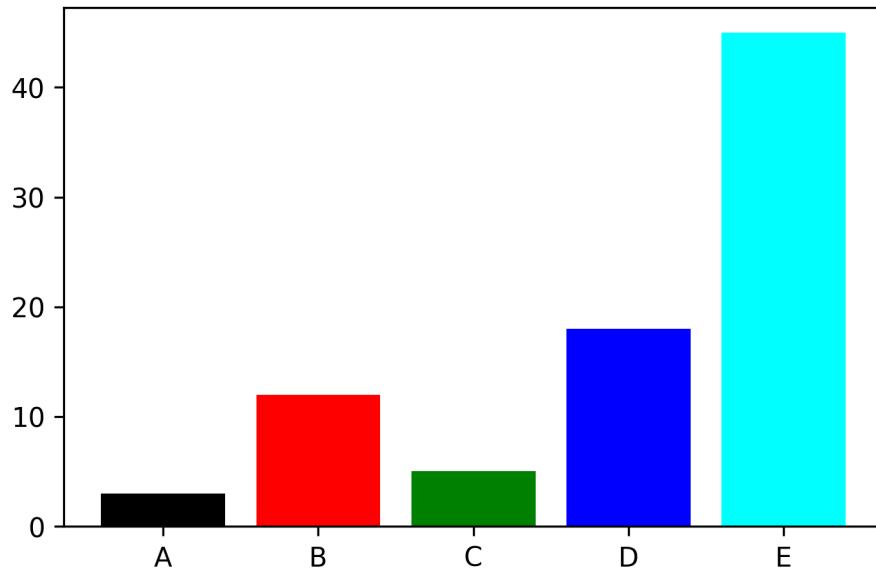
```
import numpy as np
import matplotlib.pyplot as plt

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(block=True)
```



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

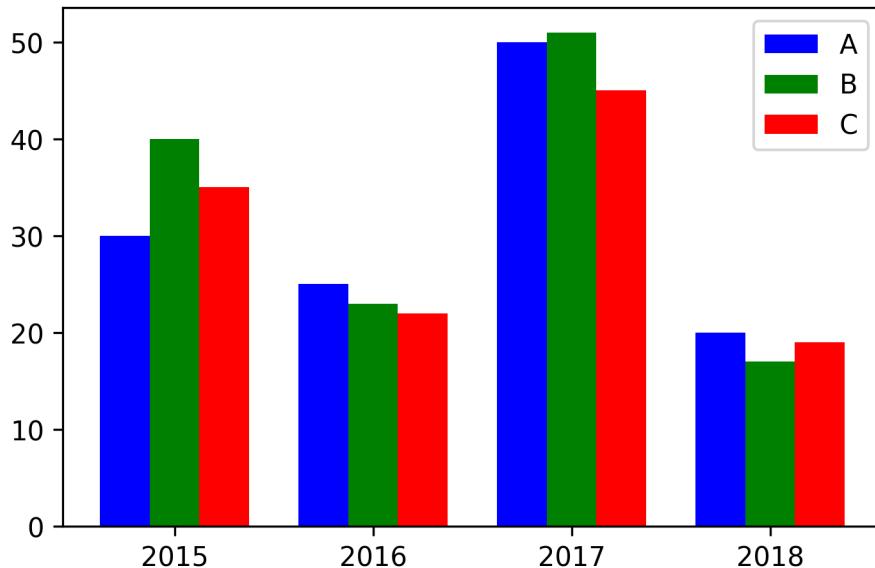
height = [3, 12, 5, 18, 45]
bars = ('A', 'B', 'C', 'D', 'E')
y_pos = np.arange(len(bars))
plt.bar(y_pos, height, color=['black', 'red', 'green', 'blue', 'cyan'])
plt.xticks(y_pos, bars)
plt.show(block=True)
```



```
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(block=True)
```



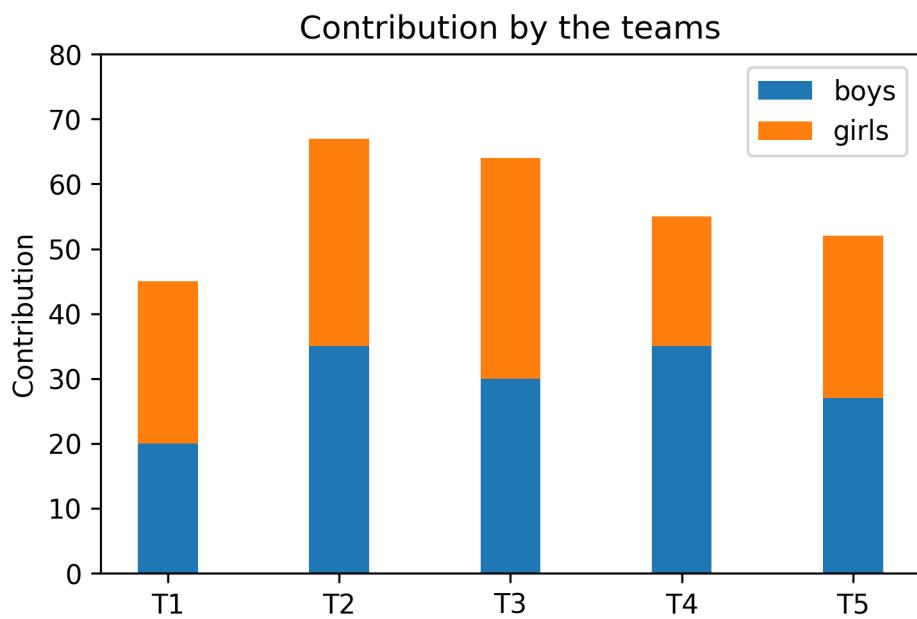
```
import numpy as np
import matplotlib.pyplot as plt

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(block=True)
```



WYKRES STUPKOWY

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.

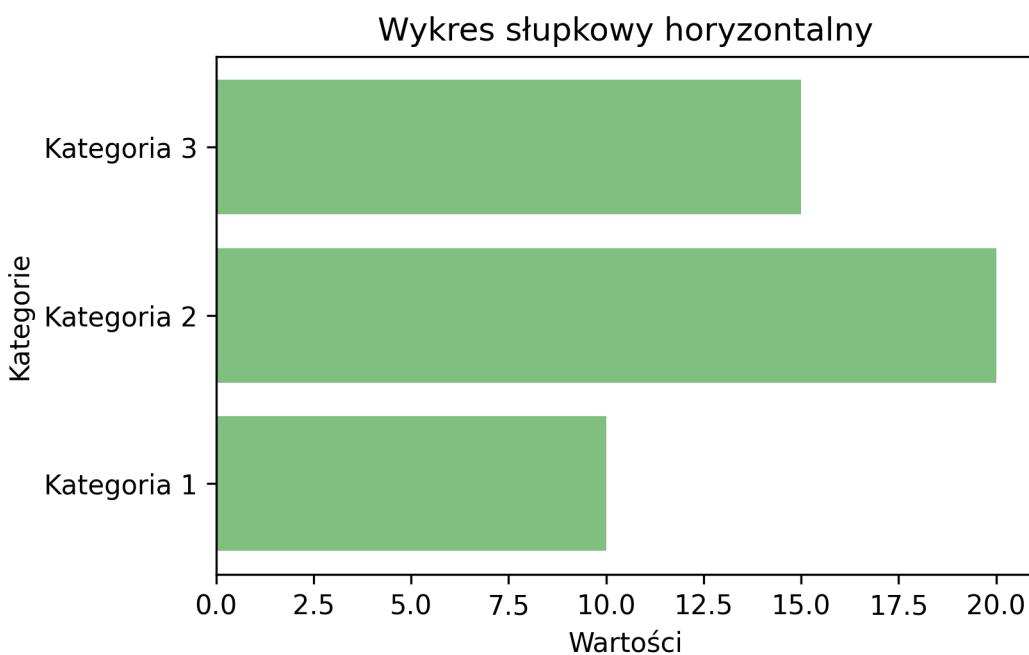
```
import matplotlib.pyplot as plt

# 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(block=True)
```

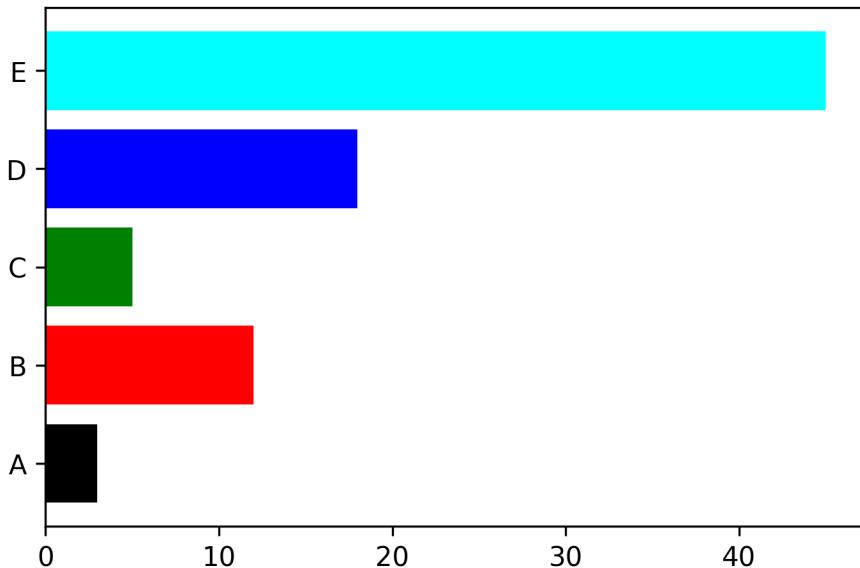


```

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(block=True)

```



```

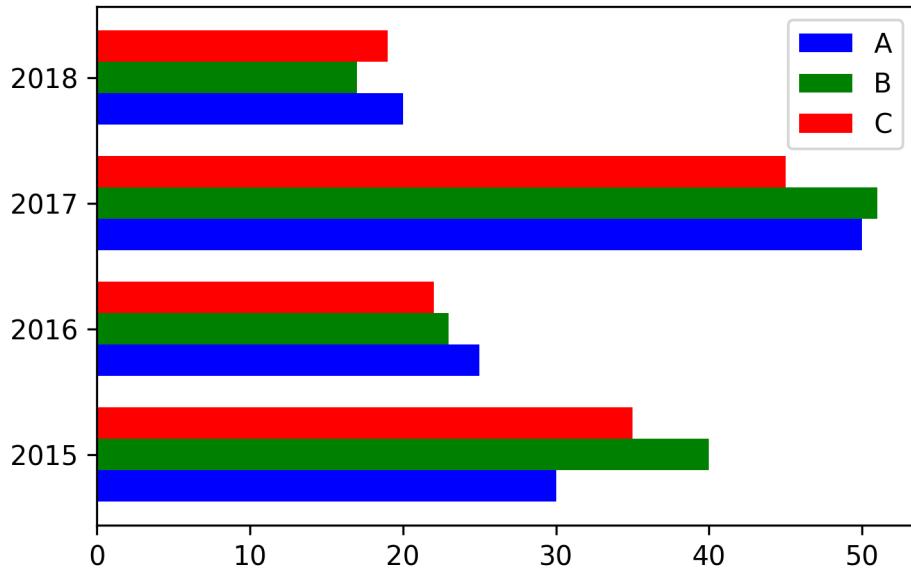
import numpy as np
import matplotlib.pyplot as plt

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(block=True)
```



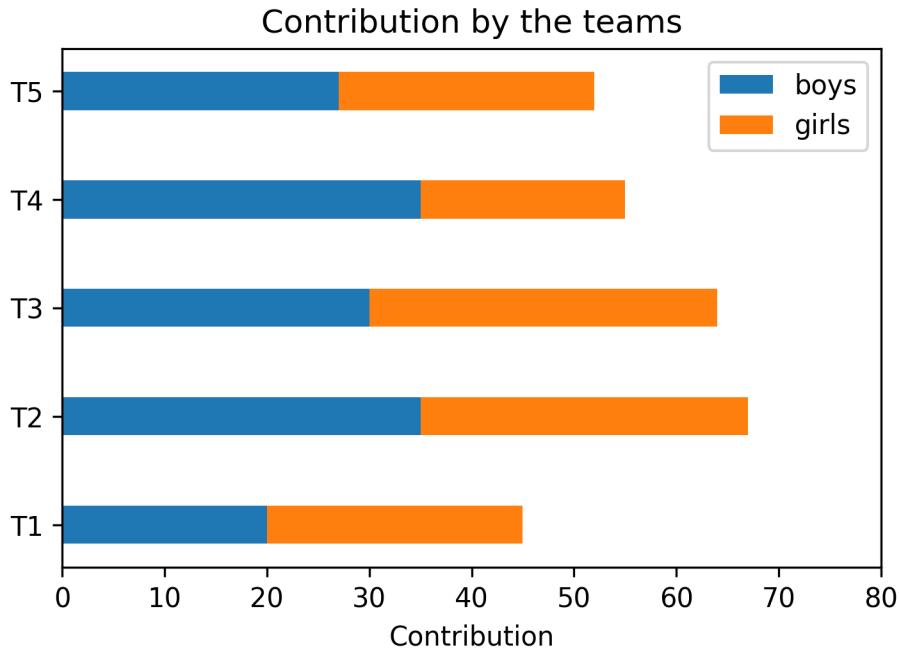
```
import numpy as np
import matplotlib.pyplot as plt

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(block=True)
```



```

import pandas as pd
import matplotlib.pyplot as plt

# Tworzenie przykładowej ramki danych
data = {
    'Grupa': ['Grupa A', 'Grupa B', 'Grupa C', 'Grupa D'],
    'Mężczyźni': [20, 35, 30, 35],
    'Kobiety': [25, 32, 34, 20]
}
df = pd.DataFrame(data)

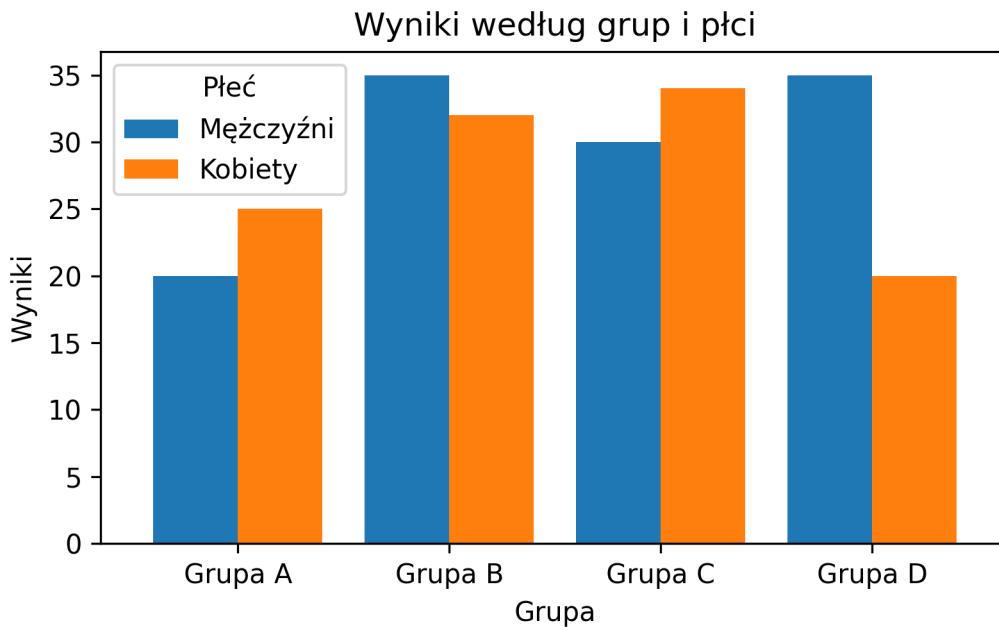
# Ustawienie indeksu na kolumnę 'Grupa'
df.set_index('Grupa', inplace=True)

# Tworzenie wykresu
ax = df.plot(kind='bar', width=0.8)

# Dodanie etykiet, tytułu i legendy
ax.set_ylabel('Wyniki')
ax.set_title('Wyniki według grup i płci')
ax.set_xticklabels(df.index, rotation=0)
ax.legend(title='Płeć')

```

```
plt.tight_layout()  
plt.show(block=True)
```



```
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Tworzenie przykładowej ramki danych  
data = {  
    'Grupa': ['Grupa A', 'Grupa B', 'Grupa C', 'Grupa D'],  
    'Mężczyźni': [20, 35, 30, 35],  
    'Kobiety': [25, 32, 34, 20]  
}  
df = pd.DataFrame(data)  
  
# Ustawienie indeksu na kolumnę 'Grupa'  
df.set_index('Grupa', inplace=True)  
  
# Tworzenie wykresu  
fig, ax = plt.subplots()  
  
# Słupki dla mężczyzn  
rects1 = ax.bar(df.index, df['Mężczyźni'], label='Mężczyźni')
```

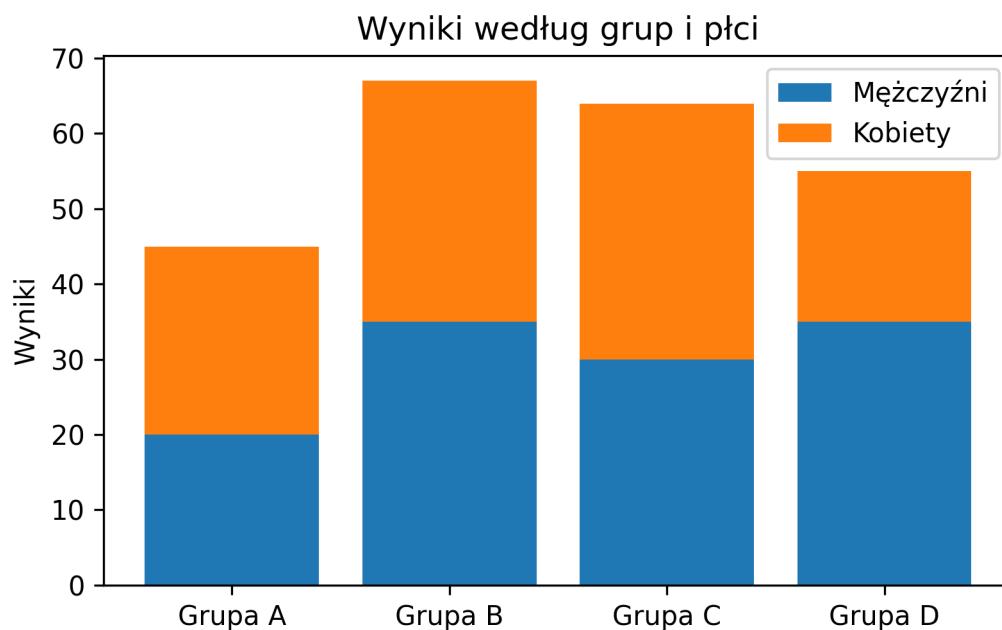
```

# Słupki dla kobiet, nałożone na słupki dla mężczyzn
rects2 = ax.bar(df.index, df['Kobiety'], bottom=df['Mężczyźni'], label='Kobiety')

# Dodanie etykiet, tytułu i legendy
ax.set_ylabel('Wyniki')
ax.set_title('Wyniki według grup i płci')
ax.legend()

plt.tight_layout()
plt.show(block=True)

```



37.4 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 kwartyl (Q1), medianę, trzeci kwartyl (Q3) i maksimum.

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

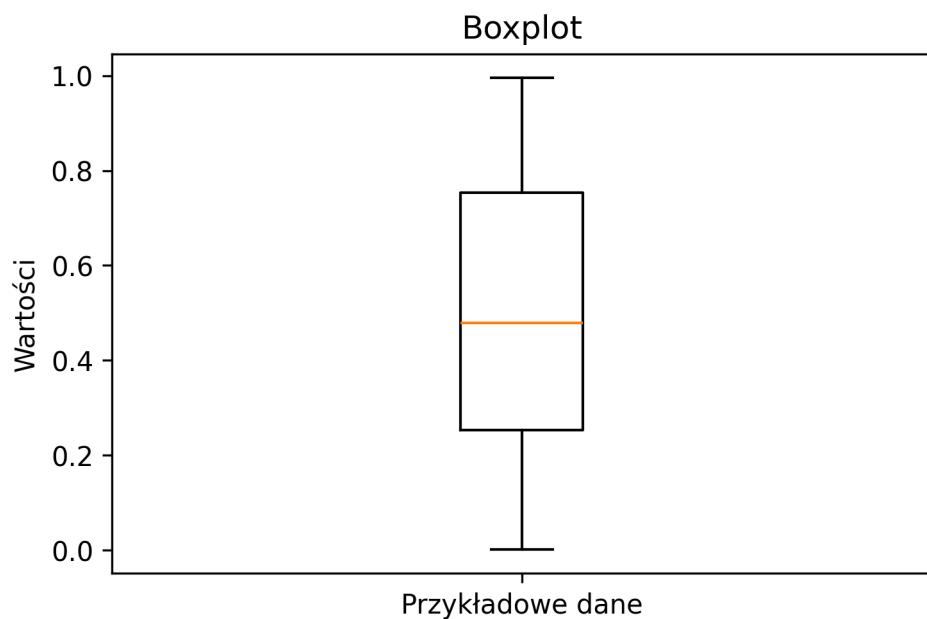
# Przykładowe dane
data = np.random.rand(100)

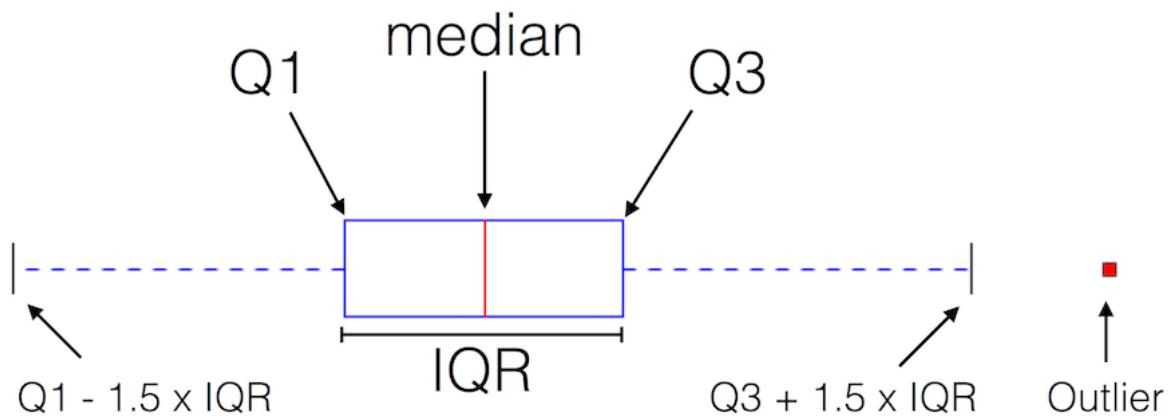
# Tworzenie wykresu
fig, ax = plt.subplots()

# Rysowanie boxplota
ax.boxplot(data)

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

# Wyświetlanie wykresu
plt.show(block=True)
```





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
 $\text{value} < Q1 - 1.5 \times IQR$ or
 $\text{value} > Q3 + 1.5 \times IQR$



Sebastian Raschka, 2016

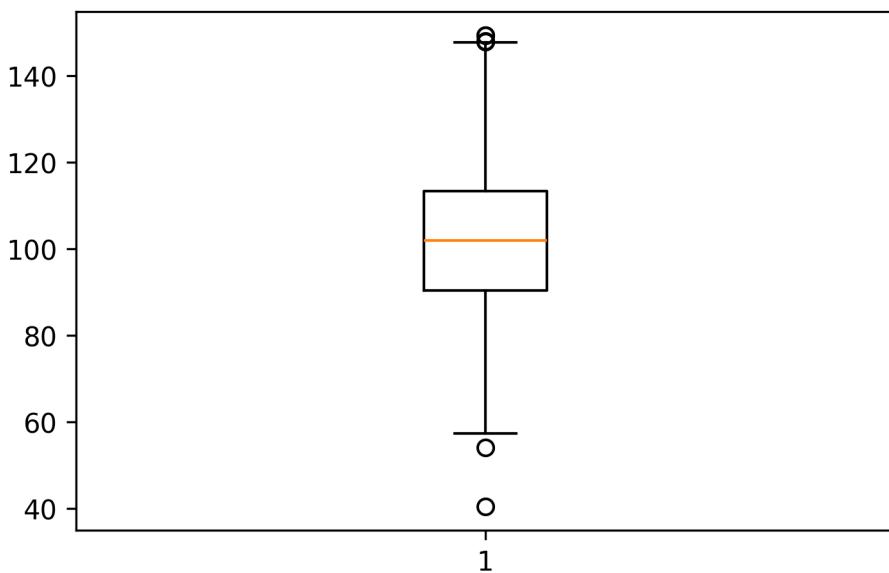
This work is licensed under a Creative Commons Attribution 4.0 International License

```
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(block=True)
```



37.5 Histogram

Wykres histogramu jest stosowany do przedstawiania rozkładu danych liczbowych, zarówno ciągłych, jak i dyskretnych. Histogram pokazujeczęstość 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 długie ogony (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.

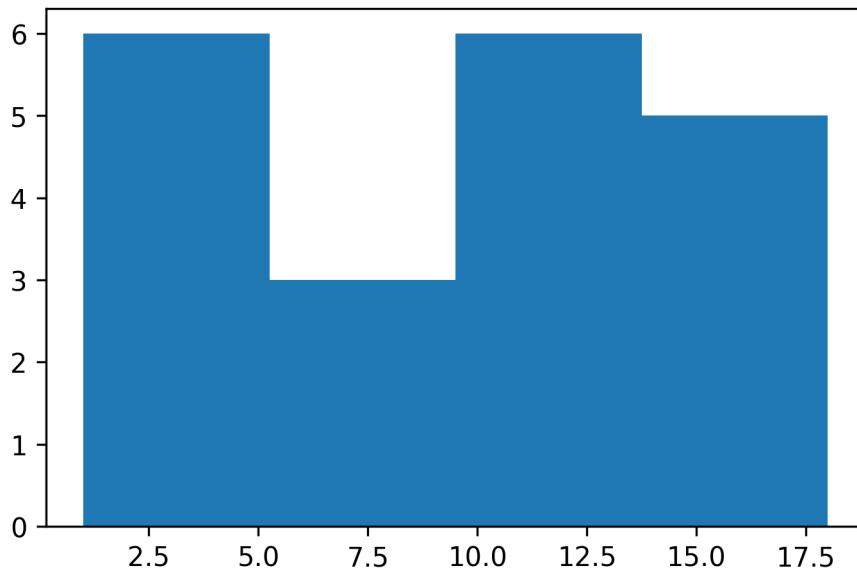
4. Szacowanie parametrów: Histogram może pomóc w szacowaniu parametrów rozkładu, takich jak średnia, mediana czy wariancja, co może być użyteczne w analizie statystycznej.

Warto zauważyc, że histogram jest odpowiedni dla danych liczbowych, ale nie jest przeznaczony do przedstawiania danych kategorialnych. 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(block=True)
```

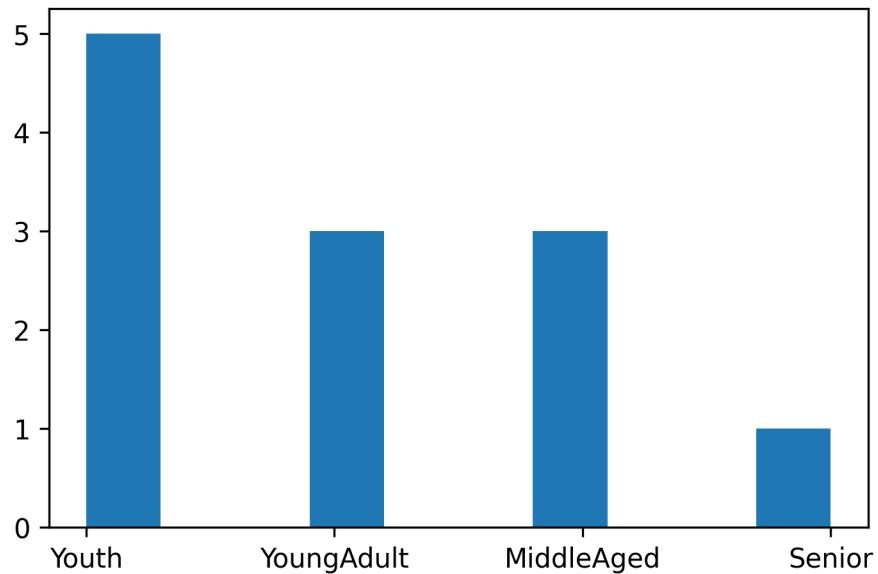


```
import pandas as pd
import matplotlib.pyplot as plt

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)
```

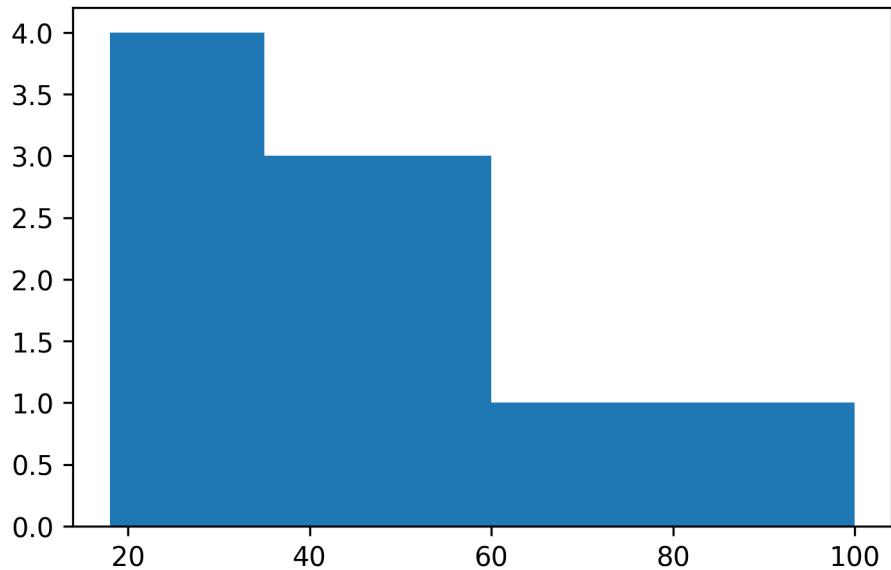
```
plt.hist(data)  
plt.show(block=True)
```

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



```
import matplotlib.pyplot as plt

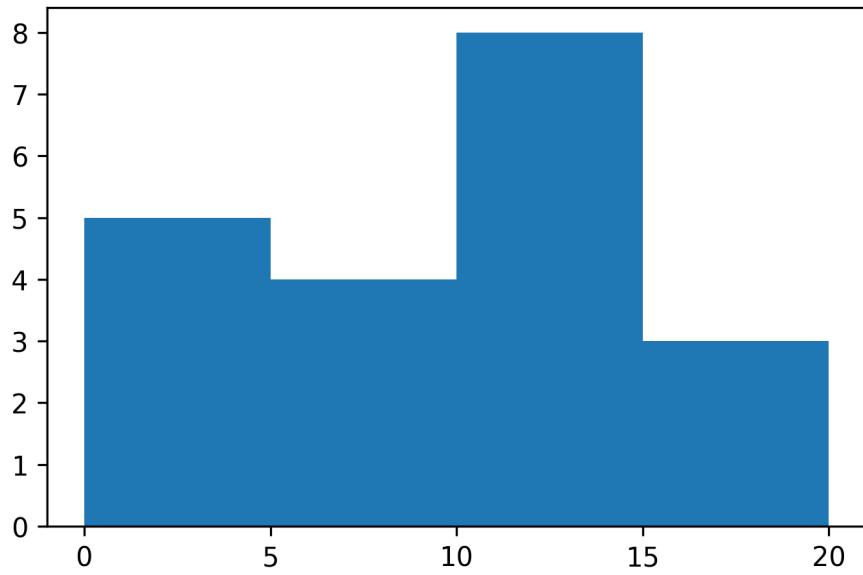
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
plt.hist(ages, bins=bins)
plt.show(block=True)
```



```
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]

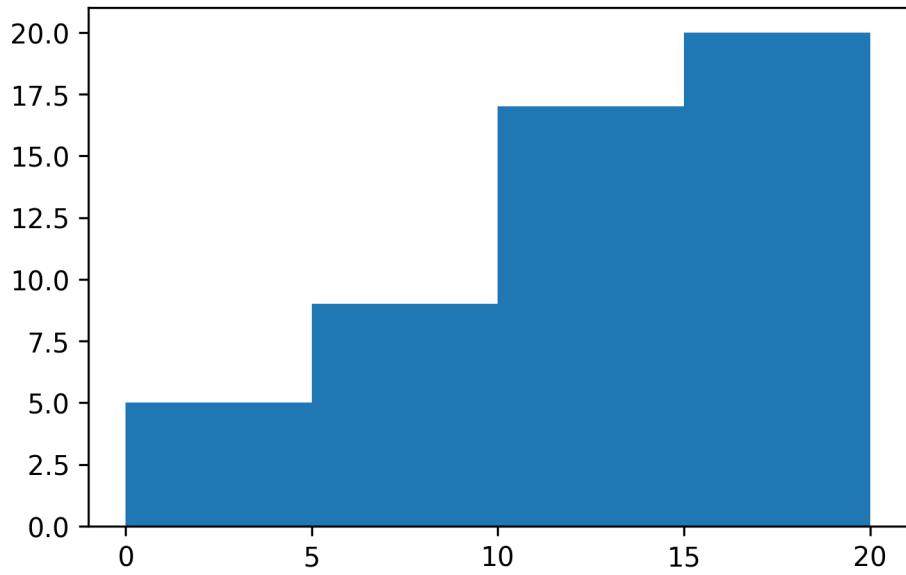
plt.hist(x, bins=[0, 5, 10, 15, 20])
plt.xticks([0, 5, 10, 15, 20])
plt.show(block=True)
```



```
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]

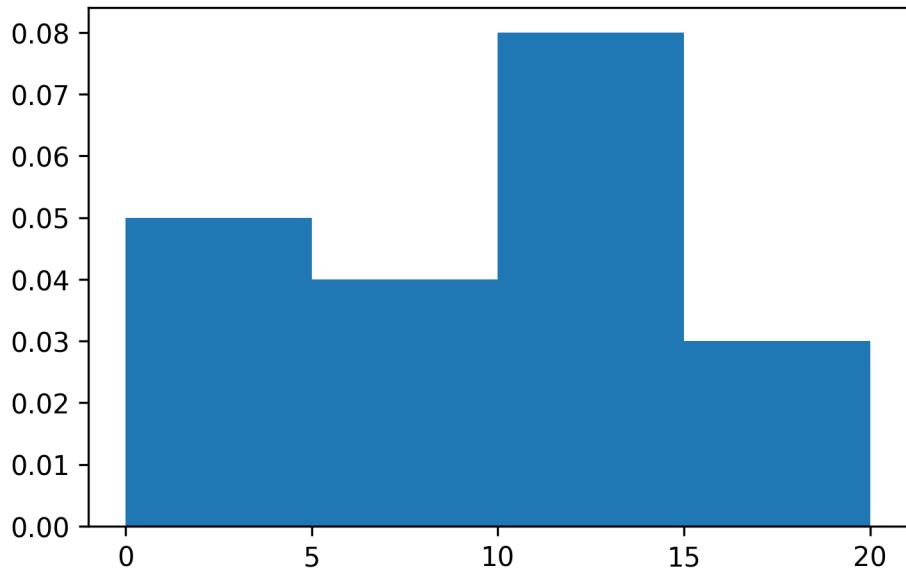
plt.hist(x, bins=[0, 5, 10, 15, 20], cumulative=True)
plt.xticks([0, 5, 10, 15, 20])
plt.show(block=True)
```



```
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]

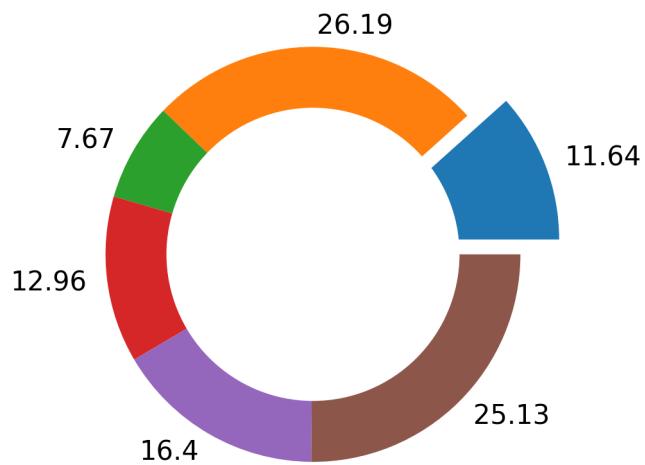
plt.hist(x, bins=[0, 5, 10, 15, 20], density=True)
plt.xticks([0, 5, 10, 15, 20])
plt.show(block=True)
```



37.6 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(block=True)
```



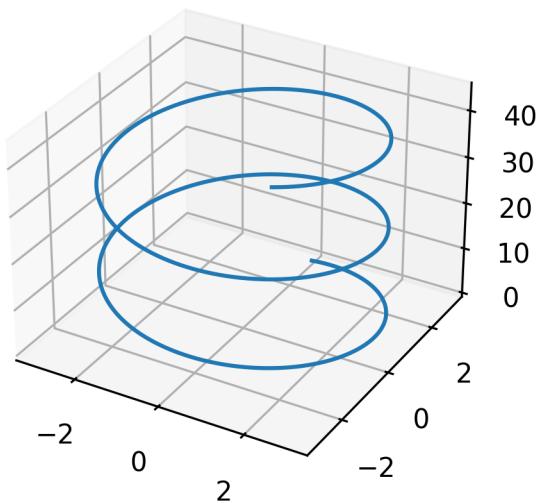
37.7 Wykresy w przestrzeni

37.7.1 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(block=True)
```

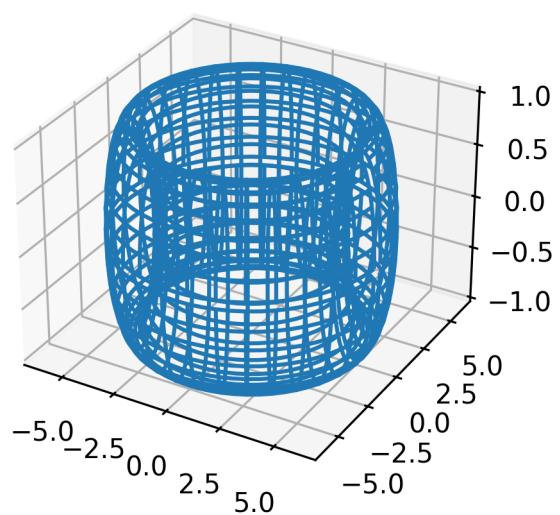


37.7.2 Torus

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

```
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(block=True)
```



Część VII

Zadania problemowe

38 Problem #1

Pliki są dostępne tutaj <https://github.com/pjastr/aiwd-book/tree/main/dataset>

Cel: wykres liniowy

Cel dodatkowy: różnice miedzy danymi szerokimi a długimi

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

# Wczytanie danych
dane = pd.read_csv('dataset/sprzedaz.csv')

# Dostosowanie kolorów dla lepszej czytelności
colors = ['steelblue', 'crimson', 'forestgreen']
markers = ['o', 's', '^']

# Utworzenie wykresu dla każdego regionu
for i, region in enumerate(dane['Region']):
    wartosci = dane.loc[dane['Region'] == region, ['Styczeń', 'Luty', 'Marzec', 'Kwiecień']]
    miesiące = ['Styczeń', 'Luty', 'Marzec', 'Kwiecień']

    plt.plot(miesiące, wartosci, marker=markers[i], linestyle='-', linewidth=2.5,
              color=colors[i], label=region, markersize=8)

# Dostosowanie wykresu
plt.title('Sprzedaż w poszczególnych regionach w pierwszym kwartale', fontsize=16, pad=20)
plt.xlabel('Miesiąc', fontsize=12)
plt.ylabel('Wartość sprzedaży', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)

# Poprawienie legendy
plt.legend(title='Region', fontsize=10, title_fontsize=12)
```

```

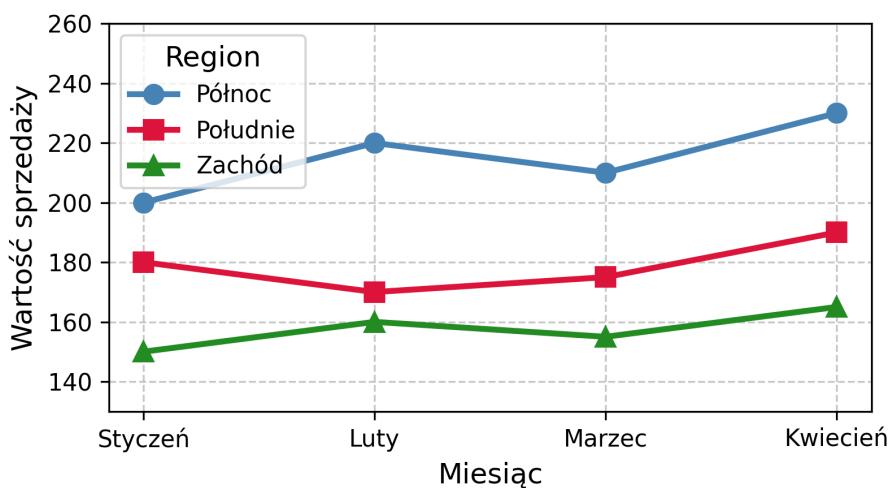
# Dostosowanie zakresu osi Y dla lepszej prezentacji
plt.ylim(min(dane[['Styczeń', 'Luty', 'Marzec', 'Kwiecień']].values.flatten()) - 20,
         max(dane[['Styczeń', 'Luty', 'Marzec', 'Kwiecień']].values.flatten()) + 30)

# Poprawienie układu
plt.tight_layout()

# Wyświetlenie wykresu
plt.show()

```

Sprzedaż w poszczególnych regionach w pierwszym kwartale



```

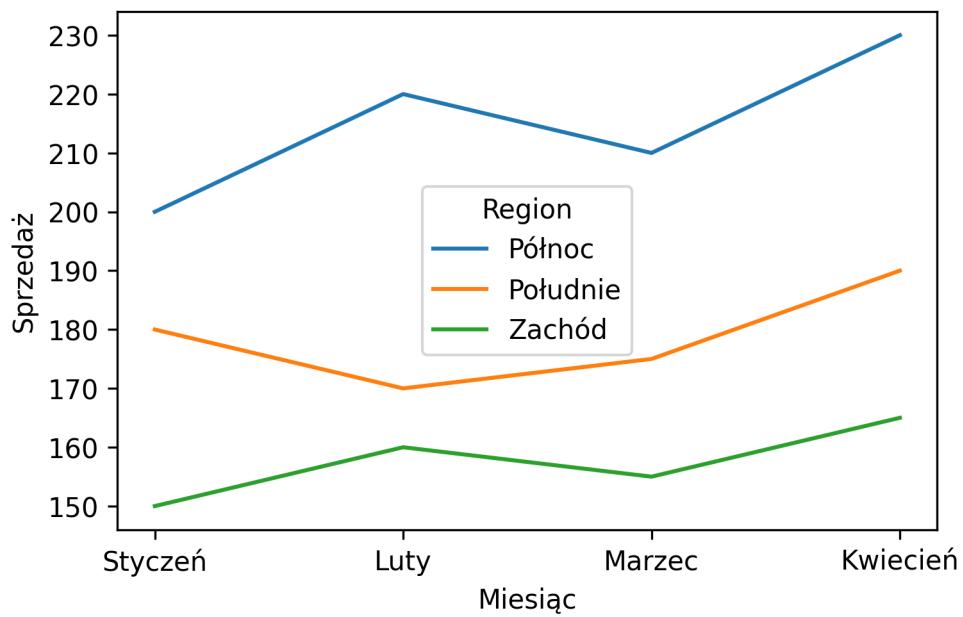
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('dataset/sprzedaz.csv')
df_long = df.melt(
    id_vars='Region',
    var_name='Miesiąc',
    value_name='Sprzedaż'
)
print(df_long)
sns.lineplot(
    data=df_long,
    x='Miesiąc',
    y='Sprzedaż',

```

```
    hue='Region'  
)  
plt.show()
```

	Region	Miesiąc	Sprzedaż
0	Północ	Styczeń	200
1	Południe	Styczeń	180
2	Zachód	Styczeń	150
3	Północ	Luty	220
4	Południe	Luty	170
5	Zachód	Luty	160
6	Północ	Marzec	210
7	Południe	Marzec	175
8	Zachód	Marzec	155
9	Północ	Kwiecień	230
10	Południe	Kwiecień	190
11	Zachód	Kwiecień	165



39 Problem #2

Pliki są dostępne tutaj <https://github.com/pjastr/aiwd-book/tree/main/dataset>

Cel: wykres liniowy

Cel dodatkowy: użycie funkcji pivot

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

# Wczytaj dane do DataFrame
df = pd.read_csv('dataset/economic_data.csv')

# Wyświetl pierwsze kilka wierszy oryginalnego DataFrame
print("Oryginalny DataFrame:")
print(df.head())

# Użyj pivot do przekształcenia danych
# Dla każdego wskaźnika tworzymy osobny pivot, aby otrzymać dane w formie odpowiedniej do wykresu

# Przekształcenie danych za pomocą pivot dla każdego wskaźnika
gdp_growth_pivot = df[df['Indicator'] == 'GDP Growth'].pivot(index='Year', columns='Country')
inflation_pivot = df[df['Indicator'] == 'Inflation'].pivot(index='Year', columns='Country')
unemployment_pivot = df[df['Indicator'] == 'Unemployment'].pivot(index='Year', columns='Country')

# Wyświetl przekształcone DataFrames
print("\nDataFrame GDP Growth po transformacji pivot:")
print(gdp_growth_pivot)

print("\nDataFrame Inflation po transformacji pivot:")
print(inflation_pivot)

print("\nDataFrame Unemployment po transformacji pivot:")
```

```

print(unemployment_pivot)

# Ustawienie stylu dla lepszej estetyki
sns.set_style("whitegrid")
plt.figure(figsize=(18, 12))

# Lista kolorów dla lepszego rozróżnienia krajów
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']

# Wykres wzrostu PKB
plt.subplot(3, 1, 1)
for i, country in enumerate(gdp_growth_pivot.columns):
    plt.plot(gdp_growth_pivot.index, gdp_growth_pivot[country], marker='o',
              linewidth=2.5, label=country, color=colors[i])

plt.title('Wzrost PKB (%) według krajów (2018-2023)', fontsize=16, fontweight='bold')
plt.ylabel('Wzrost PKB (%)', fontsize=14)
plt.grid(True, alpha=0.3)
plt.legend(loc='upper left', fontsize=12)
plt.axhline(y=0, color='black', linestyle='-', alpha=0.2) # Linia na poziomie 0%

# Wykres inflacji
plt.subplot(3, 1, 2)
for i, country in enumerate(inflation_pivot.columns):
    plt.plot(inflation_pivot.index, inflation_pivot[country], marker='s',
              linewidth=2.5, label=country, color=colors[i])

plt.title('Stopa inflacji (%) według krajów (2018-2023)', fontsize=16, fontweight='bold')
plt.ylabel('Stopa inflacji (%)', fontsize=14)
plt.grid(True, alpha=0.3)
plt.legend(loc='upper left', fontsize=12)

# Wykres bezrobocia
plt.subplot(3, 1, 3)
for i, country in enumerate(unemployment_pivot.columns):
    plt.plot(unemployment_pivot.index, unemployment_pivot[country], marker='^',
              linewidth=2.5, label=country, color=colors[i])

plt.title('Stopa bezrobocia (%) według krajów (2018-2023)', fontsize=16, fontweight='bold')

```

```

plt.xlabel('Rok', fontsize=14)
plt.ylabel('Stopa bezrobocia (%)', fontsize=14)
plt.grid(True, alpha=0.3)
plt.legend(loc='upper left', fontsize=12)

plt.tight_layout(pad=3.0)
plt.savefig('wskazniki_ekonomiczne.png', dpi=300, bbox_inches='tight')
plt.show()

```

Oryginalny DataFrame:

	Year	Country	Indicator	Value
0	2018	USA	GDP Growth	2.9
1	2018	USA	Inflation	2.4
2	2018	USA	Unemployment	3.9
3	2018	Germany	GDP Growth	1.1
4	2018	Germany	Inflation	1.9

DataFrame GDP Growth po transformacji pivot:

Country	Germany	Japan	USA
Year			
2018	1.1	0.6	2.9
2019	0.6	0.3	2.3
2020	-4.6	-4.5	-3.4
2021	2.9	1.6	5.7
2022	1.8	1.0	2.1
2023	0.3	1.9	2.5

DataFrame Inflation po transformacji pivot:

Country	Germany	Japan	USA
Year			
2018	1.9	1.0	2.4
2019	1.4	0.5	1.8
2020	0.5	0.0	1.2
2021	3.1	0.3	4.7
2022	6.9	2.5	8.0
2023	5.9	3.1	4.1

DataFrame Unemployment po transformacji pivot:

Country	Germany	Japan	USA
Year			
2018	3.4	2.4	3.9
2019	3.2	2.4	3.7

2020	4.2	2.8	8.1
2021	3.6	2.8	5.4
2022	3.0	2.6	3.6
2023	3.1	2.5	3.7

