Technologie informacyjne i komunikacyjne

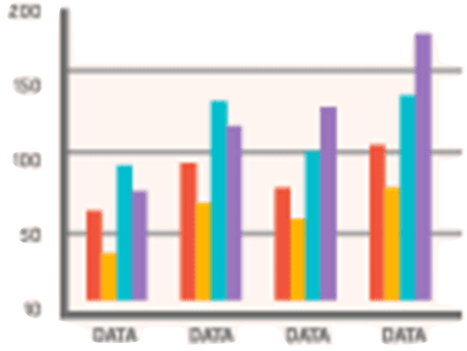
Wykład 10, dn. 18.05.2026

Źródło obrazka: pngtree

Wizualizacja danych w Matplotlib

Po co wizualizować dane?

- wykrywanie trendów,
- znajdowanie anomalii,
- komunikowanie wyników,
- wspieranie decyzji.



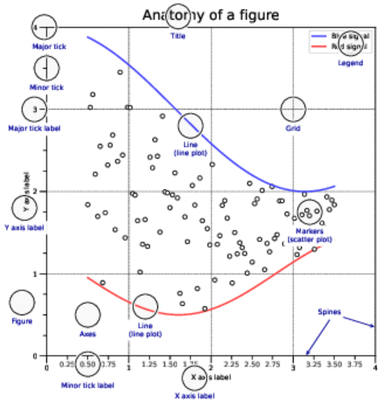
matplotlib

Cheat sheet — Version 3.9.4

Quick start

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


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure



Subplots layout

`subplot[s](rows, cols, …)` API
`fig, axs = plt.subplots(3, 3)`

`G = gridspec(rows,cols, …)` API
`ax = G[0, :]`

`ax.inset_axes(extent)` API

`d=make_axes_locatable(ax)` API
`ax = d.new_horizontal('10%')`

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X], Y, [fmt], …)` API
X, **Y**, fmt, color, marker, linestyle

`scatter(X, Y, …)` API
X, Y, [s]izes, [c]olors, marker, cmap

`bar[h](x, height, …)` API
x, **height**, width, bottom, align, color

`imshow(Z, …)` API
Z, cmap, interpolation, extent, origin

`contour[f]([X], [Y], Z, …)` API
X, Y, **Z**, levels, colors, extent, origin

`pcolormesh([X], [Y], Z, …)` API
X, Y, **Z**, vmin, vmax, cmap

`quiver([X], [Y], U, V, …)` API
X, Y, **U**, **V**, C, units, angles

`pie(X, …)` API
Z, explode, labels, colors, radius

`text(x, y, text, …)` API
x, **y**, **text**, va, ha, size, weight, transform

`fill[_between][x](…)` API
X, Y1, Y2, color, where

Advanced plots

`step(X, Y, [fmt], …)` API
X, **Y**, fmt, color, marker, where

`boxplot(X, …)` API
X, notch, sym, bootstrap, widths

`errorbar(X,Y,xerr,yerr, …)` API
X, **Y**, xerr, yerr, fmt

`hist(X, bins, …)` API
X, bins, range, density, weights

`violinplot(D, …)` API
D, positions, widths, vert

`barbs([X], [Y], U, V, …)` API
X, Y, **U**, **V**, C, length, pivot, sizes

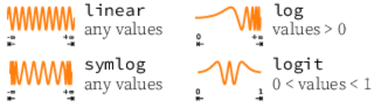
`eventplot(positions, …)` API
positions, orientation, lineoffsets

`hexbin(X, Y, C, …)` API
X, **Y**, C, gridsize, bins

Scales

API

`ax.set_[xy]scale(scale, …)`



Projections

API

`subplot(…, projection=p)`

p='polar'

p='3d' API

p=ccrs.Orthographic() API
import cartopy.crs as ccrs

Lines

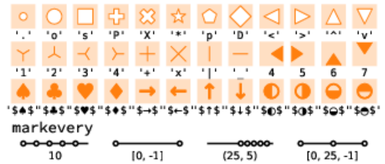
API

`linestyle or ls`

`capstyle or dash_capstyle`

Markers

API



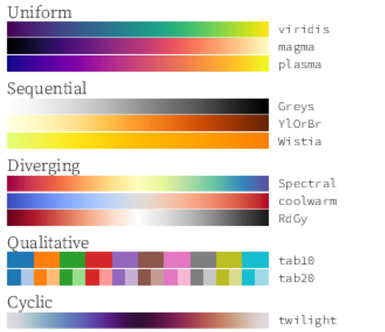
Colors

API

Colormaps

API

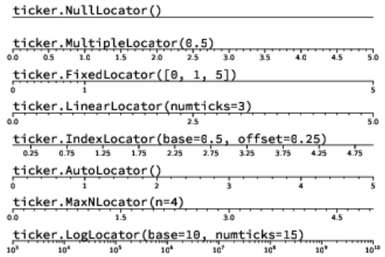
`plt.get_cmap(name)`



Tick locators

API

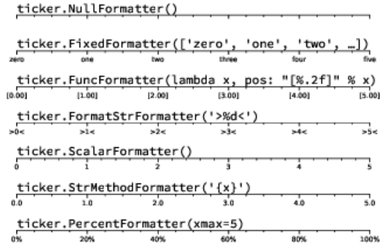
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



Tick formatters

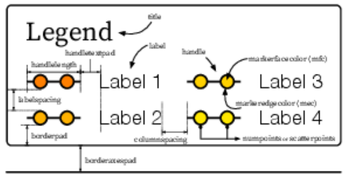
API

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

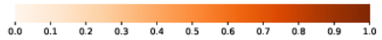


Ornaments

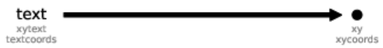
`ax.legend(…)` API
handles, labels, loc, title, frameon



`ax.colorbar(…)` API
mappable, ax, cax, orientation



`ax.annotate(…)` API
text, **xy**, **xytext**, xycoords, textcoords, arrowprops



Event handling

API

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

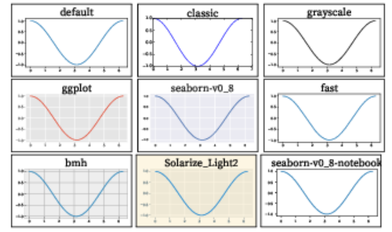
API

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

API

`plt.style.use(style)`



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

API

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

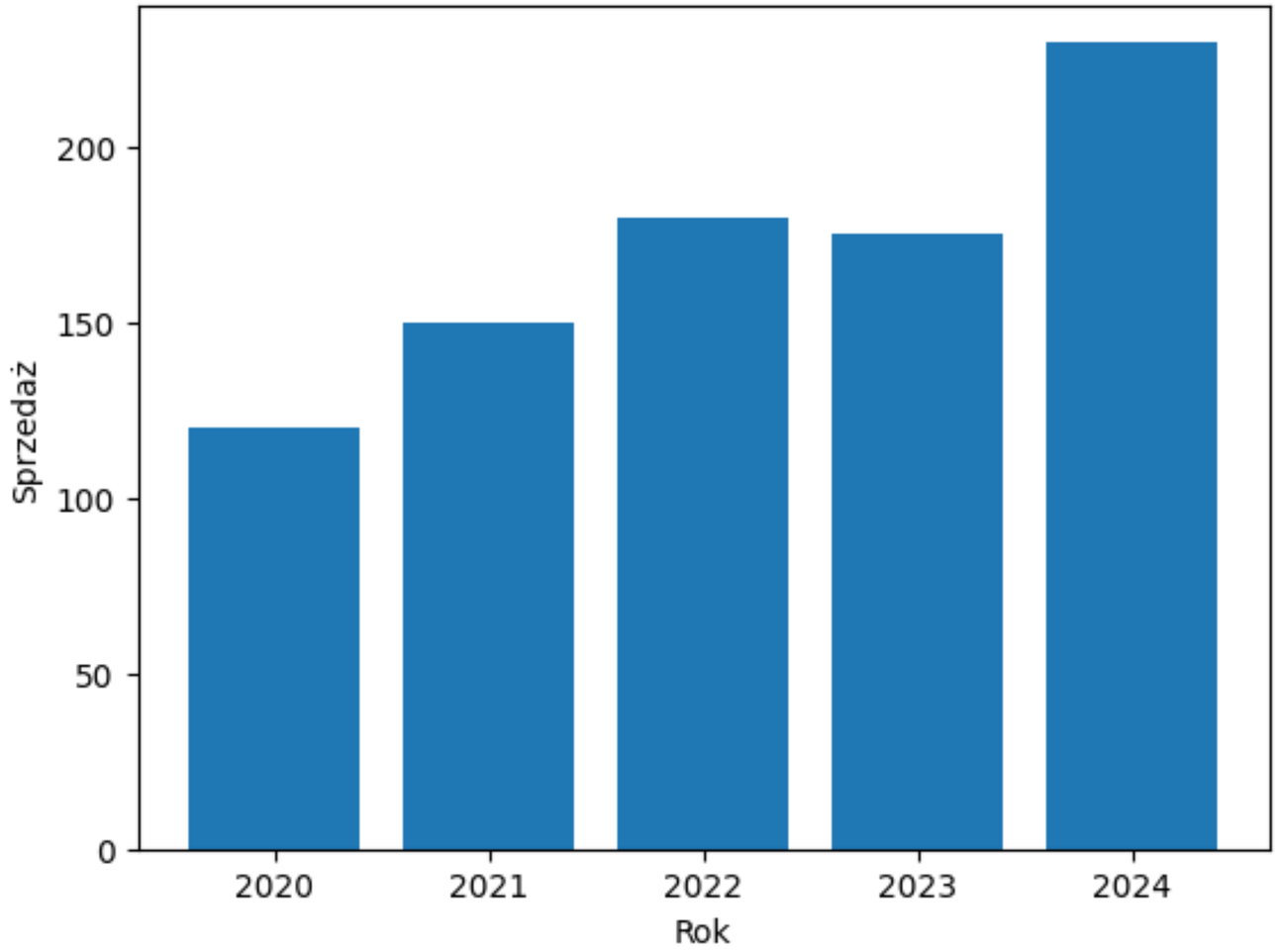
Ten simple rules

READ

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

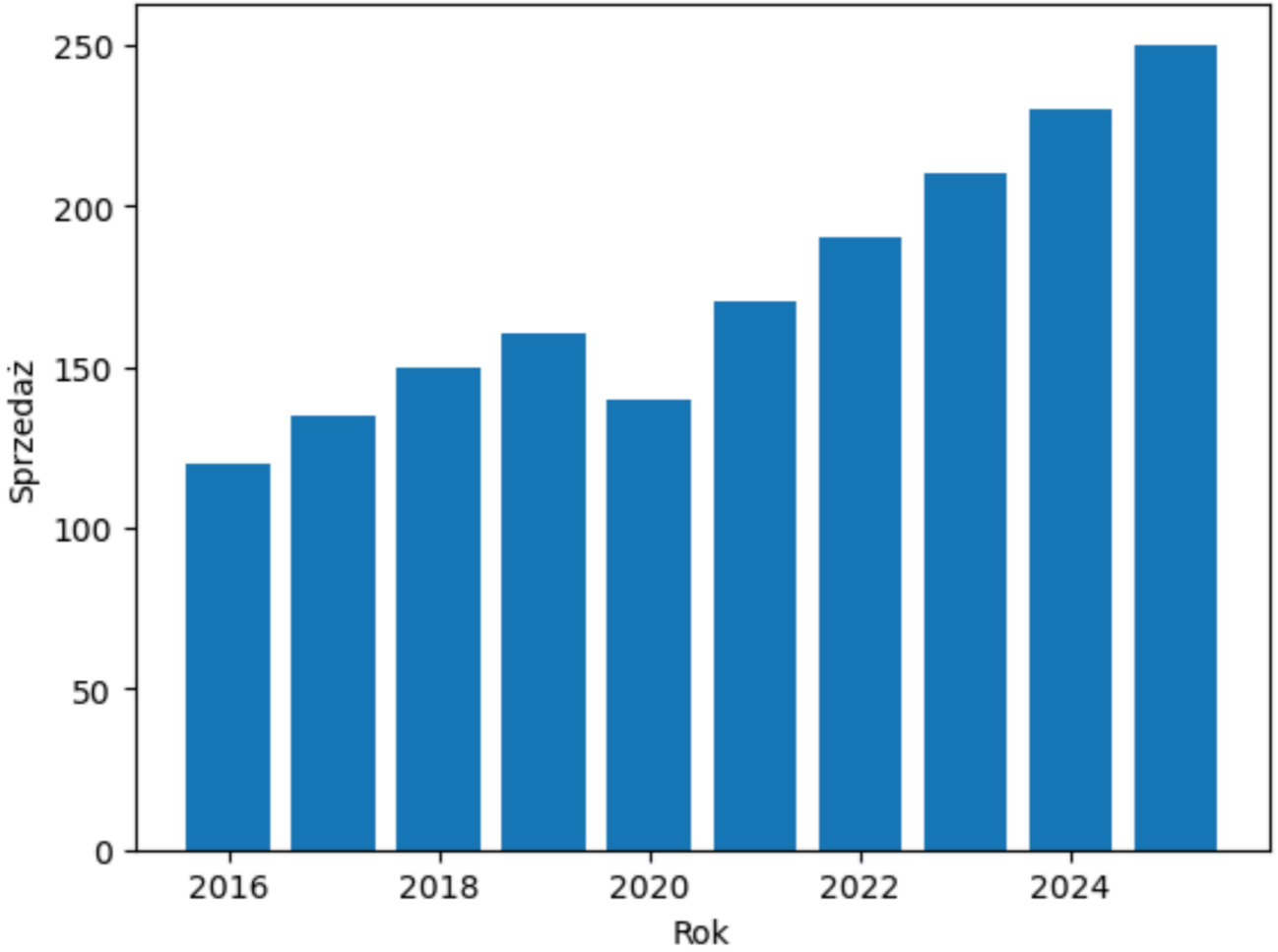
Wizualizacja danych

Rok	Sprzedaż
2020	120
2021	150
2022	180
2023	175
2024	230



Wizualizacja danych

Rok	Sprzedaż
2016	135
2017	150
2018	160
2019	140
2020	170
2021	190
2022	210
2023	230
2024	250



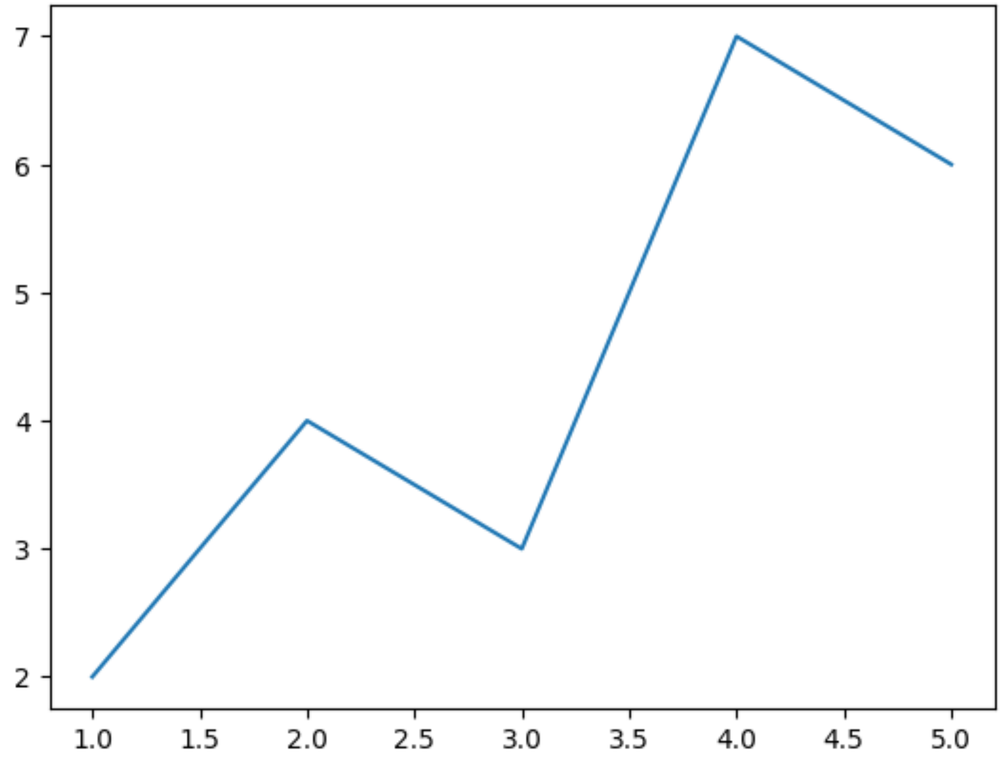
Wizualizacja danych w Matplotlib

Biblioteka matplotlib

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

Prosty wykres

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 3, 7, 6]


plt.plot(x, y)
plt.show()
```



Elementy wykresu

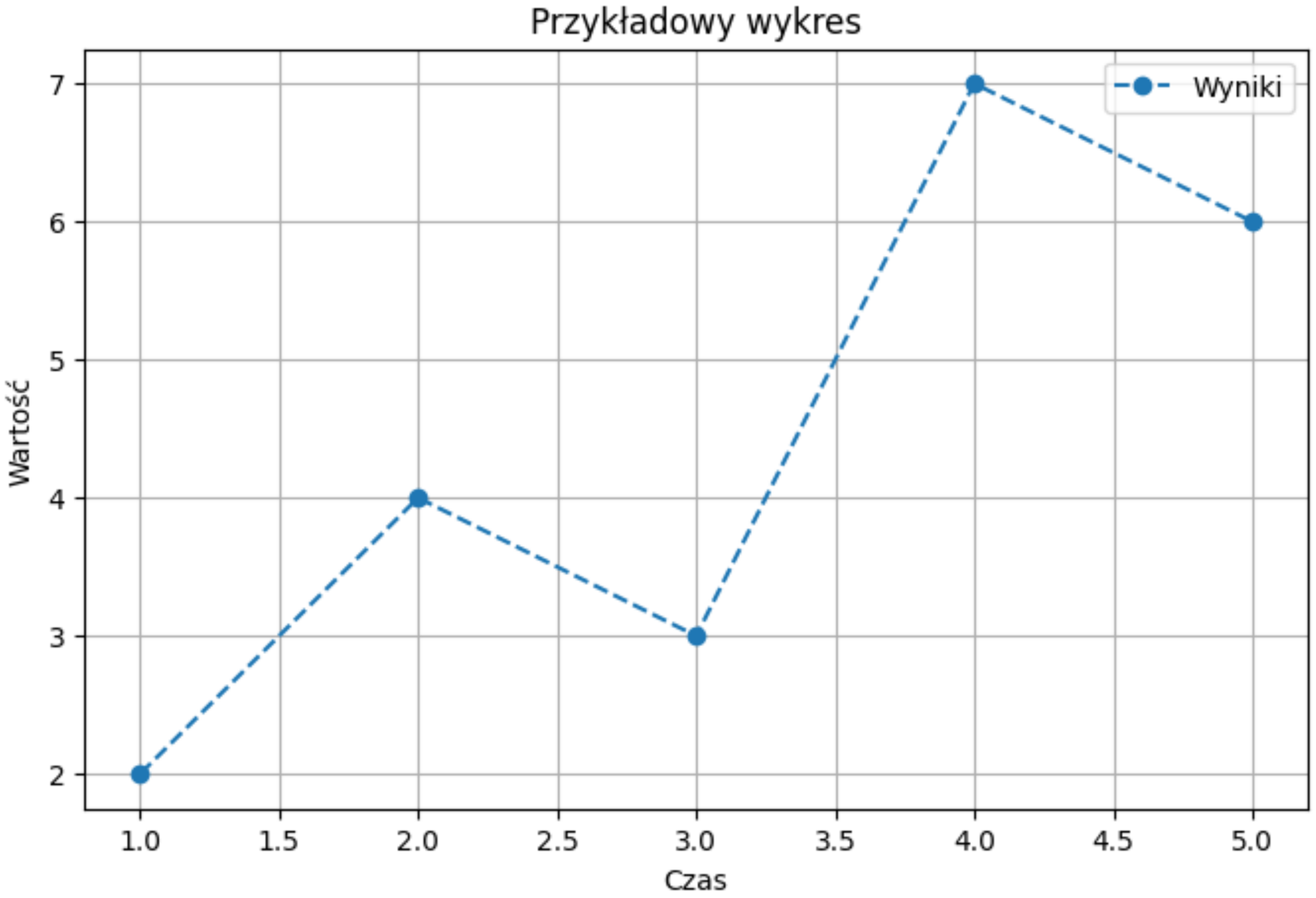
- figura (`figure`)
- osie (`axes`)
- tytuł
- etykiety osi
- legenda
- siatka
- znaczniki punktów

Elementy wykresu

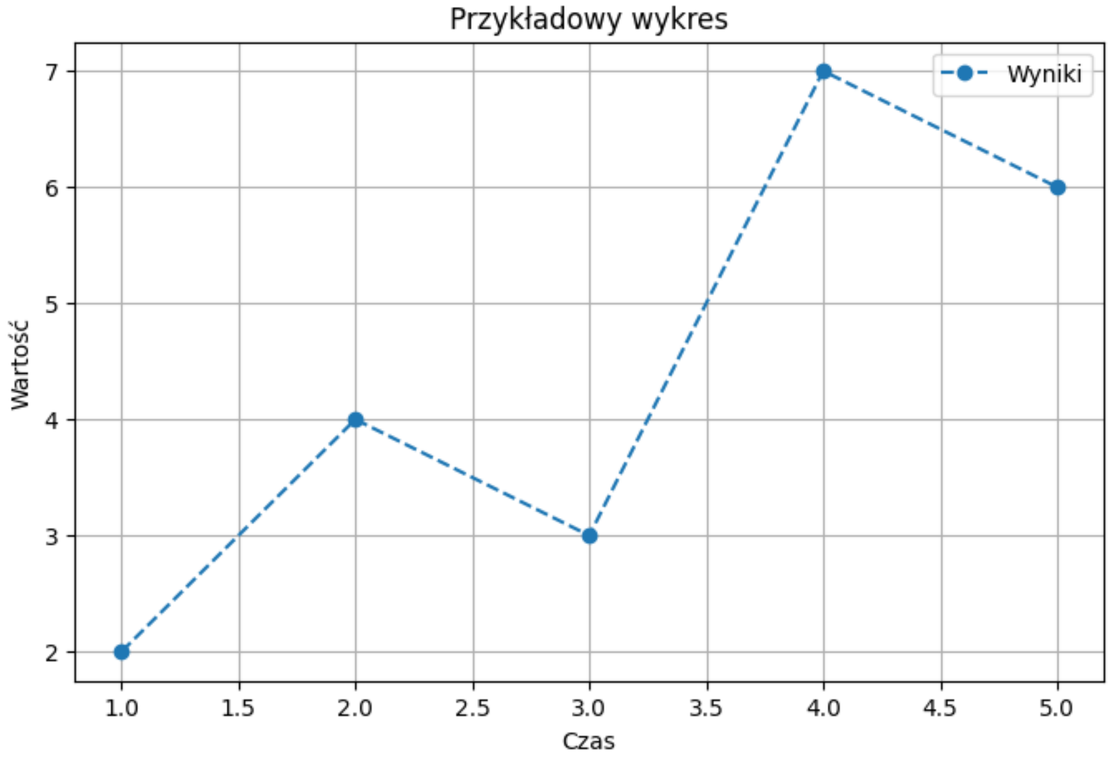
```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 3, 7, 6]

plt.figure(figsize=(8, 5))

plt.plot(
    x,
    y,
    marker='o',
    linestyle='--',
    label='Wyniki'
)
```

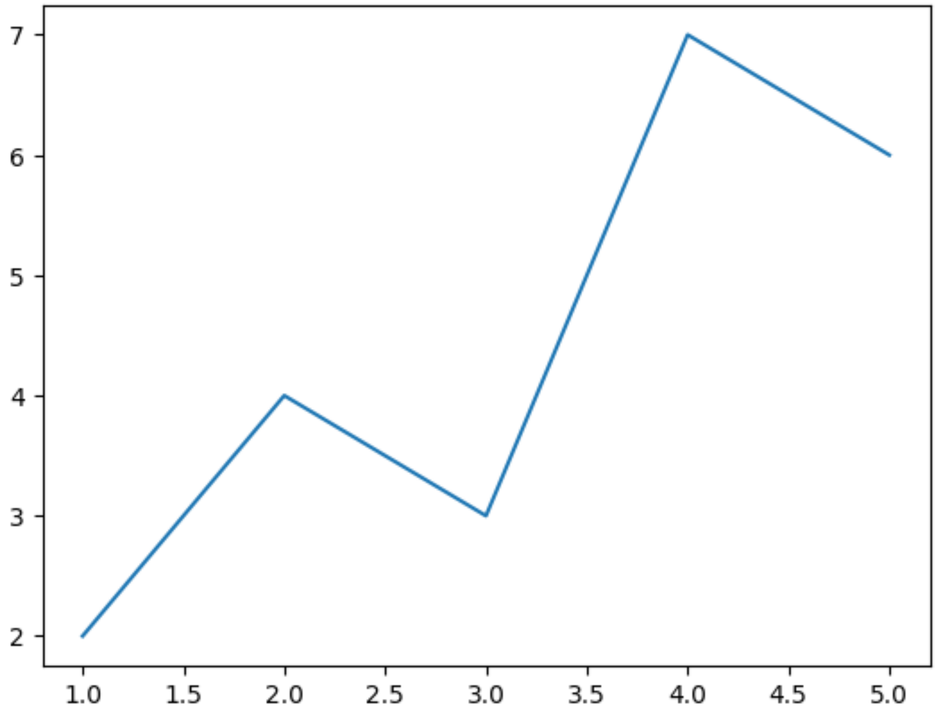
```
plt.title('Przykładowy
wykres')
plt.xlabel('Czas')
plt.ylabel('Wartość')
plt.grid(True)
plt.legend()

plt.show()
```

Elementy wykresu



Elementy wykresu





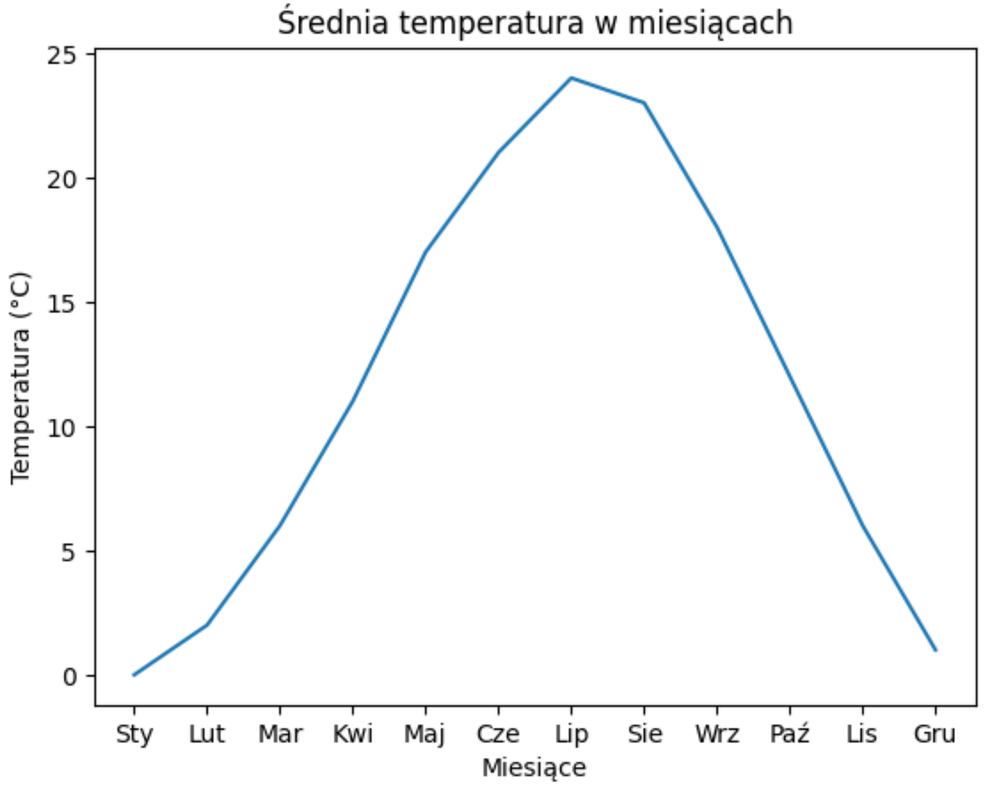
Typy wykresów: wykres liniowy

→ do trendów w czasie.

```
plt.plot(miesiace, temperatura)
```

Przykład:

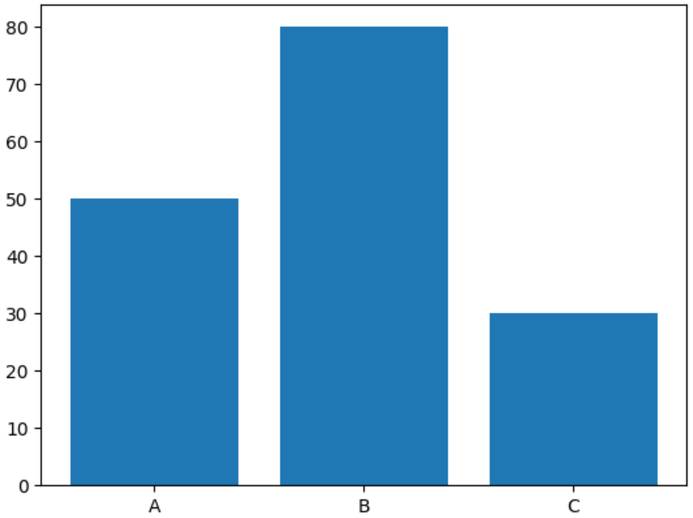
- temperatura,
- giełda,
- sprzedaż.



Typy wykresów: wykres słupkowy

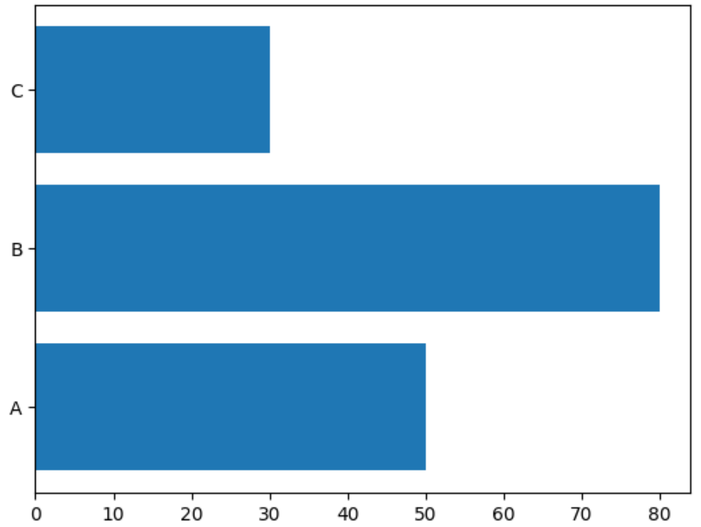
→ do porównania kategorii.

```
produkty = ['A', 'B', 'C']

sprzedaz = [50, 80, 30]

plt.bar(produkty, sprzedaz)

plt.show()
```



Dyskusja:

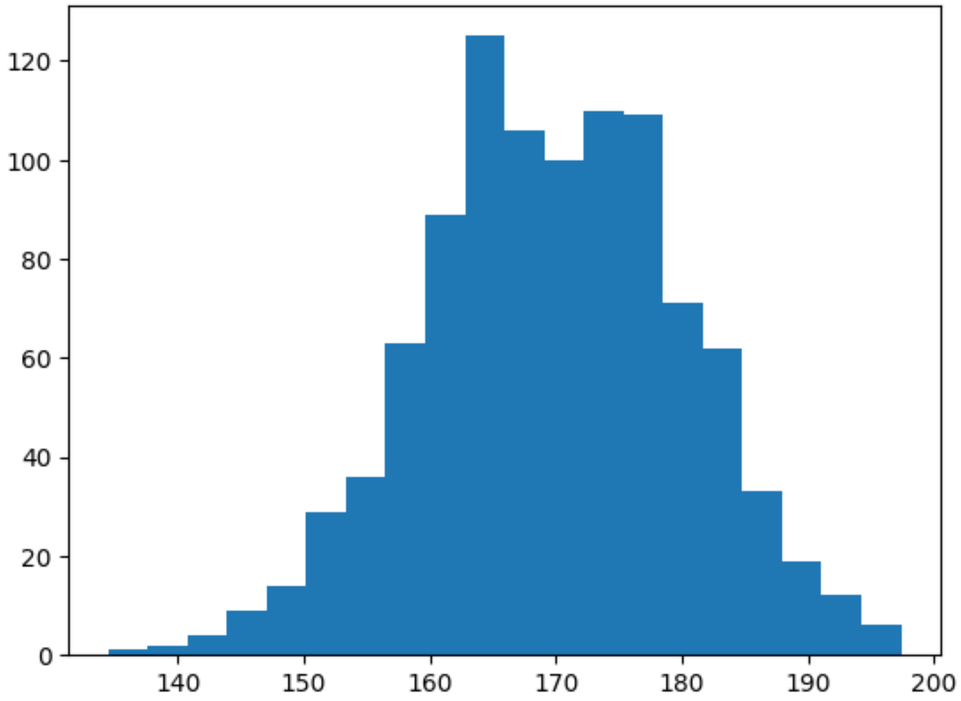
- pionowy vs poziomy (`barh()`)



Typy wykresów: histogram

→ do rozkładu danych.

```
import numpy as np
dane = np.random.normal(170, 10, 1000)
plt.hist(dane, bins=20)
plt.show()
```



Tematy:

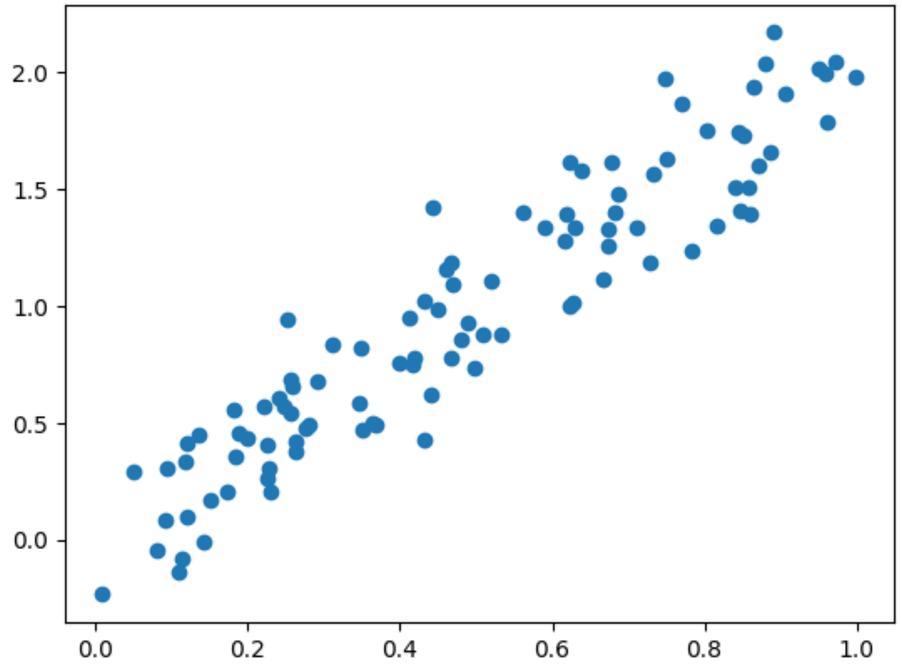
- rozkład normalny,
- liczba koszy (`bins`).

Typy wykresów: scatter plot

→ do zależności między zmiennymi.

```
x = np.random.rand(100)
y = x * 2 + np.random.normal(0, 0.2, 100)
plt.scatter(x, y)
plt.show()
```

Czy widać korelację?



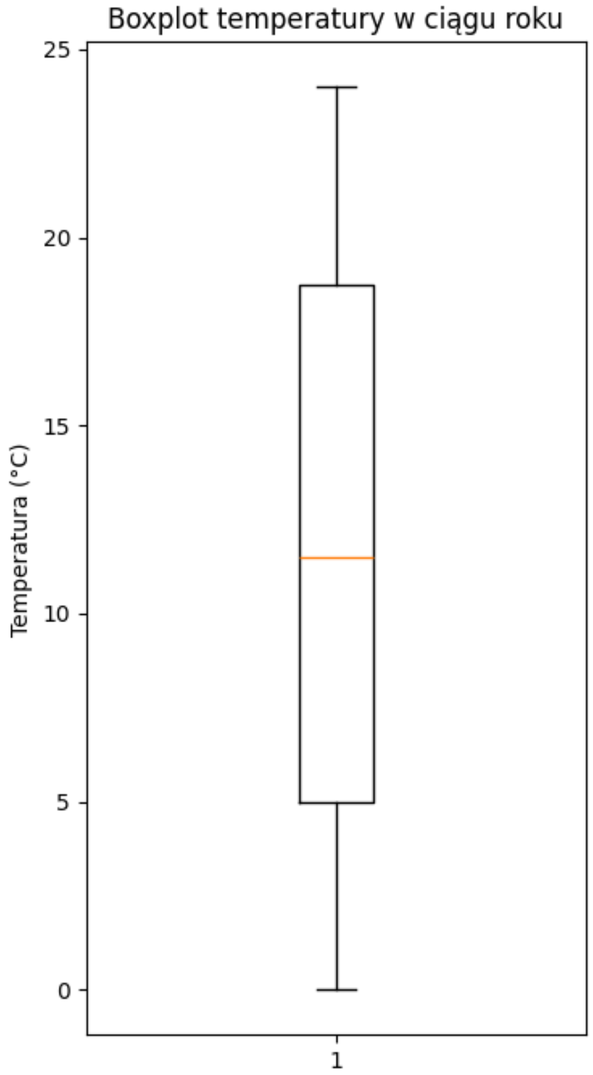
Typy wykresów: boxplot

→ do statystyki i wartości odstających.

```
plt.boxplot(dane)
plt.show()
```

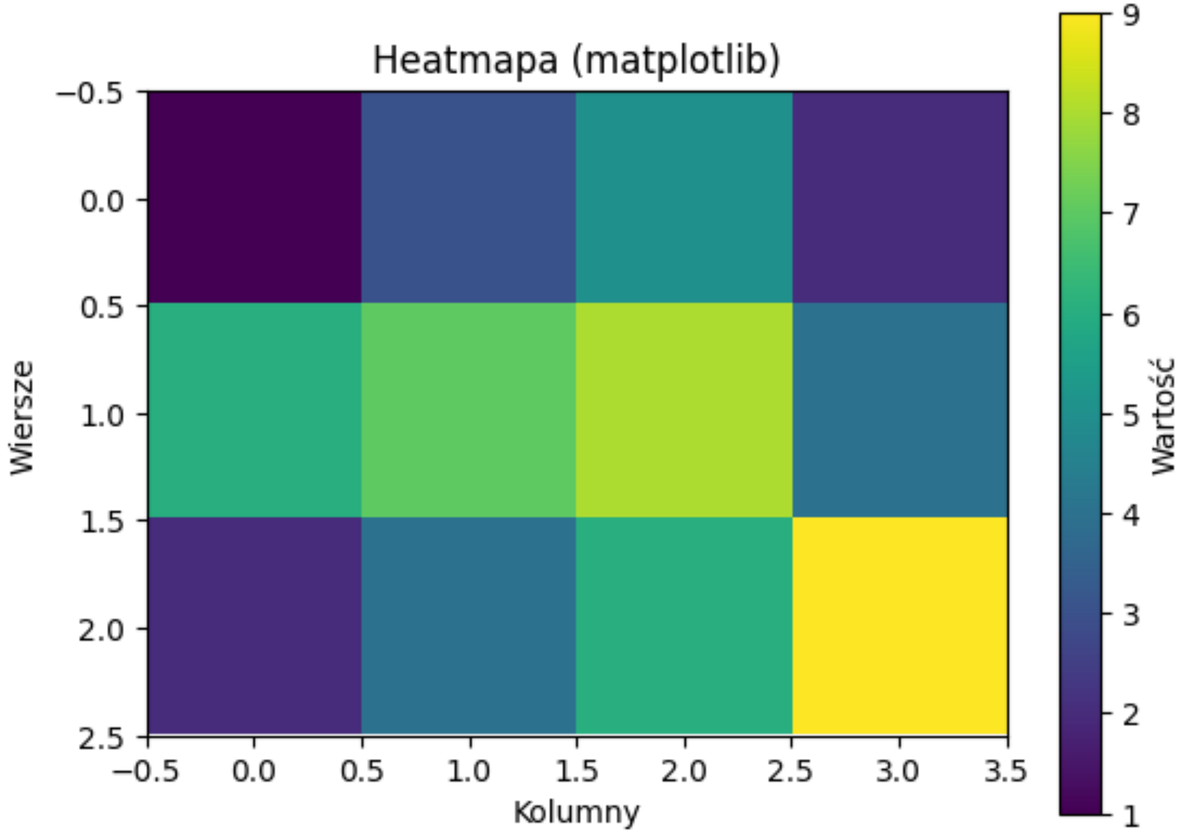
Przykład:

```
temperatura = [0, 2, 6, 11, 17, 21, 24, 23,
18, 12, 6, 1]
plt.boxplot(temperatura)
plt.title('Boxplot temperatury w ciągu roku')
plt.ylabel('Temperatura (°C)')
plt.show()
```



Typy wykresów: heatmap*

```
import matplotlib.pyplot as plt
import numpy as np
dane = np.array([
    [1, 3, 5, 2],
    [6, 7, 8, 4],
    [2, 4, 6, 9]
])
plt.imshow(dane, cmap='viridis')
plt.colorbar(label='Wartość')
plt.title('Heatmapa (matplotlib)')
plt.xlabel('Kolumny')
plt.ylabel('Wiersze')
plt.show()
```



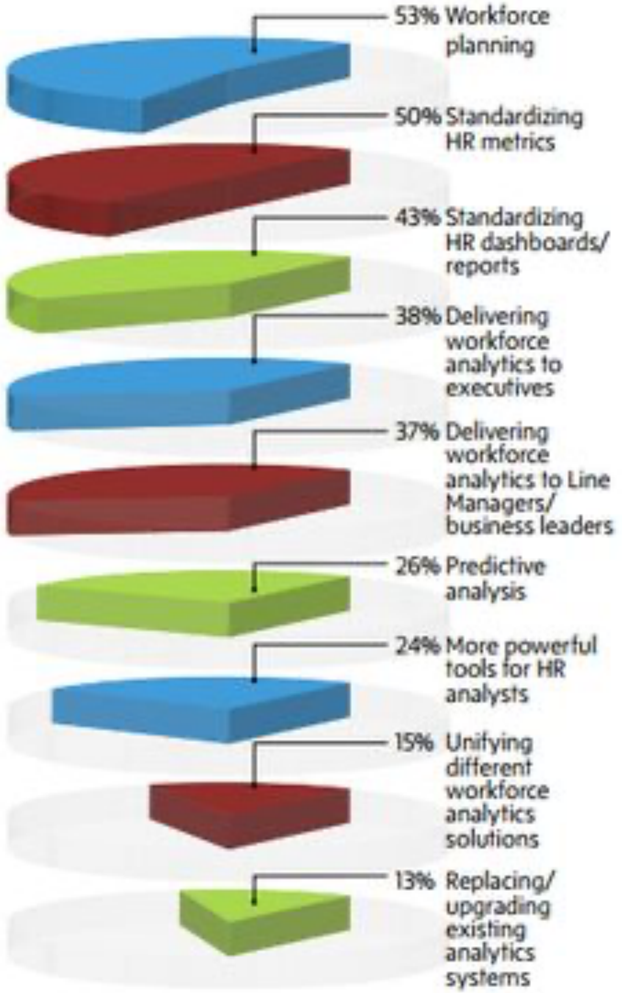
Stylizacja i estetyka

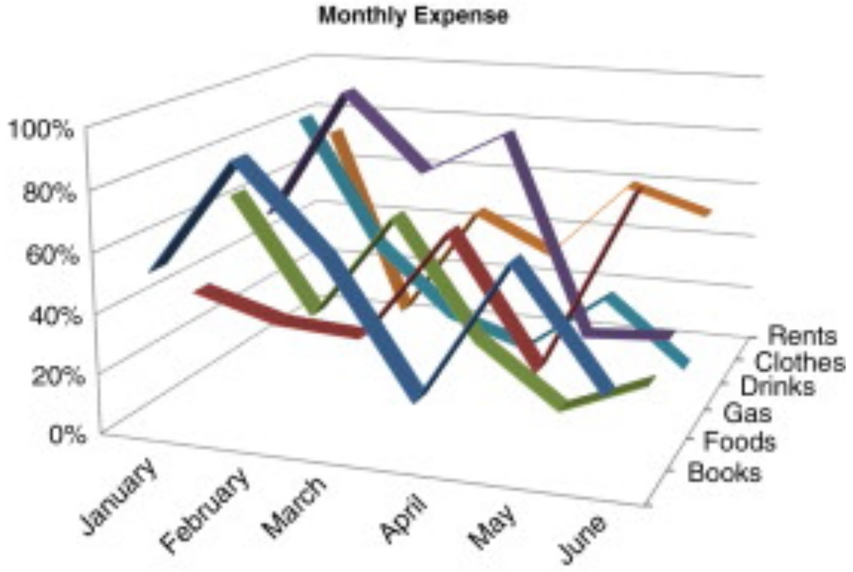
Dobry wykres ≠ kolorowy wykres

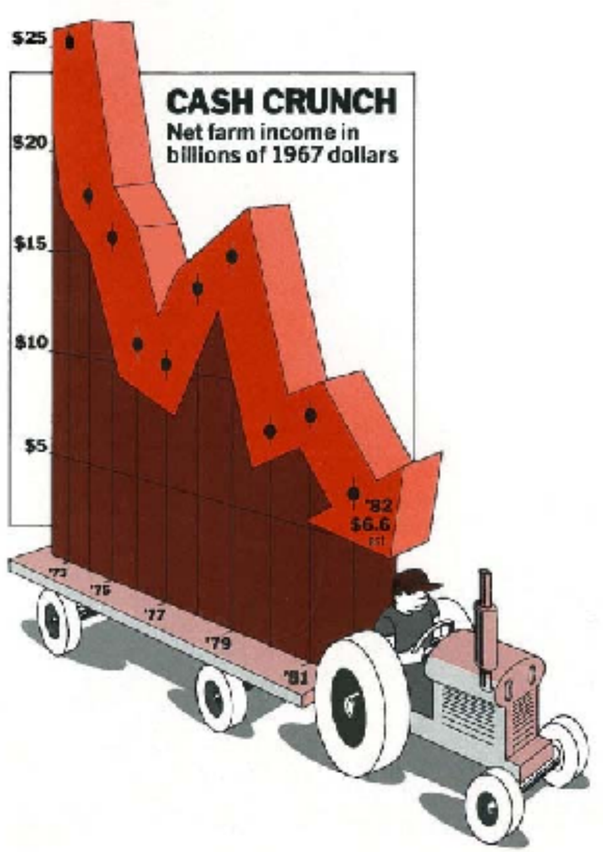
Zasady:

- minimalizm,
- czytelność,
- odpowiednie etykiety,
- unikanie „chartjunk” (wykres śmieciowy).

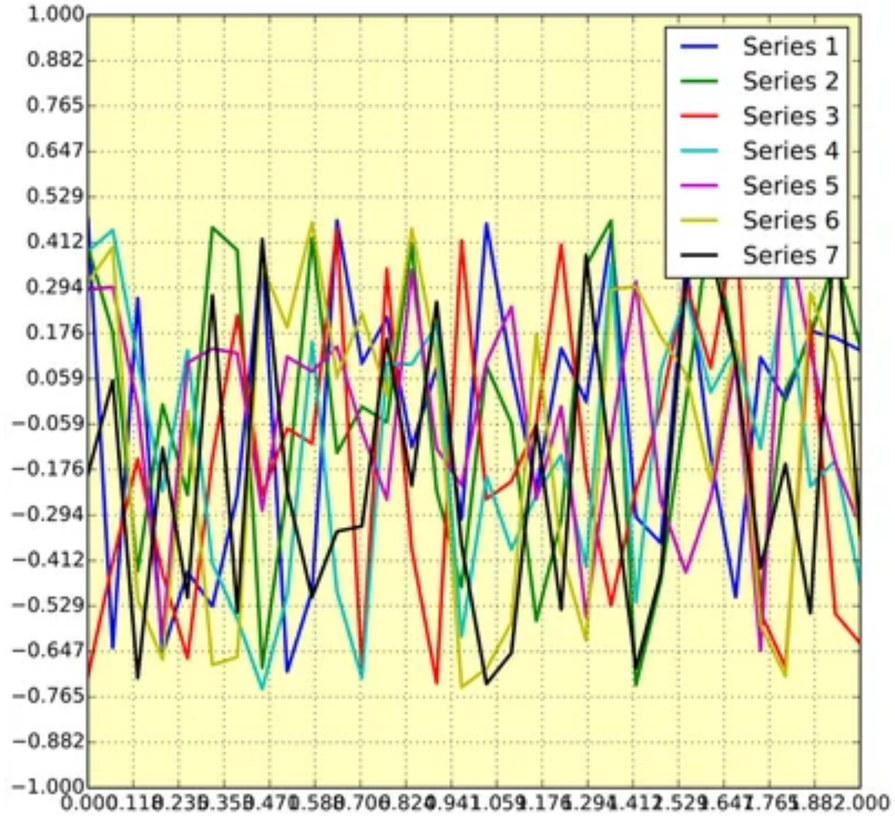
Stylizacja i estetyka

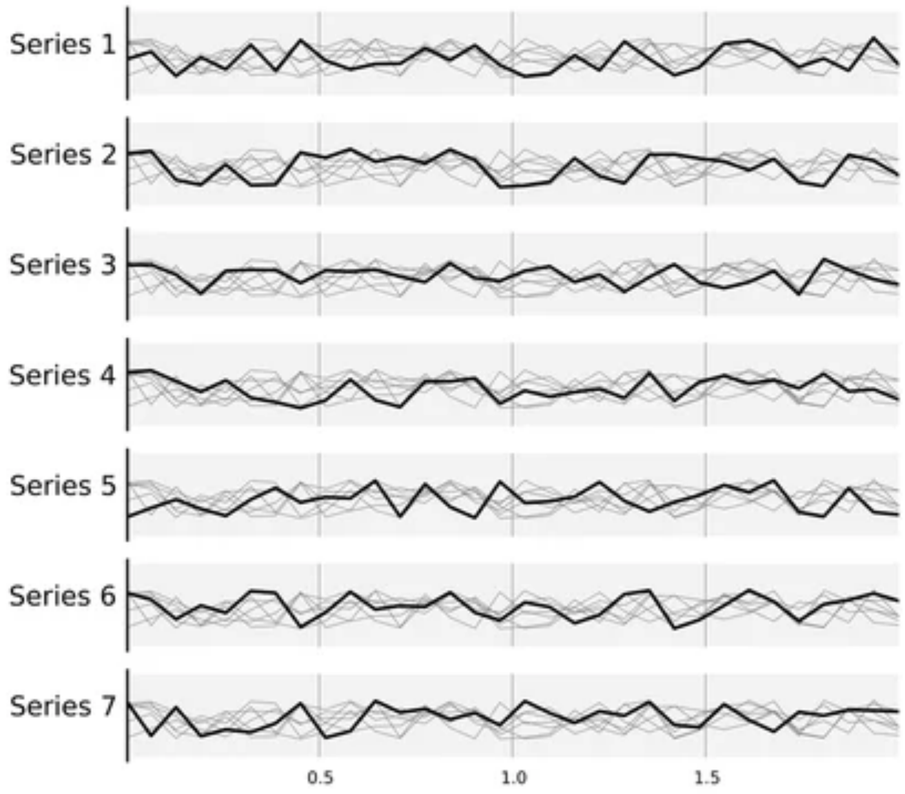






Stylizacja i estetyka

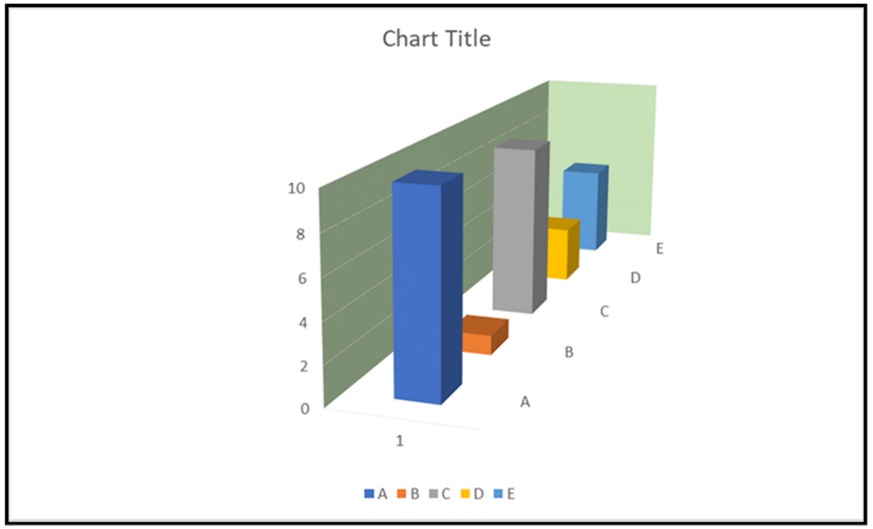




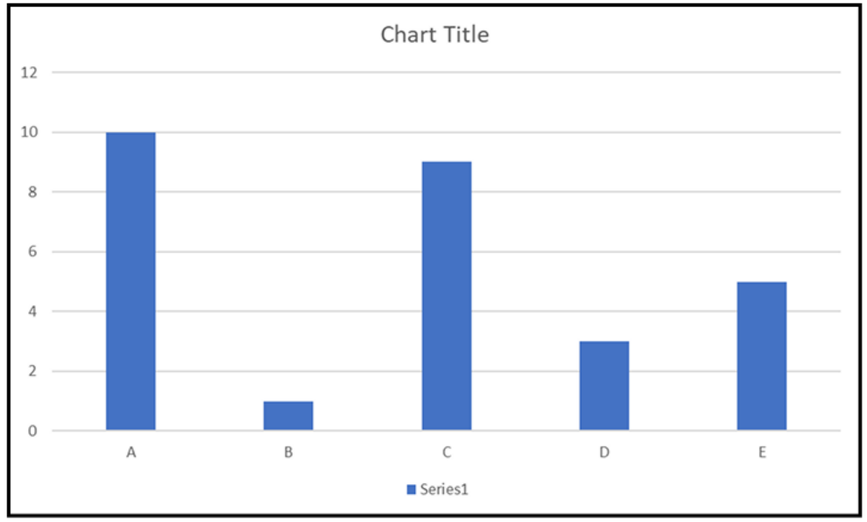
Vasilevsky, N. A., et al. (2014). *Ten Simple Rules for Better Figures*. PLOS Computational Biology, 10(9), e1003833. https://doi.org/10.1371/journal.pcbi.1003833

Stylizacja i estetyka

More chartjunk, **harder** to understand



Less chartjunk, **easier** to understand.



Lai, M., & Morrison, M. (b.d.). *Minimalist data visualizations aren't necessarily ideal: The Story of Goldilocks and the Three Charts*.

Stylizacja i estetyka

Style

```
plt.style.use('ggplot')
```

lub:

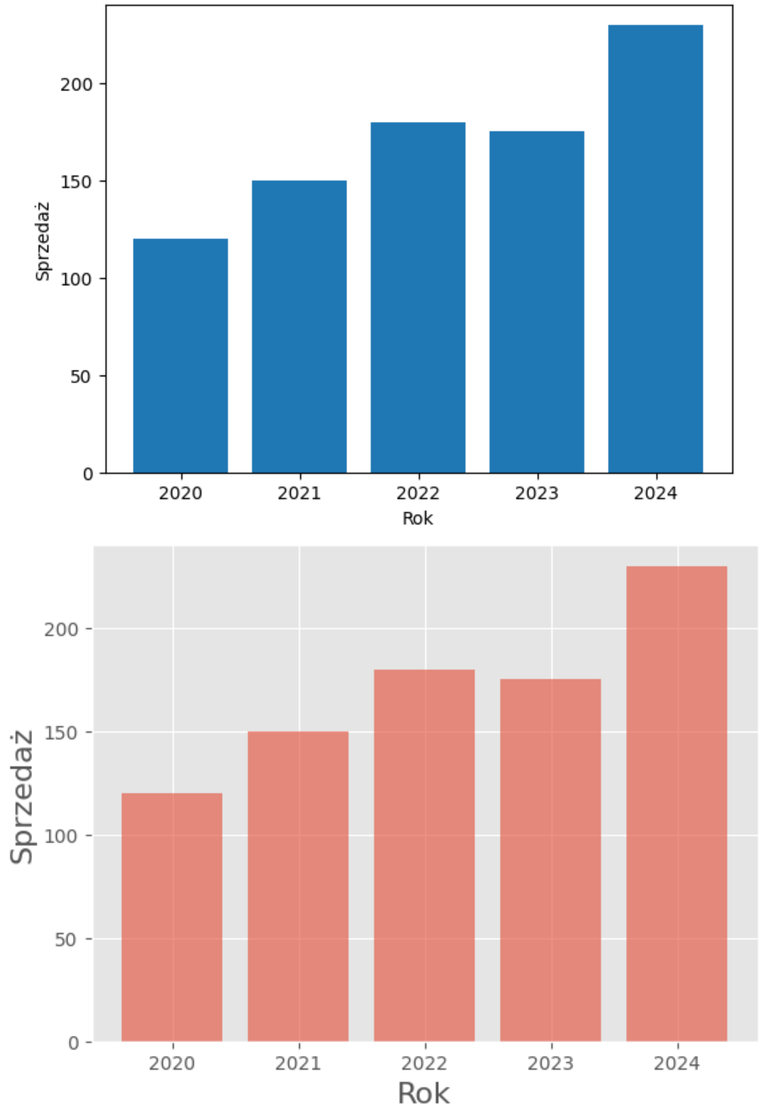
```
plt.style.available
```

Kolory i przezroczystość

```
plt.plot(x, y, color='red', alpha=0.6)
# alpha to przezroczystość tekstu, tu 60%
```

Rozmiar czcionki

```
plt.xlabel("Rok", fontsize=16)
```



Wiele wykresów jednocześnie

Funkcja **subplots**

```
fig, ax = plt.subplots(2, 2, figsize=(10, 8))
ax[0, 0].plot(x, y)
ax[0, 1].bar(produkty, sprzedaz)
ax[1, 0].hist(dane)
ax[1, 1].scatter(x2, y2)
plt.tight_layout()
plt.show()
```

Ważna idea:

Figure → Axes → Plot