



# Wizualizacja Danych 2024

Piotr Jastrzębski

2024-04-23

# Spis treści

<b>1</b>	<b>Wizualizacja Danych 2024</b>	<b>4</b>
<b>2</b>	<b>Trochę teorii...</b>	<b>5</b>
2.1	Test racjonalnego myślenia . . . . .	5
2.2	Analiza danych - podstawowe pojęcia . . . . .	5
2.2.1	Współczesne znaczenia słowa “statystyka”: . . . . .	5
2.2.2	“Masowość” . . . . .	6
2.2.3	Podział statystyki . . . . .	6
2.2.4	Zbiorowość/populacja . . . . .	6
2.2.5	Jednostka statyczna . . . . .	6
2.2.6	Cechy statystyczne . . . . .	7
2.2.7	Skale . . . . .	8
2.3	Rodzaje badań statystycznych . . . . .	10
2.4	Etapy badania statystycznego . . . . .	10
2.5	Analiza danych zastanych . . . . .	10
2.6	Proces analizy danych . . . . .	11
2.6.1	Zdefiniowanie wymagań . . . . .	11
2.6.2	Gromadzenie danych . . . . .	12
2.6.3	Przetwarzanie danych . . . . .	12
2.6.4	Właściwa analiza danych . . . . .	12
2.6.5	Raportowanie i dystrybucja wyników . . . . .	12
2.7	Skąd brać dane? . . . . .	13
2.8	Koncepcja “Tidy data” . . . . .	13
2.8.1	Zasady “czystych danych” . . . . .	13
2.8.2	Przykłady nieuporządkowanych danych . . . . .	14
2.8.3	Długie czy szerokie dane? . . . . .	14
2.9	Parę rad na dobre prezentacje . . . . .	14
2.9.1	Współczynnik kłamstwa . . . . .	14
2.9.2	Współczynnik kłamstwa . . . . .	15
2.10	Jak tworzyć? . . . . .	16
2.11	Bibliografia . . . . .	16
<b>3</b>	<b>NumPy</b>	<b>17</b>
3.1	Import biblioteki NumPy . . . . .	17
3.2	Lista a tablica . . . . .	18

3.3	Atrybuty tablic <code>ndarray</code> . . . . .	19
3.4	Typy danych . . . . .	20
3.5	Tworzenie tablic . . . . .	21
3.6	Indeksowanie, “krojenie” . . . . .	29
3.7	Modyfikacja kształtu i rozmiaru . . . . .	33
3.8	Broadcasting . . . . .	39
3.9	Funkcje uniwersalne . . . . .	42
3.10	Statystyka i agregacja . . . . .	42
3.11	Wyrażenia warunkowe . . . . .	43
3.12	Działania na zbiorach . . . . .	43
3.13	Operacje tablicowe . . . . .	43
3.14	Alegbra liniowa . . . . .	43
3.15	Funkcja na stringach . . . . .	43
3.16	Data i czas . . . . .	43
3.17	Pseudolosowe . . . . .	44
<b>4</b>	<b>NumPy - zadania</b>	<b>45</b>
<b>5</b>	<b>Pandas</b>	<b>46</b>
5.1	Podstawowe byty . . . . .	46
5.2	Uzupełnianie braków . . . . .	52
5.3	Obsługa plików csv . . . . .	53
5.4	Obsługa plików z Excela . . . . .	53
5.5	Repozytorium z testowymi plikami . . . . .	54
5.6	Operacje manipulacyjne . . . . .	54
5.7	“Tidy data” . . . . .	64
5.8	Obsługa brakujących danych . . . . .	64
5.9	Usuwanie duplikatów . . . . .	66
5.10	Zastępowanie wartościami . . . . .	67
5.11	Dyskretyzacja i podział na koszyki . . . . .	68
5.12	Wykrywanie i filtrowanie elementów odstających . . . . .	70
5.13	Zmiana typu w kolumnie . . . . .	71
5.14	Zmiana znaku kategoriach . . . . .	73

# 1 Wizualizacja Danych 2024

Materiały na semestr letni - rok akademicki 2023/24.

## 2 Trochę teorii...

### 2.1 Test racjonalnego myślenia

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

---

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

### 2.2 Analiza danych - podstawowe pojęcia

#### 2.2.1 Współczesne znaczenia słowa “statystyka”:

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

### 2.2.2 “Masowość”

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

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

### 2.2.3 Podział statystyki

Statystyka - dyscyplina naukowa - podział:

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

### 2.2.4 Zbiorowość/populacja

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

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

### 2.2.5 Jednostka statyczna

Jednostka statyczna: każdy z elementów zbiorowości statystycznej.

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

## 2.2.6 Cechy statystyczne

### Cechy statystyczne

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

### Cechy stałe

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

### Cechy zmienne

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

### Ważne:

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

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

### Podział cech zmiennych:

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

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

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

### **Podział cech mierzalnych:**

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

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

## **2.2.7 Skale**

### **Skala pomiarowa**

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

### **Skala nominalna**

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

Religia	Kod
Chrześcijaństwo	1
Islam	2
Buddyzm	3

### **Skala porządkowa**



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

Wykształcenie	Kod
Podstawowe	1
Średnie	2
Wyższe	3

Dochód	Kod
Niski	1
Średni	2
Wysoki	3

### Skala przedziałowa (interwałowa)

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

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

### Skala ilorazowa (stosunkowa)

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

Produkt	Cena w zł
Chleb	3
Masło	8
Gruszki	5

## 2.3 Rodzaje badań statystycznych

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

## 2.4 Etapy badania statystycznego

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

## 2.5 Analiza danych zastanych

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

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

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

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

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

## 2.6 Proces analizy danych

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

### 2.6.1 Zdefiniowanie wymagań

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

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

### **2.6.2 Gromadzenie danych**

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

### **2.6.3 Przetwarzanie danych**

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

### **2.6.4 Właściwa analiza danych**

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

### **2.6.5 Raportowanie i dystrybucja wyników**

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

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

## 2.7 Skąd brać dane?

Darmowa repozytoria danych:

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

Przydatne strony:

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

## 2.8 Koncepcja “Tidy data”

Koncepcja czyszczenia danych (ang. tidy data):

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

### 2.8.1 Zasady “czystych danych”

Idealne dane są zaprezentowane w tabeli:

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

Na co powinniśmy zwrócić uwagę?

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

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

### 2.8.2 Przykłady nieuporządkowanych danych

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

Nagłówki kolumn muszą odpowiadać cechom, a nie wartościom zmiennych.

### 2.8.3 Długie czy szerokie dane?

[https://seaborn.pydata.org/tutorial/data\\_structure.html#long-form-vs-wide-form-data](https://seaborn.pydata.org/tutorial/data_structure.html#long-form-vs-wide-form-data)

## 2.9 Parę rad na dobre prezentacje

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

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

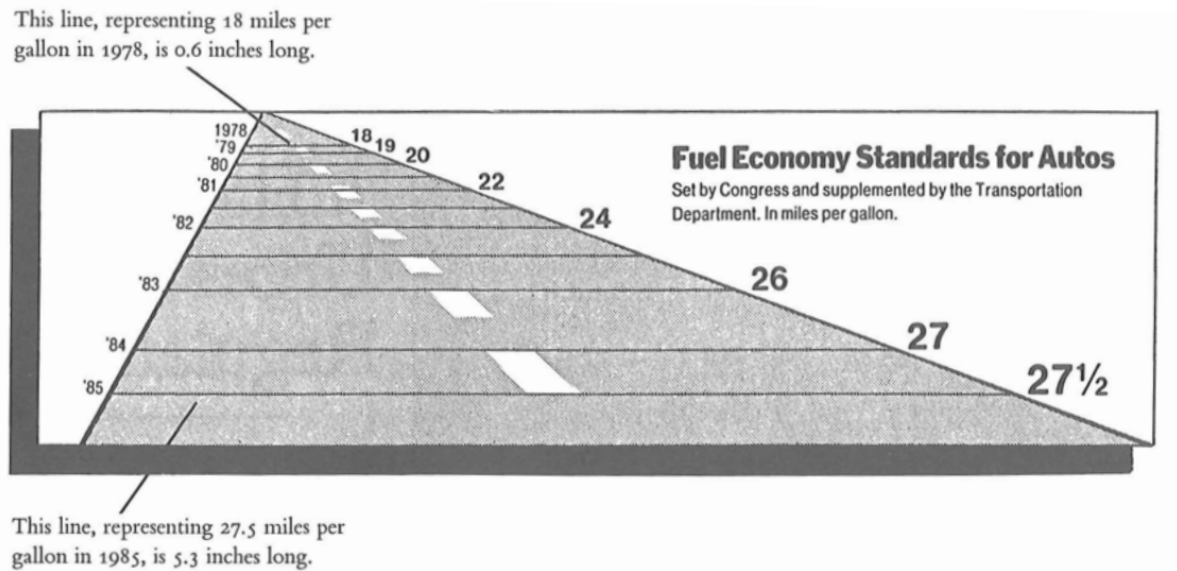
### 2.9.1 Współczynnik kłamstwa

[https://www.facebook.com/janinadaily/photos/a.1524649467770881/2836063543296127/?pav=0&eav=AfbVIDx5un8ZOklKI9c-B1jP4nOoNa2QMmJmjoA-291JNNgM1L\\_NmoCGMS\\_mJOy4xjo&\\_rdr](https://www.facebook.com/janinadaily/photos/a.1524649467770881/2836063543296127/?pav=0&eav=AfbVIDx5un8ZOklKI9c-B1jP4nOoNa2QMmJmjoA-291JNNgM1L_NmoCGMS_mJOy4xjo&_rdr)

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

[https://infovis-wiki.net/wiki/Lie\\_Factor](https://infovis-wiki.net/wiki/Lie_Factor)

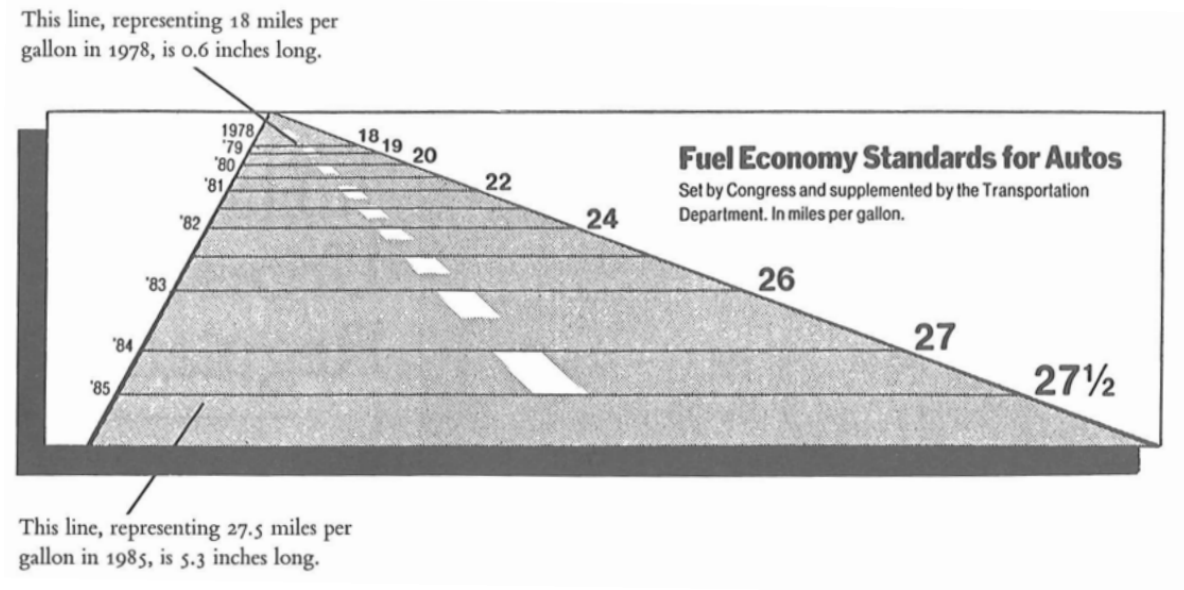
## 2.9.2 Współczynnik kłamstwa



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

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

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



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

## 2.10 Jak tworzyć?

- [https://bookdown.org/rudolf\\_von\\_ems/jak\\_sie\\_nie\\_dac/stats\\_graphs.html](https://bookdown.org/rudolf_von_ems/jak_sie_nie_dac/stats_graphs.html)
- <https://www.data-to-viz.com/>
- <https://100.datavizproject.com/>

## 2.11 Bibliografia

- <https://pl.wikipedia.org/wiki/Wizualizacja>
- [https://mfiles.pl/pl/index.php/Analiza\\_danych](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](https://en.wikipedia.org/wiki/Cognitive_reflection_test), dostęp online 20.03.2023.
- <https://qlikblog.pl/edward-tufte-dobre-praktyki-prezentacji-danych/>, dostęp online 20.03.2023.



## 3 NumPy

NumPy jest biblioteką Pythona służącą do obliczeń naukowych.

Zastosowania:

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

### 3.1 Import biblioteki NumPy

```
import numpy as np
```

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

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

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

```
import numpy as np
```

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

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

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

①

②

```

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

```

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

```

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

```

## 3.2 Lista a tablica

```

import numpy as np
import time

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

```

```

--- 0.0010008811950683594 seconds ---
--- 0.0524141788482666 seconds ---

```

### 3.3 Atrybuty tablic ndarray

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

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

```

import numpy as np

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

```

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

```
import numpy as np

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

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

NumPy nie wspiera postrzępionych tablic! Poniższy kod wygeneruje błąd:

```
import numpy as np

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

### 3.4 Typy danych

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

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

---

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

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

```
import numpy as np

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

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

### 3.5 Tworzenie tablic

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

```
import numpy as np

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

```

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

```

```

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

```

`np.zeros` - tworzy tablicę wypełnioną zerami

```

import numpy as np

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

```

```

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

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

```

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

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

```
import numpy as np

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

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

`np.diag` - tworzy tablicę odpowiadającą macierzy diagonalnej

```
import numpy as np

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

```

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

```

```

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

```

`np.arange` - tablica wypełniona równomiernymi wartościami

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

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

```

import numpy as np

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

```



```

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

```

```

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

```

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

```

import numpy as np

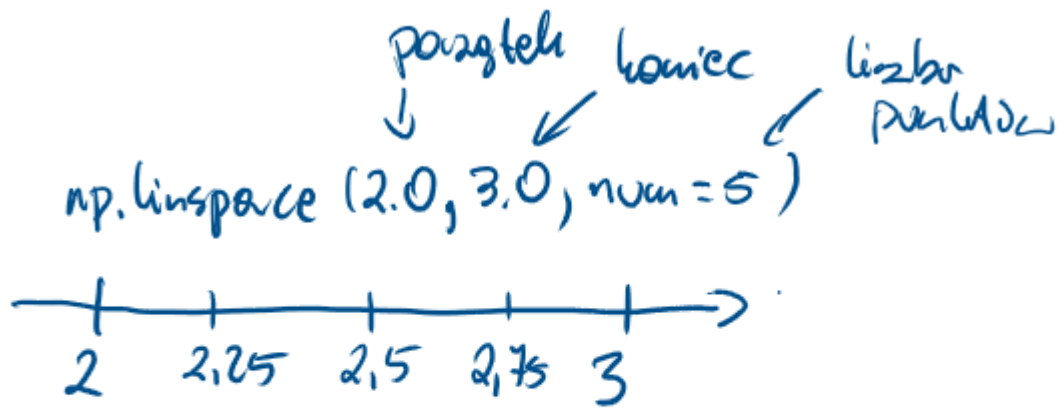
a = np.linspace(2.0, 3.0, num=5)
print(a)
b = np.linspace(2.0, 3.0, num=5, endpoint=False)
print(b)
c = np.linspace(10, 20, num=4)
print(c)
d = np.linspace(10, 20, num=4, dtype=int)
print(d)

```

```

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

```



Wzrost: odcinek jest dzielony  
na  $\text{num}-1$  części!

`endpoint = False`

- wyłącza ostatni punkt  
(z prawej strony)

Wtedy podział odbywa się  
na  $\text{num}$  części.

`dtype` ← ustala typ

zwykle używa się `int`  
(wyniki mają uciętą część  
ułamkową)

`np.logspace` - tablica wypełniona wartościami wg skali logarytmicznej

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

axis=0)

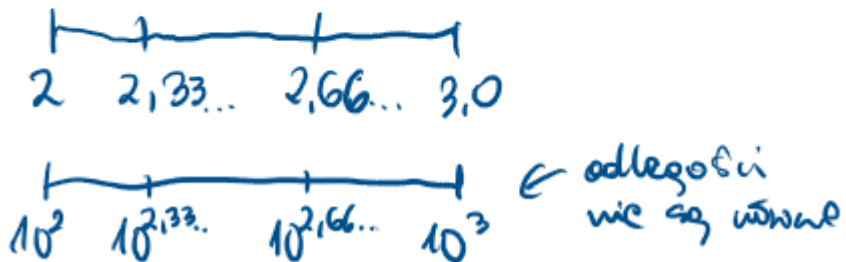
```
import numpy as np

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

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

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

*Domyślnie podstawa logarytmu 10*



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

```
import numpy as np

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

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

`np.identity` - tablica przypominająca macierz jednostkową

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

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

```
a  
[[1. 0. 0. 0.]  
 [0. 1. 0. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 0. 1.]]
```

```
b  
[[0. 1. 0. 0.]  
 [0. 0. 1. 0.]  
 [0. 0. 0. 1.]  
 [0. 0. 0. 0.]]
```

```
c  
[[0. 0. 1. 0.]  
 [0. 0. 0. 1.]  
 [0. 0. 0. 0.]  
 [0. 0. 0. 0.]]
```

```
d  
[[0. 0. 0. 0.]  
 [1. 0. 0. 0.]  
 [0. 1. 0. 0.]  
 [0. 0. 1. 0.]]
```

## 3.6 Indeksowanie, “krojenie”

```
import numpy as np

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

①

②

③

④

⑤

⑥

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

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

```
import numpy as np

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

①

②

③

④

⑤

⑥

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

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

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

```
import numpy as np

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

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

```
import numpy as np

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

```

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

```

```

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

```

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

```

import numpy as np

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

```

```

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

```

Naprawa:

```
import numpy as np

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

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

Indeksowanie logiczne (fancy indexing)

```
import numpy as np

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

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

```
import numpy as np

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



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

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

```
import numpy as np

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

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

### 3.7 Modyfikacja kształtu i rozmiaru

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

```
import numpy as np

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

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

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

```
import numpy as np

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

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

```
import numpy as np
```

```

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

```

```

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

```

```

import numpy as np

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

```

```

h

```

```

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

```

```

import numpy as np

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

```

```

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

```

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

```
import numpy as np

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

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

```
import numpy as np

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

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

```
import numpy as np

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

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

```
import numpy as np

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

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

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

### 3.8 Broadcasting

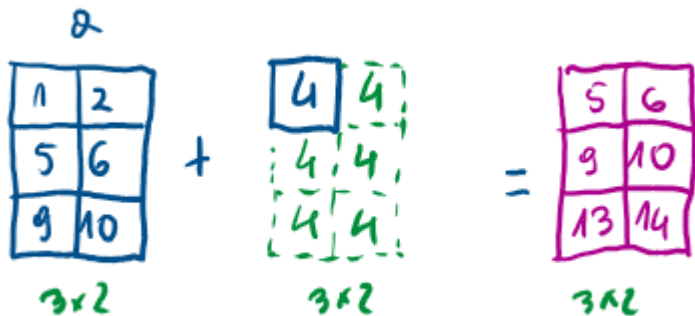
Rozważane warianty są przykładowe.

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

```
import numpy as np

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

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



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

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

```

import numpy as np

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

```

```

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

```



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

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

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]
```

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

Diagram illustrating matrix multiplication:

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

Matrix  $a$  values:  $\begin{bmatrix} 5 \\ 2 \\ -3 \end{bmatrix}$

Matrix  $b$  values:  $\begin{bmatrix} 3 & -2 & 1 & 2 & 4 \end{bmatrix}$

Resulting matrix values:  $\begin{bmatrix} 8 & 3 & 6 & 7 & 9 \\ 5 & 0 & 3 & 4 & 6 \\ 0 & -5 & -2 & -1 & 1 \end{bmatrix}$

### 3.9 Funkcje uniwersalne

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

### 3.10 Statystyka i agregacja

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

### 3.11 Wyrażenia warunkowe

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

### 3.12 Działania na zbiorach

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

### 3.13 Operacje tablicowe

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

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

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

### 3.14 Algebra liniowa

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

### 3.15 Funkcja na stringach

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

### 3.16 Data i czas

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

## 3.17 Pseudolosowe

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

Bibliografia:

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

## 4 NumPy - zadania

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

# 5 Pandas

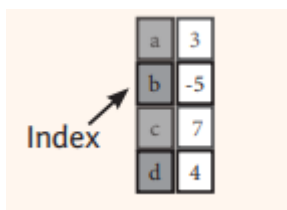
Pandas jest biblioteką Pythona służącą do analizy i manipulowania danymi

Import:

```
import pandas as pd
```

## 5.1 Podstawowe byty

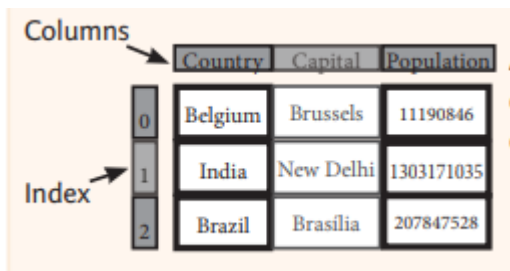
Seria - Series



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

a	3
b	-5
c	7
d	4

Ramka danych - DataFrame



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

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

```
import pandas as pd
import numpy as np

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

```

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

```

```

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

```

```

import pandas as pd
import numpy as np

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

```

```

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

```

```

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

```

```

import pandas as pd

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

```

```

key1    350
key2    700
key3     70
dtype: int64

```

```

import pandas as pd

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

```



```

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

```

```

import pandas as pd

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

```

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

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

```

import pandas as pd

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

```

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

```
Country
0 Belgium
--
```

```
Country
0 Belgium
--
```

```
Country      Brazil
Population    207847528
Capital      Brasília
Name: 2, dtype: object
--
```

```
0 Brussels
1 New Delhi
2 Brasília
Name: Capital, dtype: object
--
```

```
New Delhi
```

#### 1. loc:

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

#### 2. iloc:

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

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

```
import pandas as pd

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

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

```
import pandas as pd

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

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

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

## 5.2 Uzupełnianie braków

```
import pandas as pd

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

```
a    10.0
```

```

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

```

## 5.3 Obsługa plików csv

Funkcja `pandas.read_csv`

Dokumentacja: [link](#)

Wybrane argumenty:

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

Zapis `pandas.DataFrame.to_csv`

Dokumentacja: [link](#)

## 5.4 Obsługa plików z Excela

Funkcja `pandas.read_excel`

[https://pandas.pydata.org/docs/reference/api/pandas.read\\_excel.html](https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html)

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

Wybrane argumenty:

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

## 5.5 Repozytorium z testowymi plikami

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

## 5.6 Operacje manipulacyjne

Ściągowka [https://pandas.pydata.org/Pandas\\_Cheat\\_Sheet.pdf](https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf)

- `merge`

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

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

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

Gdzie:

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

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

```
import pandas as pd

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

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

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

```

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

```

import pandas as pd

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

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

print(df1)

print(df2)

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

print("Inner join")
print(inner_merged_df)

print("Outer join")
print(outer_merged_df)

print("Left join")
print(left_merged_df)

print("Right join")
print(right_merged_df)

```

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

	key	C	D
0	K0	C0	D0



```

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

```

- `join`

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

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

Podstawowe użycie tej metody wygląda następująco:

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

Gdzie:

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

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

```
import pandas as pd

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

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

print(df1)

print(df2)

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

```

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

- **concat**

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

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

Podstawowe użycie tej metody wygląda następująco:

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

Gdzie:

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

```
import pandas as pd

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

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

print(df1)

print(df2)
```

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

```

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

```
import pandas as pd

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

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

print(df1)

print(df2)

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

print(concatenated_df_axis1)
print(concatenated_df_keys)
```

	A	B
0	A0	B0
1	A1	B1
2	A2	B2

	C	D
0	C0	D0
1	C1	D1
2	C2	D2

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

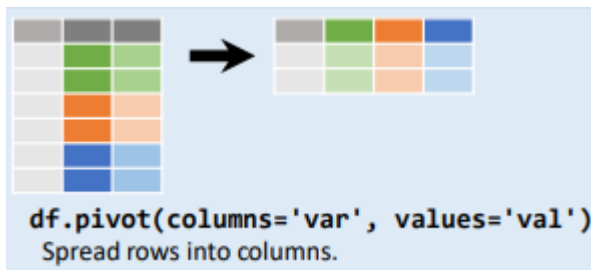
	A	B	C	D
df1 0	A0	B0	NaN	NaN
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN

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

- `pivot`

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



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

Podstawowe użycie tej metody wygląda następująco:

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

Gdzie:

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

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

```
import pandas as pd

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

print(df)

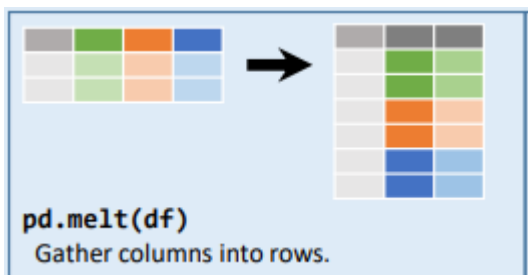
pivot_df = df.pivot(index='foo', columns='bar', values='baz')
print(pivot_df)
```

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

- **wide\_to\_long**

[https://pandas.pydata.org/docs/reference/api/pandas.wide\\_to\\_long.html](https://pandas.pydata.org/docs/reference/api/pandas.wide_to_long.html)

- **melt**



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

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

Podstawowe użycie tej metody wygląda następująco:

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

Gdzie:

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

```
import pandas as pd

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

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

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

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

## 5.7 “Tidy data”

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

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

## 5.8 Obsługa brakujących danych

```
import numpy as np
import pandas as pd

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

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



3 avocado  
dtype: object

---

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

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

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

```

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

```

## 5.9 Usuwanie duplikatów

```

import pandas as pd

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

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

```

```

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

```

## 5.10 Zastępowanie wartościami

```
import pandas as pd
import numpy as np

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

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

```

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

```

## 5.11 Dyskretyzacja i podział na koszyki

```

import pandas as pd

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

```

```

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

```

---

```

import pandas as pd

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

```

```

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

```

```

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

```

---

```

import pandas as pd
import numpy as np

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

```

```

[(0.5, 0.72], (0.5, 0.72], (0.058, 0.28], (0.058, 0.28], (0.5, 0.72], ..., (0.5, 0.72], (0.058, 0.28],
Length: 20
Categories (4, interval[float64, right]): [(0.058, 0.28] < (0.28, 0.5] < (0.5, 0.72] < (0.72, 1.0]]

```

---

```

import pandas as pd
import numpy as np

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

```

```

[(-2.8129999999999997, -0.591], (0.726, 2.691], (0.0845, 0.726], (-2.8129999999999997, -0.591],
Length: 1000
Categories (4, interval[float64, right]): [(-2.8129999999999997, -0.591] < (-0.591, 0.0845] < (0.0845, 0.726] < (0.726, 2.691]
(-2.8129999999999997, -0.591]      250
(-0.591, 0.0845]                  250
(0.0845, 0.726]                   250
(0.726, 2.691]                    250
Name: count, dtype: int64

```

## 5.12 Wykrywanie i filtrowanie elementów odstających

```
import pandas as pd
import numpy as np

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

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.005420	0.031651	0.020118	0.030635
std	1.010807	0.982027	1.032444	0.972840
min	-2.963569	-3.415895	-3.278964	-2.695732
25%	-0.677131	-0.627356	-0.689803	-0.622615
50%	0.047005	0.005774	0.011071	0.006582
75%	0.690098	0.699685	0.702852	0.720922
max	3.641686	3.138257	3.801324	2.754552

----

162	-3.024924			
189	-3.278964			
220	-3.184234			
892	3.205727			
977	3.801324			

Name: 2, dtype: float64

----

	0	1	2	3
162	-0.163157	-0.320521	-3.024924	0.775903
189	0.629337	-0.562096	-3.278964	-1.562634
220	1.612155	-0.587312	-3.184234	1.329267
293	-0.610784	-3.116327	1.597597	-1.218383
323	1.241982	-3.415895	-0.668706	0.894169
367	3.641686	0.367046	-0.717903	2.427464
835	-1.896540	-3.118797	-0.751782	0.586118
852	0.413255	3.138257	-1.107876	1.417230
892	0.698543	-0.201659	3.205727	-0.723692
964	3.281576	-0.546970	0.092283	1.663353

977 0.810734 0.910280 3.801324 0.933268

## 5.13 Zmiana typu w kolumnie

```
import pandas as pd

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

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

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

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

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

Oryginalna ramka danych:

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

Ramka danych po zmianie typów:

	A	B	C
0	1	1.0	x

```

1  2  2.0  y
2  3  3.0  z
3  4  4.0  x
4  5  5.0  y
5  6  6.0  z

```

```

import pandas as pd

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

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

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

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

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

```

Oryginalna ramka danych:

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

Ramka danych po zmianie typów:

	A	B	C
0	1	7.5	x



1	2	8.5	y
2	3	9.5	z
3	4	10.5	x
4	5	11.5	y
5	6	12.5	z

## 5.14 Zmiana znaku kategoriach

```
import pandas as pd

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

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

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

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

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

Oryginalna ramka danych:

	A	B	C
0	abc	1.23	xyz
1	def	4.56	uvw
2	ghi	7.89	rst
3	jkl	0.12	opq
4	mno	3.45	lmn
5	pqr	6.78	ijk

Ramka danych po modyfikacji:

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

Bibliografia:

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