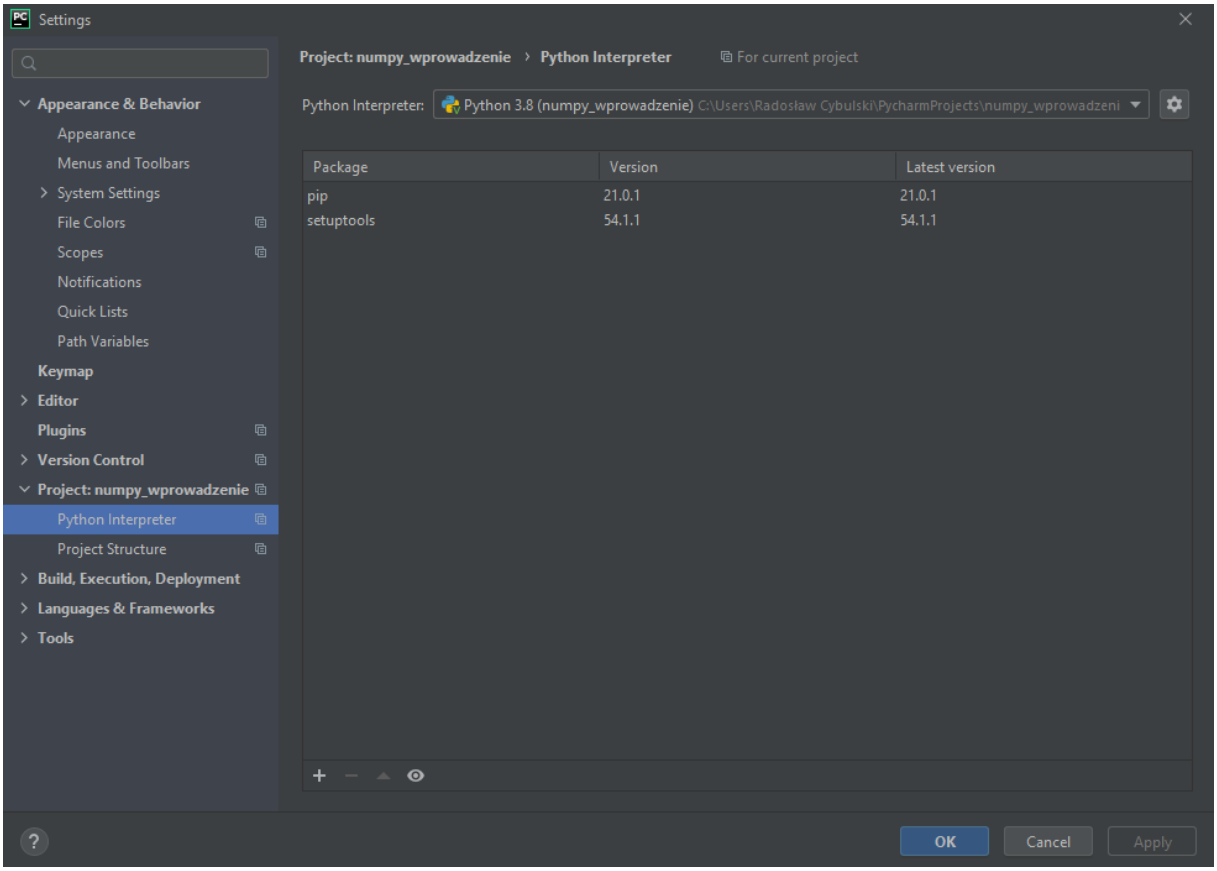
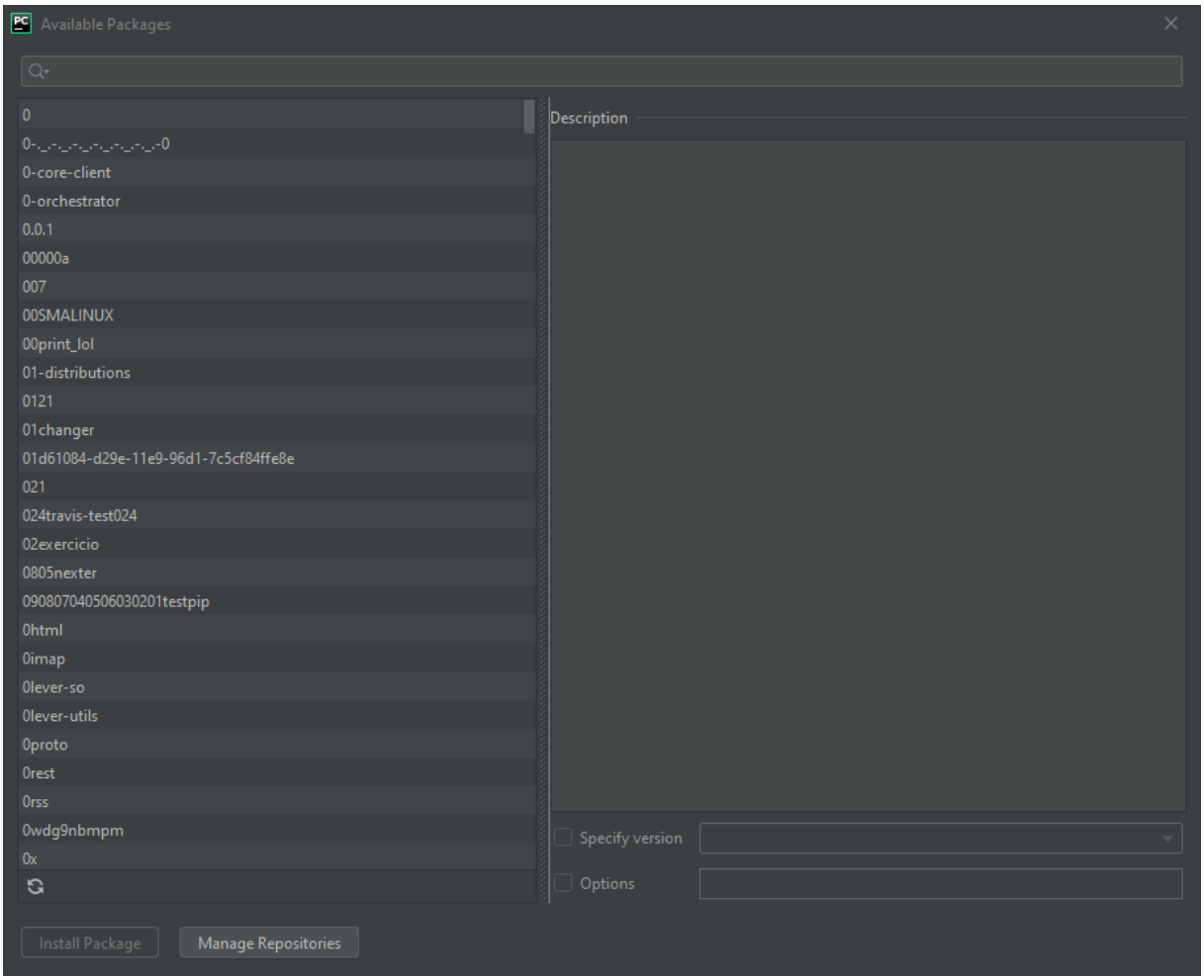
Instalacja biblioteki numpy

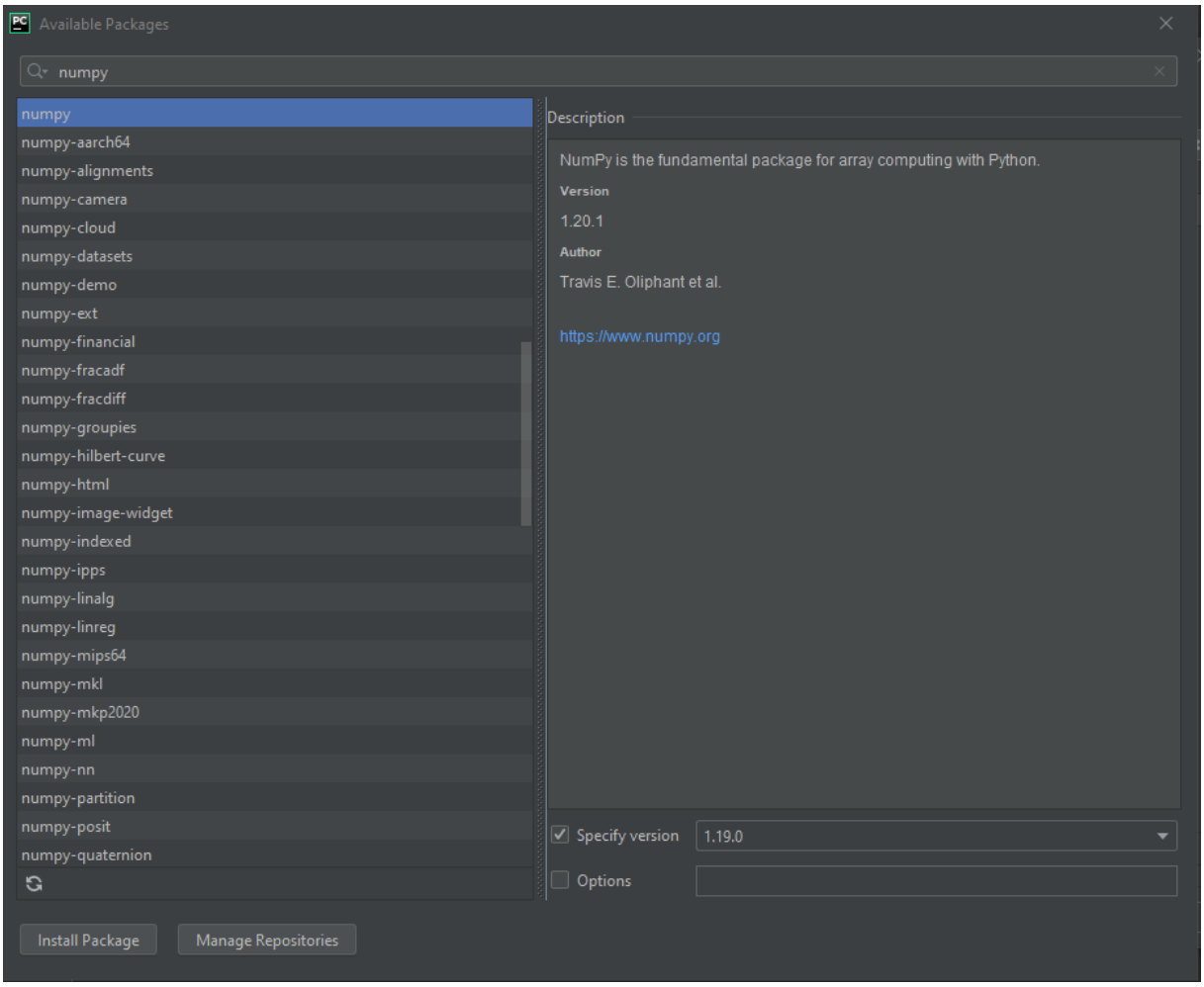
W nowo utworzonym projekcie przechodzimy do File/Settings/Project: project_name i wybieramy Python Interpreter



Następnie klikamy przycisk plusa w celu dodania do projektu wybranej biblioteki. Po naciśnięciu plusa otworzy się następujące okno.



Wpisujemy numpy w wyszukiwarkę, zaznaczamy Specify version i wybieramy wersję 1.19.0. Widok jak na zdjęciu poniżej.



Klikamy na install package i czekamy na zakończenie procesu instalacji.

Tworzenie tablic w numpy

Tablice biblioteki Numpy to kolekcje, które mogą przechowywać dane jednorodne, czyli dane tego samego typu. Taki stan rzeczy powoduje, że w kwestii przechowywania danych nie są tak uniwersalne jak listy, ale z racji tego, że znając typ danych, który będzie przechowywany można łatwo obliczyć jaki będzie rozmiar tablicy w pamięci. Dzięki temu Numpy może wykonywać operacje na całych wektorach wartości a nie na pojedynczych elementach jak w przypadku list. Biblioteka Numpy w znakomitej części jest napisana w języku C co zapewnia bardzo wysoką wydajność większości operacji. Deklaracja tablicy korzystającej z podobnego mechanizmu działania jak funkcja range():

```
import numpy as np
a = np.arange(2)

print(a)
```

Po wypisaniu zmiennej otrzymamy informację postaci [0 1].

Faktyczną nazwą klasy dla tablicy Numpy jest ndarray co stanowi skrót od n-dimensional array czyli tablicę n-wymiarową.

Przykład 1.

```
import numpy as np
#inicjalizacja tablicy
a = np.array([0, 1])
print(a)
#lub drugi sposób
a = np.arange(2)
print(a)
#wypisanie typu zmiennej tablicy (nie jej elementów) - ndarray
print(type(a))
#sprawdzenie typu danych tablicy
print(a.dtype)
#inicjalizacja tablicy z konkertnym typem danych
a = np.arange(2, dtype ='int64')
print(a.dtype)
#zapisywanie kopii tablicy jako tablicy z innym typem
b = a.astype('float')
print(b)
print(b.dtype)
#wypisanie rozmiaru tablicy
print(b.shape)
# można też sprawdzić ilość wymiarów tablicy
print(a.ndim)
#stworzenie tablicy wielowymiarowej może wyglądać tak
#parametrem przekazywanym do funkcji array jest obiekt, który
zostanie skonwertowany na tablicę
#może to być Pythonowa lista
m = np.array([np.arange(2), np.arange(2)])
print(m.shape)
#ponownie typem jest ndarray
print(type(m))
```

Pełna lista typów danych, które możemy umieścić w tablicach Numpy znajduje się pod adresem https://numpy.org/doc/

Przykład 2

```
import numpy as np
#możemy w łatwy sposób stworzyć macierz danego rozmiaru
wypełnioną zerami lub jedynkami
zera = np.zeros((5,5))
jedynki = np.ones((4,4))
print(zera)
print(jedynki)
#warto sprawdzić jaki jest domyślny typ danych takich tablic
```

```
print(zera.dtype)
print(jedynki.dtype)
#można również stworzyć "pustą" macierz o podanych wymiarach,
która pusta wcale nie jest
#wartości umieszczane są losowe, najpierw podawana jest ilość
wierszy tablicy, potem ilość kolumn
pusta = np.empty((2,2))
print(pusta)
#do elementów tablicy możemy odwołać się tak jak do elementów
np. listy czyli podając indeksy
poz_1 = pusta[1,1]
poz_2 = pusta[0,1]
print(poz_1)
print(poz_2)
#tworzenie macierzy 2x2 wraz z uzupełnieniem
macierz = np.array([[1,2],[3,4]])
print(macierz)
#funkcja arange potrafi takrze tworzyć ciągi liczb
zmiennoprzecinkowych
liczby = np.arange(1,2,0.1)
print(liczby)
#podobnie działa funkcja linspace, które działanie jest
równoważne tej samej funkcji w MATLAB-ie
liczby_lin = np.linspace(1,2,5)
print(liczby_lin)
#a teraz możemy utworzyć dwie macierze, najpierw wartości
interowane są w kolumnie a następnie w wierszu
z = np.indices((5,3))
print(z)
#wielowymiarowe macierze możemy również generować funkcją
mgrid
x, y = np.mgrid[0:5, 0:5]
print(x)
print(y)
#podobnie jak w MATLAB-ie możemy tworzyć macierze diagonalne
mat_diag = np.diag([a for a in range(5)])
print(mat_diag)
#w powyższym przykładzie stworzony wektor wartości zostanie
umieszczony na głównej przekątnej macierzy
#możemy podać drugi parametr funkcji diag, który określa
indeks przekątnej względem głównej przekątnej
#która zostanie wypełniona wartościami podanego wektora
mat_diag_k = np.diag([a for a in range(5)],-1)
print(mat_diag_k)
#Numpy jest wstanie stworzyć tablicę jednowymiarową z
dowolnego obiektu iterowalnego (iterable)
z = np.fromiter(range(5), dtype='int32')
print(z)
#ciekawą funkcją Numpy jest funkcja frombuffer, dzięki której
możemy stworzyć np. tablicę znaków
marcin = b'Marcin'
```

```
# mar = np.frombuffer(marcin,dtype='S1')
# print(mar)
# mar_2 = np.frombuffer(marcin,dtype='S2')
# print(mar_2)
#powyższa funkcja ma jednak pewną wadę dla pythona 3.x, która
powodóje, że trzeba jawnie określić
#iż ciąg znaków przekazujemy jako ciąg bajtów co osiągamy po
przez podanie litery 'b' przed wartością
#zmiennej tekstowej. Można podobne efekty osiągnąć inaczej
marcin = 'Marcin'
mar_3 = np.array(list(marcin))
mar_4 = np.array(list(marcin), dtype='S1')
mar_5 = np.array(list(b'Marcin'))
mar_6 = np.fromiter(marcin,dtype='S1')
mar_7 = np.fromiter(marcin,dtype='U1')
print(mar_3)
print(mar_4)
print(mar_5)
print(mar_6)
print(mar_7)
#talice w Numpy możemy w prosty sposób do siebie dodawać,
odejmować, mnożyć, dzielić
mat = np.ones((2,2))
mat_1 = np.ones((2,2))
mat = mat + mat_1
print(mat)
print(mat - mat_1)
print(mat*mat_1)
print(mat/mat_1)
```

Indeksowanie i cięcia tablic

Cięcie i indeksowanie danych w tablicy Numpy jest możliwe do wykonania na bardzo wiele sposobów. Poniżej przykłady niektórych z nich.

```
import numpy as np
#cięcie (slicing) tablicy numpy można wykonać za pomocą
wartości z funkcji slice lub range
a = np.arange(10)
print(a)
s = slice(2,7,2)
print(a[s])
s = range(2,7,2)
print(a[s])
#możemy ciąć tablice również w sposób znany z cięcia list
(efekt jak wyżej)
print(a[2:7:2])
#lub tak
print(a[1:])
print(a[2:5])
#w podobny sposób postępujemy w przypadku tablic
```

```
wielowymiarowych
mat = np.arange(25)
#teraz zmieniamy kształt tablicy jednowymiarowej na macierz
5x5
mat = mat.reshape((5,5))
print(mat)
print(mat[1:]) #od drugiego wiersza
print(mat[:,1]) #druga kolumna jako wektor
print(mat[:,-1:]) #ostatnia kolumna
print(mat[2:6, 1:3]) # 2 i 3 kolumna dla 3,4,5 wierszy
print(mat[:, range(2,6,2)]) # 3 i 5 kolumna
print('')
#bardziej zaawansowane, lecz trudniejsze do zrozumienia cięcia
można osiągnąć wg. poniższego przykładu
#y będzie tablicą zawierającą wierzchołki macierzy x
x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
print(x)
rows = np.array([[0, 0], [3, 3]])
cols = np.array([[0, 2], [0, 2]])
y = x[rows,cols]
print(y)
```

Zadania

Zad1.

Za pomocą funkcji arange stwórz tablicę numpy składającą się z 20 kolejnych wielokrotności liczby 4.

Zad2.

Stwórz listę składającą się z wartości zmiennoprzecinkowych a następnie zapisz do innej zmiennej jej kopię przekonwertowaną na typ int32

Zad3.

Napisz funkcję, która będzie:

- Przyjmowała jeden parametr ‘n’ w postaci liczby całkowitej
- Zwracała tablicę Numpy o wymiarach n*n kolejnych potęg liczby 2

Zad4.

Napisz funkcję, która będzie przyjmowała 2 parametry: liczbę, która będzie podstawą operacji potęgowania oraz ilość kolejnych potęg do wygenerowania. Korzystając z funkcji logspace generuj tablicę jednowymiarową kolejnych potęg podanej liczby, np. generuj(2,4) -> [2,4,8,16]

Zad5.

Napisz funkcję, która:

- Na wejściu przyjmuje jeden parametr określający długość wektora
- Na podstawie parametru generuj wektor, ale w kolejności odwróconej
- Generuj macierz diagonalną z w/w wektorem na przekątnej oddalonej o 2 w górę od głównej przekątnej macierzy

Zad6.

Stwórz skrypt który na wyjściu wyświetli macierz numpy (tablica wielowymiarowa) w postaci wykreślanki, gdzie jedno słowo będzie wypisane w kolumnie, jedno w wierszu i jedno po ukosie. Jedno z tych słów powinno być wypisane od prawej do lewej.

Zad7.

Napisz funkcję, która wygeneruje macierz wielowymiarową postaci:

[[2 4 6]
 [4 2 4]
 [6 4 2]]
Przy założeniach:

funkcja przyjmuje parametr n, który określa wymiary macierzy jako n*n i umieszcza wielokrotność liczby 2 na kolejnych jej przekątnych rozchodzących się od głównej przekątnej.

Zadanie 8
Napisz funkcję, która:

- jako parametr wejściowy będzie przyjmowała tablicę wielowymiarową numpy oraz parametr ‘kierunek’,
- parametr kierunek określa czy tablica wejściowa będzie dzielona w pionie czy poziomie

- funkcja dzieli tablicę wejściową na pół (napisz warunek, który wyświetli komunikat, że ilość wierszy lub kolumn, w zależności od kierunku podziału, nie pozwala na operację)

Zadanie 9
Wykorzystaj poznane na zajęciach funkcje biblioteki Numpy i stwórz macierz 5x5, która będzie zawierała kolejne wartości ciągu arytmetycznego.

1. Operacje na tablicach.

Operacje arytmetyczne wykonywane są na każdym elemencie tablicy

1.1. Operacje arytmetyczne

Przykład 1

```
import numpy as np
#inicjujemy dane
a = np.array([20, 30, 40, 50])
b = np.arange(4)
print(a)
print(b)
#wykonujemy operację i zapisujemy do nowej zmiennej
c = a - b
print(c)
#wykonujemy operację: kwadrat zawartości
print(b**2)
#możemy również zmodyfikować obecne zmienne
print(a)
a += b
print(a)
a -= b
print(a)
```

1.2. Operacje na osiach

Podczas wykonywania operacji na macierzy można podać parametr axis który pozwoli określić po której osi ma zostać wykonana operacja.

Przykład 2

```
import numpy as np
a = np.arange(12).reshape((3,4))
print(a)
#suma całej macierzy
print(a.sum())
#suma każdej z kolumn
print(a.sum(axis=0))
#minimum każdego rzędu
print(a.min(axis=1))
#skummulowana suma dla rzędu
print(a.cumsum(axis=1))
```

1.3. Mnożenie macierzy

Jeżeli chcemy wykonać operację mnożenia macierzy mamy dwie możliwości:

Przykład 3

```
import numpy as np
#inicjujemy dane
a = np.arange(3)
b = np.arange(3)
print(a)
print(b)
print(a.dot(b)) # iloczyn macierzy
print(np.dot(a,b)) #inny sposób
```

1.4. Operacje na tablicach różnej dokładności (upcasting)

Należy pamiętać, że przy operacjach na tablicach różnej dokładności wynik zawsze będzie podnoszony do wyższego.

Przykład 4

```
import numpy as np
#macierz całkowita
a = np.ones(3, dtype='int32')
print(a.dtype)
#macierz zmiennoprzecinkowa
b = np.linspace(0,np.pi,3)
print(b.dtype)
#wynikiem jest macierz zmiennoprzecinkowa
c = a+b
print(c)
print(c.dtype)
#wynikiem jest macierz liczb zespolonych
d = np.exp(c*1j)
print(d)
print(d.dtype)
```

2. Funkcje uniwersalne

Funkcje uniwersalne działają na każdym elemencie tablicy oraz tworzą nową tablicę wynikową.

Przykład 5

```
import numpy as np
b = np.arange(3)
print(b)
print(np.exp(b))
print(np.sqrt(b))
c = np.array([2., -1., 4.])
print(np.add(b,c))
```

3. Iteracja tablic

Tablice wielowymiarowe iterujemy w odniesieniu do pierwszej osi!

Przykład 6

```
import numpy as np
#generujemy macierz 3x2
a = np.arange(6).reshape((3,2))
print(a)
for b in a:
    #iterujemy wzdłuż pierwszej osi
    print(b)
    print("")
```

Możemy również iterować wszystkie elementy macierzy jakby była macierzą płaską.

Przykład 7

```
import numpy as np
#generujemy macierz 3x2
a = np.arange(6).reshape((3,2))
print(a)
for b in a.flat:
    #iterujemy jak by to była macierz płaska
    print(b)
    print("")
```

4. Przekształcenia

Kształt tablicy jest określany po ilości wymiarów na każdej z osi.

Przykład 8

```
import numpy as np
#generujemy macierz 1x6
a = np.arange(6)
print(a)
print(a.shape)
#generujemy macierz 3x3
b = np.array([np.arange(3), np.arange(3), np.arange(3)])
print(b)
print(b.shape)
```

Macierz jednego kształtu możemy zmodyfikować do zupełnie nowego na kilka różnych sposobów.

Przykład 9

```
import numpy as np
#generujemy macierz 1x6
a = np.arange(6)
print(a)
print(a.shape)
print("")
#przekształcamy ją na macierz 2x3
b = a.reshape((2,3))
print(b)
print(b.shape)
print("")
#przekształcamy ją na macierz 3x2
c = b.reshape((3,2))
print(c)
print(c.shape)
print("")
#spłaszczamy macierz zyskując pierwotny kształt ze zmiennej a
d = c.ravel()
print(d)
print(d.shape)
print("")
#transpozycja macierzy
e = b.T
print(e)
print(e.shape)
```

Zadania

Zadanie 1
Utwórz dwie macierze 1x3 różnych wartości a następnie wykonaj operację mnożenia.

Zadanie 2
Utwórz macierz 3x3 oraz 4x4 i znajdź najniższe wartości dla każdej kolumny i każdego rzędu.

Zadanie 3
Wykorzystaj macierze z zadania pierwszego i wykonaj iloczyn macierzy.

Zadanie 4
Utwórz dwie macierze 1x3 gdzie pierwsza z nich będzie zawierała liczby całkowite, a druga liczby rzeczywiste. Następnie wykonaj na nich operację mnożenia.

Zadanie 5
Utwórz macierz 2x3 różnych wartości a następnie wylicz sinus dla każdej z jej wartości i zapisz do zmiennej “a”.

Zadanie 6
Utwórz nową macierz 2x3 różnych wartości a następnie wylicz cosinus dla każdej z jej wartości i zapisz do zmiennej “b”.

Zadanie 7
Wykonaj funkcję dodawania obu macierzy zapisanych wcześniej do zmiennych a i b.

Zadanie 8
Wygeneruj macierz 3x3 i wyświetl w pętli każdy z rzędów.

Zadanie 9
Wygeneruj macierz 3x3 i wyświetl w pętli każdy element korzystając z opcji “spłaszczenia” macierzy. (użyj funkcji flat())

Zadanie 10
Wygeneruj macierz 9x9 a następnie zmień jej kształt. Jakie mamy możliwości i dlaczego?

Zadanie 11
Wygeneruj macierz płaską (wektor) i przekształć ją na macierz 3x4. Wygeneruj w ten sposób jeszcze kombinacje 4x3 i 2x6. Spłaszcz każdą z nich i wyświetl wyniki. Czy są identyczne?

Biblioteka pandas, część 1.

1. Struktury danych w bibliotece pandas.

Serie danych (Series) – to jednowymiarowa tablica z etykietami, która może przechowywać dowolny typ danych. Dokumentacja dla serii danych https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

Ramka danych (DataFrame) – dwuwymiarowa struktura z etykietami, mogąca przechowywać kolumny z różnymi typami danych. Dokumentacja dla ramek - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

Listing 1 – tworzenie Series i DataFrames

```
import pandas as pd
import numpy as np

#Series
s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)
s = pd.Series([10, 12, 8, 14], index=['Ala', 'Marek',
'Wiesiek', 'Eleonora'])
print(s)


#DataFrame
#tworzenie dataframe na podstawie słownika
data = {'Kraj': ['Belgia', 'Indie', 'Brazylia'],
        'Stolica': ['Bruksela', 'New Delhi', 'Brasilia'],
        'Populacja': [11190846, 1303171035, 207847528]}
df = pd.DataFrame(data)
print(df)
#DataFrame przechowuje typ dla każdej kolumny co możemy
sprwadzić wpisując
print(df.dtypes)
#możemy również w prosty sposób stworzyć serię danych - czyli
próbki dla kolejnych
#dat, pomiarów czasu
daty = pd.date_range('20210324', periods=5)
print(daty)
df = pd.DataFrame(np.random.randn(5,4), index=daty,
columns=list('ABCD'))
print(df)

#biblioteka Pandas umożliwia również tworzenie DataFrame'ów z
zewnętrznych źródeł danych
#CSV, odczyt i zapis
df = pd.read_csv('dane.csv', header=0, sep=";",decimal=',')
print(df)
df.to_csv('plik.csv', index=False)
```

```
#Excel - wymagana jest biblioteka openpyxl
#trzeba ją zainstalować


xlsx = pd.ExcelFile('dane.xlsx')
df = pd.read_excel(xlsx, header=0)
print(df)
df.to_excel('wyniki.xlsx', sheet_name='arkusz pierwszy')
```

2. Pobieranie danych ze struktur

Pandas dostarcza wielu sposobów na pobieranie pojedynczych wartości, kolumn, wierszy lub zbiorów wartości na podstawie parametrów. Poniżej znajdują się 2 listingi z najbardziej popularnymi metodami.

```
import pandas as pd
import numpy as np


s = pd.Series([10, 12, 8, 14], index=['Ala', 'Marek',
'Wiesiek', 'Eleonora'])
print(s)

data = {'Kraj': ['Belgia', 'Indie', 'Brazylia'],
        'Stolica': ['Bruksela', 'New Delhi', 'Brasilia'],
        'Populacja': [11190846, 1303171035, 207847528]}
df = pd.DataFrame(data)
print(df)

#pojedynczy element serii, odnosimy się do nazwy indeksu
print(s['Wiesiek'])
#możemy również dostać się do wartości serii jak do pola klasy
print(s.Wiesiek)

#pojedynczy elemenet ramki danych, tak jak przy cięciu tablic,
tylko tu jest oparte na indeksach
print(df[0:1])
print("")
#kolumna po etykiecie
print(df['Populacja'])
#pobieranie pojedynczej wartości po indeksie wiersza i kolumny
print(df.iloc[[0],[0]])
#pobieranie wartości po indeksie wiersza i etykiecie kolumny
print(df.loc[[0],["Kraj"]])
print(df.at[0,"Kraj"])
#podobnie jak w przypadku serii można odwoływać się do kolumn
jak do pól klasy
#dodatkowo print jest wywoływany jak w pętli dla każdego
elementu danej kolumny
print('kraj: ' + df.Kraj)
```

```
#Pandas posiada również funkcje pozwalające na losowe
pobieranie elementów lub
#w odniesieniu do procentowej wielkości całego zbioru

#jeden losowy element
print(df.sample())
#n losowych elementów
print(df.sample(2))
#ilość elementów procentowo, uwaga na zaokrąglenie
print(df.sample(frac=0.5))

#jeżeli potrzeba nam więcej próbek niż znajduje się w zbiorze
i dopuszczamy duplikaty
#to możemy użyć parametru replace, który będzie losował z
powtórzeniami
print(df.sample(n=10, replace=True))

#zamiast wyświetlać całą kolekcje możemy wyświetlić określoną
ilość elementów od początku kolekcji
#lub od jej końca
print(df.head())
print(df.head(2))
print(df.tail(1))

# Pandas jest też w stanie wyświetlić statystykę dla wartości,
które dana kolekcja zawiera, o ile są jakieś kolumny
# z danymi numerycznymi
print(df.describe())
# transpozycja to zmienna T kolekcji, podobnie jak w Numpy
print(df.T)
```

Więcej przykładów pobierania elementów poprzez indeksowanie można znaleźć pod adresem https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Listing 3 – filtrowanie, grupowanie i agregowanie danych

```
s = pd.Series([10, 12, 8, 14], index=['Ala', 'Marek',
'Wiesiek', 'Eleonora'])
print(s)

data = {'Kraj': ['Belgia', 'Indie', 'Brazylia'],
        'Stolica': ['Bruksela', 'New Delhi', 'Brasilia'],
        'Populacja': [11190846, 1303171035, 207847528]}
df = pd.DataFrame(data)
print(df)
#wyświetla dane z serii gdzie wartość większa od 1
print(s[(s>9)])
# nieco inny efekt można osiągnąć wykorzystując funkcję where,
```

```
która zwraca kolekcję w oryginalnej wielkości
# (liczebności) elementów, ale wartości nie spełniające
warunku uzupełnia wartością NaN
print(s.where(s > 10))
#możemy też podać wartość, która zostanie podstawiona zamiast
NaN
print(s.where(s>10, 'za duże'))
# jeszcze inna własność where pozwala na modyfikowanie
oryginalnego zbioru (domyślnie zwaracana jest kopia)
seria = s.copy()
seria.where(seria > 10, 'za duże', inplace=True)
print("########")
print(seria)

#wyświetla dane z serii gdzie wartość nie jest większa od 10
print(s[~(s > 10)])
#warunki możemy też łączyć
print(s[(s < 13) & (s > 8)])

#warunki dla pobierania DataFrame
print(df[df['Populacja']>1200000000])
#bardziej skomplikowane warunki
print(df[(df.Populacja > 1000000) & (df.index.isin([0,2]))])

#inny przykład z listą dopuszczalnych wartości oraz isin
zwracająca wartości boolowskie
print('#########')
szukaj = ['Belgia', 'Brasilia']
print(df.isin(szukaj))

#zmiana, usuwanie i dodawanie danych

#w przypadku serii możemy dodać/zmienić wartość poprzez
odwołanie się do elementu serii przez klucz (indeks)
s['Wiesiek'] = 15
print(s.Wiesiek)
s['Alan'] = 16
print(s)

# podobna operacja dla DataFrame ma nieco inny efekt - wartość
ustawiona dla wszystkich kolumn
df.loc[3] = 'dodane'
print(df)
# ale można dodać wiersz w postaci listy
df.loc[4] = ['Polska', 'Warszawa', 38675467]
print(df)

# usuwanie danych można wykonać przez funkcję drop, ale
pamiętajmy, że operacja nie wykonuje się in-place więc
# zwracana jest kopia DataFrame z usuniętymi wartościami
```

```
new_df = df.drop([3])
print(new_df)
# więc jeżeli chcemy zmienić pierwotny zbiór dodajemy parametr
inplace=True
df.drop([3], inplace=True)
print(df)
# można usuwać również całe kolumny po nazwie indeksu, ale
wykonanie tej komendy uniemożliwi
# wykonanie dalszego kodu (można przetstować po zakomentowaniu
dalszej części listingu)
#df.drop('Kraj', axis=1, inplace=True)

# do DataFrame możemy dodawać również kolumny zamiast wierszy
df['Kontynent'] = ['Europa', 'Azja', 'Ameryka Południowa',
'Europa']
print(df)

# Pandas ma również własne funkcje sortowania danych
print(df.sort_values(by='Kraj'))

# grupowania
grouped = df.groupby(['Kontynent'])
print(grouped.get_group('Europa'))
# można też jak w SQL czy Excelu uruchomić funkcje agregujące
na danej kolumnie
print(df.groupby(['Kontynent']).agg({'Populacja':['sum']}))
```

Zadanie 1
Wczytaj do DataFrame arkusz z narodzinami dzieci w Polsce dostępny w pliku /datasets/imiona.xlsx

Zadanie 2
Z danych z zadania 1 wyświetl (korzystając w miarę możliwości z funkcji biblioteki Pandas):

- tylko te rekordy gdzie liczba nadanych imion była większa niż 1000 w danym roku
- tylko rekordy gdzie nadane imię jest takie jak Twoje
- sumę wszystkich urodzonych dzieci w całym danym okresie,
- sumę dzieci urodzonych w latach 2000-2005
- sumę urodzonych chłopców i dziewczynek,
- najbardziej popularne imię dziewczynki i chłopca w danym roku (czyli po 2 rekordy na rok),
- najbardziej popularne imię dziewczynki i chłopca w całym danym okresie,

Zadanie 3
Wczytaj plik /datasets/zamowieniana.csv a następnie wyświetl:

- listę unikalnych nazwisk sprzedawców (przetwarzając zwróconą pojedynczą kolumnę z DataFrame)
- 5 najwyższych wartości zamówień
- ilość zamówień złożonych przez każdego sprzedawcę
- sumę zamówień dla każdego kraju
- sumę zamówień dla roku 2005, dla sprzedawców z Polski
- średnią kwotę zamówienia w 2004 roku,
- zapisz dane za 2004 rok do pliku zamówienia_2004.csv a dane za 2005 do pliku zamówienia_2005.csv

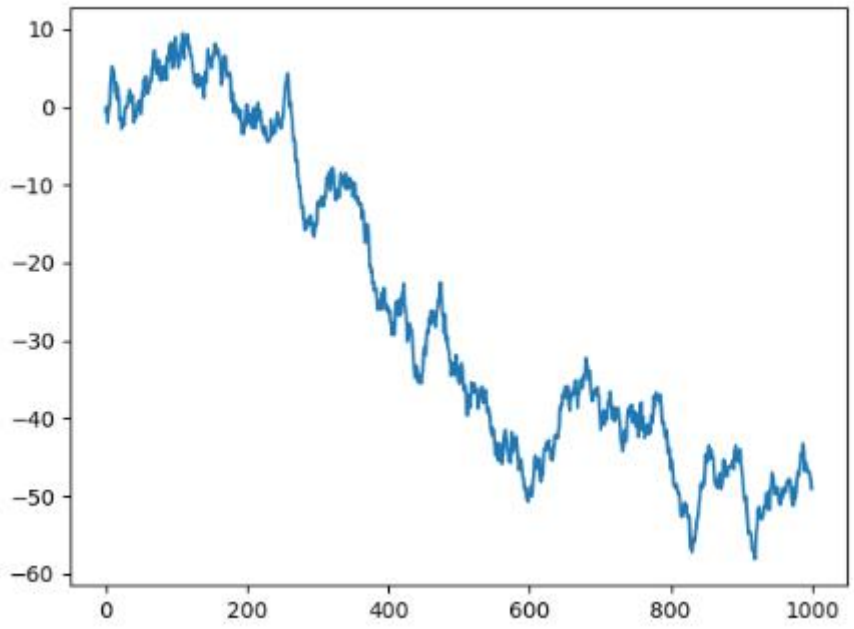
Biblioteka Pandas, część 2.

1. Pandaas i wykresy.

Listing 1 – wykres liniowy na postawie serii danych

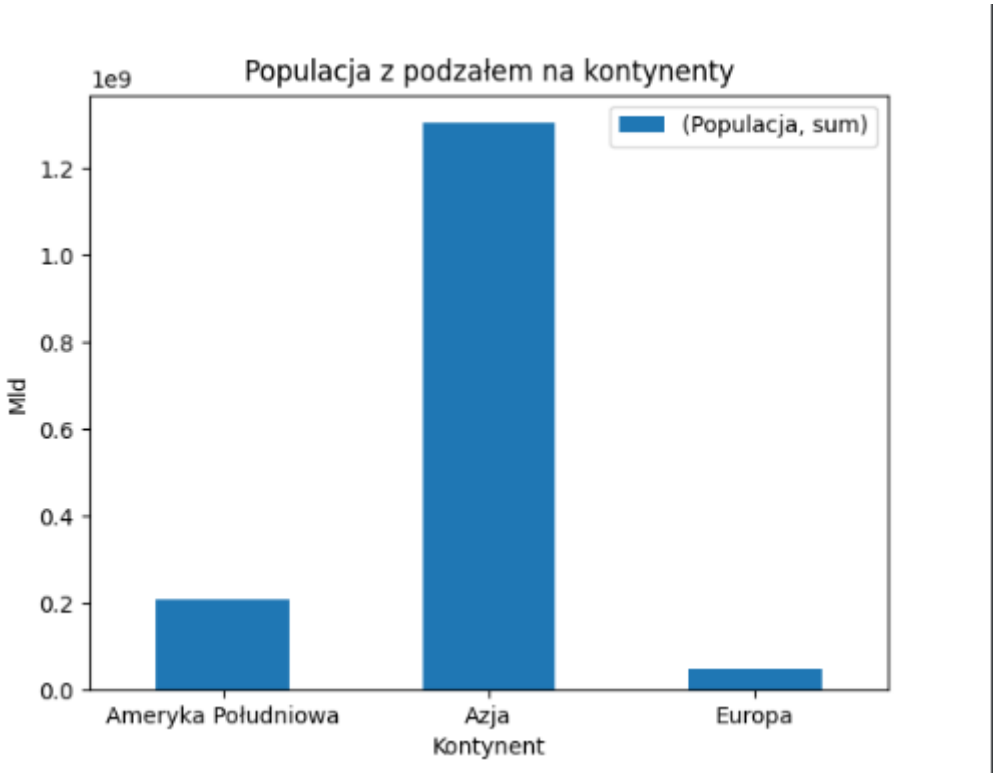
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

ts = pd.Series(np.random.randn(1000))
#funkcja biblioteki pandas generująca skumulowaną
sumę kolejnych elementów
ts = ts.cumsum()
print(ts)
ts.plot()
plt.show()
```

Wykres 1



Listing 2 – wykres kolumnowy z Pandas DataFrame

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

data = {'Kraj': ['Belgia',  'Indie',  'Brazylia',
'Polska'],
        'Stolica': ['Bruksela',  'New Delhi',
'Brasilia', 'Warszawa'],
        'Kontynent': ['Europa', 'Azja', 'Ameryka
Południowa', 'Europa'],
        'Populacja': [11190846, 1303171035,
207847528, 38675467]}

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

grupa =
df.groupby(['Kontynent']).agg({'Populacja':['sum']})
print(grupa)
# grupa.plot(kind='bar', xlabel='Kontynent',
ylabel='Mld', rot=0, legend=True, title='Populacja z
podzałem na kontynenty')
wykres = grupa.plot.bar()
wykres.set_ylabel("Mld")
wykres.set_xlabel('Kontynent')
wykres.tick_params(axis='x', labelrotation=0)
wykres.legend()
wykres.set_title('Populacja z podzałem na
kontynenty')
#zmiana kierunku tekstu etykiet słupków
# plt.xticks(rotation=0)
plt.savefig('wykres.png')
plt.show()
```



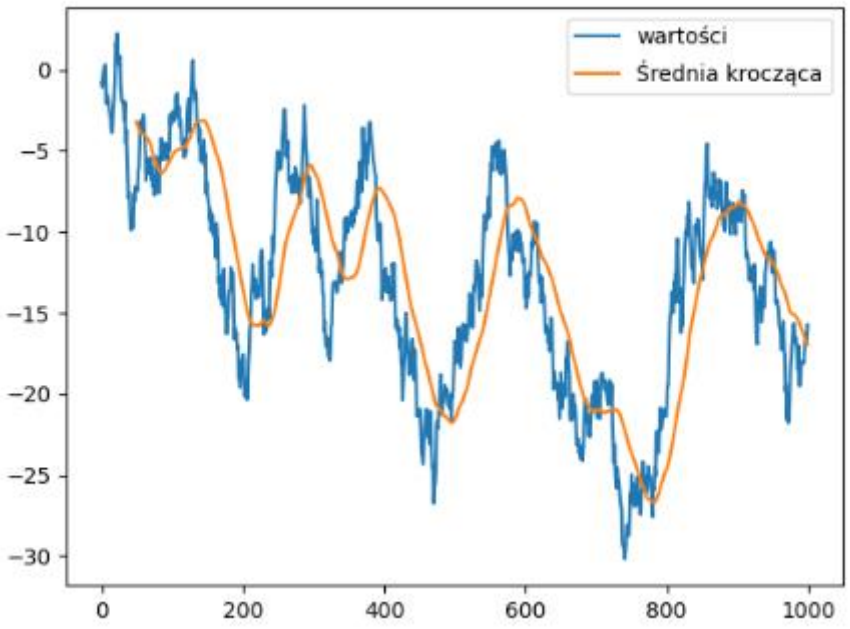
Listing 3 – wczytanie danych z pliku i wyświetlenie zgrupowanych wartości

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


df = pd.read_csv('dane.csv', header=0, sep=";",
decimal=".")
print(df)
grupa = df.groupby(['Imię i nazwisko']).agg({'Wartość
zamówienia':["sum"]})
#wykres kolumnowy z wartościami procentowymi
sformatowanymi z dokładnością do 2 miejsc po
przecinku
#figsize ustawia wielkość wykresu w calach, domyślnie
[6.4, 4.8]
grupa.plot(kind='pie', subplots=True, autopct='%.2f
%%', fontsize=20, figsize=(6,6), legend=(0, 0),
colors=['red', 'green'])
# wykres = grupa.plot.pie(subplots=True,autopct='%.2f
%%', fontsize=20, figsize=(6,6), legend=(0, 0))
# plt.legend(loc="lower right")
plt.title('Suma zamówienia dla sprzedawcy')
plt.show()
```


Suma zamówienia dla sprzedawcy
Alan Strzygło
45.48 %
(Wartość zamówienia, sum)
54.52 %
Marek Michalski
Alan Strzygło
Marek Michalski

Listing 4 – zmodyfikowana wersja listingu 1 z dodatkowym wykresem średniej kroczącej

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

#korzystając z funkcji random oraz data_range możemy
wygenerować szereg czasowy danych
ts = pd.Series(np.random.randn(1000))
#funkcja biblioteki pandas generująca skumulowaną
sumę kolejnych elementów
ts = ts.cumsum()
#rzutowanie Series na DataFrame
df = pd.DataFrame(ts, columns=['wartości'])
print(df)
# dodanie nowej kolumny i wykorzystanie funkcji
rolling do stworzenia kolejnych wartości średniej
kroczącej
df['Średnia krocząca'] = df.rolling(window=50).mean()
df.plot()
plt.legend()
plt.show()
```



Zadanie 1

Stwórz wykres liniowy, który wyświetli liczbę urodzonych dzieci dla każdego roku.

Zadanie 2

Stwórz wykres słupkowy, który wyświetli liczbę urodzonych chłopców i dziewczynek z całego zbioru.

Zadanie 3

Wykres kołowy z wartościami % ukazującymi ilość urodzonych chłopców i dziewczynek w ostatnich 5 latach z datasetu.

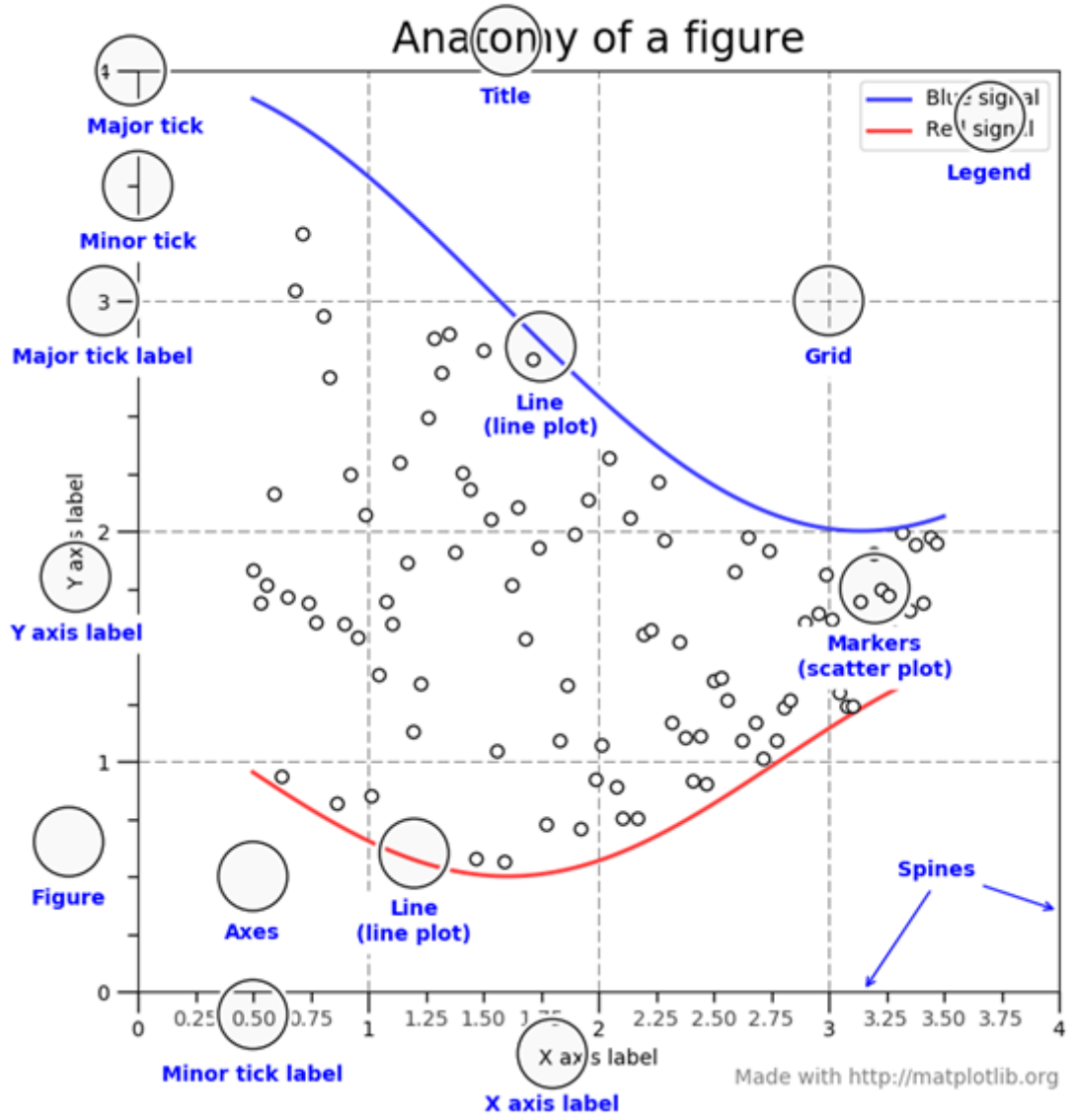
Zadanie 4

Wyświetl na pomocą wykresu słupkowego ilość złożonych zamówień przez poszczególnych sprzedawców (zbiór danych zamówienia.csv).

Matplotlib część 1

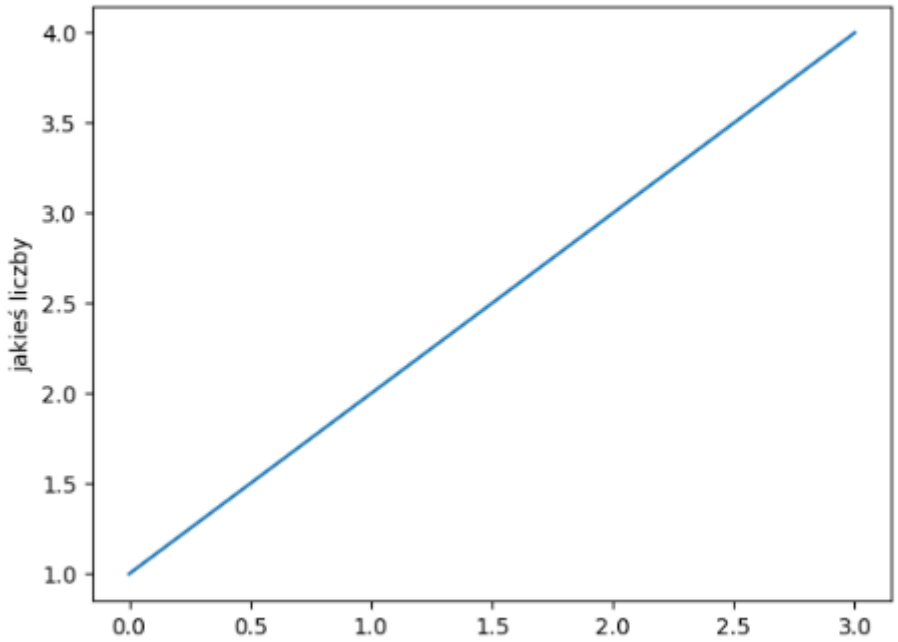
1. Wprowadzenie

Na początku warto zapoznać się z nazewnictwem (angielskim) elementów, z których składa się widok wykresu. Poniższa grafika pozwoli na ich identyfikację i możliwość dostosowania wykresu do założeń lub potrzeb danego zadania/problemu



Listing 1 – pierwszy prosty wykres liniowy

```
import matplotlib.pyplot as plt

#bardzo prosty wykres liniowy
plt.plot([1, 2, 3, 4])
plt.ylabel('jakieś liczby')
plt.show()
```

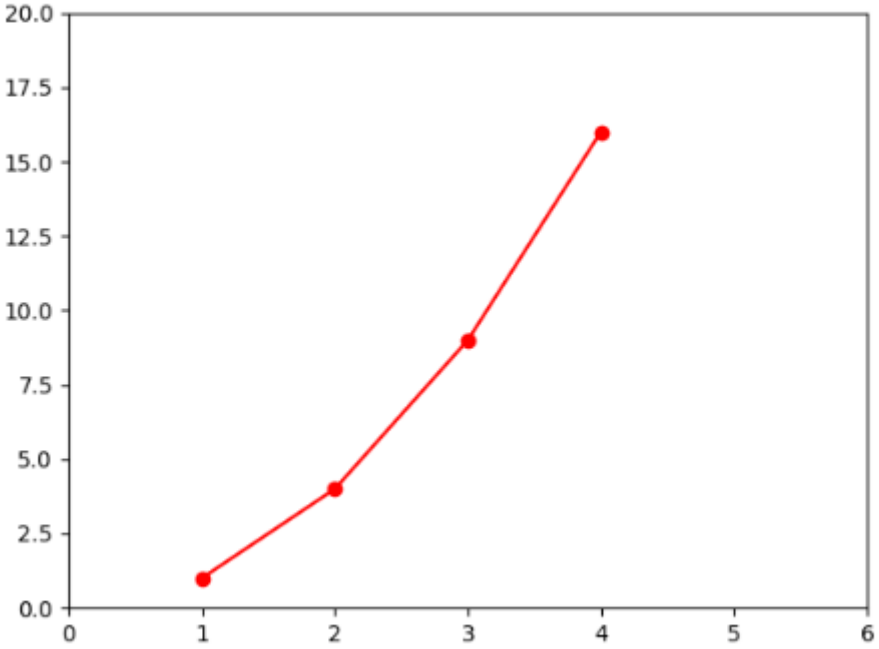


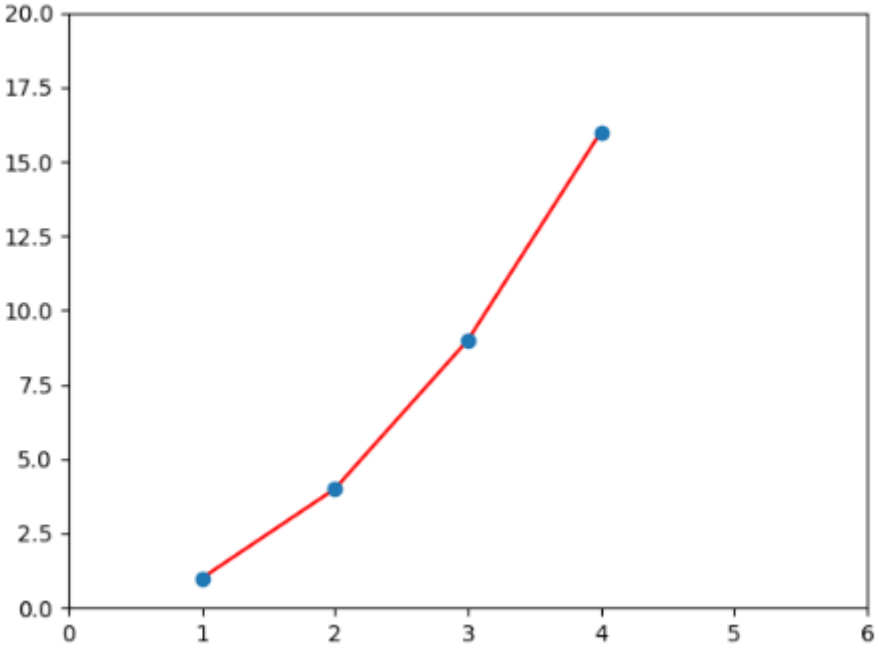
Wektor przekazanych wartości to oś Y, a oś X została wygenerowana automatycznie i tutaj dla wartości z wektora Y przyjmuje po prostu wartość indeksu z tej listy czyli dla wartości 1 przyjmuje wartość 0 itd. . Nie jest to zbyt przydatne w tym konkretnym przypadku.

2. Style wykresów.

Listing 2

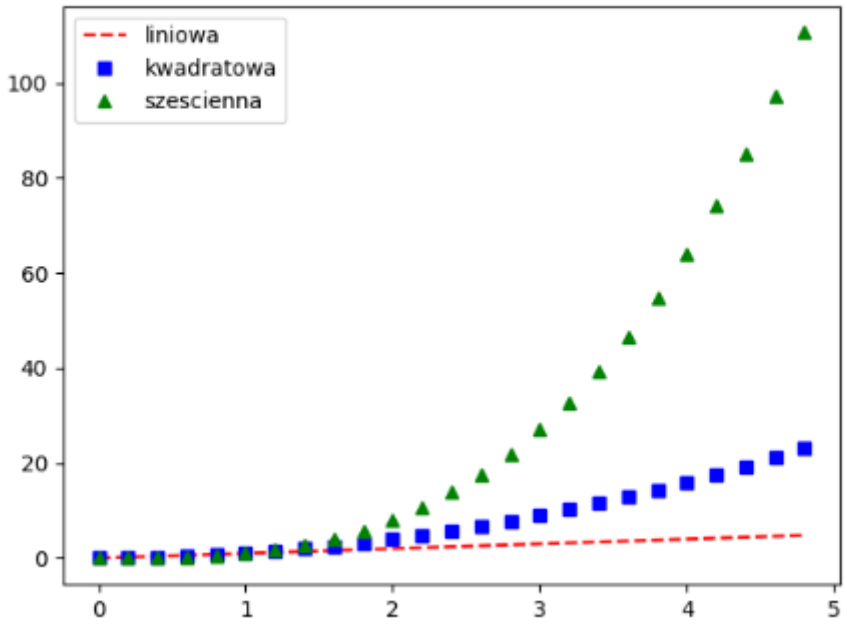
```
import matplotlib.pyplot as plt

#przekazujemy dwa wektory wartości, najpierw dla wektora x,
następnie dla wektora y
#dodatkowo mamy tutaj przekazywany parametr w postaci stringa,
który określa styl wykresu
#dla pełnej listy sprawdź dokumentację pod adresem
#https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.htm
l#matplotlib.pyplot.plot
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro-')
#tutaj określamy listę parametrów w postaci [xmin, xmax, ymin,
ymax]
plt.axis([0, 6, 0, 20])
plt.show()

#możemy też ustawić różne kolory dla poszczególnych elementów
nakładając na siebie dwa wykresy
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'r')
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'o')

plt.axis([0, 6, 0, 20])
plt.show()
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
3
4
5
6


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
3
4
5
6

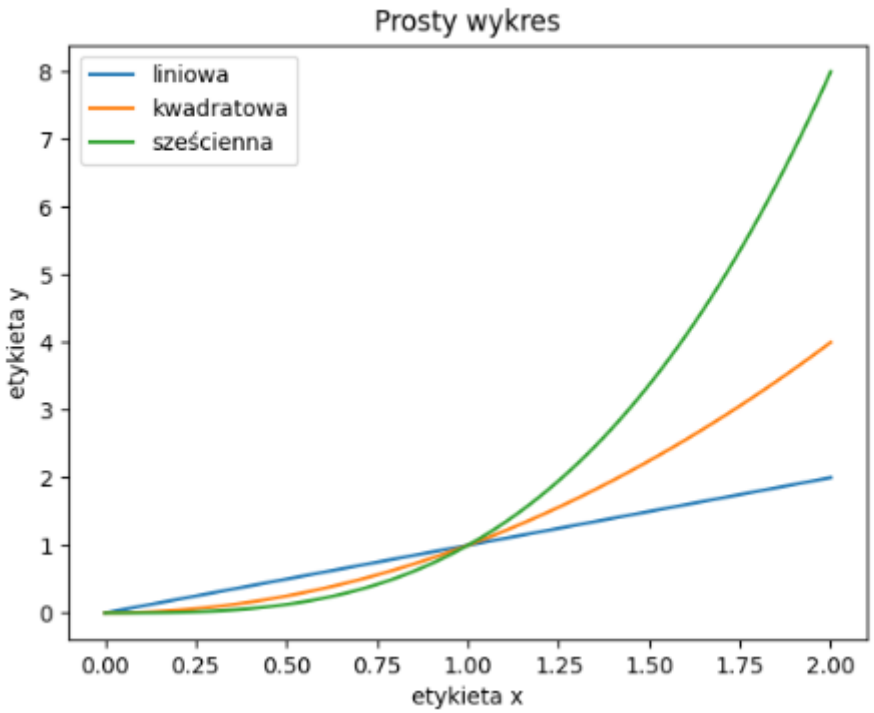
Listing 3

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

#bazowy wektor wartości
t = np.arange(0., 5., 0.2)

#za pomocą pojedynczego wywołania funkcji plot() możemy
wygenerować wiele wykresów na jednym płótnie (ang. canvas)
#każdorazowo podając niezbędne wartości: wartości dla osi x,
wartości dla osi y, styl wykresu, ...
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.legend(labels=['liniowa', 'kwadratowa', 'szescienna'])
plt.show()
```



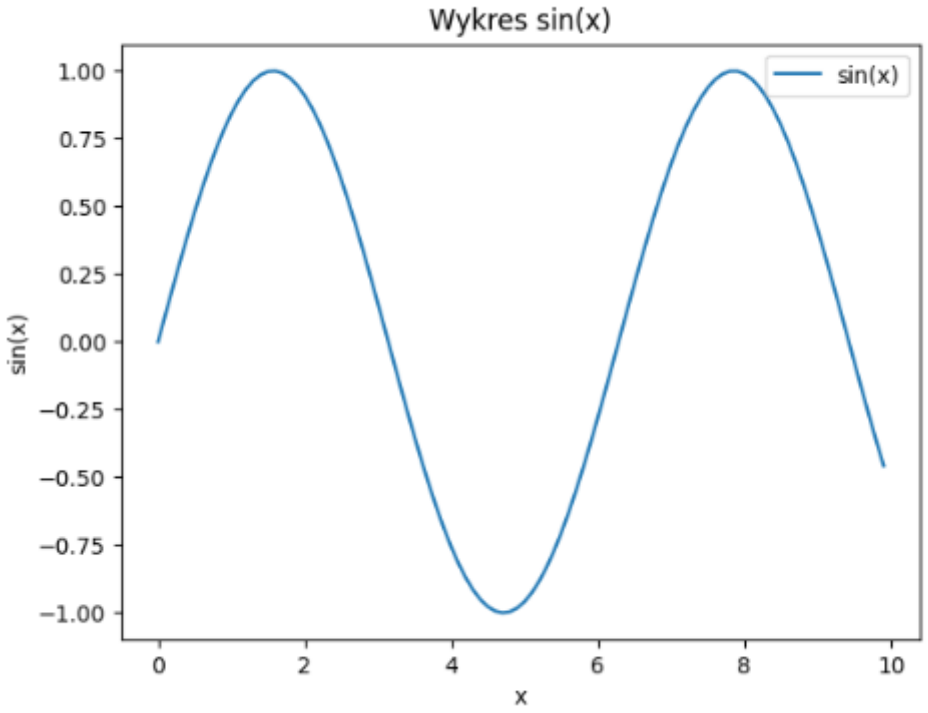
Listing 4

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image


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

#wykresy mogą być też dodawane do płótna definicja po
definicji zamiast w pojedynczym wywołaniu funkcji
plot()
#tutaj użyty został również parametr label, który
określa etykietę danego wykresu w legendzie
plt.plot(x, x, label="liniowa")
plt.plot(x, x**2, label="kwadratowa")
plt.plot(x, x**3, label="sześcienna")

#etykiety osi
plt.xlabel('etykieta x')
plt.ylabel("etykieta y")

#tytuł wykresu
plt.title("Prosty wykres")

#włączamy pokazanie legendy
plt.legend()
plt.savefig('wykres matplot.png')

plt.show()
im1 = Image.open('wykres matplot.png')
im1 = im1.convert('RGB')
im1.save('nowy.jpg')
```


Prosty wykres
liniowa
kwadratowa
sześcienna
etykieta y
etykieta x
0
1
2
3
4
5
6
7
8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

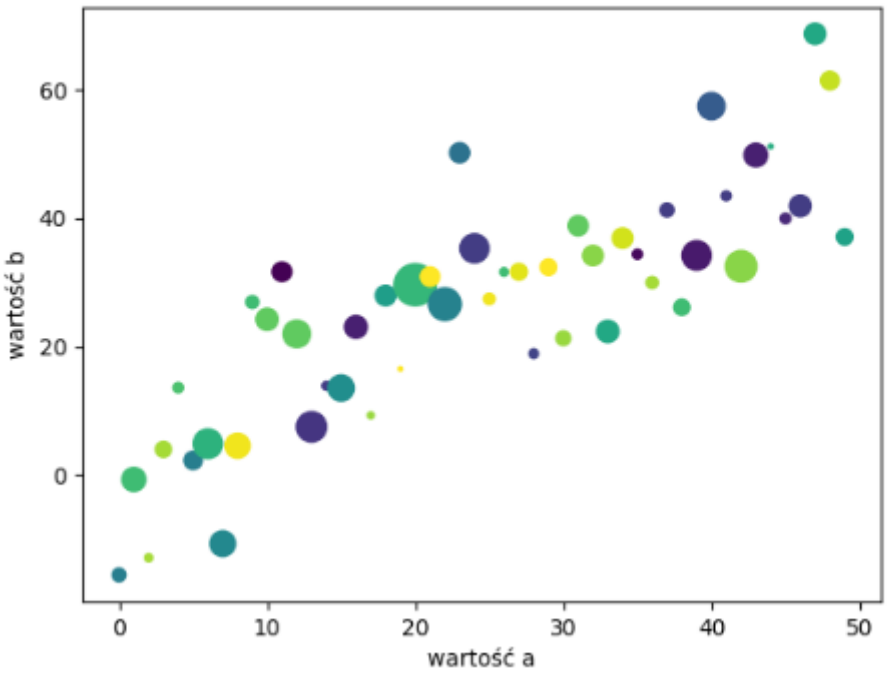
Listing 5

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

x = np.arange(0, 10, 0.1)
s = np.sin(x)
plt.plot(x, s, label="sin(x)")

#etykieta osi
plt.xlabel('x')
plt.ylabel('sin(x)')

#tytuł wykresu
plt.title('Wykres sin(x)')

#umieszczamy legendę na wykresie
plt.legend()

plt.show()
```



Listing 6

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

#dane w postaci słownika, ale równie dobrze może to być Pandas
DataFrame

data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

#aby w ten sposób przekazać parametry wykresu należy dodać
niezbędny parametr data, który zawiera dane dostępne poprzez
etykiety
#to oznacza, że 'a' jest równoważne data['a'] itd. Parametr c
to skrót od color, tutaj przekazywany w formie wektora
#wartości kolorów dla każdej kolejnej wartości wykresu.
Parametr s to scale - określa rozmiar, w tym przypadku punktu,
dla
#każdej kolejnej wartości wektora wykresu. Reasumując dla
pierwszego punktu wykresu będą brane poniższe wartości
print(f"a={data['a'][0]}, b={data['b'][0]}, c={data['c'][0]},
d={data['d'][0]}")
plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('wartość a')
plt.ylabel('wartość b')
plt.show()
```



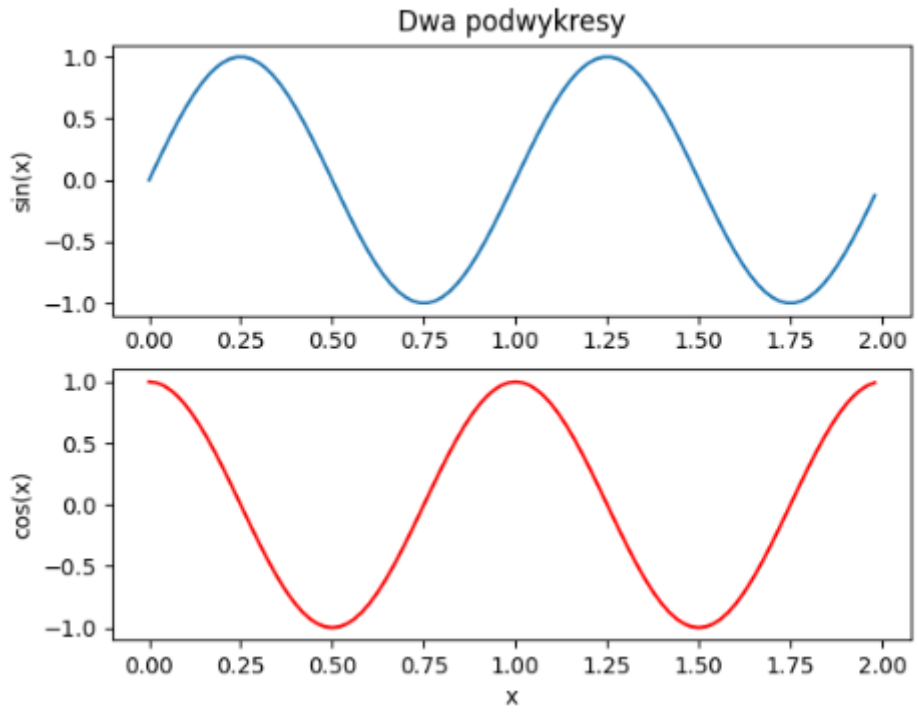
3. Podwykresy.

Podwykresy pozwalają na umieszczanie na jednym płótnie wielu wykresów zorganizowanych w formie gridu. Podajemy wymiary gridu czyli liczbę wierszy oraz liczbę kolumn. Służy to tego funkcja subplot, która przyjmuje 3 argumenty (nrows, ncols, index). Odpowiednio jest to ilość wierszy gridu, ilość kolumn oraz indeks definiowanego właśnie wykresu (indeksy rozpoczynają się od 1 i kończą na nrows*ncols).

Listing 7

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


x1 = np.arange(0, 2, 0.02)
x2 = np.arange(0, 2, 0.02)

y1 = np.sin(2 * np.pi * x1)
y2 = np.cos(2 * np.pi * x2)

plt.subplot(2, 1, 1,)
plt.plot(x1, y1, '-')
plt.title('wykres sin(x)')
plt.xlabel('x')
plt.ylabel('sin(x)')
```

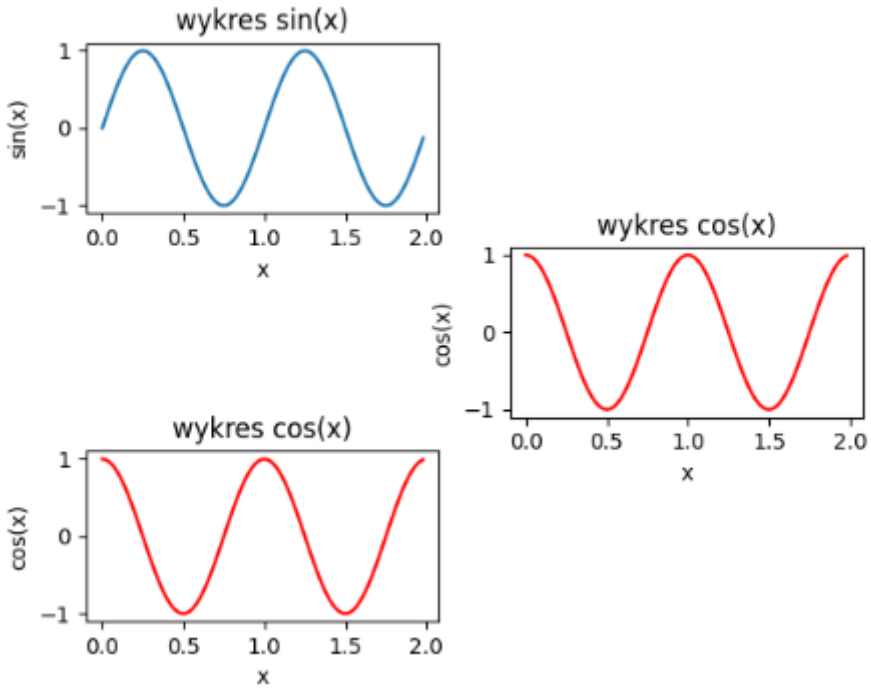
```
ax = plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r-')

plt.xlabel('x')
plt.ylabel('cos(x)')
plt.title('wykres cos(x)')
plt.subplots_adjust(hspace=0.5)
plt.show()
```



Listing 8

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

x1 = np.arange(0.0, 2.0, 0.02)
x2 = np.arange(0.0, 2.0, 0.02)
y1 = np.sin(2 * np.pi * x1)
y2 = np.cos(2 * np.pi * x2)

fig, axs = plt.subplots(3, 2, )
axs[0, 0].plot(x1, y1, '-')
axs[0, 0].set_title('wykres sin(x)')
axs[0, 0].set_xlabel('x')
```

```
axs[0, 0].set_ylabel('sin(x)')


axs[1, 1].plot(x2, y2, 'r-')
axs[1, 1].set_title('wykres cos(x)')
axs[1, 1].set_xlabel('x')
axs[1, 1].set_ylabel('cos(x)')

axs[2, 0].plot(x2, y2, 'r-')
axs[2, 0].set_title('wykres cos(x)')
axs[2, 0].set_xlabel('x')
axs[2, 0].set_ylabel('cos(x)')

fig.delaxes(axs[0, 1])
fig.delaxes(axs[1, 0])
fig.delaxes(axs[2, 1])
plt.show()
```

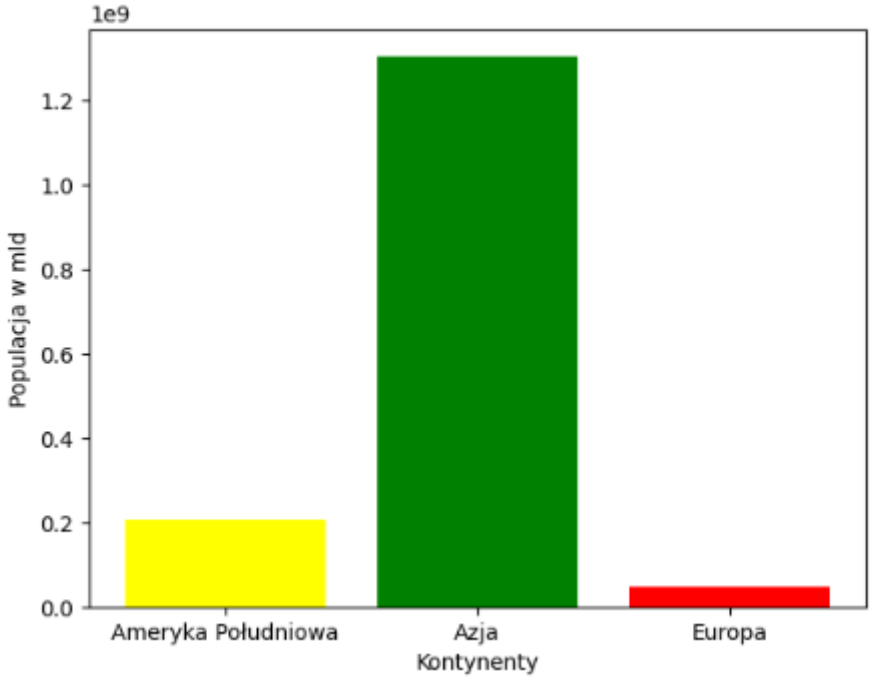


Poniższy listing przedstawia prosty przykład wykresu słupkowego.

Listing 9

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

```
data = {'Kraj': ['Belgia', 'Indie', 'Brazylia',
'Polska'],
        'Stolica': ['Bruksela', 'New Delhi',
'Brasilia', 'Warszawa'],
        'Kontynent': ['Europa', 'Azja', 'Ameryka
Południowa', 'Europa'],
        'Populacja': [11190846, 1303171035,
207847528, 38675467]}
df = pd.DataFrame(data)
print(df)
grupa = df.groupby('Kontynent')
etykiety = list(grupa.groups.keys())
wartosci = list(grupa.agg('Populacja').sum())

plt.bar(x=etykiety, height=wartosci, color=['yellow',
'green', 'red'])
plt.xlabel('Kontynenty')
plt.ylabel('Populacja w mld')
plt.show()
```

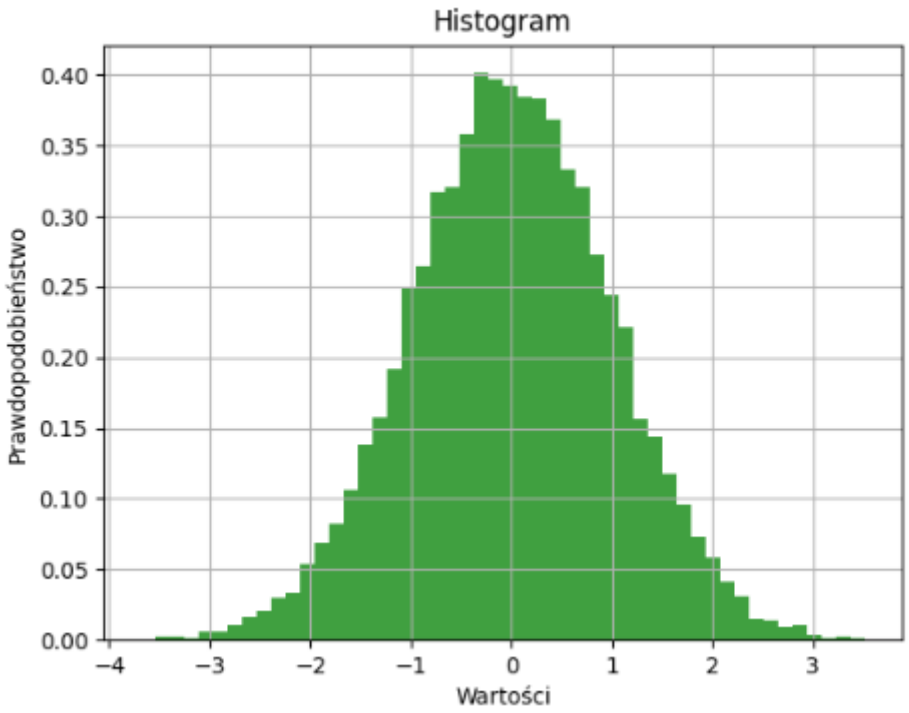


Popularnym typem wykresów dla zaprezentowania rozkładów prawdopodobieństwa są histogramy.

Listing 10

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

```
x = np.random.randn(10000)

# bins oznacza ilość "koszy" czyli słupków, do których mają
wpadać wartości z wektora x
# facekolor oznacza kolor słupków
# alpha to stopień przezroczystości wykresu
# density oznacza czy suma ilości zostanie znormalizowana do
rozkładu prawdopodobieństwa (czyli przedział 0, 1)
plt.hist(x, bins=50, facecolor='g', alpha=0.75, density=True)

plt.xlabel('Wartości')
plt.ylabel('Prawdopodobieństwo')
plt.title('Histogram')
#wyświetlanie siatki
plt.grid()
plt.show()
```



Listing 11

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

df = pd.read_csv('dane.csv', header=0, sep=";",
decimal=".")
print(df)
seria = df.groupby('Imię i nazwisko')['Wartość
```

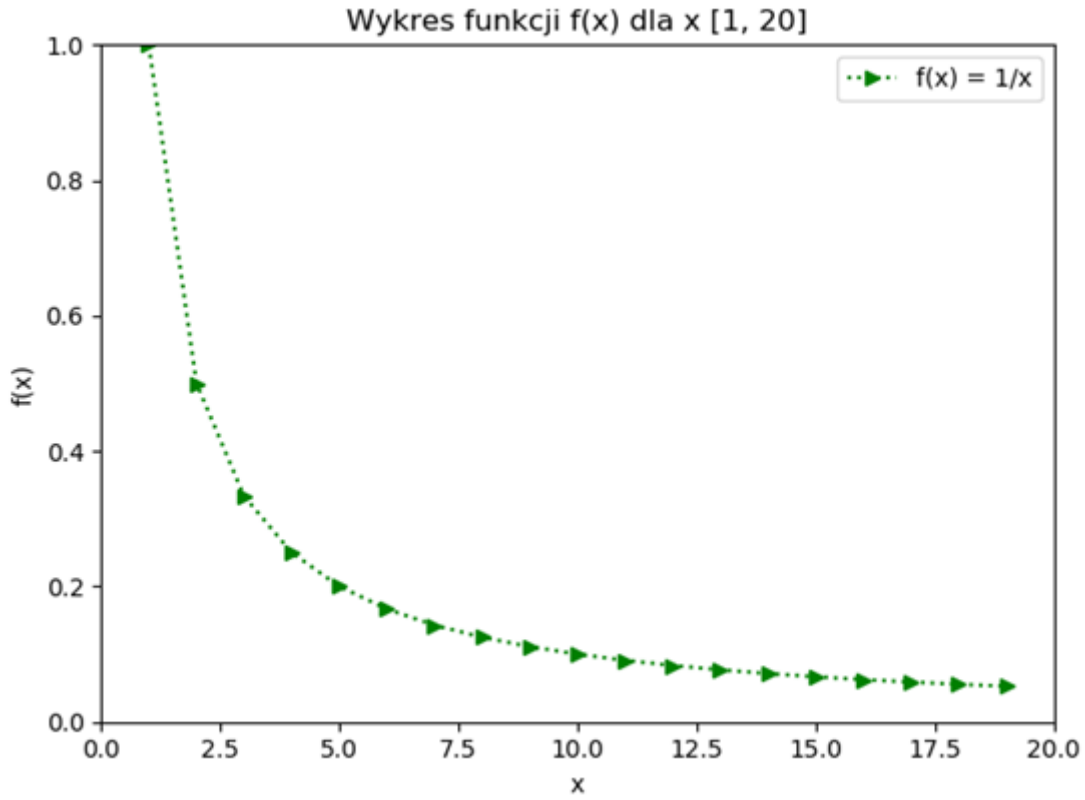
```
zamówienia'].sum()
wedges, texts, autotext = plt.pie(x=seria,
labels=seria.index, autopct=lambda pct:
"{:.1f}%".format(pct),

textprops=dict(color="black"), colors=['red',
'green'])
plt.title('Suma zamówień dla sprzedawców')
plt.legend(loc='lower right')
plt.ylabel('Procentowy wynik wartości zamówienia')
plt.show()
```

Zadanie 1

Na wykresie wyświetl wykres liniowy funkcji f(x) = 1/x dla x ϵ [1, 20]. Dodaj etykietę do linii wykresu i wyświetl legendę. Dodaj odpowiednie etykiety do osi wykresu (‘x’, ’f(x)’) oraz ustaw zakres osi na (0, 1) oraz (1, długość wektora x).

Zadanie 2

Zmodyfikuj wykres z zadania 1 tak, żeby styl wykresu wyglądał tak jak na poniższym zrzucie ekranu.



Zadanie 3

Na jednym wykresie wygeneruj wykresy funkcji sin(x) oraz cos(x) dla x ϵ [0, 30] z krokiem 0.1. Dodaj etykiety i legendę do wykresu.

Zadanie 4

Korzystając ze zbioru danych Iris (https://archive.ics.uci.edu/ml/datasets/iris) wygeneruj wykres punktowy, gdzie wektor x to wartość ‘sepal length’ a y to ‘sepal width’, dodaj paletę kolorów c na przykładzie listingu 6 a parametr s niech będzie wartością absolutną z różnicy wartości poszczególnych elementów wektorów x oraz y.

Zadanie 5

Korzystając z biblioteki pandas wczytaj zbiór danych z narodzinami dzieci przedstawiony w lekcji 8. Następnie na jednym płótnie wyświetl 3 wykresy (jeden wiersz i 3 kolumny). Dodaj do wykresów stosowne etykiety. Poustawiaj różne kolory dla wykresów.

1 wykres – wykres słupkowy przedstawiający ilość narodzonych dziewczynek i chłopców w całym okresie.

2 wykres – wykres liniowy, gdzie będą dwie linie, jedna dla ilości urodzonych kobiet, druga dla mężczyzn dla każdego roku z osobna. Czyli y to ilość narodzonych kobiet lub mężczyzn (dwie linie), x to rok.

3 wykres – wykres słupkowy przedstawiający sumę urodzonych dzieci w każdym roku.

Zadanie 6

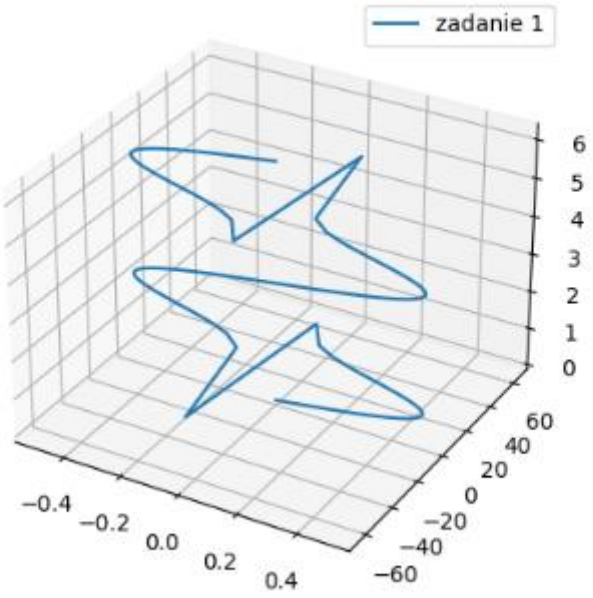
Korzystając z pliku zamówienia.csv (Pandas) policz sumy zamówień dla każdego sprzedawcy i wyświetl wykres kołowy z procentowym udziałem każdego sprzedawcy w ogólnej sumie zamówień. Poszukaj w Internecie jak dodać cień do takiego wykresu i jak działa atrybut ‘explode’ tego wykresu. Przetestuj ten atrybut na wykresie.

Matplotlib wykresy 3D

1. Wykresy 3D

Przykład 1 Wykresy liniowe

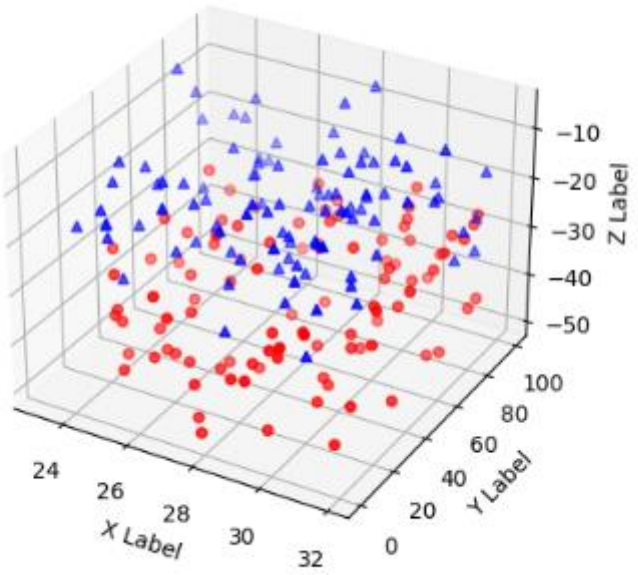
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
print(type(ax))
t = np.linspace(0, 2 * np.pi, 100)
z = t
x = np.sin(t)*np.cos(t)
y = np.tan(t)
ax.plot(x, y, z, label='zadanie 1')
ax.legend()
plt.show()
```



Przykład 2 Wykresy punktowe

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


# Ustawiamy seed by za każdym razem wyglądało identycznie
np.random.seed(19680801)


def randrange(n, vmin, vmax):
```

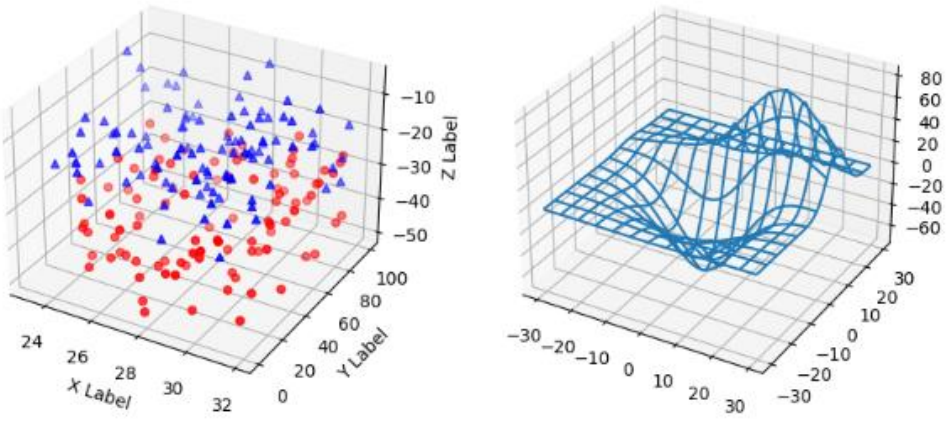
```
    '''
    Funkcja wspomagająca może tworzyć macierz losowych liczb o
    kształcie(n, )
    '''
    return (vmax - vmin)*np.random.rand(n) + vmin


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100

# Dla każdego zbioru styli i zakresów wygeneruje n losowych
punktów
# zdefiniowane przez x z [23, 32], y z [0, 100], z z [zlow,
zhigh].
for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -
30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zlow, zhigh)
    ax.scatter(xs, ys, zs, c=c, marker=m)

ax.set_xlabel( 'X Label' )
ax.set_ylabel( 'Y Label' )
ax.set_zlabel( 'Z Label' )
plt.show()
```



2. Wiele wykresów w jednym wywołaniu.

Przykład 3

```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
from mpl_toolkits.mplot3d.axes3d import get_test_data


# szerokość 2 razy większa niż wysokość
fig = plt.figure(figsize=plt.figaspect(0.5))
#===============
# Pierwszy wykres
#===============
# osie dla pierwszego wykresu
ax = fig.add_subplot(1, 2, 1, projection='3d')
np.random.seed(19680801)
def randrange(n, vmin, vmax):
    ''' Funkcja wspomagająca może tworzyć macierz losowych
liczb o kształcie(n, ) '''
    return (vmax - vmin)*np.random.rand(n) + vmin

n = 100
# Dla każdego zbioru styli i zakresów wygeneruje n losowych
punktów
# zdefiniowane przez x z [23, 32], y z [0, 100], z z [zlow,
zhigh].
for c, m, zlow, zhigh in [('r', 'o', -50, -25), ('b', '^', -
30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zlow, zhigh)
    ax.scatter(xs, ys, zs, c=c, marker=m)


ax.set_xlabel( 'X Label' )
ax.set_ylabel( 'Y Label' )
ax.set_zlabel( 'Z Label' )
#===============
# Drugi wykres
#===============
# Osie dla drugiego wykresu
ax = fig.add_subplot(1, 2, 2, projection='3d')
X, Y, Z = get_test_data()
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.show()
```

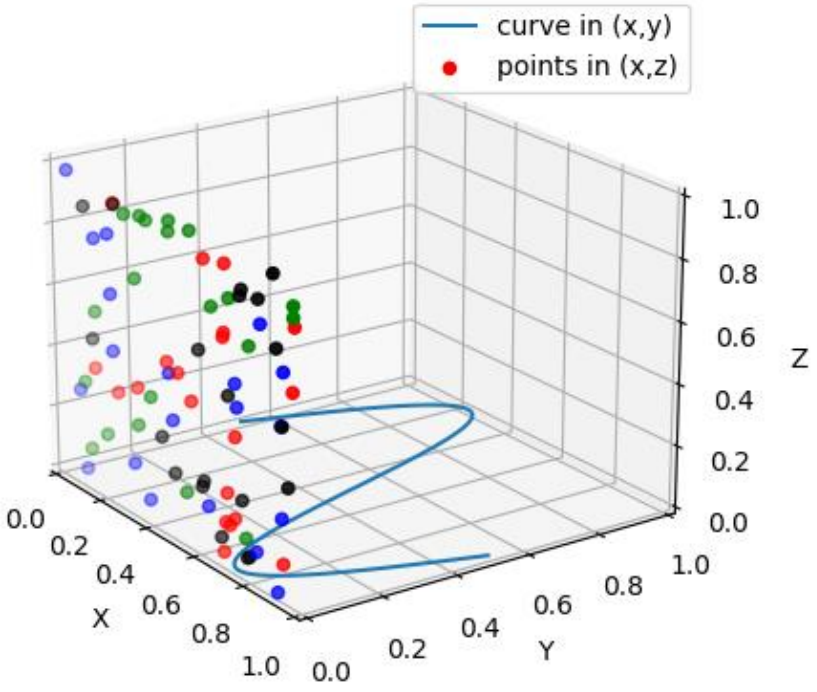


3. Wiele typów wykresów w jednej przestrzeni.

Przykład 4

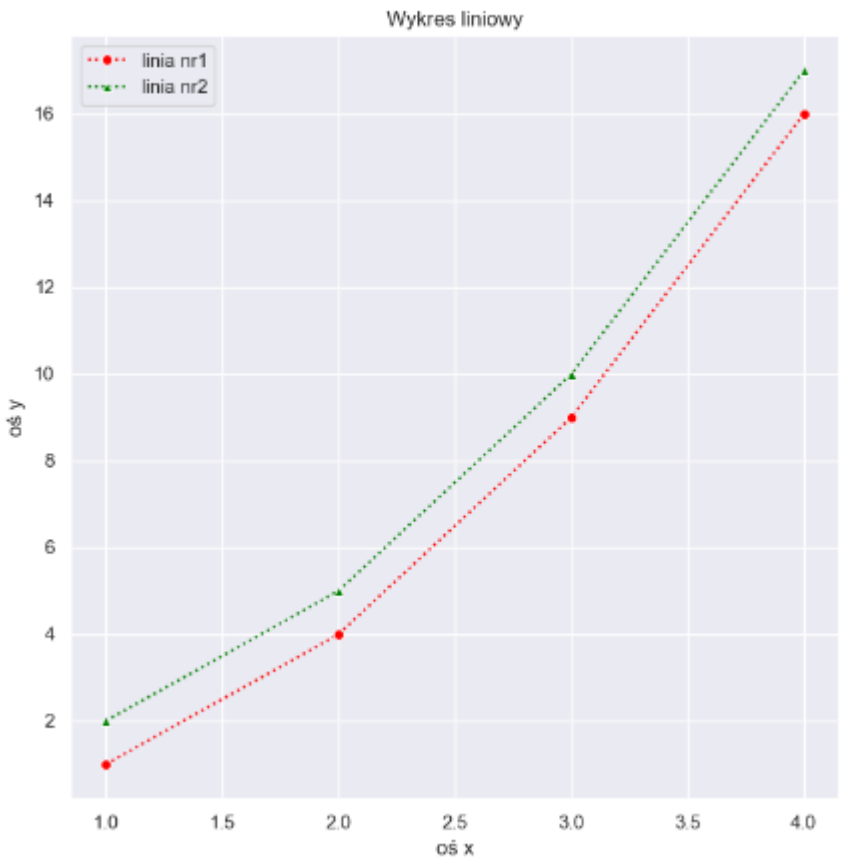
```
import numpy as np
import matplotlib.pyplot as plt


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(0, 1, 100)
y = np.sin(x * 2 * np.pi) / 2 + 0.5
ax.plot(x, y, zs=0, zdir='z', label='curve in (x,y)')
colors = ('r', 'g', 'b', 'k')
np.random.seed(19680801)
x = np.random.sample(20 * len(colors))
y = np.random.sample(20 * len(colors))
c_list = []
for c in colors:
    c_list.extend([c] * 20)

# przez użycie zdir='y', wartość y dla tych punktów jest równe
zs czyli 0
# punkty (x,y) są nakładane na osiach x i z.
ax.scatter(x, y, zs=0, zdir='y', c=c_list, label='points in
(x,z)')

# Limity dla legendy
ax.legend()
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_zlim(0, 1)
ax.set_xlabel('X')
ax.set_ylabel('Y')
```

```
ax.set_zlabel('Z')
# Ustawienie kąta nachylenia przy generowaniu wykresu
# oś y=0
ax.view_init(elev=20., azim=-35)
plt.show()
```



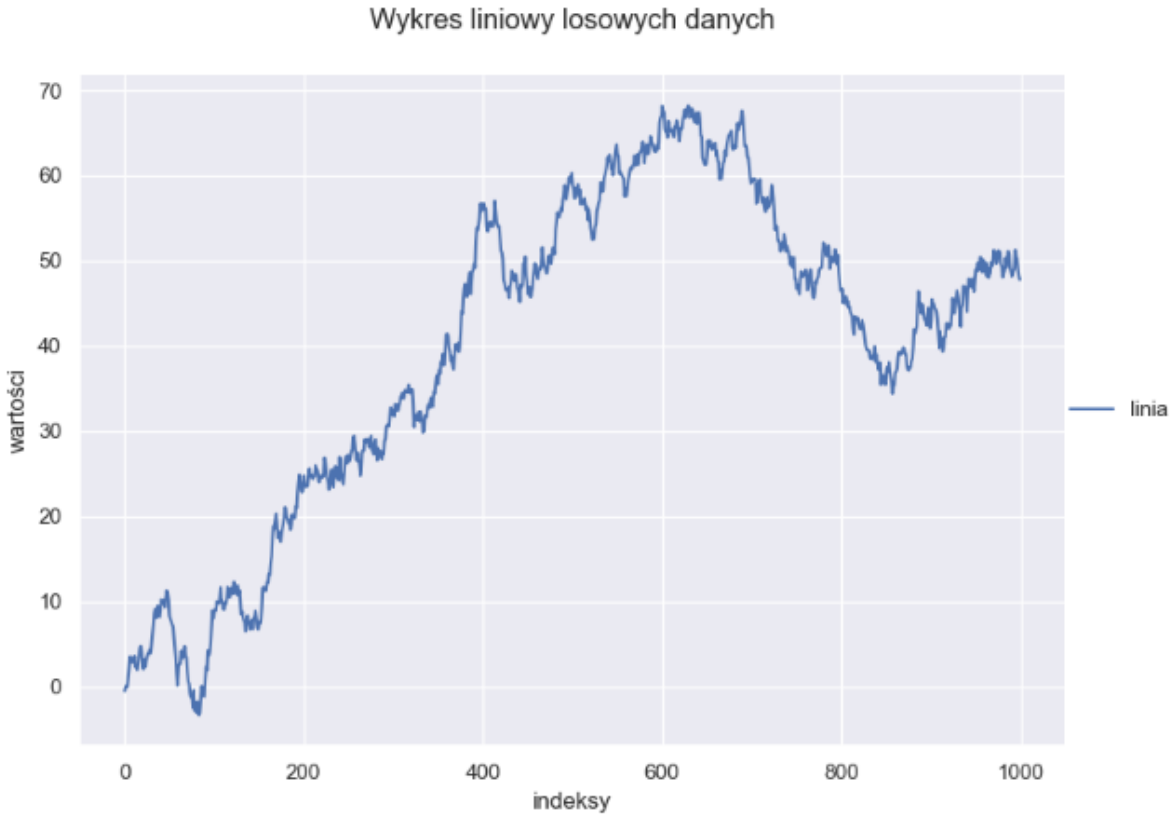
1. Wykres liniowy

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

#wykres liniowy
sns.set(rc={'figure.figsize': (8, 8)})
sns.lineplot(x=[1, 2, 3, 4], y=[1, 4, 9, 16],
             label='linia nr1', color='red', marker='o', linestyle=':')
sns.lineplot(x=[1, 2, 3, 4], y=[2, 5, 10, 17],
             label='linia nr2', color='green', marker='^',
linestyle=':')
plt.xlabel('oś x')
plt.ylabel('oś y')
plt.title('Wykres liniowy')
plt.show()
```



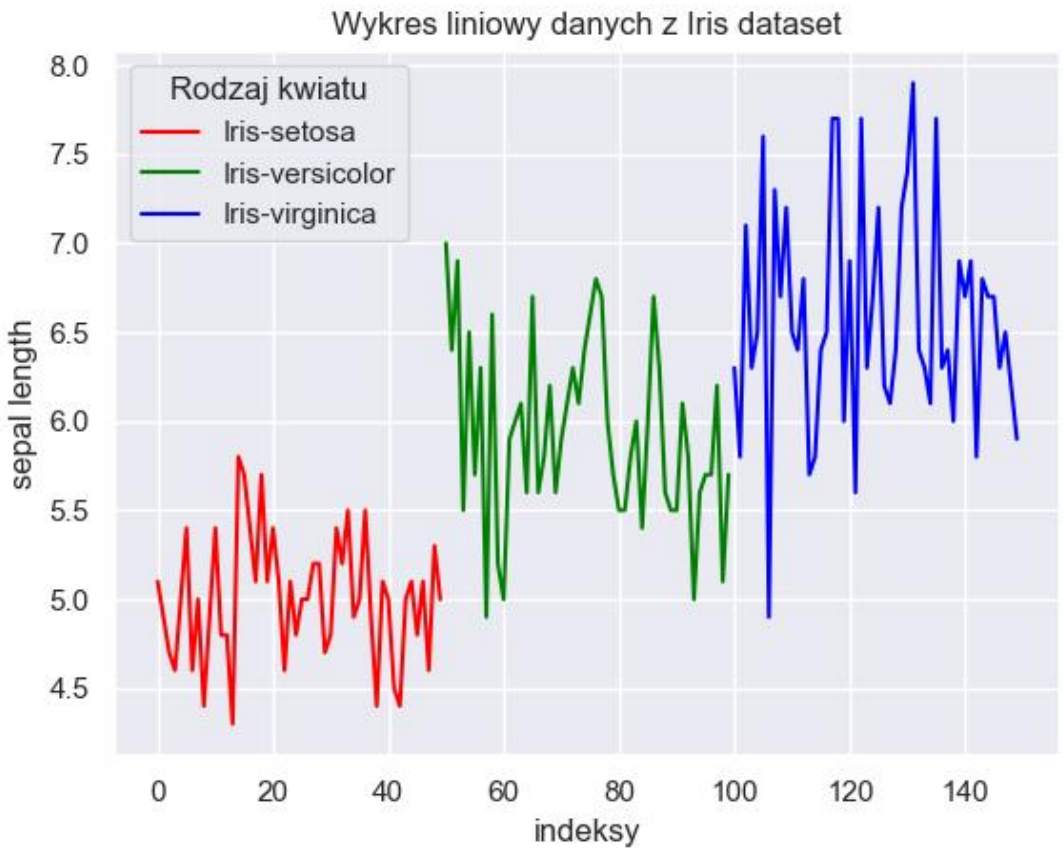
2. Wykres liniowy z wykorzystaniem serii danych

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

s = pd.Series(np.random.randn(1000))
s = s.cumsum()
sns.set()
wykres = sns.relplot(kind='line', data=s, label='linia')
wykres.fig.set_size_inches(8, 6)
wykres.fig.suptitle('Wykres liniowy losowych danych')
wykres.set_xlabels('indeksy')
wykres.set_ylabels('wartości')
wykres.add_legend()
wykres.figure.subplots_adjust(left=0.1, right=0.9,
bottom=0.1, top=0.9)
plt.show()
```



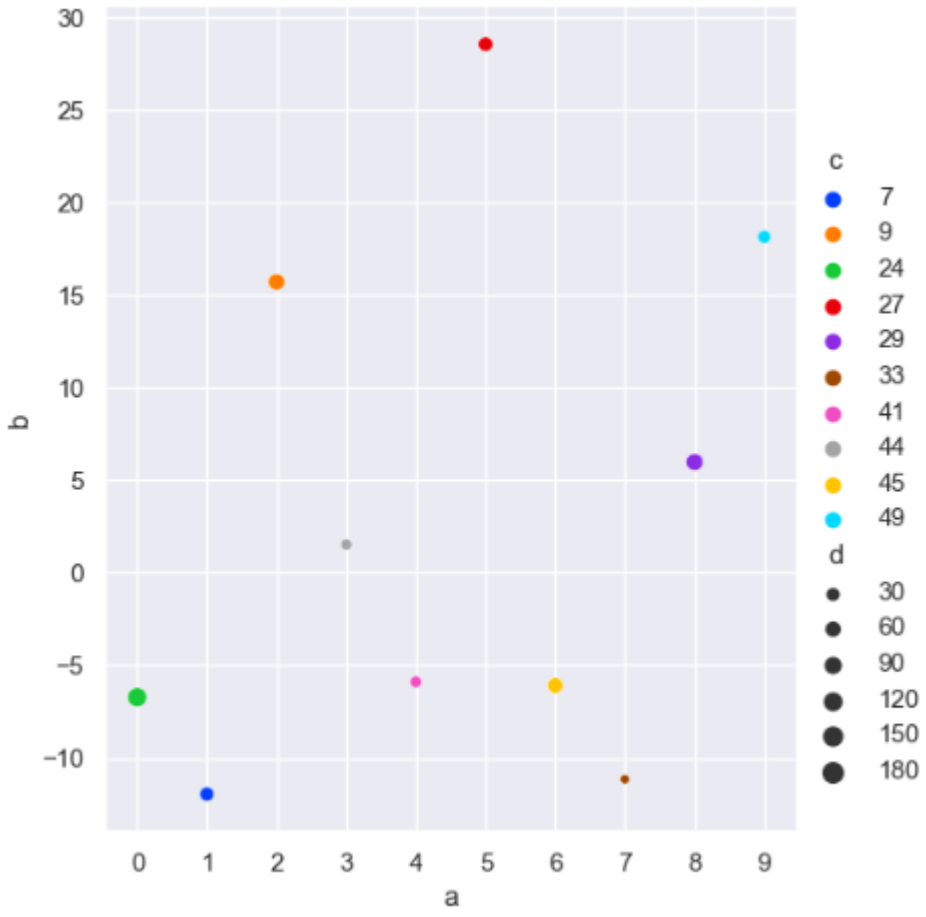
3. Wykres liniowy z wykorzystaniem ramki danych

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

sns.set()
df = pd.read_csv('iris.data', header=0, sep=',',
decimal='.')
print(df)
wykres = sns.lineplot(data=df, x=df.index, y='sepal length',
hue='class')
wykres.set_xlabel('indeksy')
wykres.set_title('Wykres liniowy danych z Iris dataset')
wykres.legend(title='Rodzaj kwiatu')
plt.show()
```



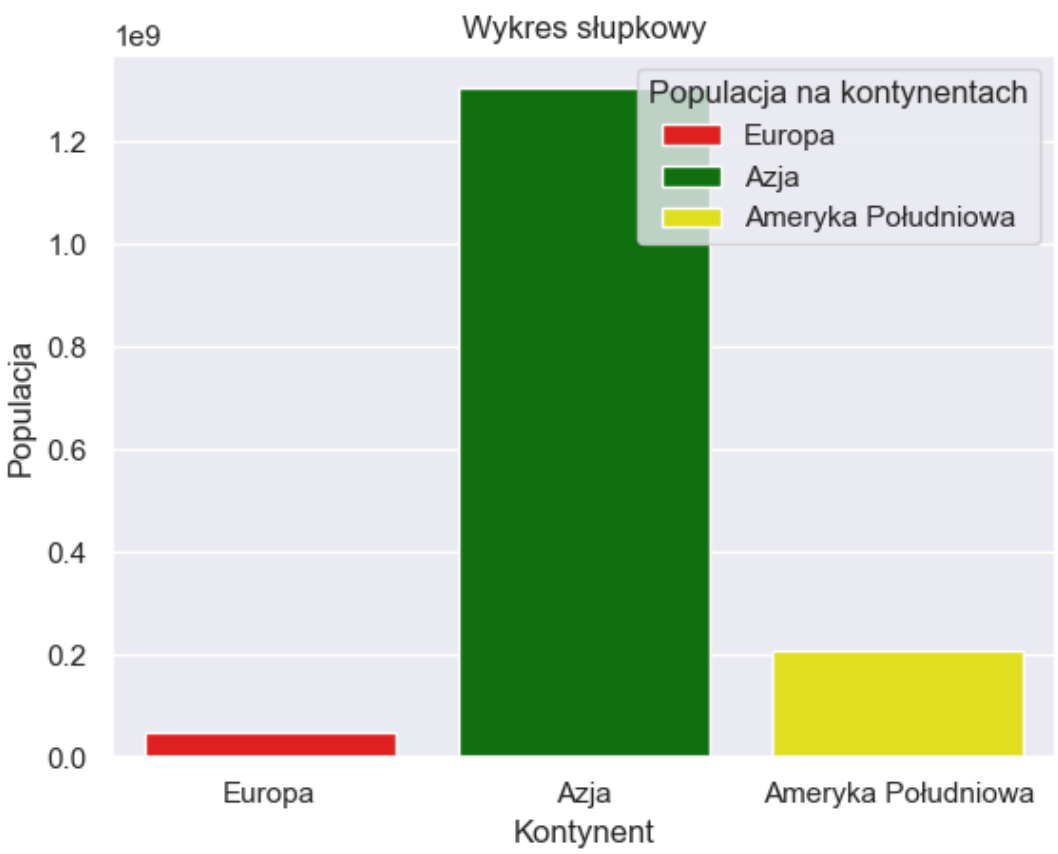
4. Wykres punktowy

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


sns.set()
data = {'a': np.arange(10),
        'c': np.random.randint(0, 50, 10),
        'd': np.random.randn(10)}
data['b'] = data['a'] + 10 * np.random.randn(10)
data['d'] = np.abs(data['d']) * 100
print(data['c'])
print(data['d'])
df = pd.DataFrame(data)
plot = sns.relplot(data=df, x="a", y="b", hue="c",
palette='bright', size="d", legend=True)
plot.fig.set_size_inches(6, 6)
plot.set(xticks=data['a'])
plt.show()
```



5. Wykres kolumnowy

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


data = {'Kraj': ['Belgia', 'Indie', 'Brazylia', 'Polska'],
        'Stolica': ['Bruksela', 'New Delhi', 'Brasilia',
'Warszawa'],
        'Kontynent': ['Europa', 'Azja', 'Ameryka
Południowa', 'Europa'],
        'Populacja': [11190846, 1303171035, 207847528,
38675467]}
df = pd.DataFrame(data)
sns.set()
# plot = sns.catplot(data=df, x='Kontynent', y='Populacja',
kind='bar', ci=None, hue='Kontynent', estimator=np.sum,
#                    dodge=False, palette=['red', 'green',
'yellow'], legend_out=False)
# plot.fig.set_size_inches(7, 6)
# plot.add_legend(title='Populacja na kontynentach',
loc='upper right')
# plot.fig.suptitle('Populacja na kontynentach')
plot = sns.barplot(data=df, x='Kontynent', y='Populacja',
ci=None, hue='Kontynent', estimator=np.sum,
                   dodge=False, palette=['red', 'green',
'yellow'])
plot.legend(title='Populacja na kontynentach')
plot.set(title='Wykres słupkowy')
plt.show()
```


Wykres słupkowy
1e9
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Populacja
Populacja na kontynentach
Europa
Azja
Ameryka Południowa
Europa
Azja
Ameryka Południowa
Kontynent