# Matematyczne aspekty analizy danych
# semestr zimowy 2024/2025

Dr Anna Muranova
UWM w Olsztynie

Wykład 12

# Rozdział 8. Python: biblioteka Pandas

# Biblioteka `Pandas`

Pandas to biblioteka Python, która umożliwia efektywną pracę z danymi w postaci tabelarycznej. Pandas współpracuje z biblioteką Matplotlib i umożliwia szybkie generowanie wykresów.
"the name is derived from the term "panel data", an econometrics term for multidimensional structured data sets."
Ściągawka: `https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf`

```
import pandas as pd
```

Dzięki Pandas łatwo jesteśmy w stanie przeprowadzać takie operacje jak: czyszczenie danych, normalizacja danych, wizualizacja danych, analiza statyczna, ładowanie oraz zapisywanie danych i wiele więcej.
Głównym celem biblioteki Pandas jest ułatwienie pracy z danymi, dlatego Pandas wprowadza dwie struktury danych: Series i DataFrame. Zrozumienie tych struktur jest kluczowe do efektywnego korzystania z tej biblioteki.

## Series

Series to jednowymiarowa struktura danych, a właściwie tablicy (ndarray), podobna do listy lub kolumny w tabeli. Każdy element (np. liczby całkowite, listy, obiekty, tuple) w Series ma przypisany identyfikator, który nazywany jest indeksem. Series przechowuje dane jednego typu.

```
import pandas as pd

s_int = pd.Series([1, 32, -37, 91, 12, 11, -5],
                          index = ['a','b','c','d','e','f','g'])
s_str = pd.Series(['My', 'name','is', 'Anna', 'Muranova','.'])
print(s_int)
print(s_str)
my_list = [4, 3, 2, 1, 0 ,-1, -2, -3, -4]
s_list = pd.Series(my_list)
print(s_list)
int_list = s_int.tolist()
print(int_list)
s_float = pd.Series([1.5, 32.3, -37.1, 91, 12.9, 11, -5.2],
                index = ['a','b','c','d','e','f','g'])
print(s_float[(s_float < 0)])
```

## Data Frame

DataFrame to dwuwymiarowa struktura danych podobna do tabeli w bazie danych lub arkusza kalkulacyjnego Excela. DataFrame składa się z wierszy i kolumn – każda kolumna w DataFrame to Series. Jak pewnie się domyślasz, mimo że dana kolumna zawiera tylko jeden typ danych, to DataFrame może zawierać wiele kolumn, z których każda ma dane innego typu.

```
import pandas as pd

my_list = [1, 32, -37, 91, 12, 11, -5]
df_list = pd.DataFrame (my_list, index = ['a','b','c','d','e','f','g'],
                columns = ['numbers'])
print(df_list)
df = pd.DataFrame ([[1, 2, 4, 5],[-3, 8, 0.5, 10],[2, -5, 7, 3]],
            index = ['l1','l2','l3'], columns = ['a','b','c','d'])
print(df)
print(df.iloc[1,1:3])
#print(df[1,1:3])
print(df.max())
print(df['a'].max())
#print(df['l2'].max())
print(df.sort_values('a'))
```

# Przykład: Data Frame z Series

```
import pandas as pd

x = pd.Series([-2,-1,0,1,2])
y = pd.Series([-3,-1,1,2.5,3.5])
data = pd.DataFrame ({'x':x,'y':y})

print(data)
```

# Przykład

Dla podanych tablic stworzymy ramki danych (numery jako indeksy, nazwy `Name`, `Age`, `...` jako nazwy kolumn)

| ID | Name | Age |
|---|---|---|
| 2312 | Anna | 21 |
| 2336 | Zofia | 40 |
| 2942 | Sylwia | 13 |
| 9840 | Katarzyna | 31 |
| 2794 | Teresa | 34 |
| 2933 | Zenon | 28 |

| ID | Name | W | H | Glasses |
|---|---|---|---|---|
| 2942 | Sylwia | 64 | 151 | F |
| 9840 | Katarzyna | 69 | 177 | T |
| 2794 | Teresa | 74 | 170 | F |
| 8891 | Tomasz | 61 | 157 | T |
| 8111 | Cezary | 66 | 151 | F |
| 2933 | Zenon | 61 | 153 | T |

## Tworzenie

```
import pandas as pd

df1 = pd.DataFrame([[2942,9840,2794,8891,8111,2933,8301,9125],
        [ 'Sylwia', 'Katarzyna', 'Teresa', 'Tomasz', 'Cezary',
        'Zenon', 'Filip', 'Adrian'],[13, 31, 34, 14, 13, 28, 20, 15]]).
df1.columns = ['ID', 'Name','Age']
weight = [65, 80, 64, 69, 74, 61, 66, 61]
height = [179, 179, 151, 177, 170, 157, 151, 153]
glasses = [False, True, False, True, False, True, False, True]
df2 = pd.DataFrame([[2312,2336,2942,9840,2794,8891,8111,2933],
       ['Anna', 'Zofia', 'Sylwia', 'Katarzyna', 'Teresa', 'Tomasz',
        'Cezary', 'Zenon'],weight,height,glasses]).T
df2.columns = ['ID','Name','W', 'H','Gl']
print(df2)
```

## Rożny przykłady

```
#Laczenie inner
df0 = pd.merge(df1,df2)
print(df0)
#sortowanie imion
print(df0.sort_values(by=['Name']))
#imiona osób noszących okulary
DfGl=df0[df0['Gl']]
print(DfGl)
# imiona osób w wieku z przedziału lat $[20, 30]$
DfA=df0[(df0['Age']>20)*(df0['Age']<30)]
print(DfA)
#kolumną z bmi
df0['bmi']=(df0['W']*10000/df0['H']**2).astype(float).round()
                  .astype(int)
print(df0)
średni wiek i wyświetl na konsoli.
print(df0['Age'].mean().round(decimals=2))
#Grupowanie
print(df0.groupby(by='Gl'))
print(df0.groupby(by='Gl')['bmi'].mean().round(decimals=2))
print(df0.groupby(by='Gl')['Age'].mean().round(decimals=2))
```

# Wczytywanie danych z pliku

```
import pandas as pd

data = pd.read_csv('penguins.csv', sep=',', index_col=False,
          encoding='UTF-8')
print(data)
```

## Przykłady

```
#średnia waga w każdej plecie
print(data.groupby('sex')['body_mass_g'].mean())

#średnia waga w każdej plecie,
print(data.dropna().groupby('sex')['body_mass_g'].mean())
#średnia waga z podziałem na płeć  i gatunek
print(data.dropna().groupby(by=['sex','species'])
['body_mass_g'].mean())

#wszystkie wartości dla pingwinów z najmniejsza waga.
print(data[data['body_mass_g']==data['body_mass_g'].min()])
print(data[data['body_mass_g']==data['body_mass_g'].min()].to_string())

#ilość pingwinów gatunku 'Adelie' na każdej wyspie
print(data[data['species']=='Adelie'].groupby('island').size())

#ilość pingwinów każdego gatunku na każdej wyspie
print(data.groupby(by =['species','island']).size())
```

## Regresja liniowa: pandas i numpy 1

```
import pandas as pd
import numpy as np

x = pd.Series([-2,-1,0,1,2])
y = pd.Series([-3,-1,1,2.5,3.5])
beta = np.polyfit(x,y,1)[::-1]
print(beta)#[0.6  1.65]
print(beta[1]*x+beta[0])
```

# Regresja liniowa: pandas i numpy 2

```
import pandas as pd
import numpy as np

x = pd.Series([-2,-1,0,1,2])
y = pd.Series([-3,-1,1,2.5,3.5])
data = pd.DataFrame ({'x':x,'y':y})

print(data)
beta = np.polyfit(data['x'],data['y'],1)[::-1]
print(beta)#[0.6  1.65]
data[['z']]=pd.Series([beta[1]*x+beta[0]])
print(data)
```

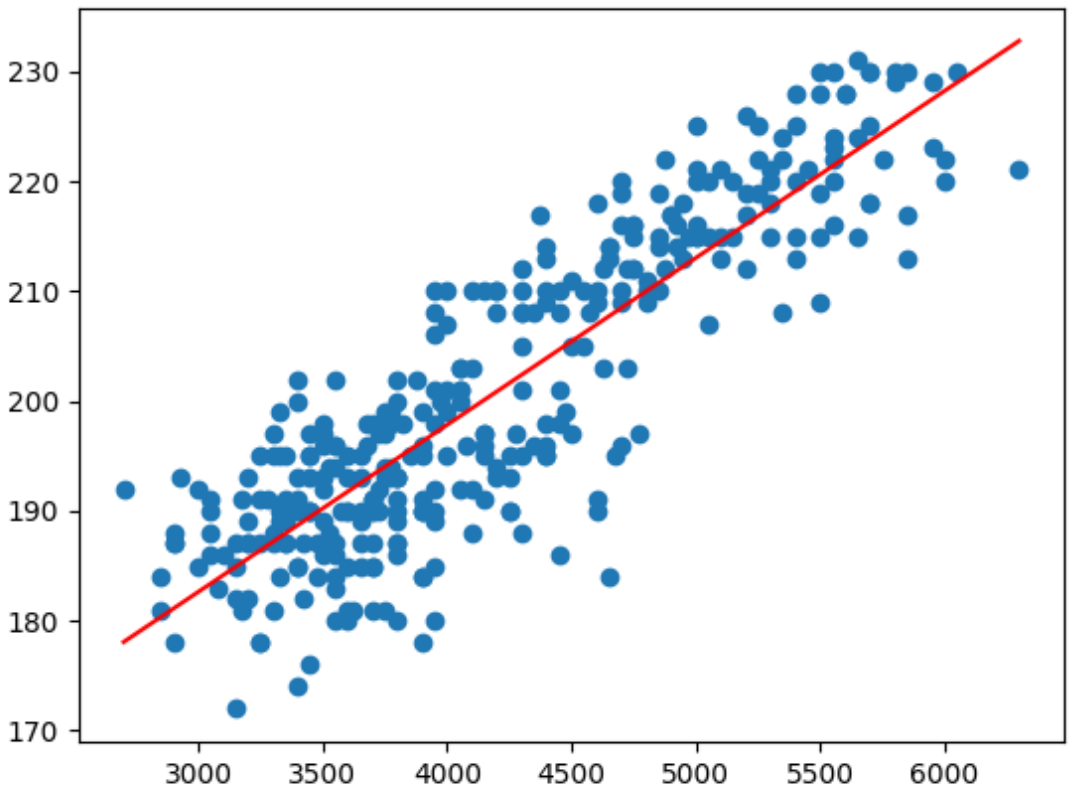
## Regresja liniowa: pandas i stats

```
import pandas as pd
from scipy import stats
x = pd.Series([-2,-1,0,1,2])
y = pd.Series([-3,-1,1,2.5,3.5])
data = pd.DataFrame ({'x':x,'y':y})

xi = data['x']#data[['x']]blad
yi = data['y']
print(stats.linregress(xi, yi))
```

## Regresja liniowa: pandas i stats

```
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import numpy as np

data = pd.read_csv('penguins.csv', sep=',',
          encoding='UTF-8')
data = data.dropna()
print(data)
flipper_length_mm=data['flipper_length_mm']
body_mass_g=data['body_mass_g']

regr = stats.linregress( body_mass_g, flipper_length_mm)
print(regr)
x = np.linspace(min(body_mass_g),max(body_mass_g))
y = regr.intercept + x*regr.slope
plt.plot(body_mass_g,flipper_length_mm, 'o')
plt.plot(x,y,'r')
plt.show()
```

# Regresja liniowa: pandas i stats, obrazek



# Rozdział 9. Python: biblioteka matplotlib

# Biblioteka `matplotlib`

`https://matplotlib.org/`

Zazwyczaj importuje się główny moduł tej biblioteki, czyli ‘pyplot’. Zawiera on niezbędne funkcje do generowania wykresów.

```
import matplotlib.pyplot as plt
```

Szczególnie wydajne jest połączenie możliwości tej biblioteki z innymi bibliotekami naukowymi, takimi jak NumPy, Pandas czy SciPy.

`https://www.w3schools.com/python/matplotlib_intro.asp`

## Stylu

Sama biblioteka korzysta z domyślnych stylów podczas „upiększania” i modyfikacji wykresów. Są to np. modyfikacje kolorystyczne, czcionki, dodawanie dodatkowych informacji, legend itd. Jeżeli chcesz zapoznać się z różnymi stylami, które oferuje Matplotlib, to zajrzyj do dokumentacji lub skorzystaj z komendy ‘print’

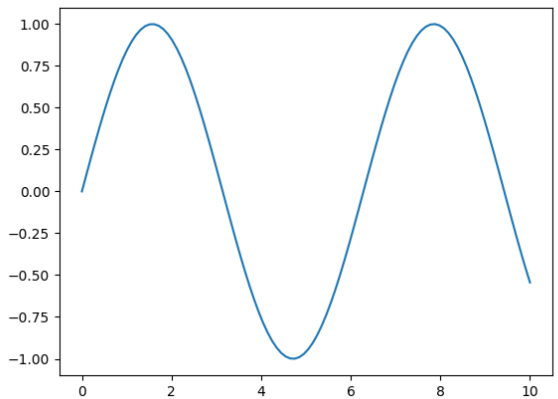
```
print(plt.style.available)
```

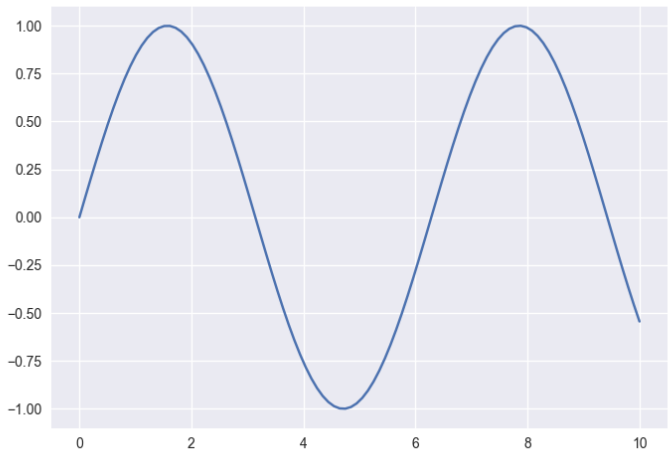
W celu użycia konkretnego stylu w twoim projekcie skorzystaj z funkcji

```
plt.style.use('nazwa_stylu')  # np. 'seaborn-v0_8'
```

# Wykres liniowy

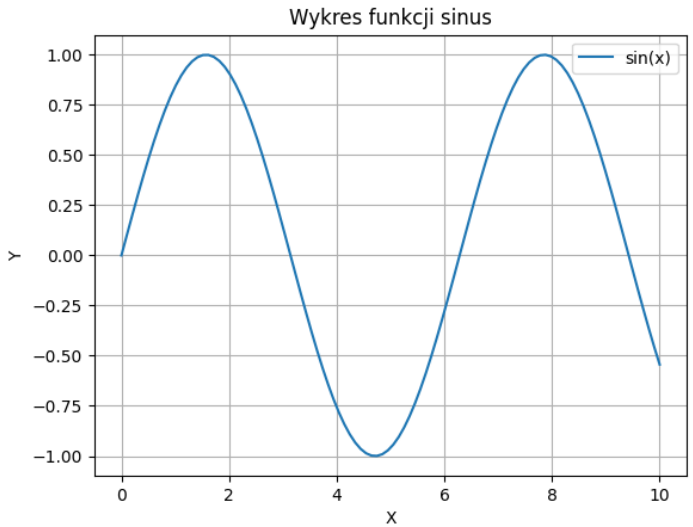
```
import matplotlib.pyplot as plt
import numpy as np

#plt.style.use('seaborn-v0_8')

# Przygotowanie danych
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)  # Rysowanie wykresu liniowego
plt.show()  # Wyświetlenie wykresu
```



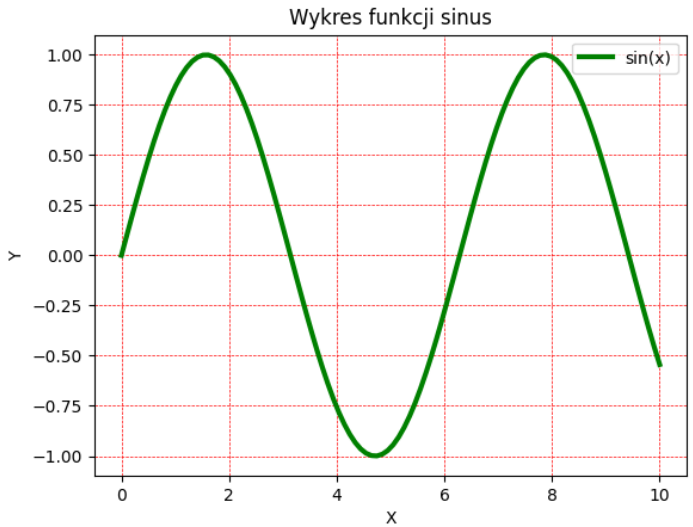
# Wykres liniowy: podpis

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

#plt.style.use('seaborn-v0_8')

# Przygotowanie danych
x = np.linspace(0, 10, 100)
y = np.sin(x)

#Dodanie etykiety dla serii danych (do legendy)
plt.plot(x, y, label='sin(x)')
plt.xlabel('X')  # Etykieta osi X
plt.ylabel('Y')  # Etykieta osi Y

# Tytuł wykresu
plt.title('Wykres funkcji sinus')

plt.legend()  # Dodanie legendy
plt.grid() #siatka
plt.show()  # Wyświetlenie wykresu
```



# Wykres liniowy: opcje zaawansowane

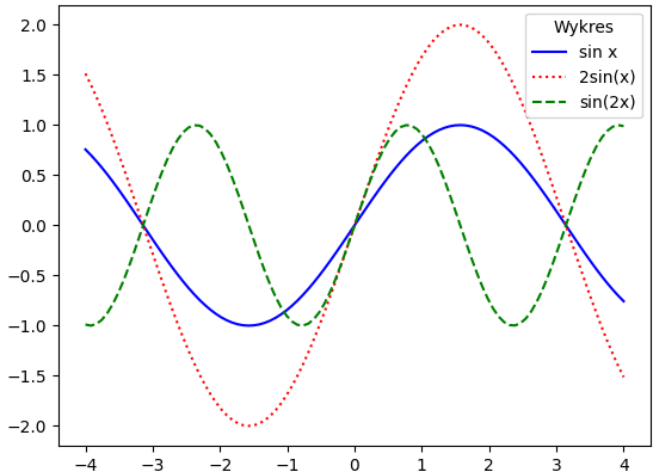
```
import matplotlib.pyplot as plt
import numpy as np



#plt.style.use('seaborn-v0_8')

# Przygotowanie danych
x = np.linspace(0, 10, 100)
y = np.sin(x)
```




```
plt.plot(x, y, label='sin(x)', color='green', linewidth=3)
       #Dodanie etykiety dla serii danych (do legendy)
plt.xlabel('X')  # Etykieta osi X
plt.ylabel('Y')  # Etykieta osi Y
plt.title('Wykres funkcji sinus')  # Tytuł wykresu
plt.legend()  # Dodanie legendy
plt.grid(color='r', linestyle='--', linewidth=0.5) #siatka
plt.show()  # Wyświetlenie wykresu
```

# Kilka wykresów w jednym okienku na jednej skale

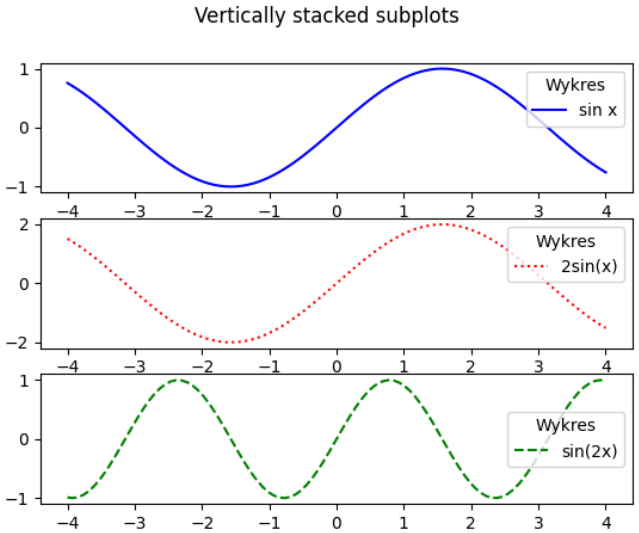
```
import numpy as np
import matplotlib.pyplot as plt
```




```
x = np.linspace(-4,4,100)
y = np.sin(2 * x)
y1 = 2*np.sin(x)
y2 = np.sin(x)
plt.plot(x, y2, 'blue', linestyle="-", label="sin x")
plt.plot(x, y1, 'red', linestyle=":", label="2sin(x)")
plt.plot(x, y, 'green', linestyle="--", label="sin(2x)")
plt.legend(title='Wykres')
plt.show()
```

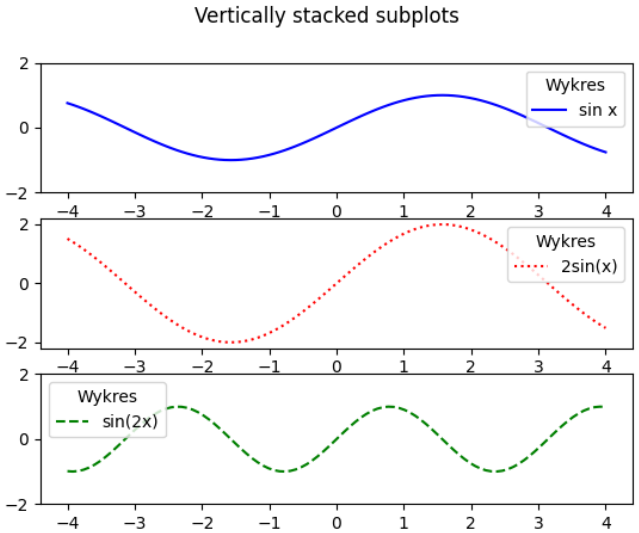
# Kilka wykresów w jednym okienku na rożnych skalach

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


x = np.linspace(-4,4,100)
y = np.sin(2 * x)
y1 = 2*np.sin(x)
y2 = np.sin(x)
fig, axs = plt.subplots(3)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(x, y2, 'blue', linestyle="-", label="sin x")
axs[1].plot(x, y1, 'red', linestyle=":", label="2sin(x)")
axs[2].plot(x,y, 'green', linestyle="--", label="sin(2x)")

#plt.ylim(-2,2)
axs[0].legend(title='Wykres')
axs[1].legend(title='Wykres')
axs[2].legend(title='Wykres')
plt.show()
```



# Drugi sposób na zakresy

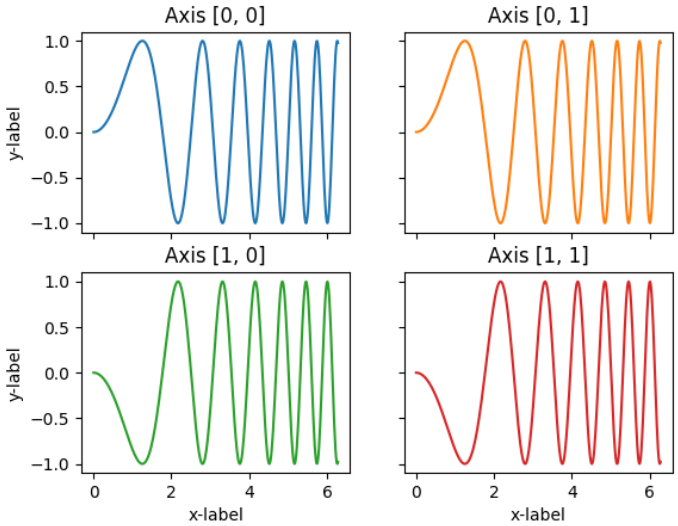
```
import numpy as np
import matplotlib.pyplot as plt
```




```
x = np.linspace(-4,4,100)
y = np.sin(2 * x)
y1 = 2*np.sin(x)
y2 = np.sin(x)
fig, axs = plt.subplots(3)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(x, y2, 'blue', linestyle="-", label="sin x")
axs[0].set_ylim(-2,2)
axs[1].plot(x, y1, 'red', linestyle=":", label="2sin(x)")
axs[2].plot(x,y, 'green', linestyle="--", label="sin(2x)")
axs[2].set_ylim(-2,2)
axs[0].legend(title='Wykres')
axs[1].legend(title='Wykres')
axs[2].legend(title='Wykres')
plt.show()
```

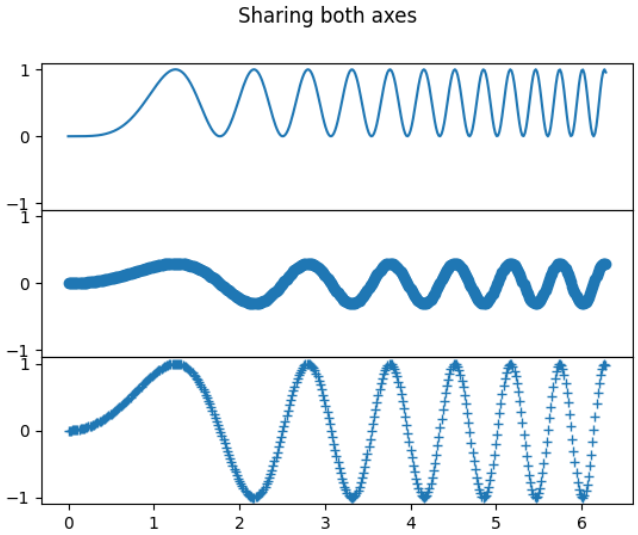
# Jeszcze przykład 1

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x, y)
axs[0, 0].set_title('Axis [0, 0]')
axs[0, 1].plot(x, y, 'tab:orange')
axs[0, 1].set_title('Axis [0, 1]')
axs[1, 0].plot(x, -y, 'tab:green')
axs[1, 0].set_title('Axis [1, 0]')
axs[1, 1].plot(x, -y, 'tab:red')
axs[1, 1].set_title('Axis [1, 1]')
for ax in axs.flat:
    ax.set(xlabel='x-label', ylabel='y-label')
# Hide x labels and tick labels for top plots
# and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()
plt.show()
```



# Jeszcze przykład 2

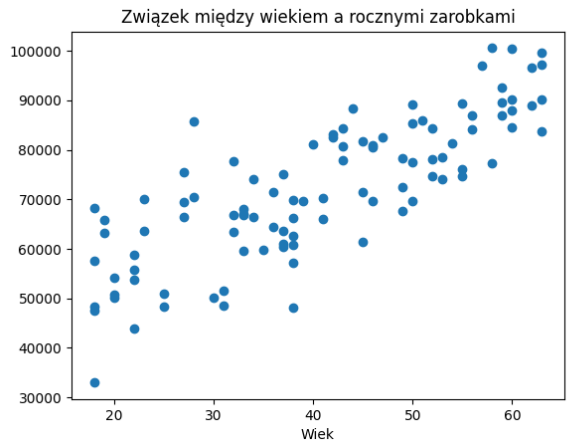
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)

fig = plt.figure()
gs = fig.add_gridspec(3, hspace=0)
axs = gs.subplots(sharex=True, sha
fig.suptitle('Sharing both axes')
axs[0].plot(x, y ** 2)
axs[1].plot(x, 0.3 * y, 'o')
axs[2].plot(x, y, '+')

# Hide x labels and tick labels for all but bottom plot.
for ax in axs:
    ax.label_outer()


plt.show()
```



# Inne wykresy: wykres punktowy

Zbiór punktów, które nie są ze sobą połączone. Używany w wizualizacji korelacji pomiędzy zmiennymi, rozkładu wartości albo określania grup. Działa tez na `pd.Series`

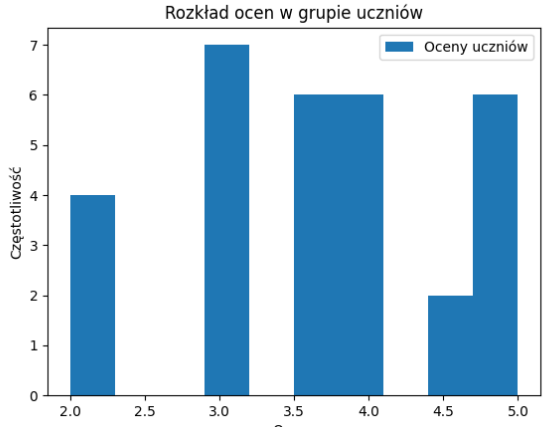
```
import numpy as np
import matplotlib.pyplot as plt
# Przykładowe dane
wiek = np.random.randint(18, 65, 100)
zarobki = np.random.normal(50000, 10000, 100) + (wiek - 18) * 1000
plt.scatter(wiek, zarobki)
plt.xlabel('Wiek')
plt.title('Związek między wiekiem a rocznymi zarobkami')
plt.show()
```



# Inne wykresy: histogram

Histogram – jest rodzajem wykresu słupkowego, który ilustruje rozkład wartości danych. Wykorzystywany do wizualizacji częstotliwości występowania wartości w wybranym zbiorze danych.

```
import numpy as np
import matplotlib.pyplot as plt
oceny = np.array([2,2,2,2,3,3,3,3,3,3,3,3.5,3.5,3.5,3.5,3.5,3.5,4,4,
                                4,4,4,4,4.5,4.5,5,5,5,5,5,5])
plt.hist(oceny, label='Oceny uczniów',align='mid')
plt.xlabel('Ocena')
plt.ylabel('Częstotliwość')
plt.title('Rozkład ocen w grupie uczniów')
plt.legend()
plt.show()
```

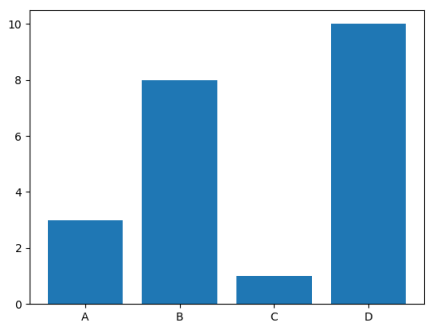


# Inne wykresy: wykres słupkowy

Wykres słupkowy – jeden z najpopularniejszych wykresów, który przedstawia dane za pomocą poziomych lub pionowych słupków. Zazwyczaj ułatwia porównanie danych, które pogrupowano w konkretne kategorie.

```
import numpy as np
#import pandas as pd
import matplotlib.pyplot as plt
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
#x = pd.Series(["A", "B", "C", "D"])
#y = pd.Series([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```

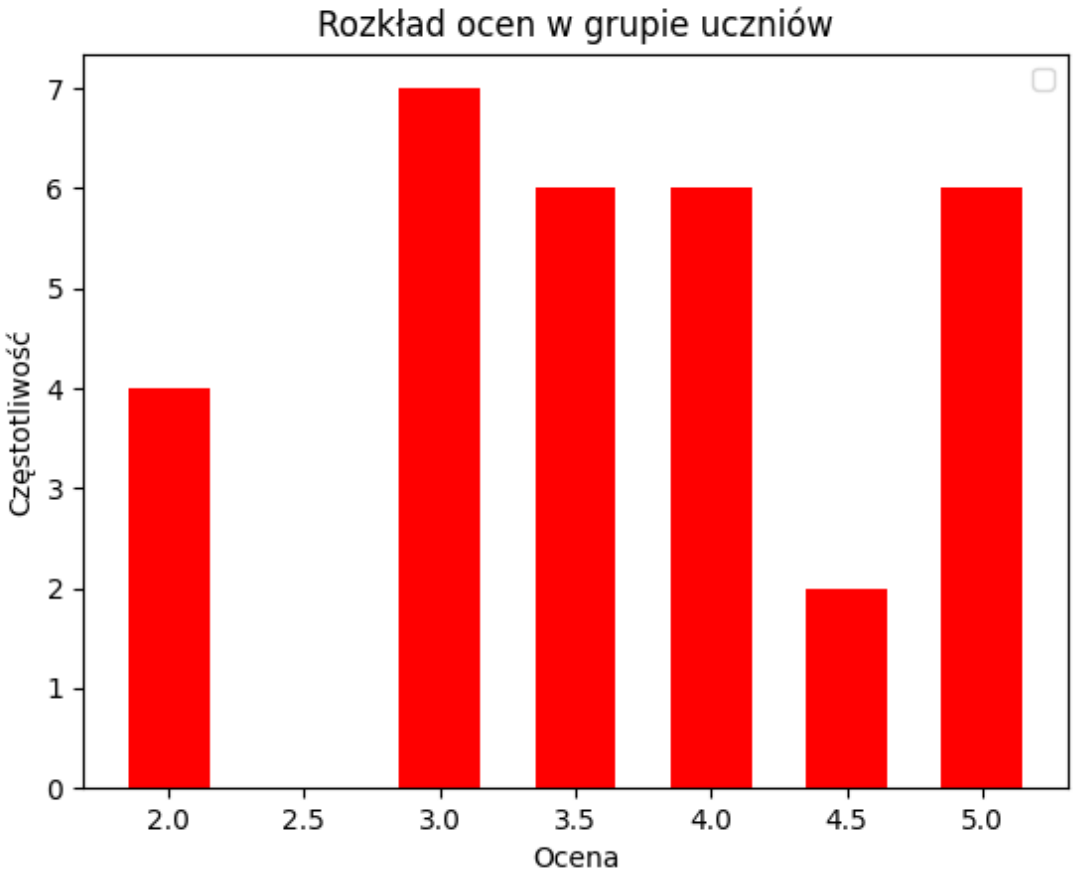
Kolory: `https://www.w3schools.com/python/matplotlib_bars.asp`



# Inne wykresy: histogram przy pomocy bar

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

oceny = np.array([2,2,2,2,3,3,3,3,3,3,3,3.5,3.5,3.5,3.5,3.5,3.5,
                              4,4,4,4,4,4,4.5,4.5,5,5,5,5,5,5])
print(oceny)
x = []
y = []
for i in np.unique(oceny):
    x.append(i)
    y.append(len(oceny[oceny==i]))

plt.bar(x,y, align='center',width=0.3,color='r')
plt.xlabel('Ocena')
plt.ylabel('Częstotliwość')
plt.title('Rozkład ocen w grupie uczniów')
plt.legend()
plt.show()
```

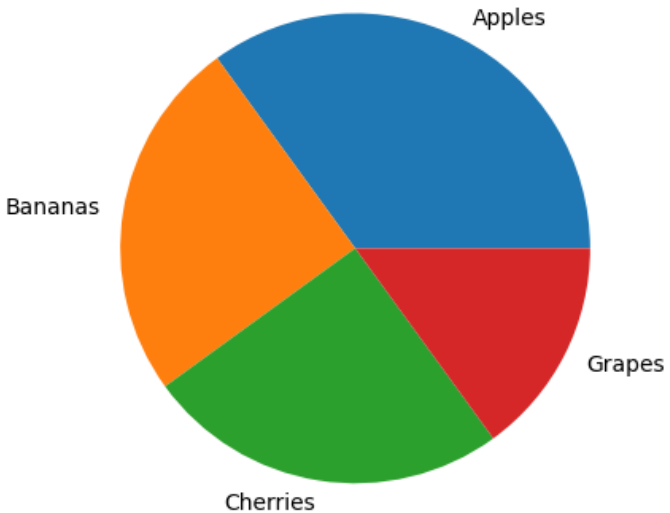
# Inne wykresy: histogram przy pomocy bar



# Inne wykresy: wykres kołowy

Wykres kołowy – dane znajdują się w sektorach w obrębie koła. Doskonale obrazuje proporcje poszczególnych elementów lub grup względem całości.

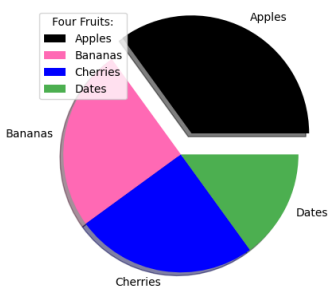
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Grapes"]
plt.pie(y, labels = mylabels)
plt.show()
```



# Inne wykresy: wykres kołowy, opcje zaawansowane

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

y = pd.Series([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True,
                colors = mycolors)
plt.legend(title = "Four Fruits:", loc = 2)
plt.show()
```

https://www.geeksforgeeks.org/change-the-legend-position-in-matplotlib/



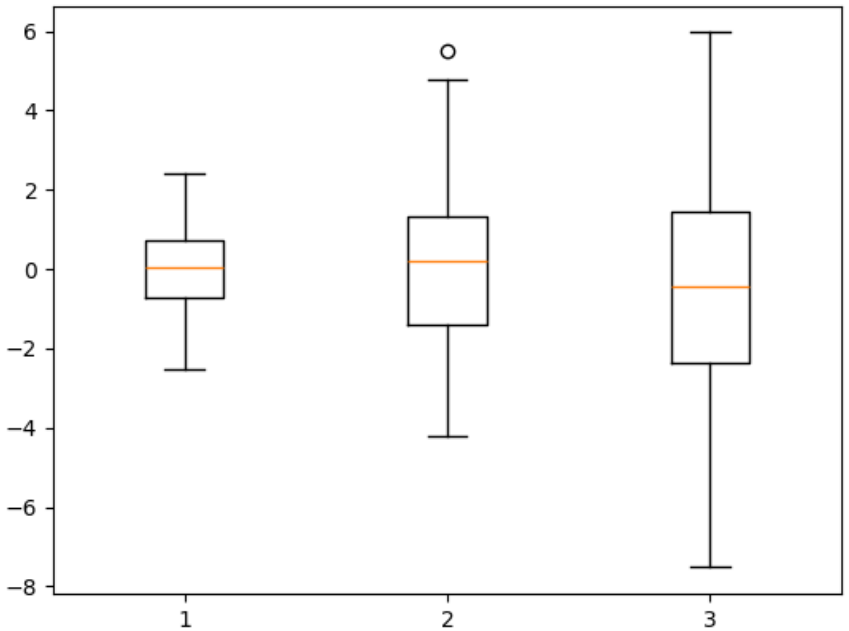
## Inne wykresy: wykres pudełkowy

Wykres pudełkowy (przedstawiony pionowo) tworzy się odkładając na pionowej osi wartości niektórych parametrów rozkładu. Nad osią umieszczony jest prostokąt (pudełko), którego dol jest wyznaczony przez pierwszy kwartyl, zaś górny bok przez trzeci kwartyl. Wysokość pudełka odpowiada wartości rozstępu ćwiartkowego. Wewnątrz prostokąta znajduje się pozioma linia, określająca wartość mediany.
Rysunek pudełka uzupełniamy odcinkami zwanymi wąsami. W najprostszym wariancie dolny koniec dolnego odcinka wyznacza najmniejszą wartość w zbiorze, natomiast górny koniec górnego odcinka to wartość największa.

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

# Generate some random data
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
print(data)
# Create a box and whisker plot
plt.boxplot(data)

plt.show()
```

## Inne wykresy: wykres pudełkowy

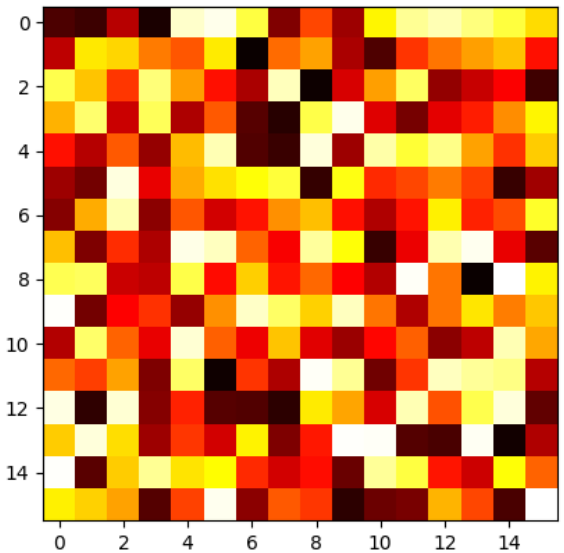
Wykres pudełkowy (przedstawiony pionowo) tworzy się odkładając na pionowej osi wartości niektórych parametrów rozkładu. Nad osią umieszczony jest prostokąt (pudełko), którego dol jest wyznaczony przez pierwszy kwartyl, zaś górny bok przez trzeci kwartyl. Wysokość pudełka odpowiada wartości rozstępu ćwiartkowego. Wewnątrz prostokąta znajduje się pozioma linia, określająca wartość mediany.
Rysunek pudełka uzupełniamy odcinkami zwanymi wąsami. W najprostszym wariancie dolny koniec dolnego odcinka wyznacza najmniejszą wartość w zbiorze, natomiast górny koniec górnego odcinka to wartość największa.



## Inne wykresy: mapa ciepła

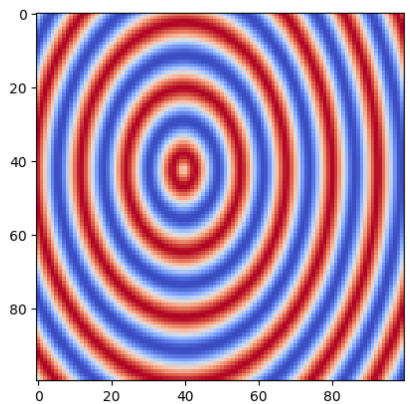
Mapa ciepła (heatmap) – nie jest typowym wykresem znanym np. ze szkoły, ponieważ występuje w postaci macierzy, na której kolory odpowiadają wartościom konkretnych komórek. Dobrze obrazuje korelacje i gradienty wartości oraz wykrywa wzorce w przypadku danych dwuwymiarowych.

```
import matplotlib.pyplot as plt
import numpy as np
a = np.random.random((16, 16))
plt.imshow(a, cmap='hot', interpolation='nearest')
plt.show()
```



# Inne wykresy: mapa ciepła

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

x = np.linspace(-10, 40, 100)
# długość geograficzna dla Europy (w przybliżeniu)
y = np.linspace(35, 70, 100)
# szerokość geograficzna dla Europy (w przybliżeniu)
X, Y = np.meshgrid(x, y)
cisnienie = 1000 + 10 * np.sin(np.sqrt((X - 10)**2 + (Y - 50)**2))

plt.imshow(cisnienie, cmap='coolwarm', interpolation='nearest')
plt.show()
```

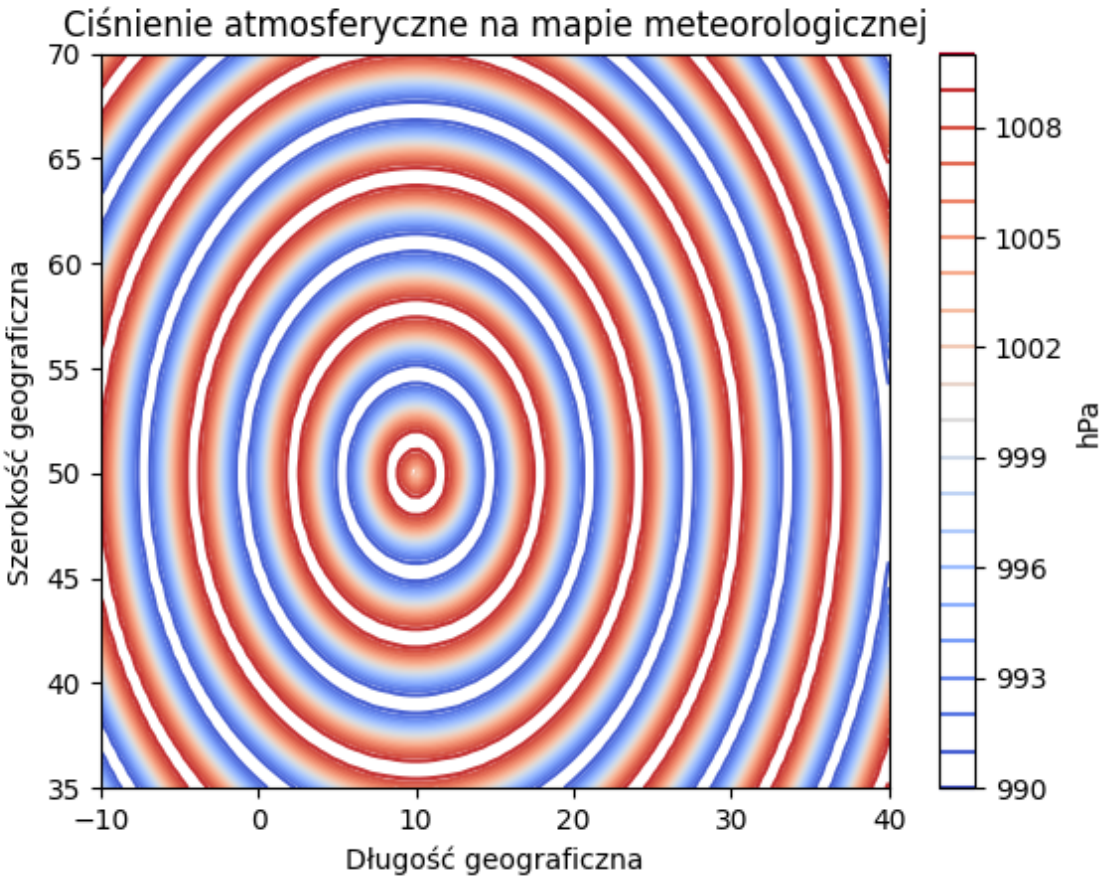


## Inne wykresy: wykres konturowy

Wykres konturowy – używany jest do obrazowania danych trójwymiarowych w postaci linii poziomicy na powierzchni dwuwymiarowej. Wizualizuje wartości, które są równe dla konkretnej funkcji na danej płaszczyźnie.
Przykładem może być np. wizualizacja danych pogodowych lub wartości ciśnienia atmosferycznego na mapie meteorologicznej.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-10, 40, 100)
# długość geograficzna dla Europy (w przybliżeniu)
y = np.linspace(35, 70, 100)
# szerokość geograficzna dla Europy (w przybliżeniu)
X, Y = np.meshgrid(x, y)
cisnienie = 1000 + 10 * np.sin(np.sqrt((X - 10)**2 + (Y - 50)**2))
plt.contour(X, Y, cisnienie, levels=20, cmap='coolwarm')
plt.xlabel('Długość geograficzna')
plt.ylabel('Szerokość geograficzna')
plt.title('Ciśnienie atmosferyczne na mapie meteorologicznej')
plt.colorbar(label='hPa')
plt.show()
```

# Inne wykresy: wykres konturowy

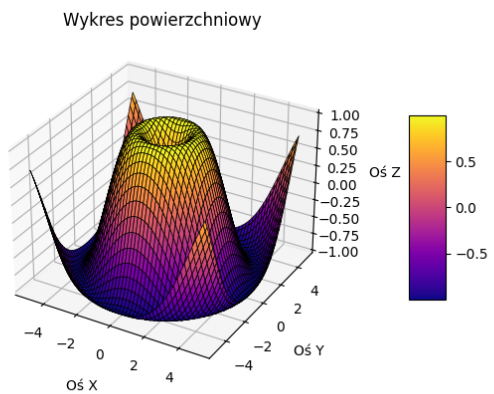


## Inne wykresy: 3D

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

# Definiowanie danych
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x**2 + y**2))

# Inicjalizacja wykresu 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Tworzenie wykresu powierzchniowego
surf = ax.plot_surface(x, y, z, cmap='plasma',
                       edgecolors='k', linewidth=0.5)

# Dodanie paska kolorów
cbar = fig.colorbar(surf, pad=0.15, shrink=0.5, aspect=5)
cbar.ax.yaxis.set_ticks_position('right')
```

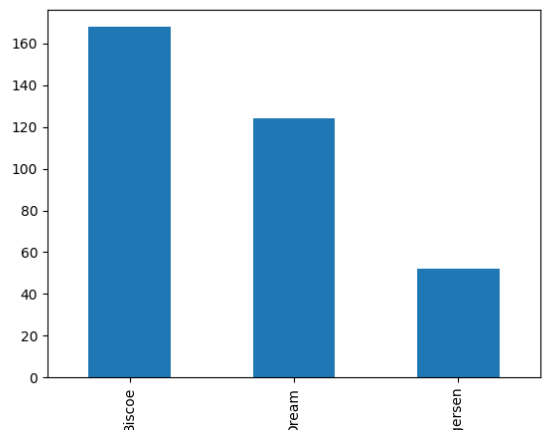
# Inne wykresy: 3D

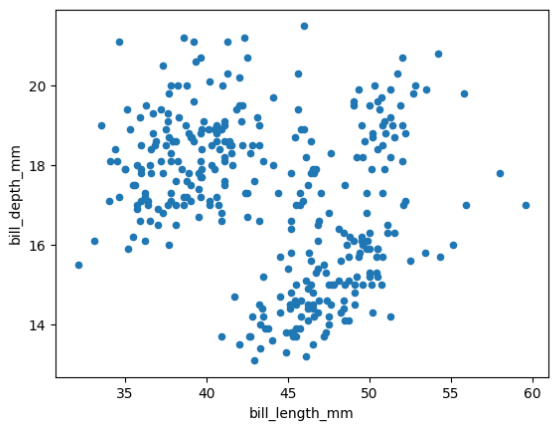
```
# Etykiety osi
ax.set_xlabel('Oś X', labelpad=10)
# labelpad zapobiega nakładaniu się osi
ax.set_ylabel('Oś Y', labelpad=10)
ax.set_zlabel('Oś Z', labelpad=10)

# Tytuł wykresu
ax.set_title('Wykres powierzchniowy')
plt.show()
```



# Wykresy z pingwinami 1

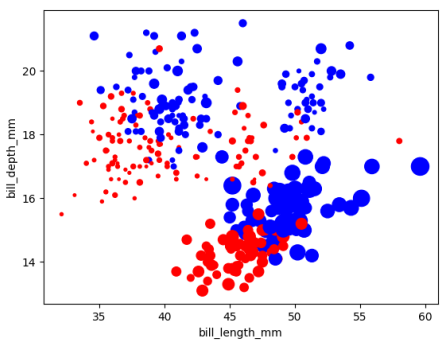
```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('penguins.csv', sep=',', index_col=False,
 encoding='UTF-8')

#wykres słupkowy ilości pingwinów w zależności od wyspy.
data.groupby(['island']).size().plot.bar()
plt.show()
#wykres punktowy zależności  szerokości dzióba od długości.
data.plot.scatter(x = 'bill_length_mm',y ='bill_depth_mm')
plt.show()
```



# Wykresy z pingwinami 2

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data = pd.read_csv('penguins.csv', sep=',', index_col=False,
                                encoding='UTF-8')
data = data.dropna()
colors = np.where(data['sex']=='male','blue','red')
weight = ((data['body_mass_g']/2000)**5).astype(float)
#wykres punktowy zależności  szerokości dzióba od długości.
#w ktorym kolor punktów zależy od plecie, a rozmiar - od wagi.
data.plot.scatter(x = 'bill_length_mm',y ='bill_depth_mm',
                            c = colors, s = weight)
plt.show()
```



# Rozdział 10. Python: biblioteka Seaborn

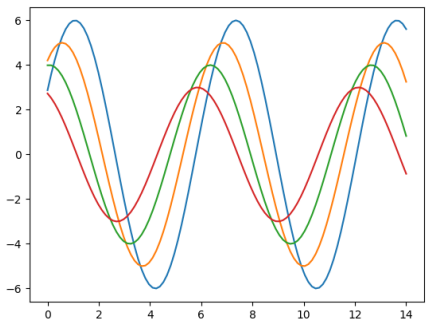
# Biblioteka Seaborn

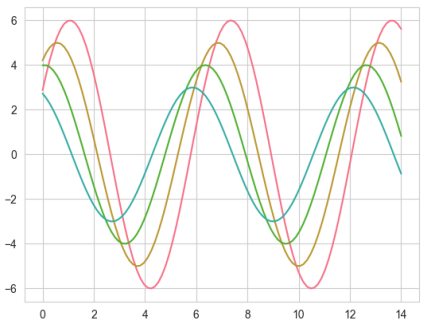
Seaborn, to zgrabna oraz efektywna biblioteka, pozwalająca na szybkie tworzenie atrakcyjnych wykresów, w Python. Została, zbudowana na bazie biblioteki Matplotlib, jednocześnie wzbogacona o dodatkowe typy wykresów.

`https://seaborn.pydata.org/`

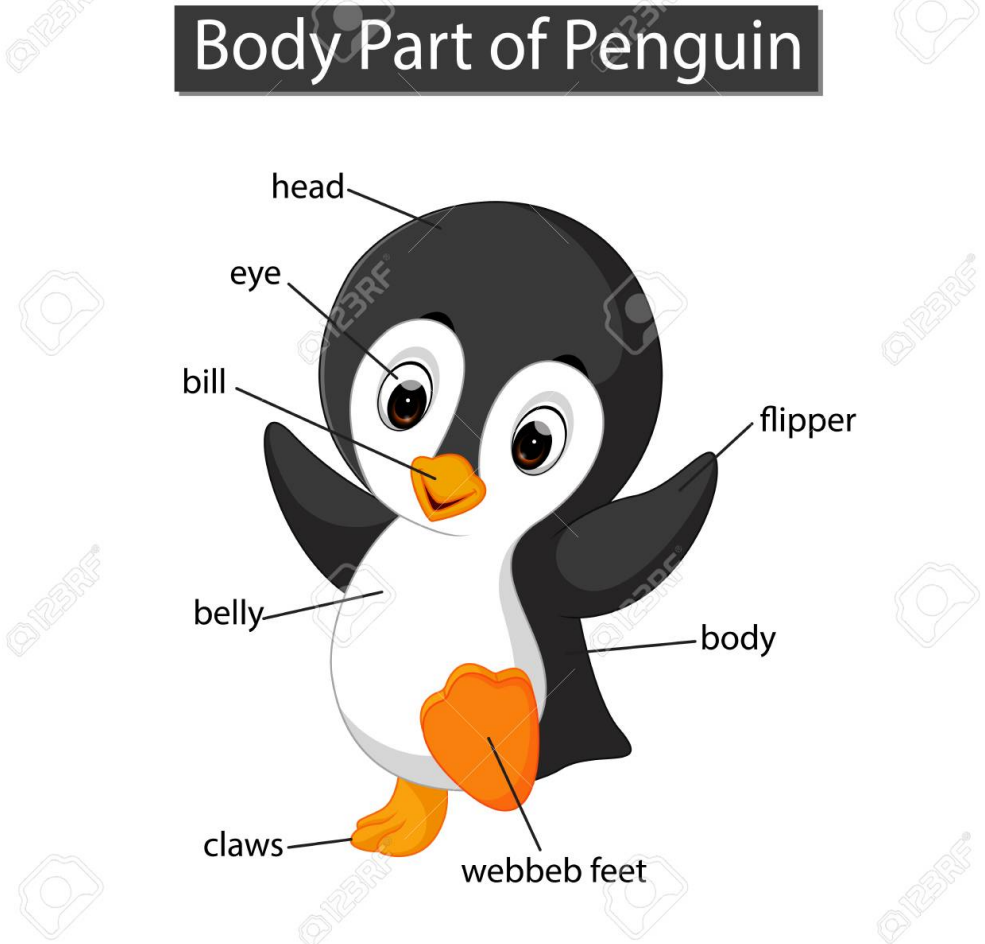
## Porównaj:

```
import numpy as np
#import seaborn as sns
import matplotlib.pyplot as plt

def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 5):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)

#sns.set_style("whitegrid")
#sns.set_palette("husl")
sinplot()
#print(sns.axes_style())
plt.show()
```



# Pingwiny

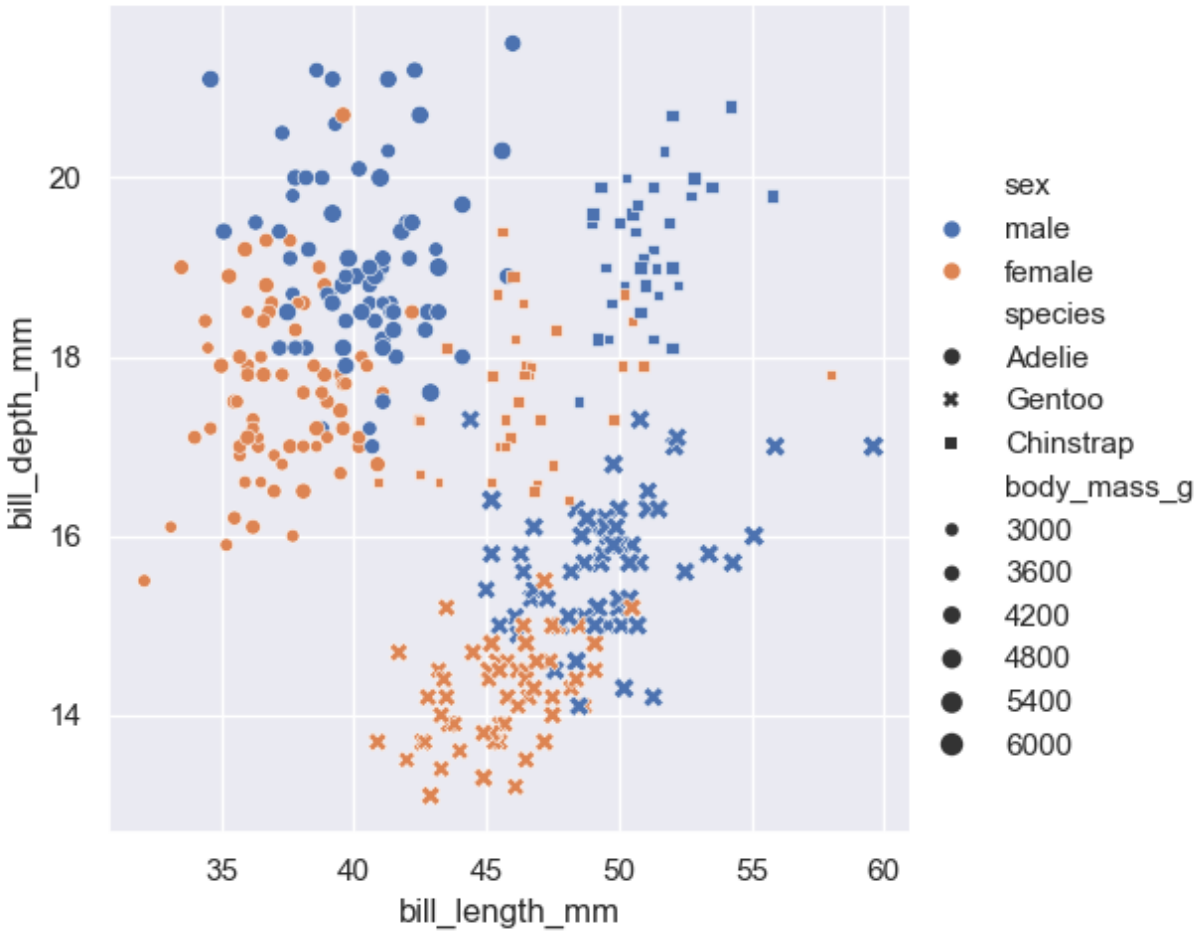



`https://github.com/mwaskom/seaborn-data`

## Seaborn: pingwiny

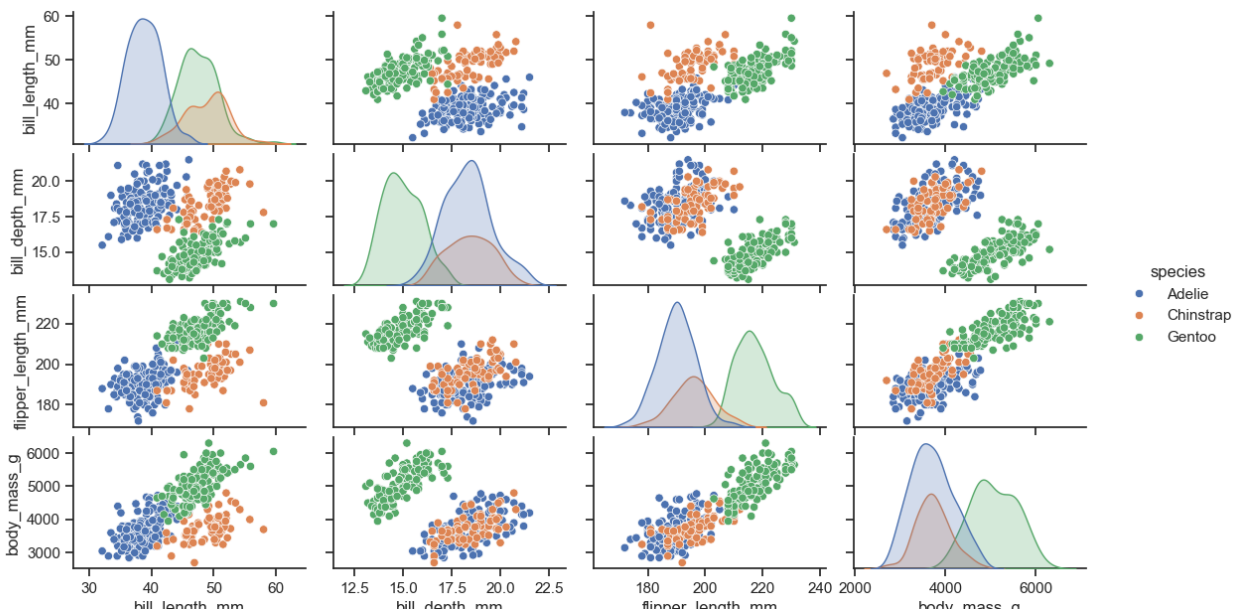
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Apply the default theme
sns.set_theme()

#penguins = sns.load_dataset("penguins")
penguins = pd.read_csv('penguins.csv', index_col=None,
encoding="UTF-8")

sns.relplot(
    data=penguins,
    x="bill_length_mm", y="bill_depth_mm",
    hue="sex", style="species", size="body_mass_g",)

plt.show()
```

# Seaborn: pingwiny



# Seaborn: jeszcze pingwiny

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks")

df = sns.load_dataset("penguins")
sns.pairplot(df, hue="species")
plt.show()
```



## 4 główne typy wykresów w Seaborn

Seaborn, posiada całkiem rozbudowany arsenał wykresów. Szczegóły dotyczące wszystkich z nich, znajdziemy na stronie produktu. My skupimy się na nauce pakietu oraz na 4 najczęściej stosowanych typach wykresów. Mianowicie

1. Wykresy relacyjne
2. Wykresy z kategoriami
3. Wykresy z regresją
4. Wykresy dystrybucji oraz korelacji (histogram)

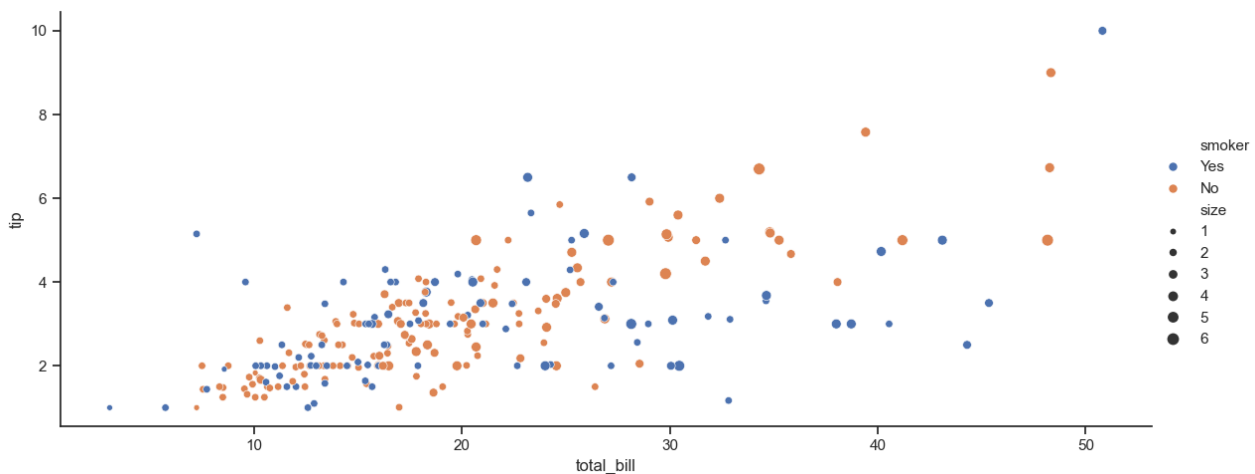
# Wykresy relacyjne: funkcja relplot()

Podstawowe parametry:

data – wskazujemy zbiór danych
x, y – wskazujemy dane, które mają być umieszczone na osi x oraz y
hue – jeżeli chcemy, aby dane były kolorystycznie różne, w zależności od wartości zmiennej, to tutaj ją podajemy
col, row – w ilu kolumnach i wierszach mają się wyświetlić wykresy
kind – typ wykresu – czy liniowy, czy z punktami
aspect – szerokość wykresu

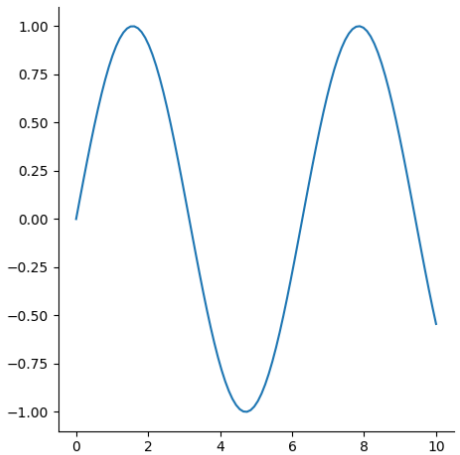
# Wykresy relacyjne: przykład 1

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks")

tips = sns.load_dataset("tips")
sns.relplot(x="total_bill", y="tip", aspect=2.5,
            data=tips, size='size', hue='smoker',
            kind="scatter");
plt.show()
```



# Wykresy relacyjne: przykład 2

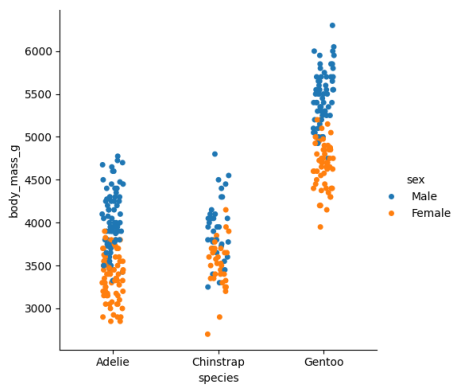
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import seaborn as sns


x0=np.linspace(0,10,100)
sns.relplot(x=x0,
            y=np.sin(x0),
            kind="line");
plt.show()
```

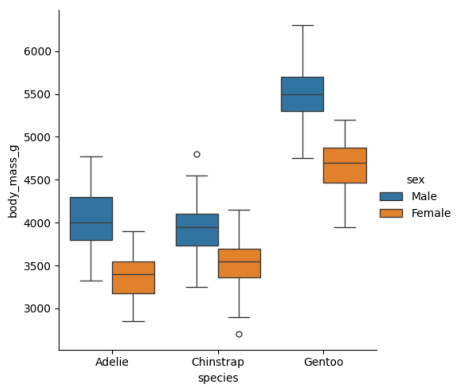


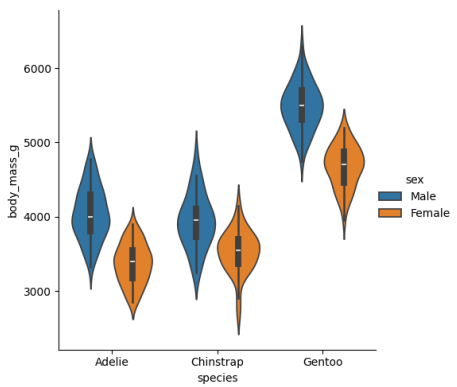
# Wykresy z kategoriami

Kolejne, popularne wykresy, to wykresy słupkowe różnego typu. Podstawową funkcją, którą powinniśmy poznać, jest 'catplot()'. Nazwa, od category plot. Podobnie, jak w przypadku funkcji 'relplot()', mamy kilka rodzajów wykresów słupkowych. Count, bar, box itd.

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

df = sns.load_dataset("penguins")
sns.catplot(x="species", y="body_mass_g",
            data=df, hue = 'sex',
#kind = 'box' #kind = 'violin'
)
plt.show()
```

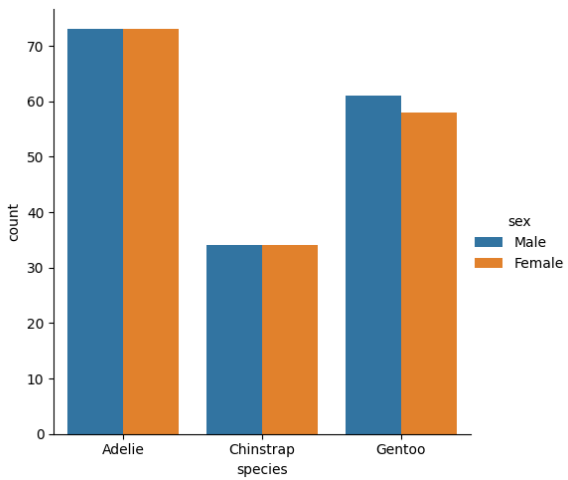


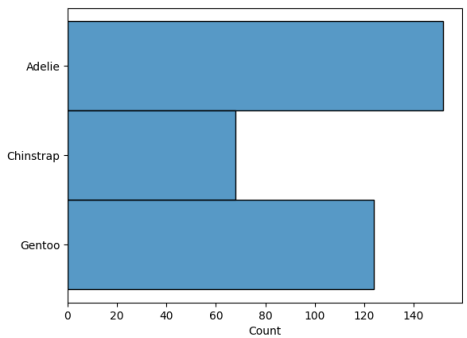
# Wykresy z kategoriami

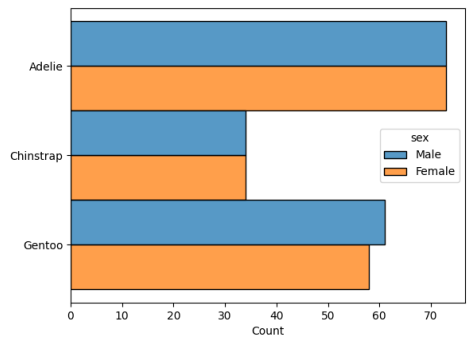
```
import matplotlib.pyplot as plt
import seaborn as sns

df = sns.load_dataset("penguins")
sns.catplot(x="species",
            data=df,
            hue = 'sex',
            kind = 'count'
            )
plt.show()
```

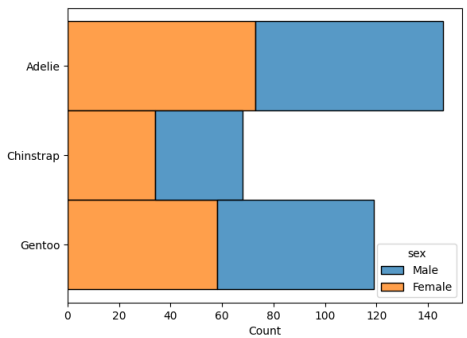


## Histogram

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

df = sns.load_dataset("penguins")
sns.histplot(data=df, y="species")
#sns.histplot(data=df, y="species", hue='sex', multiple='dodge')
#sns.histplot(data=df, y="species", hue='sex', multiple='stack')
plt.show()
```



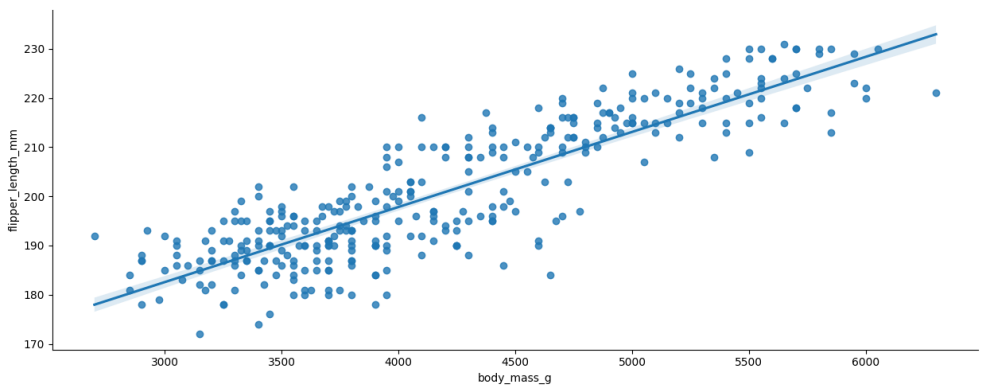



# Wykresy z regresja

Regresja liniowa:

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

df = sns.load_dataset("penguins")
sns.lmplot(data=df,
           x="body_mass_g",
           y="flipper_length_mm",
           aspect=2.5,)
plt.show()
```



# Rozdział świąteczny. choinki

# Choinka 1

# Kod

```
import matplotlib.pyplot as plt

xpoints = [0,18,2,14,4,12,6,10,9]
ypoints = [0,0,4,7,9,11,13,14,16]

plt.plot(xpoints, ypoints, 'g', linewidth="20")

ax = plt.gca()
ax.set_aspect('equal', adjustable='box')
plt.ylim(-1,18)
plt.axis('off')
plt.plot(9,16,'r',marker='*',markersize=60)
plt.show()
```

# Choinka 2

# Kod

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

ypoints = np.linspace(-0.5,-50,200)
xpoints = np.sin(ypoints)*(ypoints)*0.5

plt.plot(xpoints, ypoints, 'g',linewidth = '3' )
ax = plt.gca()
ax.set_aspect('equal', adjustable='box')
plt.axis('off')
plt.plot(0,0,'r',marker='.',markersize=30)
plt.ylim(-52,2)
plt.show()
```

# Choinka 3D

## Kod

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

ax = plt.figure().add_subplot(projection='3d')

# Prepare arrays x, y, z
theta = np.linspace(-8 * np.pi, 8 * np.pi, 200)
z = np.linspace(-3, 0, 200)
x =  z * np.sin(theta)/3
y =  z * np.cos(theta)/3

ax.plot(x, y, z, color='g')
ax.set_aspect('equal', adjustable='box')
plt.axis('off')
plt.plot(0,0,'r',marker='*',markersize=20)

plt.show()
```