



Wizualizacja Danych 2024

Piotr Jastrzębski

2024-04-02

Spis treści

1	Wizualizacja Danych 2024	4
2	Trochę teorii...	5
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
3	NumPy	17
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
4	NumPy - zadania	45

1 Wizualizacja Danych 2024

Materiały na semestr letni - rok akademicki 2023/24.

2 Trochę teorii...

2.1 Test racjonalnego myślenia

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

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

2.2 Analiza danych - podstawowe pojęcia

2.2.1 Współczesne znaczenia słowa “statystyka”:

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

2.2.2 “Masowość”

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

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

2.2.3 Podział statystyki

Statystyka - dyscyplina naukowa - podział:

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

2.2.4 Zbiorowość/populacja

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

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

2.2.5 Jednostka statyczna

Jednostka statystyczna: każdy z elementów zbiorowości statystycznej.

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

2.2.6 Cechy statystyczne

Cechy statystyczne

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

Cechy stałe

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

Cechy zmienne

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

Ważne:

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

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

Podział cech zmiennych:

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

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

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

Podział cech mierzalnych:

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

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

2.2.7 Skale

Skala pomiarowa

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

Skala nominalna

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

Religia	Kod
Chrześcijaństwo	1
Islam	2
Buddyzm	3

Skala porządkowa

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

Wykształcenie	Kod
Podstawowe	1
Średnie	2
Wyższe	3

Dochód	Kod
Niski	1
Średni	2
Wysoki	3

Skala przedziałowa (interwałowa)

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

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

Skala ilorazowa (stosunkowa)

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

Produkt	Cena w zł
Chleb	3
Masło	8
Gruszki	5

2.3 Rodzaje badań statystycznych

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

2.4 Etapy badania statystycznego

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

2.5 Analiza danych zastanych

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

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

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

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

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

2.6 Proces analizy danych

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

2.6.1 Zdefiniowanie wymagań

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

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

2.6.2 Gromadzenie danych

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

2.6.3 Przetwarzanie danych

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

2.6.4 Właściwa analiza danych

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

2.6.5 Raportowanie i dystrybucja wyników

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

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

2.7 Skąd brać dane?

Darmowa repozytoria danych:

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

Przydatne strony:

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

2.8 Koncepcja “Tidy data”

Koncepcja czyszczenia danych (ang. tidy data):

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

2.8.1 Zasady “czystych danych”

Idealne dane są zaprezentowane w tabeli:

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

Na co powinniśmy zwrócić uwagę?

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

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

2.8.2 Przykłady nieuporządkowanych danych

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

Nagłówki kolumn muszą odpowiadać cechom, a nie wartościom zmiennych.

2.8.3 Długie czy szerokie dane?

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

2.9 Parę rad na dobre prezentacje

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

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

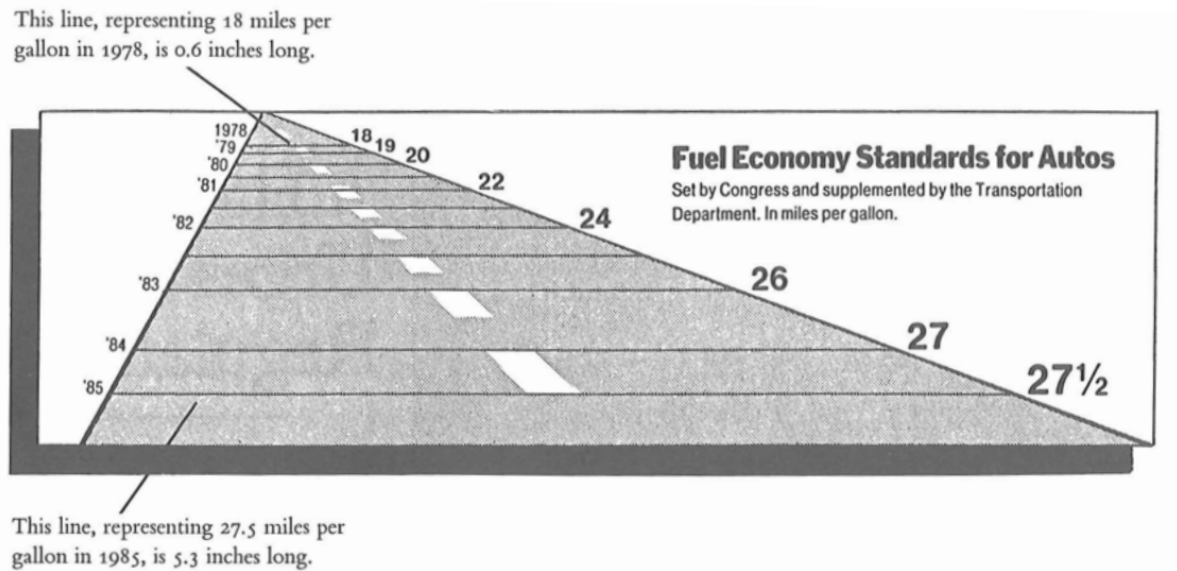
2.9.1 Współczynnik kłamstwa

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

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

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

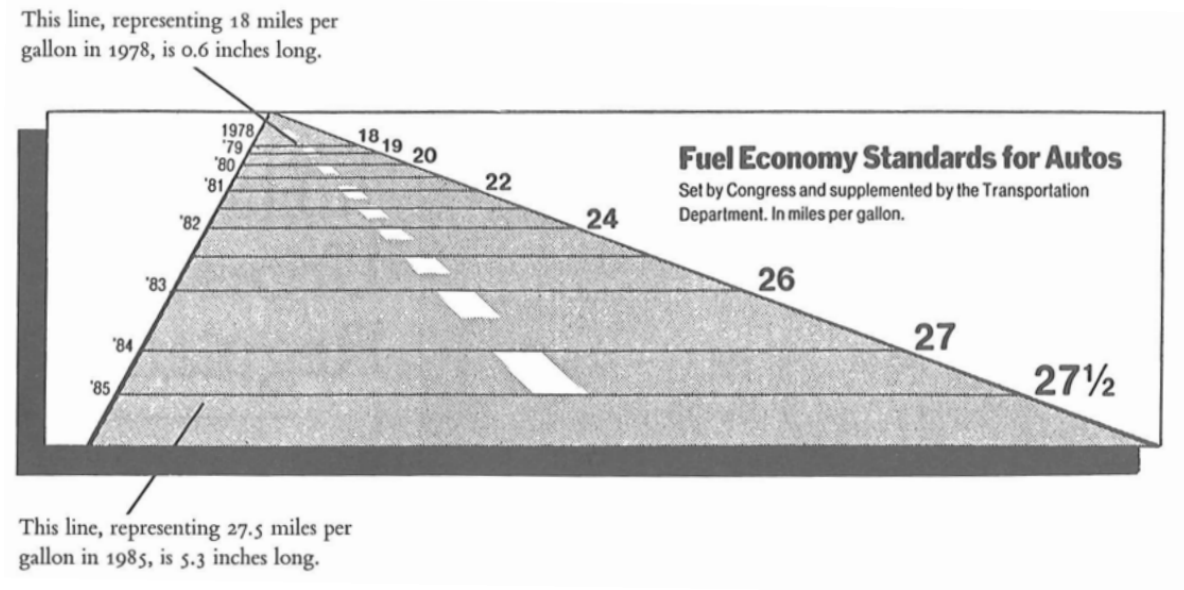
2.9.2 Współczynnik kłamstwa



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

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

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



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

2.10 Jak tworzyć?

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

2.11 Bibliografia

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

3 NumPy

NumPy jest biblioteką Pythona służącą do obliczeń naukowych.

Zastosowania:

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

3.1 Import biblioteki NumPy

```
import numpy as np
```

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

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

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

```
import numpy as np
```

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

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

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

①

②

```

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

```

- ① Standardowe domyślne.
- ② Sprawdzenie typu.
- ③ Jeden z elementów jest innego typu. Tu następuje zatem rozszerzenie do typu “największego”.
- ④ Tu otrzymamy tablicę 2x2.
- ⑤ W tej 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))

```

```

--- 0.0010004043579101562 seconds ---
--- 0.04944443702697754 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.]]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
(2, 3, 4)
```

`np.diag` - tworzy tablicę odpowiadającą macierzy diagonalnej

```
import numpy as np

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

```

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

```

```

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

```

`np.arange` - tablica wypełniona równomiernymi wartościami

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

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

```

import numpy as np

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

```



```

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

```

```

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

```

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

```

import numpy as np

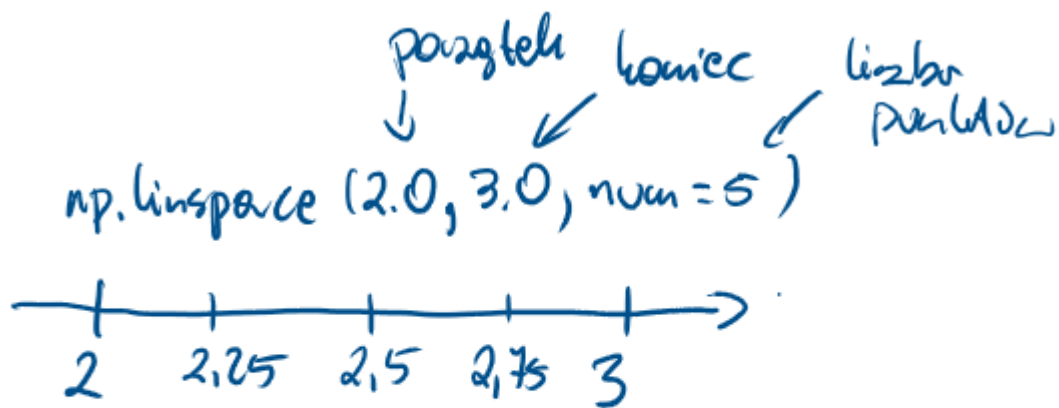
a = np.linspace(2.0, 3.0, num=5)
print(a)
b = np.linspace(2.0, 3.0, num=5, endpoint=False)
print(b)
c = np.linspace(10, 20, num=4)
print(c)
d = np.linspace(10, 20, num=4, dtype=int)
print(d)

```

```

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

```



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

`endpoint = False`

- wyłącza ostatni punkt
(z prawej strony)

Wtedy podział odbywa się
na `num` części.

`dtype` ← ustala typ

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

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

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

axis=0)

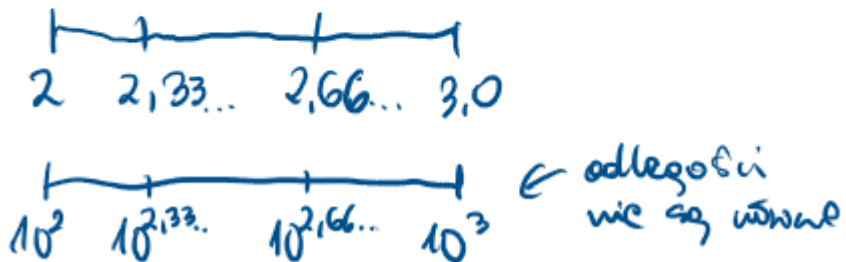
```
import numpy as np

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

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

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

Domyślnie podstawa logarytmu 10



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

```
import numpy as np

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

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

`np.identity` - tablica przypominająca macierz jednostkową

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

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

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

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

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

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

3.6 Indeksowanie, “krojenie”

```
import numpy as np

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

①
②
③
④
⑤
⑥

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

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

```
import numpy as np

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

①
②
③
④
⑤
⑥

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

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

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

```
import numpy as np

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

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

```
import numpy as np

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

```

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

```

```

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

```

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

```

import numpy as np

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

```

```

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

```

Naprawa:

```
import numpy as np

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

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

Indeksowanie logiczne (fancy indexing)

```
import numpy as np

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

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

```
import numpy as np

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



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

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

```
import numpy as np

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

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

3.7 Modyfikacja kształtu i rozmiaru

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

```
import numpy as np

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

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

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

```
import numpy as np

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

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

```
import numpy as np
```

```

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

```

```

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

```

```

import numpy as np

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

```

```

h

```

```

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

```

```

import numpy as np

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

```

```

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

```

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

```
import numpy as np

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

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

```
import numpy as np

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

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

```
import numpy as np

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

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

```
import numpy as np

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

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

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

3.8 Broadcasting

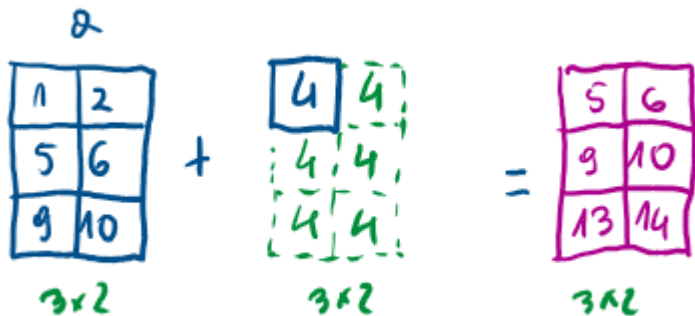
Rozważane warianty są przykładowe.

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

```
import numpy as np

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

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



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

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

```

import numpy as np

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

```

```

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

```


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

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

Wariant 3 - "kolumna" i "wiersz"

```
import numpy as np

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

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

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

Diagram illustrating matrix multiplication:

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

Matrix a 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 π do 2π i użyj tej tablicy do obliczenia i wyświetlenia sinusa dla każdego punktu. Wyniki zapisz w osobnej tablicy.
5. Stwórz tablicę składającą się z 15 punktów równomiernie rozłożonych między -5 a 5. Następnie, na podstawie tej tablicy, utwórz dwie nowe tablice: jedną zawierającą wartości funkcji eksponencjalnej e^x dla każdego z punktów, a drugą zawierającą logarytm naturalny dla tych punktów, gdzie punkty równoznaczne z wartością mniejszą lub równą 0 są pomijane.
6. Stwórz tablicę `logArray`, używając funkcji `logspace`, która zawiera 30 punktów rozłożonych logarytmicznie między 10^1 a 10^5 . Następnie oblicz średnią wartość wszystkich elementów w tej tablicy.
7. Wygeneruj tablicę `frequencies`, korzystając z funkcji `logspace`, aby otrzymać 25 punktów logarytmicznie równomiernie rozłożonych między częstotliwościami 10^2 Hz a 10^6 Hz. Użyj tej tablicy do symulacji wartości pewnego sygnału w zależności od częstotliwości i zapisz wyniki w nowej tablicy `signalValues`.
8. Korzystając z funkcji `logspace`, utwórz tablicę `resistances` reprezentującą wartości rezystancji, które są rozłożone logarytmicznie w zakresie od 1Ω do $1M\Omega$ włącznie, z 40 punktami.