

# БІБЛІОТЕКА MATPLOTLIB ТА ВІЗУАЛІЗАЦІЯ ДАНИХ

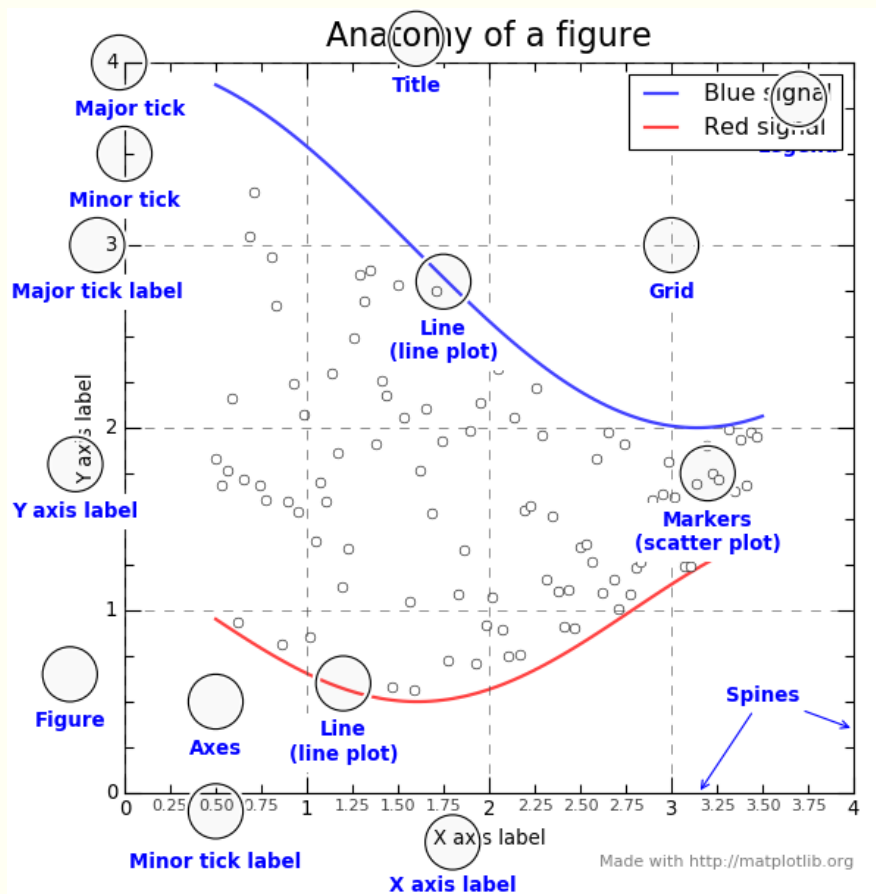
Питання 13.2.

21.04.2021

@Марченко С.В., ЧДБК, 2021

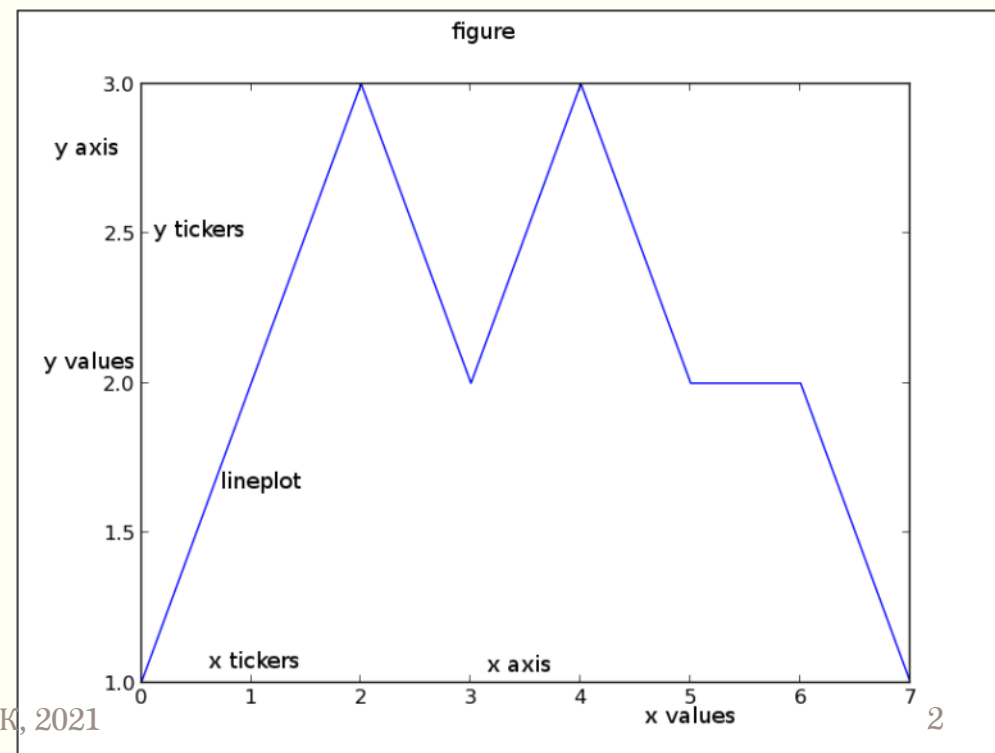


# Анатомія графіків бібліотеки matplotlib



■ **Рисунок (*Figure*)** – основний контейнер у matplotlib.

- Вісі (Axes) представляють координатну систему.
- Вона містить більшість елементів рисунка, наприклад, Axis, Line2D, Text.
- Допускається багато Axes-об'єктів на рисунку (Figure-об'єкти).
- Axis – контейнер для насічок (ticks) та написів (labels) для осей x та y на графіку (plot).



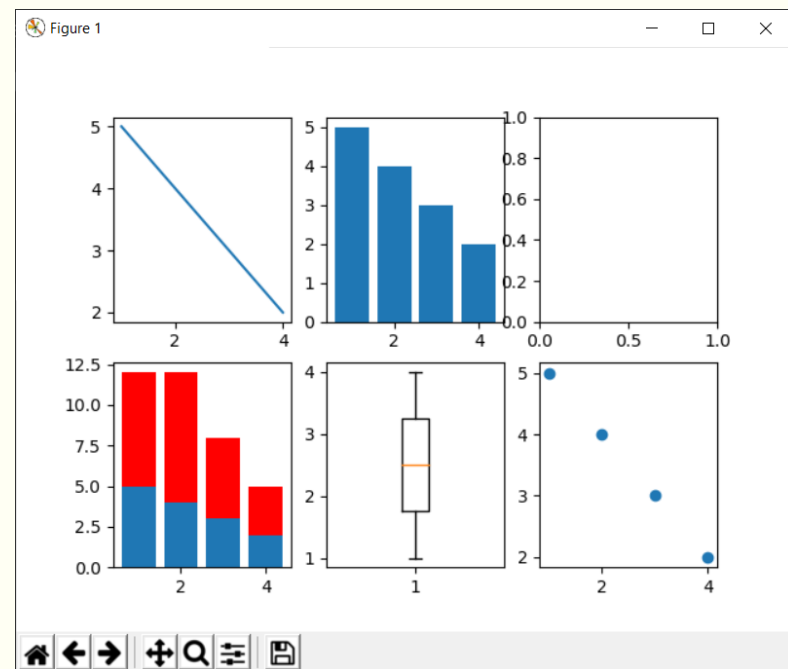
```
from matplotlib.pyplot import *
# some simple data
x = [1, 2, 3, 4]
y = [5, 4, 3, 2]
```

```
# create new figure
figure()
# divide subplots into 2 x 3 grid
# and select #1
subplot(231)
plot(x, y)
# select #2
subplot(232)
bar(x, y)
# horizontal bar-charts
subplot(233)
```

```
# create stacked bar charts
subplot(234)
bar(x, y)
# we need more data for stacked bar charts
y1 = [7, 8, 5, 3]
bar(x, y1, bottom=y, color='r')
# box plot
subplot(235)
boxplot(x)
# scatter plot
subplot(236)
scatter(x, y)
show()
barh(x, y)
```

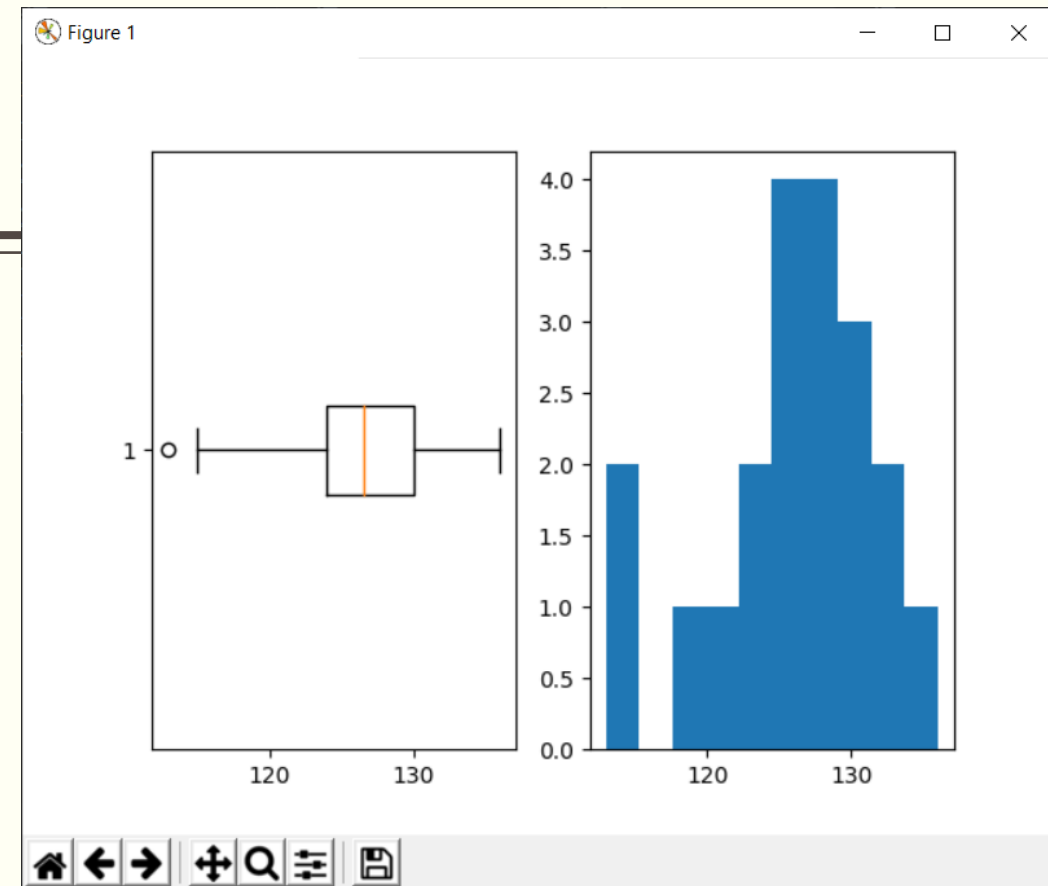
## Демонстраційний код

- За допомогою `figure()` створюємо новий рисунок.
  - Якщо передати рядковий аргумент, він стане backend title вікна.
  - Якщо викликати `figure()` з тим же параметром (це може бути й число), відповідний рисунок стане активним з подальшим рисунням на ньому.
- Розіб'ємо рисунок сіткою 2x3 за допомогою виклику `subplot(231)`.
  - Третій параметр позначає номер графіку по порядку.



# Побудова гістограм

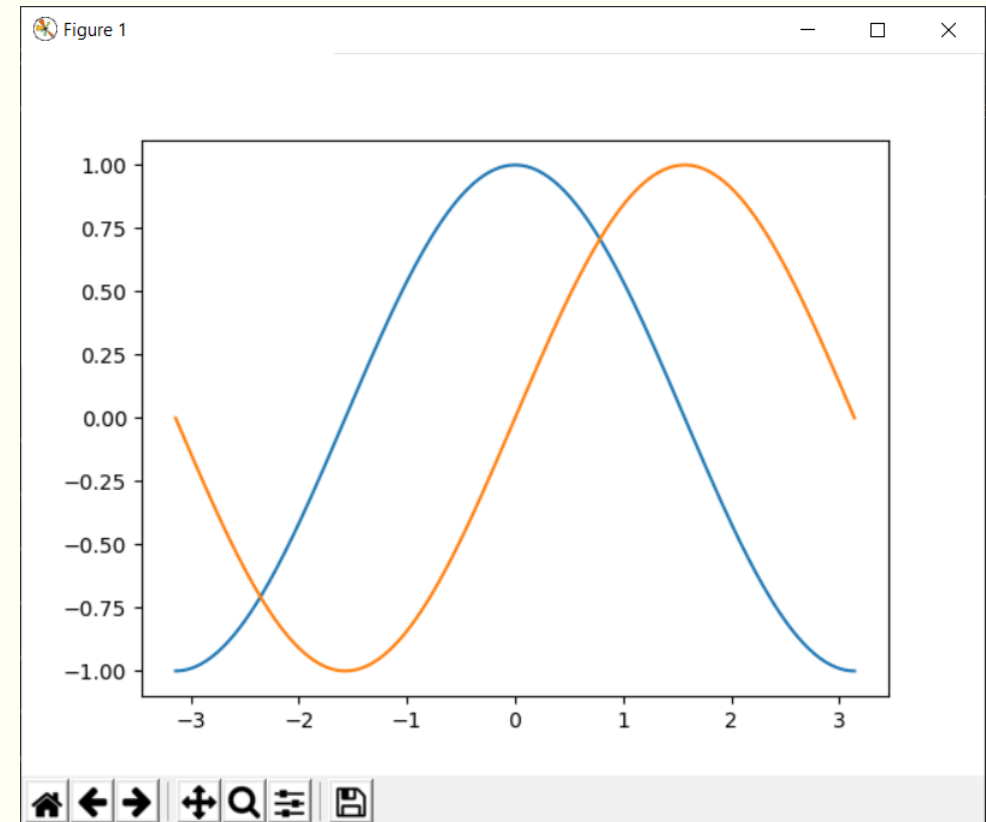
```
1 from pylab import *
2 dataset = [113, 115, 119, 121, 124,
3            124, 125, 126, 126, 126,
4            127, 127, 128, 129, 130,
5            130, 131, 132, 133, 136]
6 subplot(121)
7 boxplot(dataset, vert=False)
8 subplot(122)
9 hist(dataset)
10 show()
```



- Одні дані можна представити по-різному.
- На графіку можна представити 5 статистичних величин:
  - Мінімальне значення в наборі даних (dataset)
  - Другий квартиль: 25% з найменших значень для набору даних
  - Медіана: медіанне значення датасету (половина чисел менша за нього)
  - Третій квартиль: 25% з найбільших значень у датасеті
  - Максимальне значення в наборі даних

# Рисування простих графіків синуса та косинуса

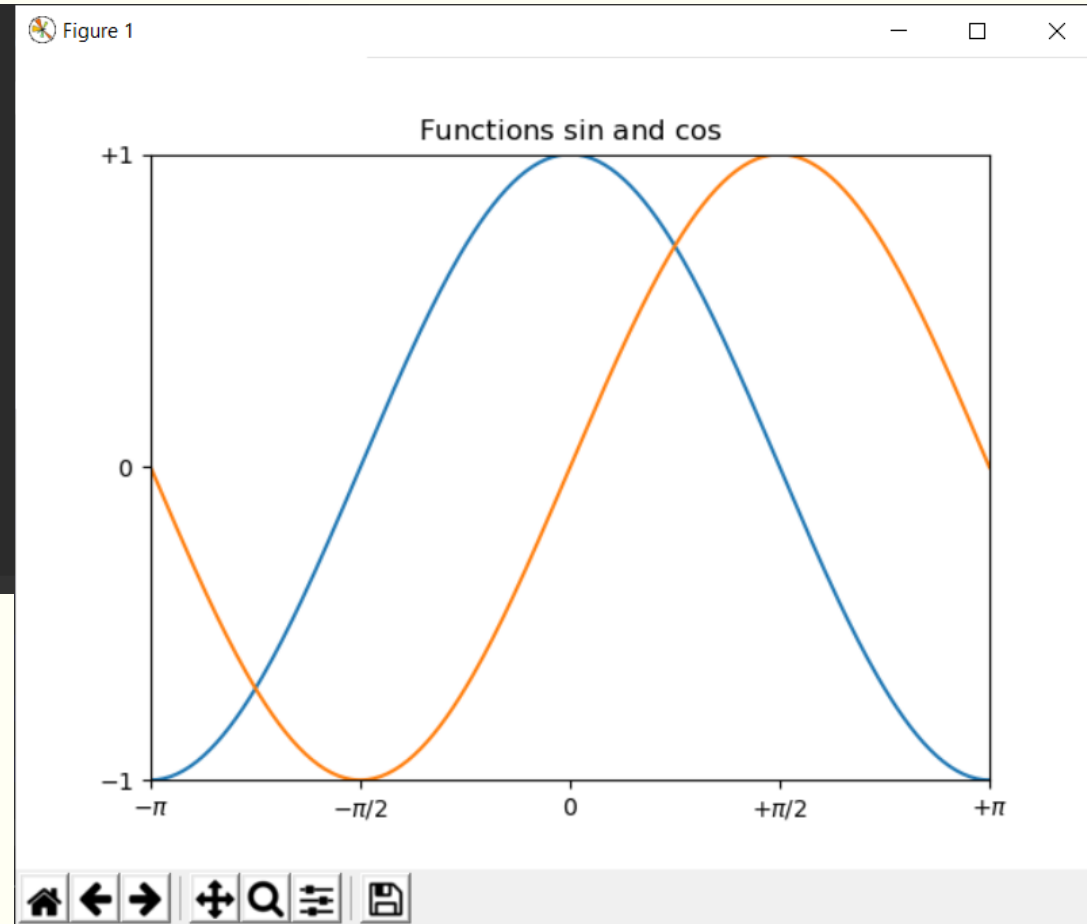
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
4 y = np.cos(x)
5 y1 = np.sin(x)
6 plt.plot(x, y)
7 plt.plot(x, y1)
8 plt.show()
```



# Додаткові налаштування для графіка

```
17 # define plot title
18 title("Functions  $\sin$  and  $\cos$ ")
19 # set x limit
20 xlim(-3.0, 3.0)
21 # set y limit
22 ylim(-1.0, 1.0)
23 # format ticks at specific values
24 xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
25 [r' $-\pi$ ', r' $-\pi/2$ ', r' $0$ ', r' $+\pi/2$ ', r' $+\pi$ '])
26 yticks([-1, 0, +1],
27 [r' $-1$ ', r' $0$ ', r' $+1$ '])
28 show()
```

- Використовуються вирази на зразок  $\sin$  або  $\pi$ , що є синтаксисом LaTeX.



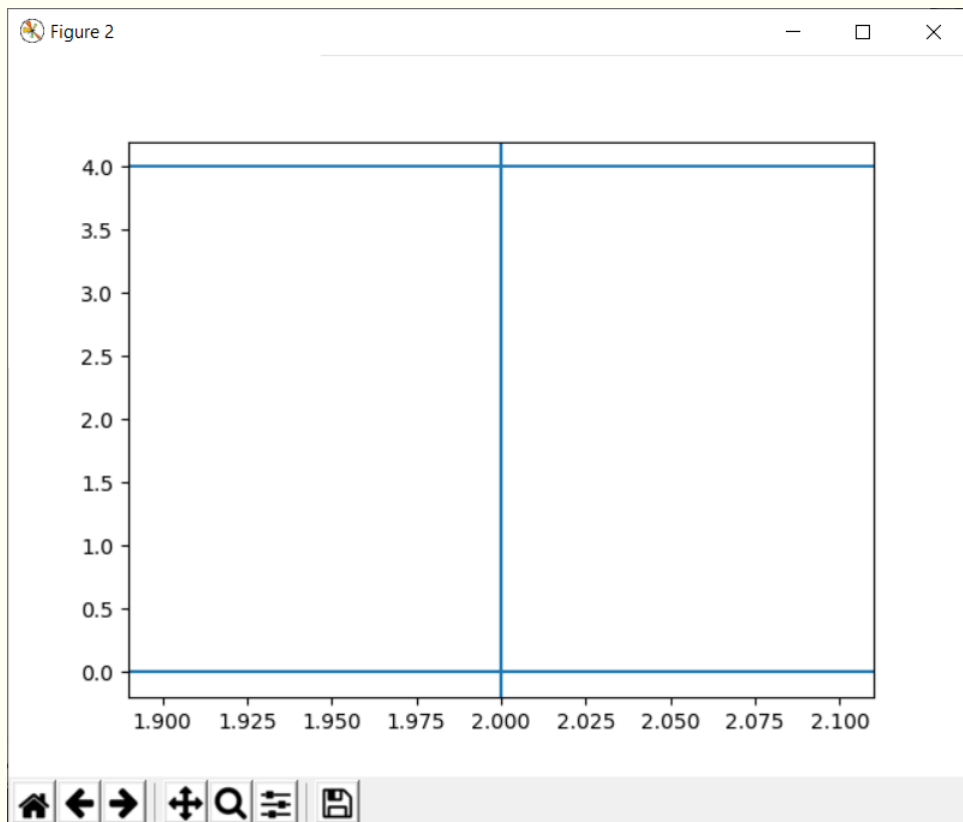
# Робота з сіткою та осями

---

- `axis([-1, 1, -10, 10])` визначає розміри осей `xmin`, `xmax`, `ymin`, `ymax`
  - Можна викликати `matplotlib.pyplot.autoscale()`, яка підбере оптимальні розміри відповідно до даних.
- Для додавання нових осей на тому ж рисунку можна використати `matplotlib.pyplot.axes()`.
  - Зазвичай до цього виклику додаються деякі властивості;
  - `rect` з атрибутами `left`, `bottom`, `width` та `height` у нормалізованих одиницях вимірювання (0, 1)
  - `axisbg` задає колір фону осей.
  - `sharex/sharey` приймає значення інших екземплярів осей та ділиться з ними поточними налаштуваннями для `x/y`.
  - `polar` – параметр, який визначає, чи потрібно використовувати полярну систему координат.
- Виклик `matplotlib.pyplot.grid()` перемикає видимість сітки на рисунку з параметрами:
  - `which`: визначає тип насічок для відрисовки (`major`, `minor` або обидва варіанти)
  - `axis`: визначає набір ліній сітки для відрисовки (`x`, `y` або обидві)
- Осі зазвичай управляються за допомогою `matplotlib.pyplot.axis()`.
  - Всередині осі представляються кількома Python-класами:
    - `matplotlib.axes.Axes` містить більшість методів для операцій з осями.
    - `matplotlib.axis.Axis` описує окрему вісь за допомогою класів `matplotlib.axis.XAxis` та `matplotlib.axis.YAxis`.

```
figure(2)
axhline(4)
axhline(4)
axvline(2)

show()
```



- Якщо потрібно додати до поточного рисунку лише одну лінію, можна застосувати функції `matplotlib.pyplot.axhline()` або `matplotlib.pyplot.axvline()`.
  - Вони рисують горизонтальні та вертикальні лінії вздовж осей x та y відповідно.
  - Найбільш важливі спільні параметри: y-координата, `xmin` та `xmax` для `axhline()`; x-координата, `ymin` та `ymax` для `axvline()`.
- Подібні функції `matplotlib.pyplot.axhspan()` та `matplotlib.pyplot.axspan()` дозволяють додавати `horizontal span` (rectangle) вздовж осей.
  - `axhspan()` потребує параметрів `ymin` та `ymax` для визначення довжини `horizontal span`.
  - `axvspan()` потребує `xmin` та `xmax` для визначення довжини `vertical span`.



# Визначення стилів ліній, властивостей та форматуваних рядків на графіку

---

- Стиль ліній на графіку можна змінити різними методами:
  - 1) Найбільш поширений спосіб – передавати параметри у функції на зразок `plot()`:
    - `plot(x, y, linewidth=1.5)`
  - 2) Оскільки `plot()` повертає екземпляр `matplotlib.lines.Line2D`, можна задати сеттери для цього екземпляру:
    - `line, = plot(x, y)`
    - `line.set_linewidth(1.5)`
  - 3) Спосіб, знайомий для користувачів MATLAB,—функція `setp()`:
    - `lines = plot(x, y)`
    - `setp(lines, 'linewidth', 1.5)` або `setp(lines, linewidth=1.5)`

Linestyle	Description
' - '	Solid
' - - '	Dashed
' - . '	Dash_dot
' : '	Dotted
'None', ' ', ''	Draw nothing

Alias	Color
b	Blue
g	Green
r	Red
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

- Можна отримати всі доступні кольори, викликавши `matplotlib.pyplot.colors()`

# Аргументи налаштування кольору

---

- Якщо базових кольорів не вистачає, можна застосувати шістнадцяткове HTML-значення, назву кольору або нормалізований RGB-кортеж (від 0 до 1):
  - `color = '#eeeeff'`
  - `color = 'red', color = 'chartreuse'`.
  - `color = (0.3, 0.3, 0.4)`
- Аргумент `color` приймається багатьма функціями, зокрема `title()`:
  - `title('Title in a custom color', color='#123456')`
- Аргумент `axisbg` може визначати фоновий колір осі у таких функціях, як `matplotlib.pyplot.axes()` або `matplotlib.pyplot.subplot()`:
  - `subplot(111, axisbg=(0.1843, 0.3098, 0.3098))`

# Маркери на графіку

---

Marker	Description
'o'	Circle
'D'	Diamond
'h'	Hexagon1
'H'	Hexagon2
'_'	Horizontal line
',' , 'None' , ' ', None	Nothing
'8'	Octagon
'p'	Pentagon
','	Pixel
'+'	Plus
'.'	Point
's'	Square
'*'	Star

Marker	Description
'd'	Thin_diamond
'v'	Triangle_down
'<'	Triangle_left
'>'	Triangle_right
'^'	Triangle_up
' '	Vertical line
'x'	X

## Доступні для зміни властивості класу `matplotlib.lines.Line2D`

Властивість	Тип значення	Опис
<code>alpha</code>	<code>float</code>	Задає рівень прозорості при змішуванні
<code>color</code> або <code>c</code>	Будь-який колір <code>matplotlib</code>	Встановлює колір лінії
<code>dashes</code>	Послідовність	Визначає рискову послідовність. Якщо <code>seq = (None, None)</code> , <code>linestyle</code> матиме значення <code>solid</code> .
<code>label</code>	Довільний рядок	Задає напис <code>s</code> для легенди
<code>linestyle</code> або <code>ls</code>	<code>['-'   '-'   '-.'   ':'   'steps'   ...]</code>	Визначає стиль лінії
<code>linewidth</code> або <code>lw</code>	<code>float</code> -значення в пунктах ( <code>pt</code> )	Встановлює товщину лінії в пунктах
<code>marker</code>	<code>[ 7   4   5   6   'o'   'D'   'h'   'H'   '_'   "   'None'   ''   None   '8'   'p'   '.'   '+'   '.'   's'   '*'   'd'   3   0   1   2   '1'   '3'   '4'   '2'   'v'   '&lt;'   '&gt;'   '^'   ' '   'x'   '\$...\$'   tuple   Nx2 array ]</code>	Задає маркер на лінії

# Доступні для зміни властивості класу `matplotlib.lines.Line2D`

Property	Value type	Description
<code>ydata</code>	<code>np.array</code>	Set the data <code>np.array</code> for y.
<code>Zorder</code>	Any number	Set the z axis order for the artist. Artists with lower <code>Zorder</code> values are drawn first.  If x and y are axes going horizontal to the right and vertical to the top of the screen, the z axis is the one extending toward the viewer. So 0 value would be at the screen, 1, one layer above, and so on.
<code>markeredgecolor</code> or <code>mec</code>	Any matplotlib color	Sets the marker edge color.
<code>markeredgewidth</code> or <code>mew</code>	float value in points	Sets the marker edge width in points.
<code>markerfacecolor</code> or <code>mfc</code>	Any matplotlib color	Set the marker face color.
<code>markersize</code> or <code>ms</code>	float	Set the marker size in points.
<code>solid_capstyle</code>	['butt'   'round'   'projecting']	Set the cap style for solid line styles.
<code>solid_joinstyle</code>	['miter'   'round'   'bevel']	Set the join style for solid line styles.
<code>visible</code>	[True   False]	Set the artist's visibility.
<code>xdata</code>	<code>np.array</code>	Set the data <code>np.array</code> for x.

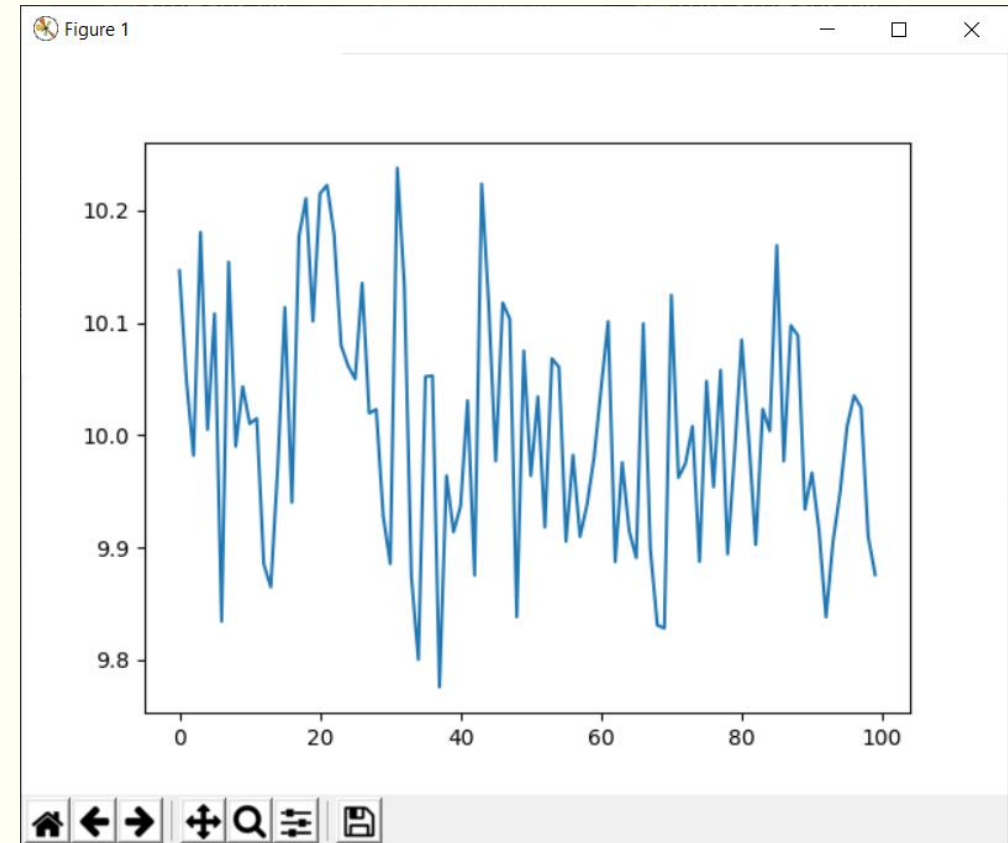
# Встановлення насічок, написів та сіток

---

- Насічки є частиною рисунку та складаються з локаторів (tick locators, розташування насічок) та форматтерів (tick formatters, визначають появу насічок).
  - Мінорні насічки (Minor ticks) за умовчанням невидимі.
  - Мажорні та мінорні насічки можна розташовувати та форматовувати незалежно одні від одних.
- Для управління поведінкою локаторів застосовується функція `matplotlib.pyplot.locator_params()`.
  - Хоч розташування насічок зазвичай визначається автоматично, кількістю насічок можна керувати.

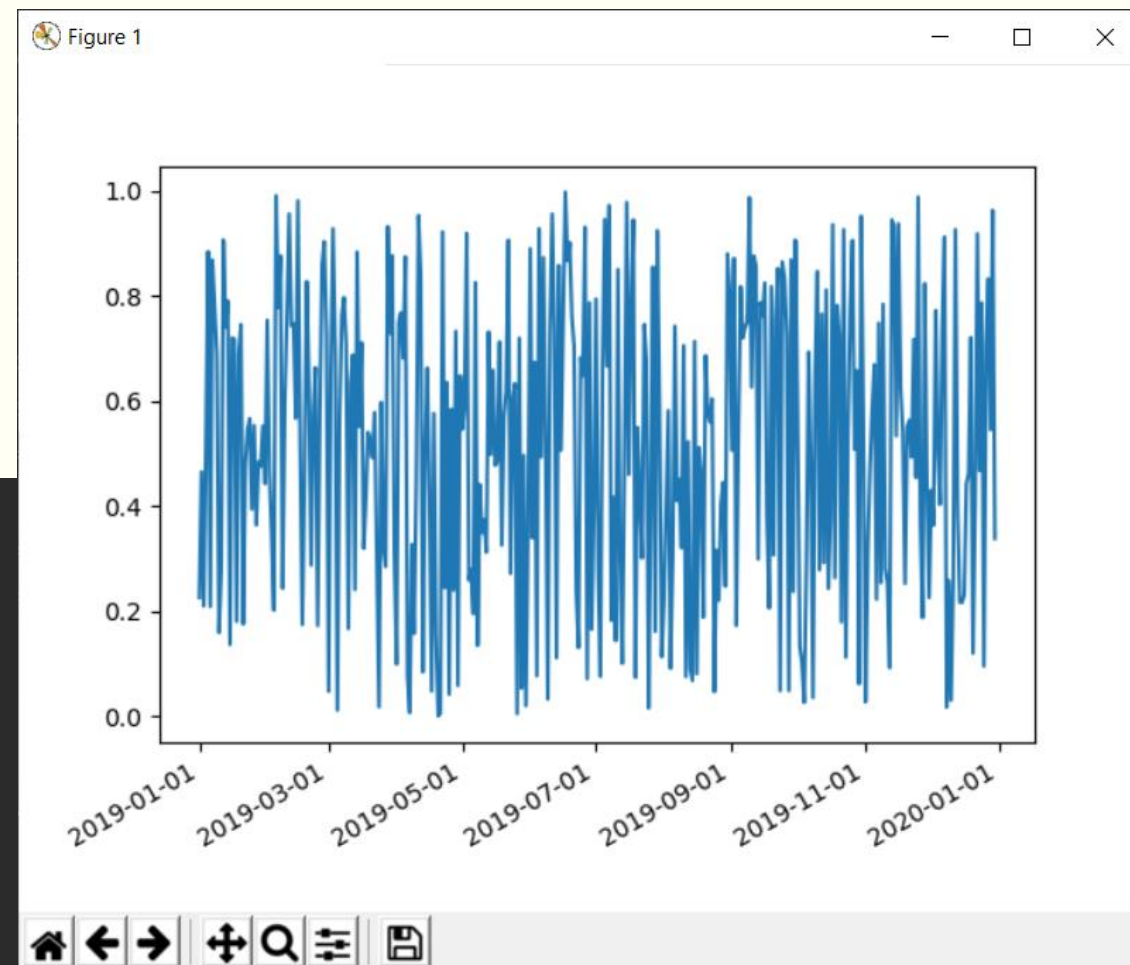
```
1  from pylab import *
2  # get current axis
3  ax = gca()
4  # set view to tight, and maximum number of tick intervals to 10
5  ax.locator_params(tight=True, nbins=10)
6  # generate 100 normal distribution values
7  ax.plot(np.random.normal(10, .1, 100))
8  show()
```

- Такі ж налаштування можна здійснити за допомогою locator-класів.
  - `ax.xaxis.set_major_locator(matplotlib.ticker.MultipleLocator(10))`
- Аналогічно можна задати форматтери насічок.
  - Форматтери визначають, як значення (зазвичай числа) будуть відображатись.
  - Наприклад, `matplotlib.ticker.FormatStrFormatter` просто задає `'%2.1f'` або `'%1.1f cm'` to be used as the label for the ticker.



## Приклад з використанням дат

```
1 from pylab import *
2 import matplotlib as mpl
3 import datetime
4 fig = figure()
5 # get current axis
6 ax = gca()
7 # set some daterange
8 start = datetime.datetime(2019, 1, 1)
9 stop = datetime.datetime(2019, 12, 31)
10 delta = datetime.timedelta(days=1)
11 # convert dates for matplotlib
12 dates = mpl.dates.drange(start, stop, delta)
13 # generate some random values
14 values = np.random.rand(len(dates))
15 ax = gca()
16 # create plot with dates
17 ax.plot_date(dates, values, linestyle='-', marker='')
18 # specify formater
19 date_format = mpl.dates.DateFormatter('%Y-%m-%d')
20 # apply formater
21 ax.xaxis.set_major_formatter(date_format)
22 # autofmt date labels
23 # rotates labels by 30 degrees by default
24 # use rotate param to specify different rotation degree
25 # use bottom param to give more room to date labels
26 fig.autofmt_xdate()
27 show()
```





```

1 from matplotlib.pyplot import *
2 # generate different normal distributions
3 x1 = np.random.normal(30, 3, 100)
4 x2 = np.random.normal(20, 2, 100)
5 x3 = np.random.normal(10, 3, 100)
6 # plot them
7 plot(x1, label='plot')
8 plot(x2, label='2nd plot')
9 plot(x3, label='last plot')
10 # generate a legend box
11 legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,
12        ncol=3, mode="expand", borderaxespad=0.)
13 # annotate an important value
14 annotate("Important value", (55, 20), xycoords='data',
15        xytext=(5, 38),
16        arrowprops=dict(arrowstyle='->'))
17 show()

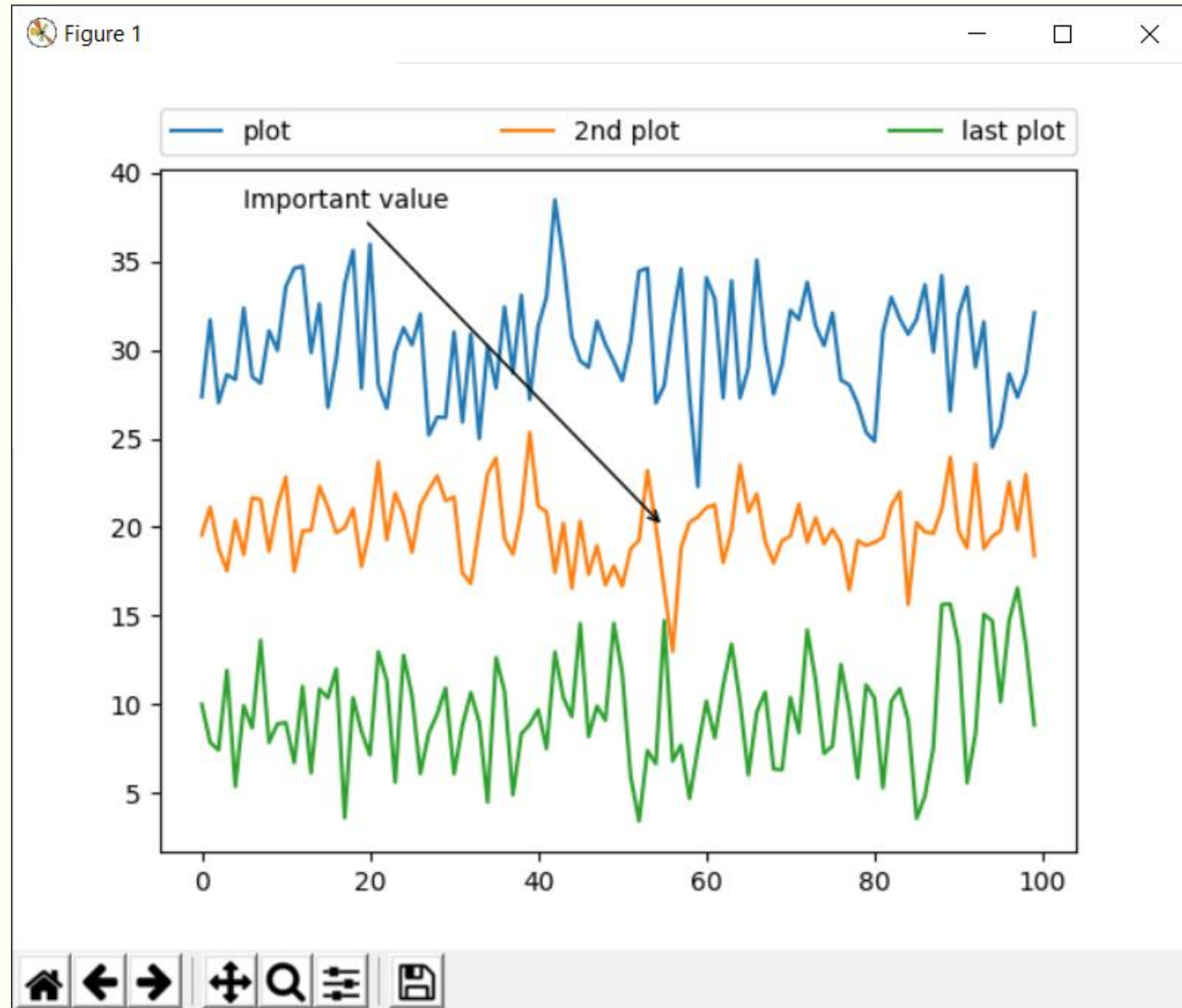
```

## Додавання легенди та annotations

String	Number value
upper right	1
upper left	2
lower left	3
lower right	4
right	5
center left	6
center right	7
lower center	8
upper center	9
center	10

- Задаємо розташування легенди за допомогою опційного параметру loc.
  - Для приховування легенди label = '\_nolegend\_'.
  - Для легенди визначили кількість стовпців ncol = 3 та задали розташування lower left.
  - Задано bounding box (bbox\_to\_anchor) з координатою верхнього лівого кута (0., 1.02), довжиною (width) 1 та шириною (height) 0.102. Координати нормалізовані.
  - Параметр mode може бути 'None' або 'expand' (horizontally filling the axis area).
  - Параметр borderaxespad визначає зміщення (padding) між осями та контуром легенди.

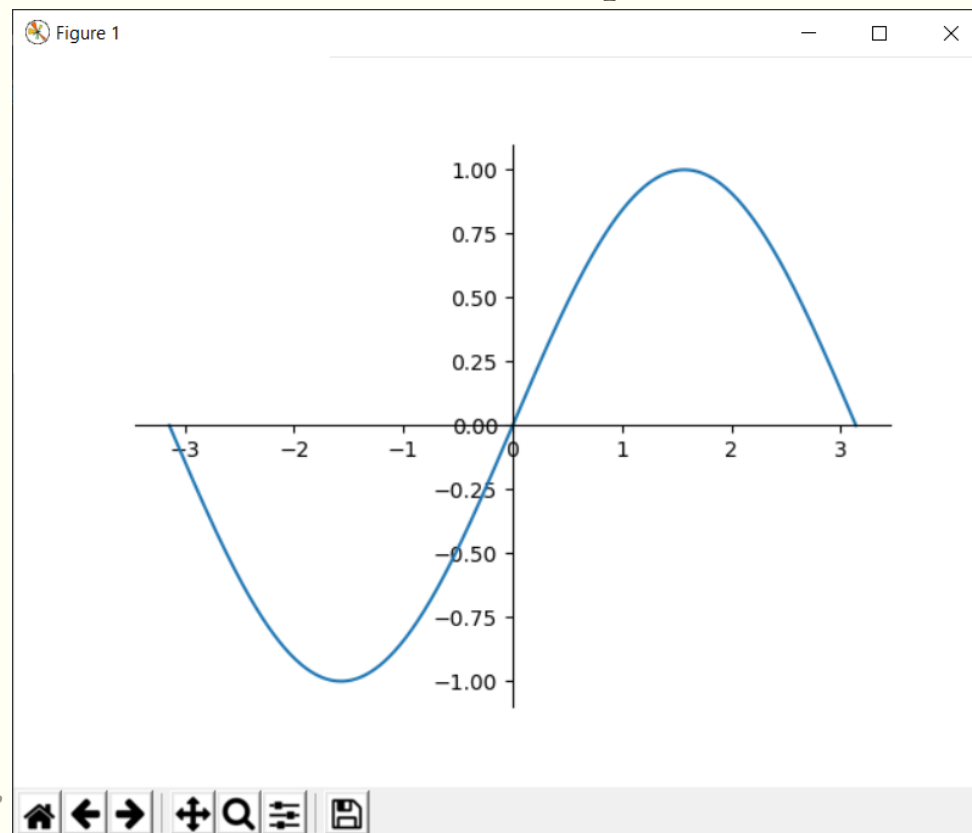
# Результат запуску коду



# Переміщення spines до центру

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 x = np.linspace(-np.pi, np.pi, 500, endpoint=True)
4 y = np.sin(x)
5 plt.plot(x, y)
6 ax = plt.gca()
7 # hide two spines
8 ax.spines['right'].set_color('none')
9 ax.spines['top'].set_color('none')
10 # move bottom and left spine to 0,0
11 ax.spines['bottom'].set_position(('data', 0))
12 ax.spines['left'].set_position(('data', 0))
13 # move ticks positions
14 ax.xaxis.set_ticks_position('bottom')
15 ax.yaxis.set_ticks_position('left')
16 plt.show()
```

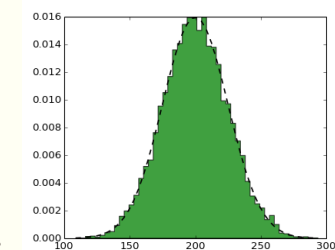
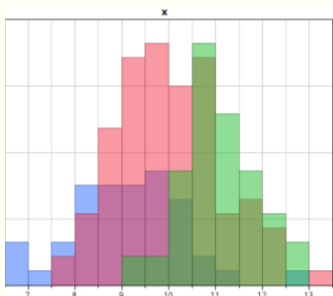
- Необхідно видалити 2 spines, приховуючи їх (присвоїти color значення 'none').
  - Після цього переміщуємо два інших в координати (0,0).
  - Координати задаються в data space coordinates.



# Створення гістограм

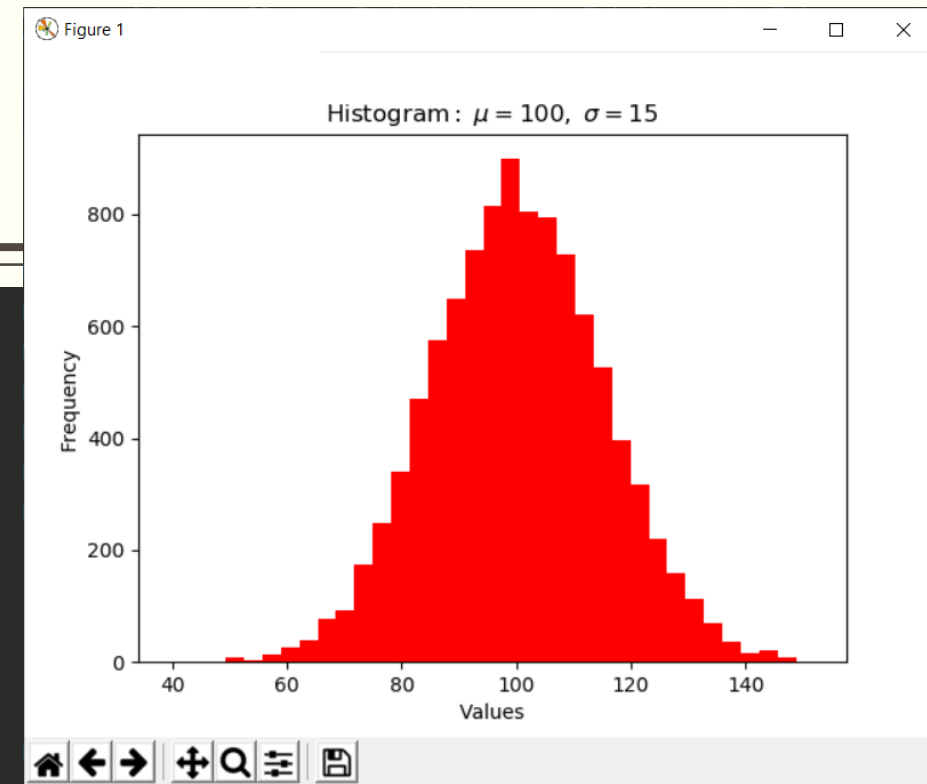
---

- Виклик `matplotlib.pyplot.hist()` з набором параметрів. Найкорисніші:
  - ***bins***: кількість бінів гістограми або послідовність, яка задає біни. За умовчанням – 10.
  - ***range***: діапазон бінів, не використовується при послідовності бінів. За умовчанням `None`, `Outliers` ігноруються.
  - ***normed***: якщо дорівнює `True`, значення гістограми нормалізуються та формують розподіл ймовірності. За умовчанням – `False`.
  - ***histtype***: стандартний вигляд гістограми – прямокутниками. Інші опції:
    - `barstacked`: стекове представлення, як на рисунку зліва.
    - `step`: лінійна гістограма, яка `left unfilled`.
    - `stepfilled`: лінійна гістограма, яка залита за умовчанням. Елемент за умовчанням – прямокутник.
  - ***align***: центрує прямокутники між границями бінів. За умовчанням - `mid`. Інші значення - `left` і `right`.
  - ***color***: задає колір гістограми. Одне значення або послідовність кольорів.
    - Якщо задано `multiple datasets`, послідовність кольорів буде застосована в тому ж порядку.
    - Якщо не задано, використовується `default line color sequence`.
  - ***orientation***: дозволяє створення горизонтальних гістограм (`'horizontal'`). За умовчанням – `'vertical'`.



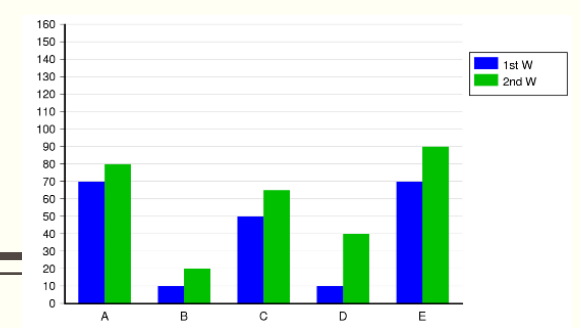
# Приклад гістограми

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 mu = 100
4 sigma = 15
5 x = np.random.normal(mu, sigma, 10000)
6 ax = plt.gca()
7 # the histogram of the data
8 ax.hist(x, bins=35, color='r')
9 ax.set_xlabel('Values')
10 ax.set_ylabel('Frequency')
11 ax.set_title(r'$\mathrm{Histogram:} \ \mu = %d, \ \sigma = %d$' % (mu, sigma))
12 plt.show()
```



- Починаємо з генерування деяких нормально розподілених даних.
  - Гістограма будується із заданою кількістю бінів—35 та нормалізується (`normed = True` або 1);
  - колір – червоний ('r').
  - Далі встановлюємо написи та заголовок графіку.

# Створення стовпчикової діаграми (bar charts)

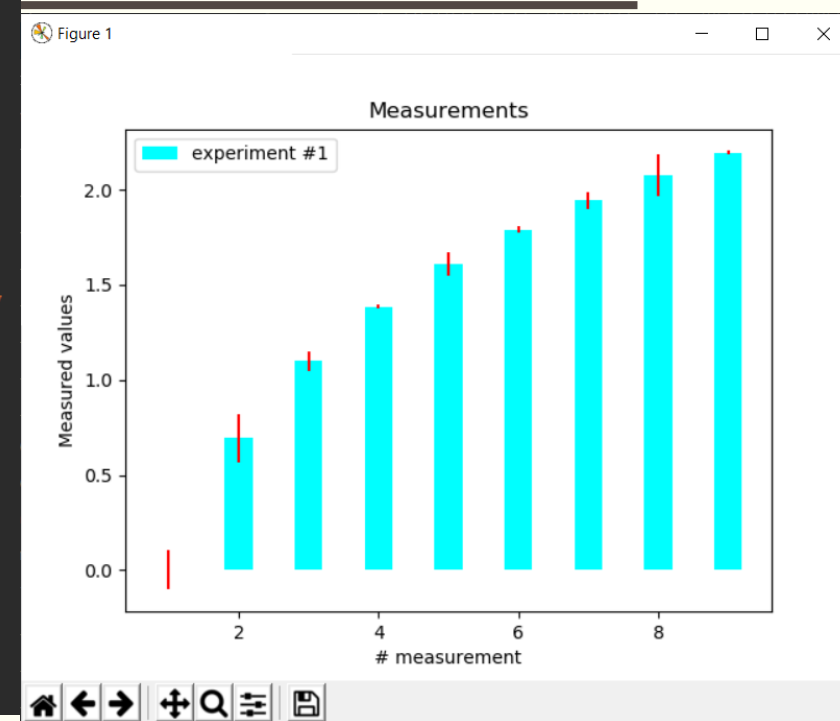


- Для візуалізації неточності вимірювання в наборі даних або для виявлення похибок можна використовувати error bars.
  - Вони можуть показати стандартне відхилення (standard deviation), стандартну похибку (standard error) або 95%-ий інтервал достовірності (confidence interval).
  - Це не стандартизовано, тому завжди явно перевіряйте, які значення (похибки) error bars показують.
- Разом з двома обов'язковими параметрами—`left` і `height`—можна використовувати й опційні:
  - ***width***: ширина прямокутника. За умовчанням – 0.8.
  - ***bottom***: якщо задане, це значення додається до `height`. За умовчанням – None.
  - ***edgcolor***: задає колір контурів прямокутників.
  - ***ecolor***: задає колір будь-якого error bar.
  - ***linewidth***: задає ширину контуру прямокутника; спеціальні значення – None (за умовчанням) та 0 (коли контури не відображаються).
  - ***orientation***: вертикальна або горизонтальна орієнтація.
  - ***xerr*** та ***yerr***: використовуються для генерування error bars на стовпчиковій діаграмі.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 # generate number of measurements
4 x = np.arange(0, 10, 1)
5 # values computed from "measured"
6 y = np.log(x)
7 # add some error samples from standard normal distribution
8 xe = 0.1 * np.abs(np.random.randn(len(y)))
9 # draw and show errorbar
10 plt.bar(x, y, yerr=xe, width=0.4, align='center', ecolor='r',
11         color='cyan', label='experiment #1');
12 # give some explanations
13 plt.xlabel('# measurement')
14 plt.ylabel('Measured values')
15 plt.title('Measurements')
16 plt.legend(loc='upper left')
17 plt.show()

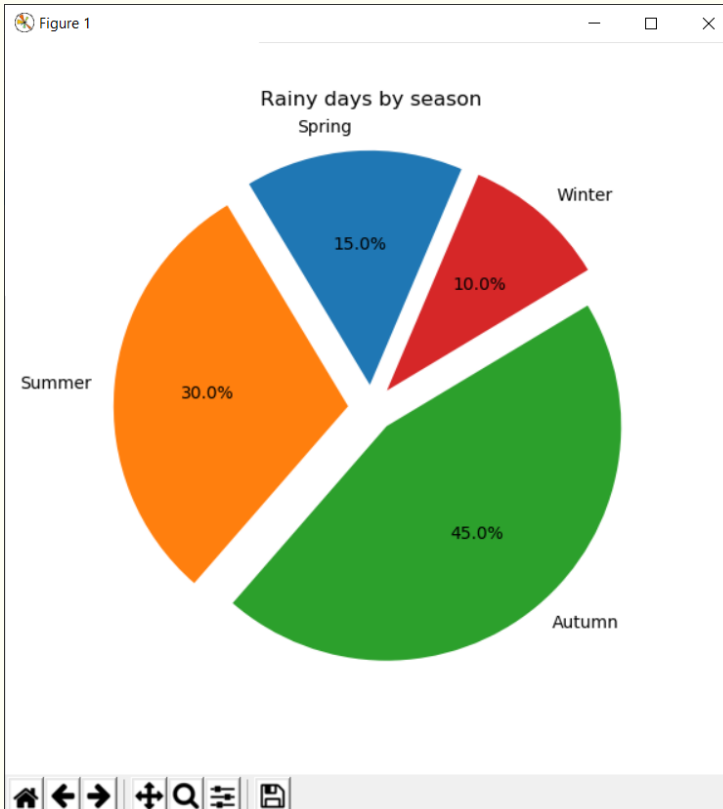
```



- Якщо природа набору даних така, що відхилення (errors) не однакові в додатному та від'ємному напрямках, можемо задати їх окремо за допомогою асиметричних error bar-ів.
  - Слід задавати xerr або yerr з використанням двоелементного списку: перший список міститиме значення від'ємних відхилень, а другий - додатніх.

Hatch value	Description
/	Diagonal hatching
\	Back diagonal
	Vertical hatching
-	Horizontal
+	Crossed
x	Crossed diagonal
o	Small circle
O	Large circle
.	Dot pattern
*	Star pattern

# Кругова діаграма



```

1  from pylab import *
2  # make a square figure and axes
3  figure(1, figsize=(6,6))
4  ax = axes([0.1, 0.1, 0.8, 0.8])
5  # the slices will be ordered
6  # and plotted counter-clockwise.
7  labels = 'Spring', 'Summer', 'Autumn', 'Winter'
8  # fractions are either x/sum(x) or x if sum(x) <= 1
9  x = [15, 30, 45, 10]
10 # explode must be len(x) sequence or None
11 explode=(0.1, 0.1, 0.1, 0.1)
12 pie(x, explode=explode, labels=labels,
13     autopct='%1.1f%%', startangle=67)
14 title('Rainy days by season')
15 show()

```

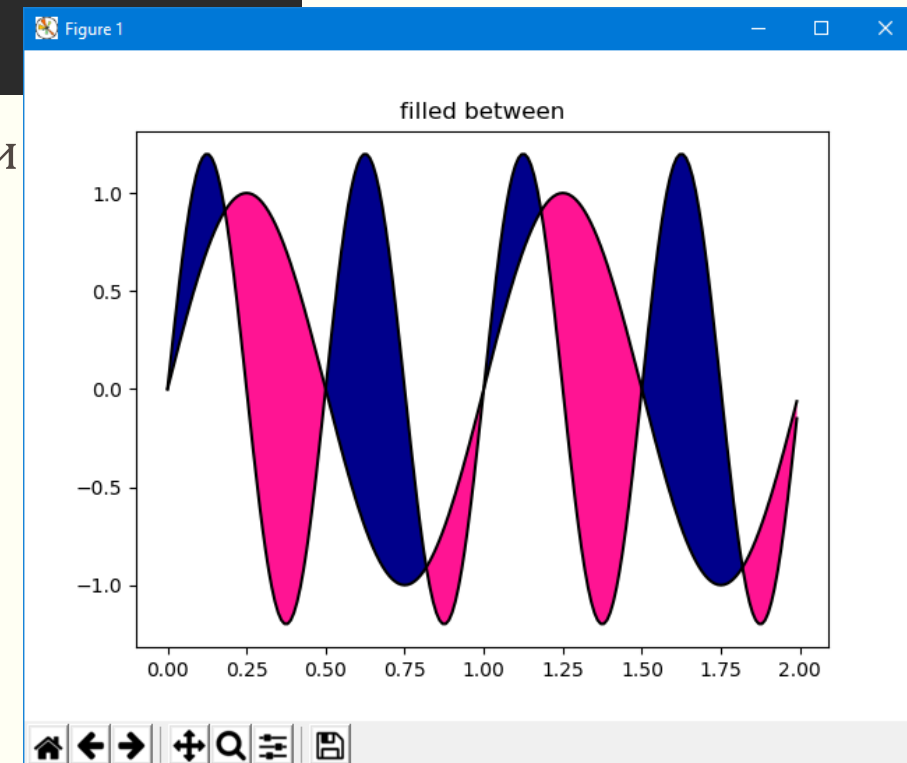


```

1 from matplotlib.pyplot import figure, show, gca
2 import numpy as np
3 x = np.arange(0.0, 2, 0.01)
4 # two different signals are measured
5 y1 = np.sin(2*np.pi*x)
6 y2 = 1.2*np.sin(4*np.pi*x)
7 fig = figure()
8 ax = gca()
9 # plot and fill between y1 and y2 where a logical condition is met
10 ax.plot(x, y1, x, y2, color='black')
11 ax.fill_between(x, y1, y2, where=y2>=y1, facecolor='darkblue', interpolate=True)
12 ax.fill_between(x, y1, y2, where=y2<=y1, facecolor='deeppink', interpolate=True)
13 ax.set_title('filled between')
14 show()

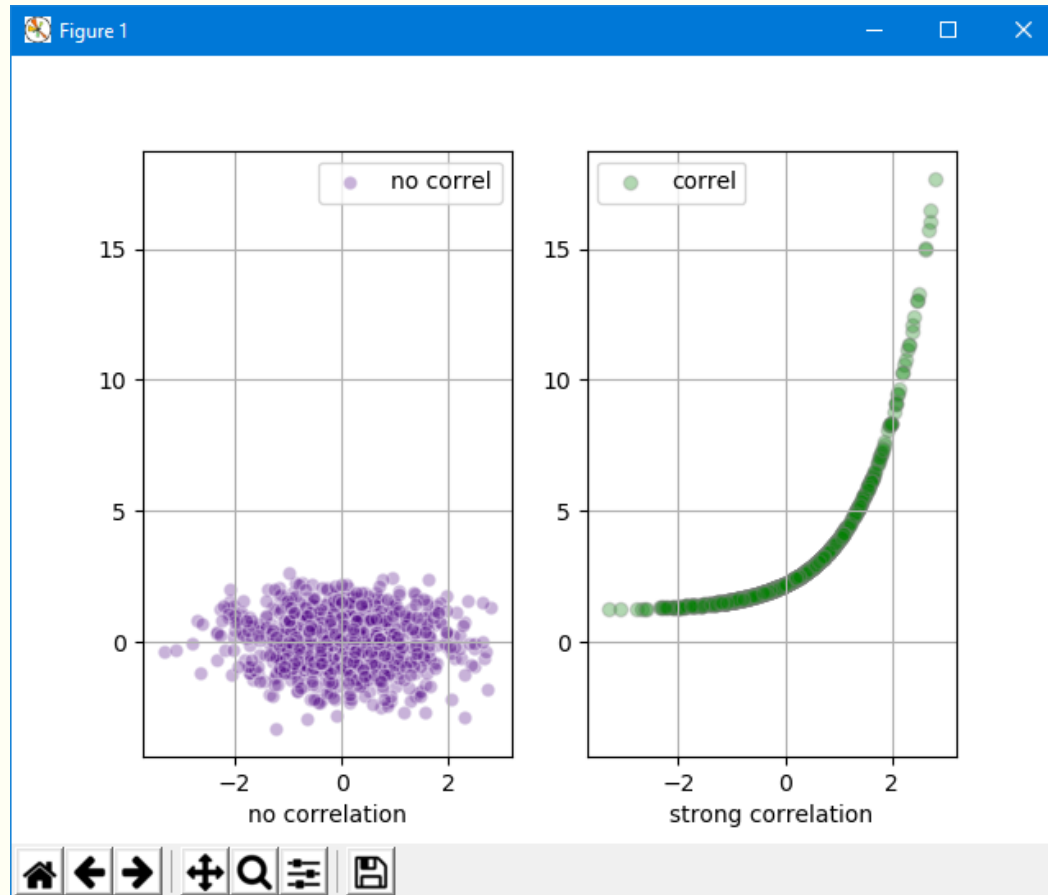
```

- Бібліотека matplotlib дозволяє заповнювати проміжки між кривими - функція fill\_between().



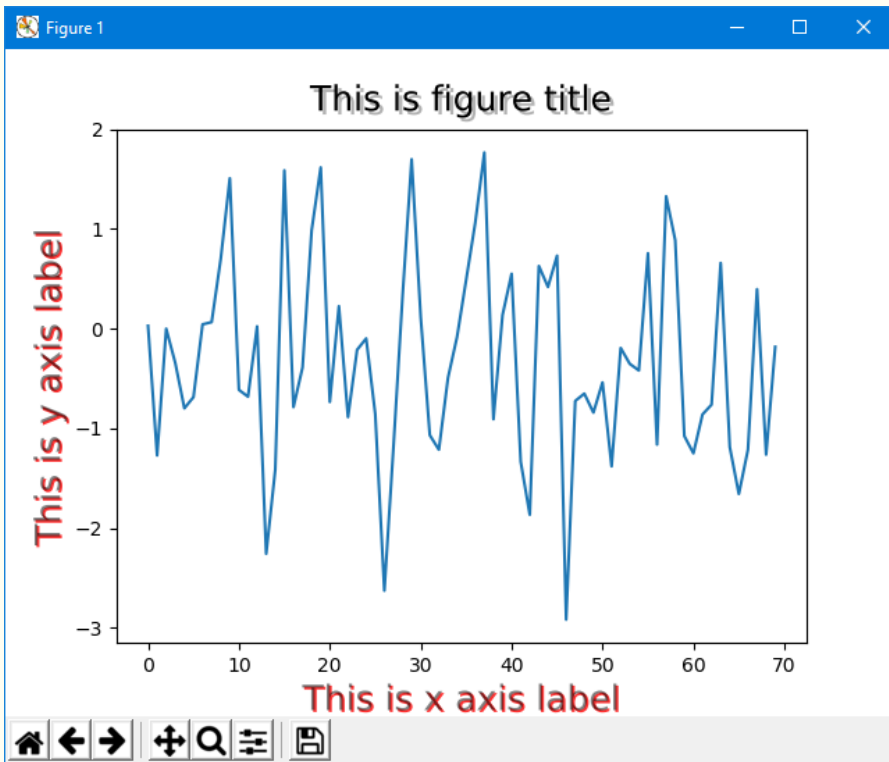
# Малювання scatter plots з кольоровими маркерами

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # generate x values
4 x = np.random.randn(1000)
5 # random measurements, no correlation
6 y1 = np.random.randn(len(x))
7 # strong correlation
8 y2 = 1.2 + np.exp(x)
9 ax1 = plt.subplot(121)
10 plt.scatter(x, y1, color='indigo', alpha=0.3, edgecolors='white',
11            label='no correl')
12 plt.xlabel('no correlation')
13 plt.grid(True)
14 plt.legend()
15 ax2 = plt.subplot(122, sharey=ax1, sharex=ax1)
16 plt.scatter(x, y2, color='green', alpha=0.3, edgecolors='grey',
17            label='correl')
18 plt.xlabel('strong correlation')
19 plt.grid(True)
20 plt.legend()
21 plt.show()
```



- Діаграми розсіювання (Scatter plots) відображають значення з 2 наборів даних.
  - Візуалізація даних виконується як хмара нез'єднаних точок.

# Налаштування прозорості та розміру підписів осей (axis labels)



■ Підписи для осей описують, які дані представлено на рисунку.

- 1. Створимо графік для деяких випадкових чисел.
- 2. Додамо заголовок та підписи осей.
- 3. Додамо налаштування прозорості (alpha).
- 4. Додамо ефекти тіней до заголовку та підписів осей.

```
1 import matplotlib.pyplot as plt
2 from matplotlib import patheffects
3 import numpy as np
4 data = np.random.randn(70)
5 fontsize = 18
6 plt.plot(data)
7 title = "This is figure title"
8 x_label = "This is x axis label"
9 y_label = "This is y axis label"
10 title_text_obj = plt.title(title, fontsize=fontsize, verticalalignment='bottom')
11 title_text_obj.set_path_effects([patheffects.withSimplePatchShadow()])
12 # offset_xy -- set the 'angle' of the shadow
13 # shadow_rgbFace -- set the color of the shadow
14 # patch_alpha -- setup the transparency of the shadow
15 offset_xy = (1, -1)
16 rgbRed = (1.0, 0.0, 0.0)
17 alpha = 0.8
```

## Продовження коду

---

```
18 # customize shadow properties
19 pe = patheffects.withSimplePatchShadow(offset = offset_xy,
20                                         shadow_rgbFace = rgbRed,
21                                         alpha = alpha)
22 # apply them to the xaxis and yaxis labels
23 xlabel_obj = plt.xlabel(x_label, fontsize=fontsize, alpha=0.5)
24 xlabel_obj.set_path_effects([pe])
25 ylabel_obj = plt.ylabel(y_label, fontsize=fontsize, alpha=0.5)
26 ylabel_obj.set_path_effects([pe])
27 plt.show()
```

Після усіх налаштувань інстанціюємо `matplotlib.patheffects.withSimplePatchShadow` та зберігаємо посилання на нього у змінній `pe` з метою повторного використання потім.

- Для накладання тіні потрібно мати доступ до об'єктів-підписів.
- За допомогою `matplotlib.pyplot.xlabel()` повертаємо посилання на об'єкт (`matplotlib.text.Text`), який використаємо при виклику `set_path_effects([pe])`.

# Додавання тіні до chart line

---

- Потрібно використовувати transformation framework з пакету `matplotlib.transforms`.
  - Перетворення (Transformations) знають, як конвертувати задані координати з їх координатної системи для відображення на екрані, і навпаки.

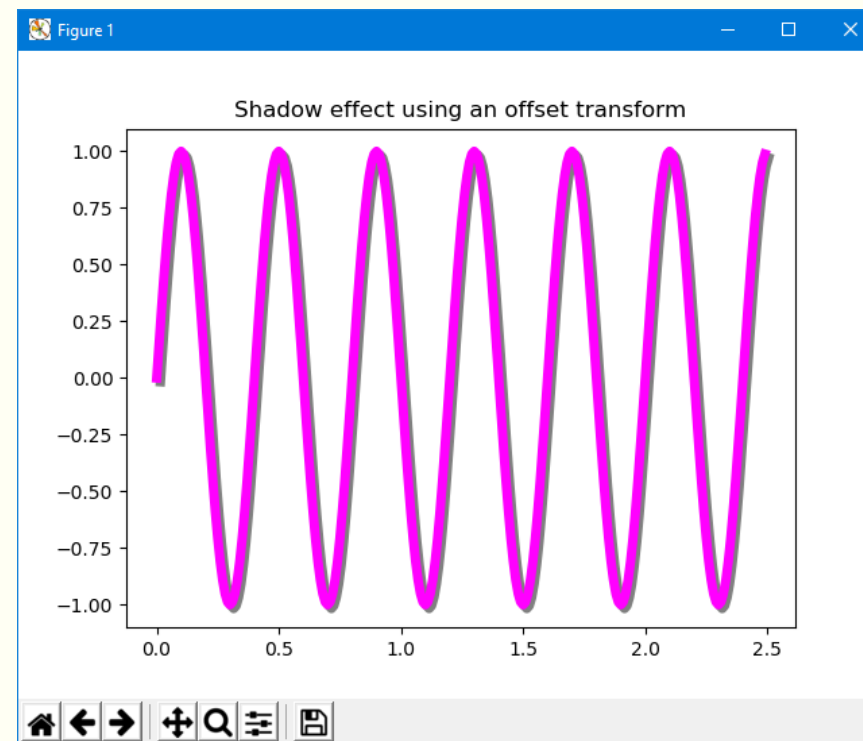
Coordinate system	Transformation object	Description
Data	<code>Axes.transData</code>	Represents the user's data coordinate system.
Axes	<code>Axes.transAxes</code>	Represents the <code>Axes</code> coordinate system, where (0,0) represents the bottom-left end of the axes and (1,1) represents the upper-right end of the axes.
Figure	<code>Figure.transFigure</code>	This is the <code>Figure</code> coordinate system, where (0,0) represents the bottom-left end of the figure and (1,1) represents the upper-right end of the figure.
Display	None	Represents the pixel coordinate system of the user display, where (0,0) represents the bottom-left of the display, and tuple (width, height) represents the upper-right of the display, where width and height are in pixels.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.transforms as transforms
4 def setup(layout=None):
5     assert layout is not None
6     fig = plt.figure()
7     ax = fig.add_subplot(layout)
8     return fig, ax
9
10 def get_signal():
11     t = np.arange(0., 2.5, 0.01)
12     s = np.sin(5 * np.pi * t)
13     return t, s
14
15 def plot_signal(t, s):
16     line, = axes.plot(t, s, linewidth=5, color='magenta')
17     return line,
18
19 def make_shadow(fig, axes, line, t, s):
20     delta = 2 / 72. # how many points to move the shadow
21     offset = transforms.ScaledTranslation(delta, -delta, fig.dpi_scale_trans)
22     offset_transform = axes.transData + offset
23     # We plot the same data, but now using offset transform
24     # zorder -- to render it below the line
25     axes.plot(t, s, linewidth=5, color='gray',
26              transform=offset_transform, zorder=0.5 * line.get_zorder())
27
28 if __name__ == "__main__":
29     fig, axes = setup(111)
30     t, s = get_signal()
31     line, = plot_signal(t, s)
32     make_shadow(fig, axes, line, t, s)
33     axes.set_title('Shadow effect using an offset transform')
34     plt.show()

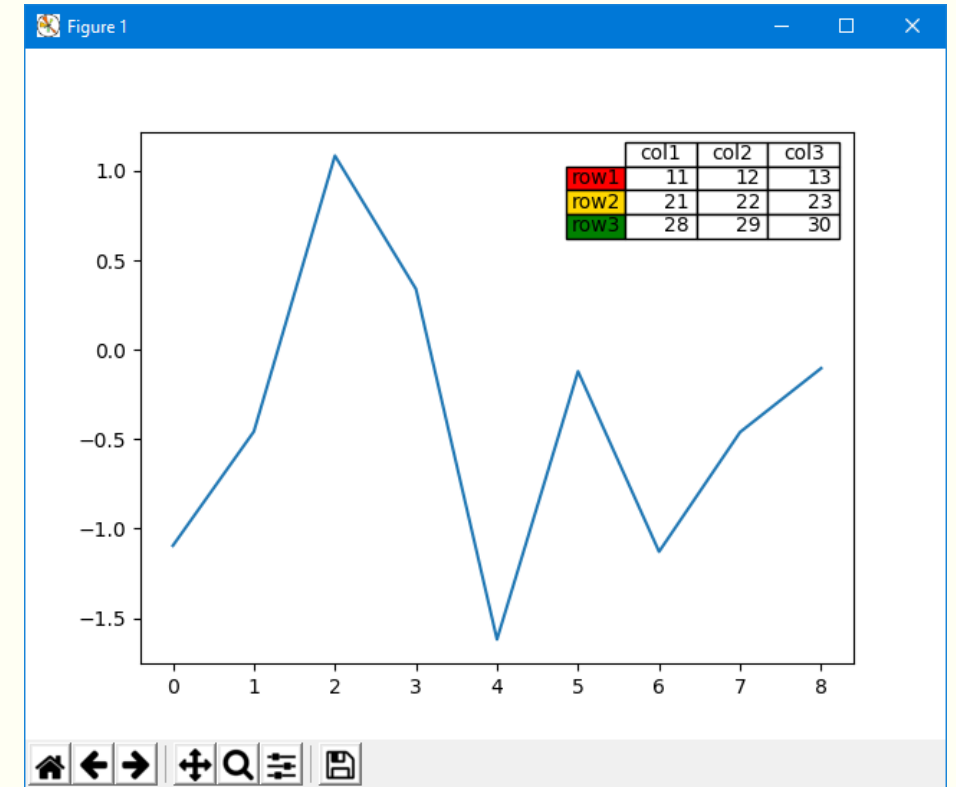
```

- Спочатку створимо рисунок у setup(); потім отримуємо сигнал (генерує дані—синусоїду).
  - Будуємо графік базового сигналу plot\_signal().
  - Далі накладаємо тінь у make\_shadow().
  - Використовуємо ефект зміщення (offset).



# Додавання таблиці до рисунку

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 plt.figure()
4 ax = plt.gca()
5 y = np.random.randn(9)
6 col_labels = ['col1', 'col2', 'col3']
7 row_labels = ['row1', 'row2', 'row3']
8 table_vals = [[11, 12, 13], [21, 22, 23], [28, 29, 30]]
9 row_colors = ['red', 'gold', 'green']
10 my_table = plt.table(cellText=table_vals,
11                     colWidths=[0.1] * 3,
12                     rowLabels=row_labels,
13                     colLabels=col_labels,
14                     rowColours=row_colors,
15                     loc='upper right')
16 plt.plot(y)
17 plt.show()
```



## ■ Базова сигнатура функції:

```
table(cellText=None, cellColours=None,
      cellLoc='right', colWidths=None,
      rowLabels=None, rowColours=None, rowLoc='left',
      colLabels=None, colColours=None, colLoc='center',
      loc='bottom', bbox=None)
```

- Функція створює та повертає екземпляр `matplotlib.table.Table`.
- Клас `Table` дозволяє напряму налаштувати таблицю до її додавання на екземпляр осі за допомогою `add_table()`.



# Використання підграфіків (subplots)

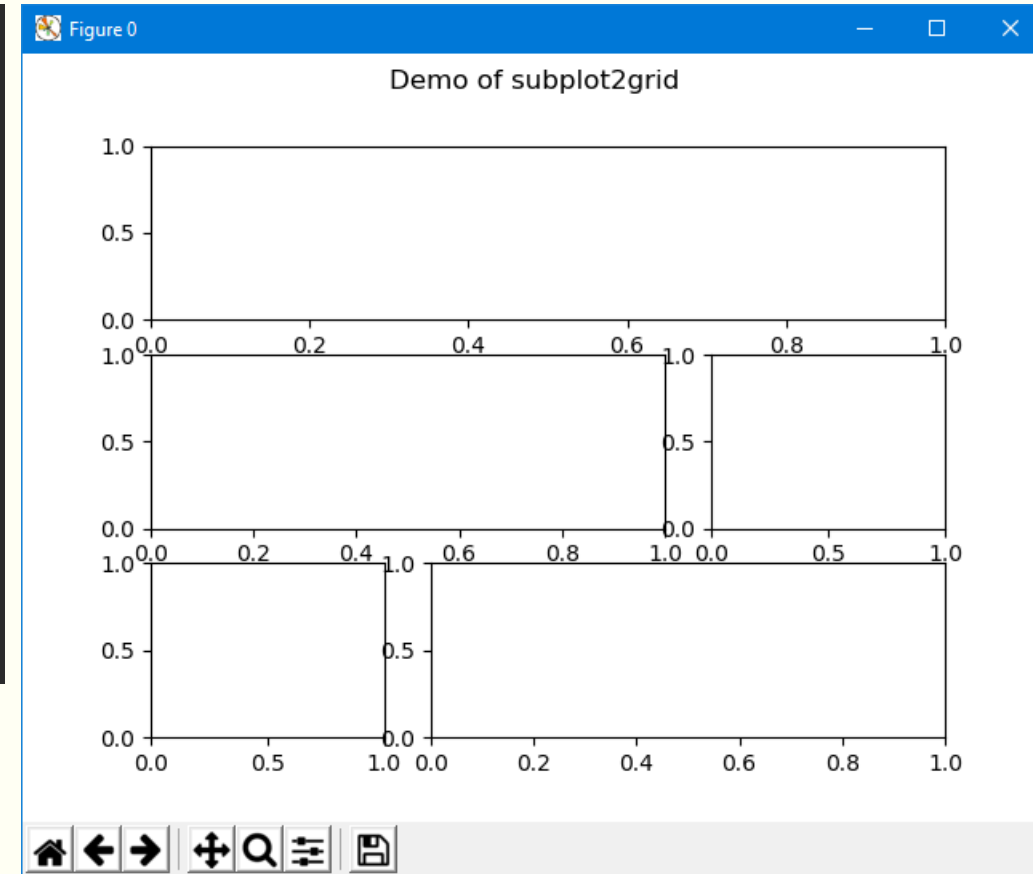
---

- Базовий клас для підграфіків – `matplotlib.axes.SubplotBase`.
  - Підграфіки є екземплярами `matplotlib.axes.Axes`, проте постачають допоміжні методи для генерування та оперування набором `Axes` всередині рисунку.
- Доступний клас `matplotlib.figure.SubplotParams`, який містить всі параметри підграфіка.
  - Одиниці вимірювання нормалізуються по ширині та висоті рисунку.
  - Як уже відомо, якщо не задати власних значень, вони зчитуються з параметрів `rc`.
- Скриптинговий прошарок (scripting layer, `matplotlib.pyplot`) містить кілька допоміжних методів для роботи з підграфіками.
  - `matplotlib.pyplot.subplots` використовується для простого створення поширених макетів підграфіків.
  - Можемо задати розмір сітки—кількість рядків та стовпців сітки підграфіку.
- `matplotlib.pyplot.subplots_adjust` використовуються для налаштування макету підграфіка.
  - Іменовані аргументи задають координати підграфіків всередині рисунку (`left`, `right`, `bottom`, `top`), нормалізовані до розміру рисунку.
  - Відступи між підграфіками можна задати за допомогою аргументів `wspace` та `hspace`.



# Інша допоміжна функція в matplotlib—subplot2grid()

```
1 import matplotlib.pyplot as plt
2 plt.figure(0)
3 axes1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
4 axes2 = plt.subplot2grid((3, 3), (1, 0), colspan=2)
5 axes3 = plt.subplot2grid((3, 3), (1, 2))
6 axes4 = plt.subplot2grid((3, 3), (2, 0))
7 axes5 = plt.subplot2grid((3, 3), (2, 1), colspan=2)
8 # tidy up tick labels size
9 all_axes = plt.gcf().axes
10 for ax in all_axes:
11     for ticklabel in ax.get_xticklabels() + ax.get_yticklabels():
12         ticklabel.set_fontsize(10)
13
14 plt.suptitle("Demo of subplot2grid")
15 plt.show()
```



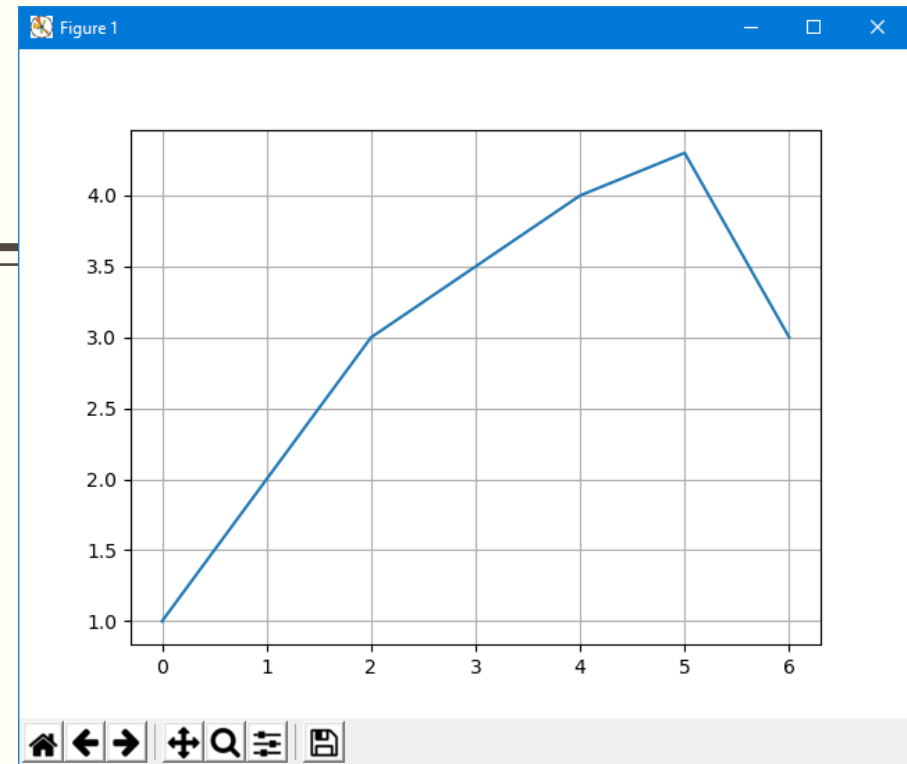
# Альтернативи

---

- `axes = fig.add_subplot(111)`  
`rectangle = axes.patch`  
`rectangle.set_facecolor('blue')`
  - Кожен екземпляр `axes` містить поле `patch`, яке посилається на об'єкт-прямокутник, що представляє фон для екземпляру поточної осі; доступні властивості для оновлення кольору фону, завантаження зображення, додавання водяного знаку тощо.
- Також можливе створення `patch`, а потім його додавання до фону осі:
- `fig = plt.figure()`  
`axes = fig.add_subplot(111)`  
`rect = matplotlib.patches.Rectangle((1,1), width=6, height=12)`  
`axes.add_patch(rect)`  
# потрібно вручну запустити відрисовку рисунку  
`axes.figure.canvas.draw()`

# Кастомізація сіток

```
1 import matplotlib.pyplot as plt
2 plt.figure(1)
3 plt.plot([1, 2, 3, 3.5, 4, 4.3, 3])
4 plt.grid()
5 plt.show()
```



- Для можливості налаштування видимості, частоти відображення та стилю сітки зазвичай використовують `matplotlib.pyplot.grid`.
  - Крім ввімкнення-вимкнення сітки, можна маніпулювати нею мажорними та/або мінорними насічками; hence, the value of function argument which can be 'major', 'minor', or 'both'.
- Усі інші властивості передаються за допомогою kwargs та представляють стандартний набір властивостей, прийнятних для екземпляру `matplotlib.lines.Line2D`, зокрема `color`, `linestyle`, `linewidth`:
  - `ax.grid(color='g', linestyle='--', linewidth=1)`

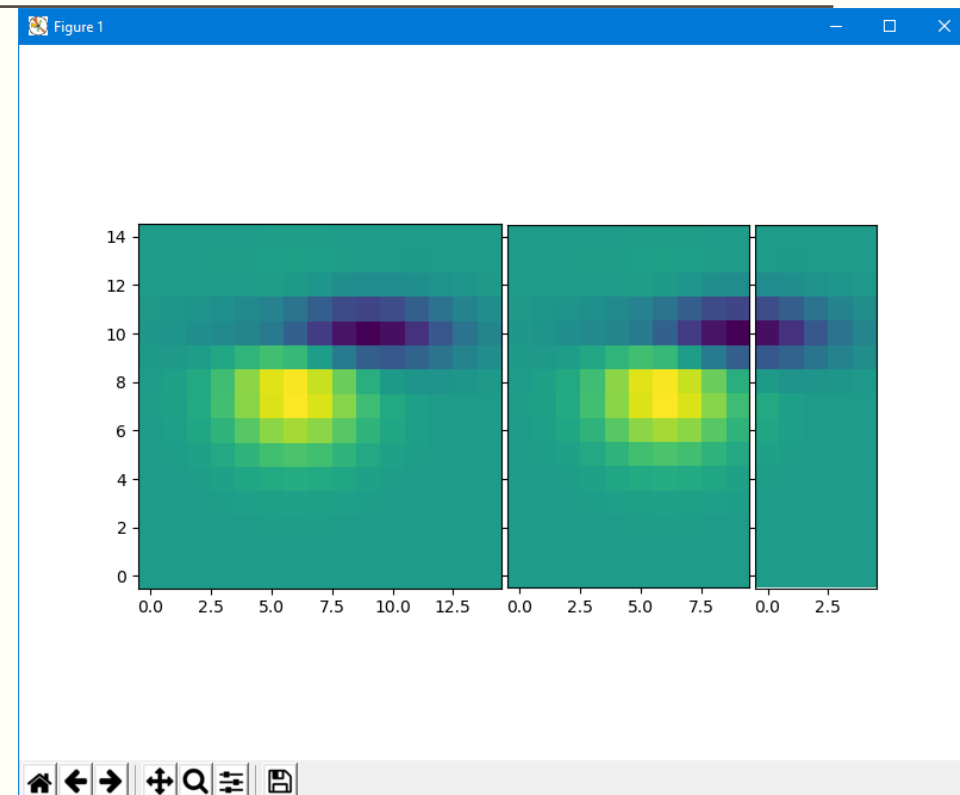
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.axes_grid1 import ImageGrid
4 from matplotlib.cbook import get_sample_data
5
6 def get_demo_image():
7     f = get_sample_data("axes_grid/bivariate_normal.npy", asfileobj=False)
8     # z is a numpy array of 15x15
9     Z = np.load(f)
10    return Z, (-3, 4, -4, 3)
11
12 def get_grid(fig=None, layout=None, nrows_ncols=None):
13     assert fig is not None
14     assert layout is not None
15     assert nrows_ncols is not None
16     grid = ImageGrid(fig, layout, nrows_ncols=nrows_ncols,
17                      axes_pad=0.05, add_all=True, label_mode="L")
18    return grid
```

- У функції `get_demo_image()` завантажуюємо дані з sample-папки з matplotlib.
  - Список `grid` містить axes grid (тут – `ImageGrid`).

```

20 def load_images_to_grid(grid, Z, *images):
21     min, max = Z.min(), Z.max()
22     for i, image in enumerate(images):
23         axes = grid[i]
24         axes.imshow(image, origin="lower", vmin=min, vmax=max,
25                     interpolation="nearest")
26
27 if name == "main":
28     fig = plt.figure(1, (8, 6))
29     grid = get_grid(fig, 111, (1, 3))
30     Z, extent = get_demo_image()
31     # Slice image
32     image1 = Z
33     image2 = Z[:, :10]
34     image3 = Z[:, 10:]
35     load_images_to_grid(grid, Z, image1, image2, image3)
36     plt.draw()
37     plt.show()

```



- Змінні `image1`, `image2` та `image3` містять розрізані (sliced) дані з `Z`, які були розбиті по багатьох осях у списку `grid`.
  - Обходячи циклом всі ґріди, будуємо графік даних з `im1`, `im2` та `im3` за допомогою стандартного виклику `imshow()`, поки `matplotlib` забезпечує все, що стосується рендерингу та вирівнювання.

# Створення контурних графіків

---

- Контурний графік відображає ізолінії – криві, в яких двопараметрична функція має однакове значення.
  - Контури представляються as a contour plot of matrix  $Z$ , де  $Z$  інтерпретується як висота над  $XY$  площиною.
- Проблема контурних графіків – їх представлення без позначення ізоліній досить даремне, оскільки неможливо декодувати висоту точок, знайти локальні мінімуми тощо.
  - Позначення (labeling) ізоліній можна здійснити або за допомогою написів (`clabel()`), або карт кольорів (`colormaps`).
- Інший ризик роботи з контурними графіками – вибір кількості ізоліній для побудови.
  - При надто великій кількості графік стає надто щільним для розбору, а при надто маленькій – неінформативним та неточним.
  - Функція `contour()` автоматично вгадує, скільки ізоліній будувати, проте можна й задавати власну кількість.
  - У `matplotlib` використовується `matplotlib.pyplot.contour`.
- Доступні подібні функції `contour()` та `contourf()` (контури з заливкою).
  - Будемо демонструвати лише `contour()`, оскільки більшість з цього застосовна і до `contourf()`.
  - Вони працюють майже з однаковими аргументами.

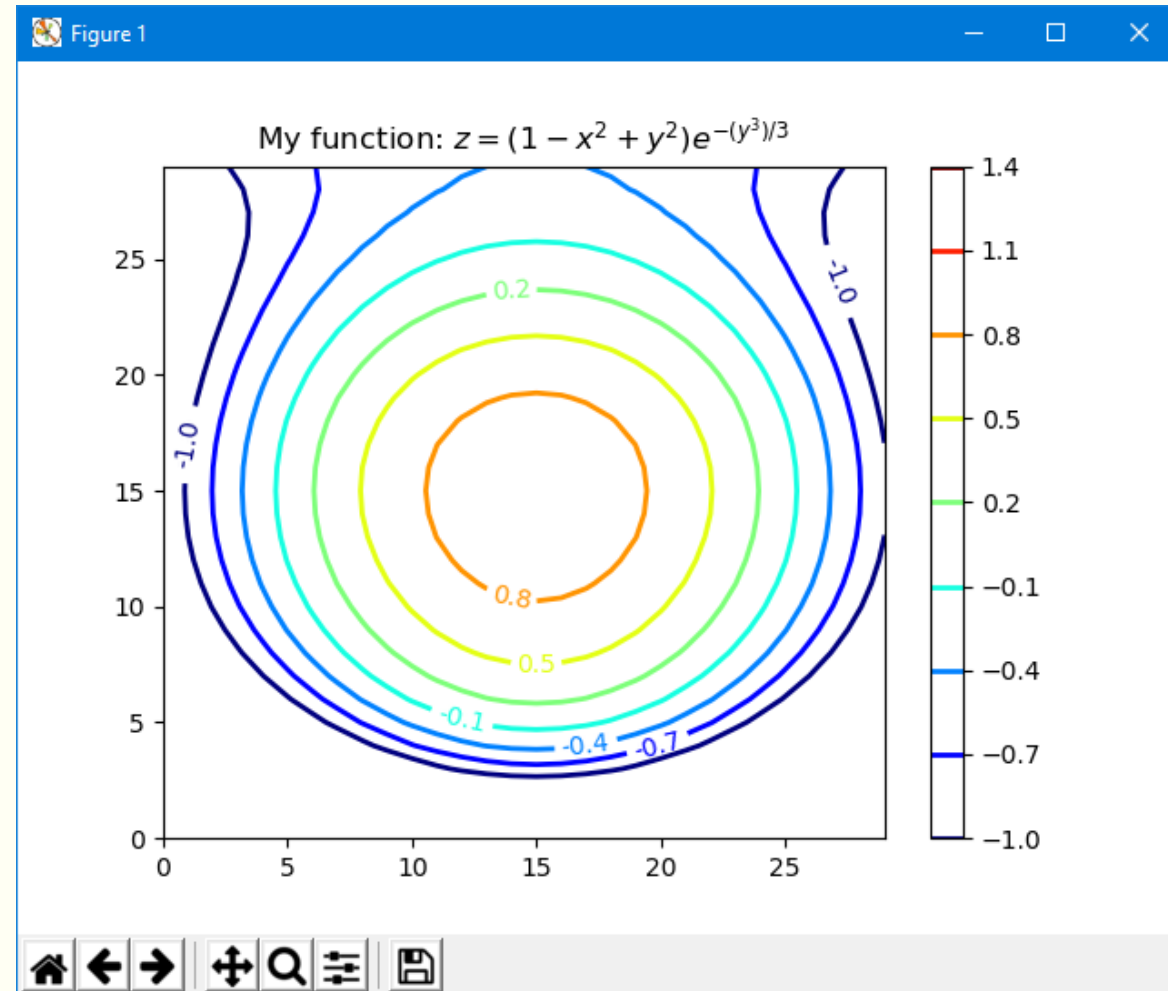
## Функція `contour()` може мати різні сигнатури виклику

Call signature	Description
<code>contour(Z)</code>	Plots the contour of <code>Z</code> (array). The level values are chosen automatically.
<code>contour(X, Y, Z)</code>	Plots the contour of <code>X</code> , <code>Y</code> , and <code>Z</code> . The arrays <code>X</code> and <code>Y</code> are (x, y) surface coordinates.
<code>contour(Z, N)</code>	Plots the contour of <code>Z</code> , where the number of levels is defined with <code>N</code> . The level values are automatically chosen.
<code>contour(X, Y, Z, N)</code>	
<code>contour(Z, V)</code>	Plots the contour lines with levels at the values specified in <code>V</code> .
<code>contour(X, Y, Z, V)</code>	
<code>contourf(..., V)</code>	Fills the <code>len(V) - 1</code> regions between the level values in sequence <code>V</code> .
<code>contour(Z, **kwargs)</code>	Uses keyword arguments to control common line properties (colors, line width, origin, color map, and so on).

```

1 import numpy as np
2 import matplotlib as mpl
3 import matplotlib.pyplot as plt
4
5 def process_signals(x, y):
6     return (1 - (x**2 + y**2)) * np.exp(-y**3 / 3)
7
8 x = np.arange(-1.5, 1.5, 0.1)
9 y = np.arange(-1.5, 1.5, 0.1)
10 # Make grids of points
11 X, Y = np.meshgrid(x, y)
12 Z = process_signals(X, Y)
13 # Number of isolines
14 N = np.arange(-1, 1.5, 0.3)
15 # adding the Contour lines with labels
16 CS = plt.contour(Z, N, linewidths=2, cmap=mpl.cm.jet)
17 plt.clabel(CS, inline=True, fmt='%1.1f', fontsize=10)
18 plt.colorbar(CS)
19 plt.title('My function: $z=(1-x^2+y^2) e^{-y^3/3}$')
20 plt.show()

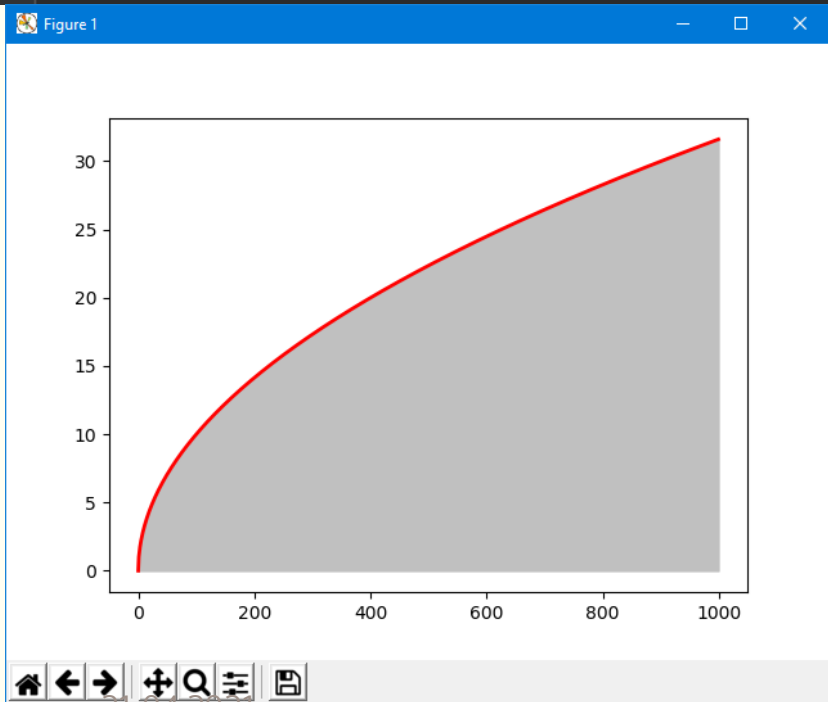
```





# Заливка under-plot області

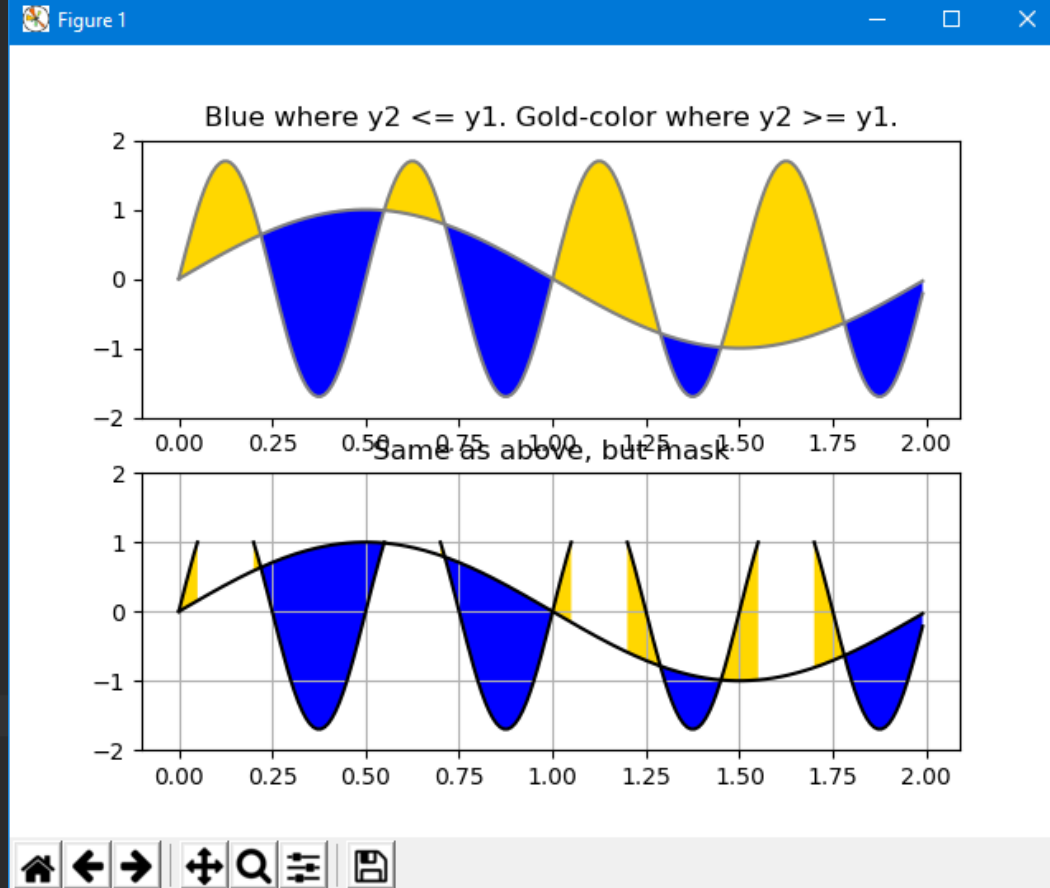
```
1 import matplotlib.pyplot as plt
2 from math import sqrt
3 t = range(1000)
4 y = [sqrt(i) for i in t]
5 plt.plot(t, y, color='red', lw=2)
6 plt.fill_between(t, y, color='silver')
7 plt.show()
```



- Базовий спосіб відрисовки залитого багатокутника в matplotlib – застосувати `matplotlib.pyplot.fill()`.
  - Дана функція приймає подібні до `matplotlib.pyplot.plot()` аргументи—багато пар `x` та `y` та інші властивості `Line2D`.
  - Повертає список екземплярів `Patch`, які було додано.
- matplotlib постачає кілька функцій, які допомагають будувати залиті фігури, наряду з функціями для побудови залитих багатокутників, зокрема `histogram()`.
- Також доступні функції `matplotlib.pyplot.fill_between()` та `matplotlib.pyplot.fill_betweenx()`.
  - Вони заливають багатокутники між 2 кривими.
  - Основна відмінність між функціями – остання здійснює заливку між значеннями по вісі `OX`, а перша – по вісі `OY`.
  - Функція `fill_between()` приймає аргументи `x`, `y1` та `y2`—масиви значень по осях.

# Інший приклад: більше умов для функції заливки

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0.0, 2, 0.01)
y1 = np.sin(np.pi*x)
y2 = 1.7*np.sin(4*np.pi*x)
fig = plt.figure()
axes1 = fig.add_subplot(211)
axes1.plot(x, y1, x, y2, color='grey')
axes1.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue', interpolate=True)
axes1.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold', interpolate=True)
axes1.set_title('Blue where y2 <= y1. Gold-color where y2 >= y1.')
axes1.set_ylim(-2,2)
# Mask values in y2 with value greater than 1.0
y2 = np.ma.masked_greater(y2, 1.0)
axes2 = fig.add_subplot(212, sharex=axes1)
axes2.plot(x, y1, x, y2, color='black')
axes2.fill_between(x, y1, y2, where=y2<=y1, facecolor='blue', interpolate=True)
axes2.fill_between(x, y1, y2, where=y2>=y1, facecolor='gold', interpolate=True)
axes2.set_title('Same as above, but mask')
axes2.set_ylim(-2,2)
axes2.grid('on')
plt.show()
```

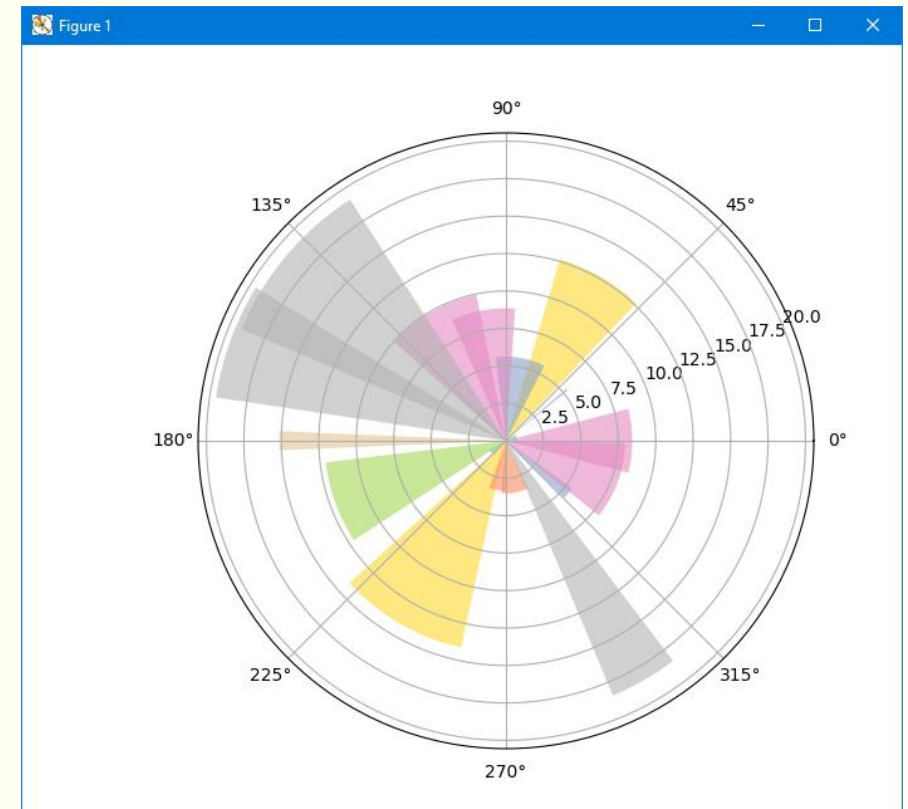


# Відрисовка в полярних координатах

- У полярній системі координат точка описується радіус-вектором  $r$  та кутом  $\theta$ .
  - Бібліотека `matplotlib` виконистовує градуси для позначення величини кутів.
  - Для відрисовки використовується функція `polar()`, яка приймає 2 масиви параметрів,  $\theta$  та  $r$ .
  - Також приймаються інші аргументи, як у функції `plot()`.

```
1 import numpy as np
2 import matplotlib.cm as cm
3 import matplotlib.pyplot as plt
4
5 figsize = 7
6 colormap = lambda r: cm.Set2(r / 20.)
7 N = 18 # number of bars
8 fig = plt.figure(figsize=(figsize, figsize))
9 ax = fig.add_axes([0.2, 0.2, 0.7, 0.7], polar=True)
10 theta = np.arange(0.0, 2*np.pi, 2*np.pi/N)
11 radii = 20*np.random.rand(N)
12 width = np.pi/4*np.random.rand(N)
13 bars = ax.bar(theta, radii, width=width, bottom=0.0)
14 for r, bar in zip(radii, bars):
15     bar.set_facecolor(colormap(r))
16     bar.set_alpha(0.6)
17 plt.show()
```

2021



# Відрисовка в полярних координатах

---

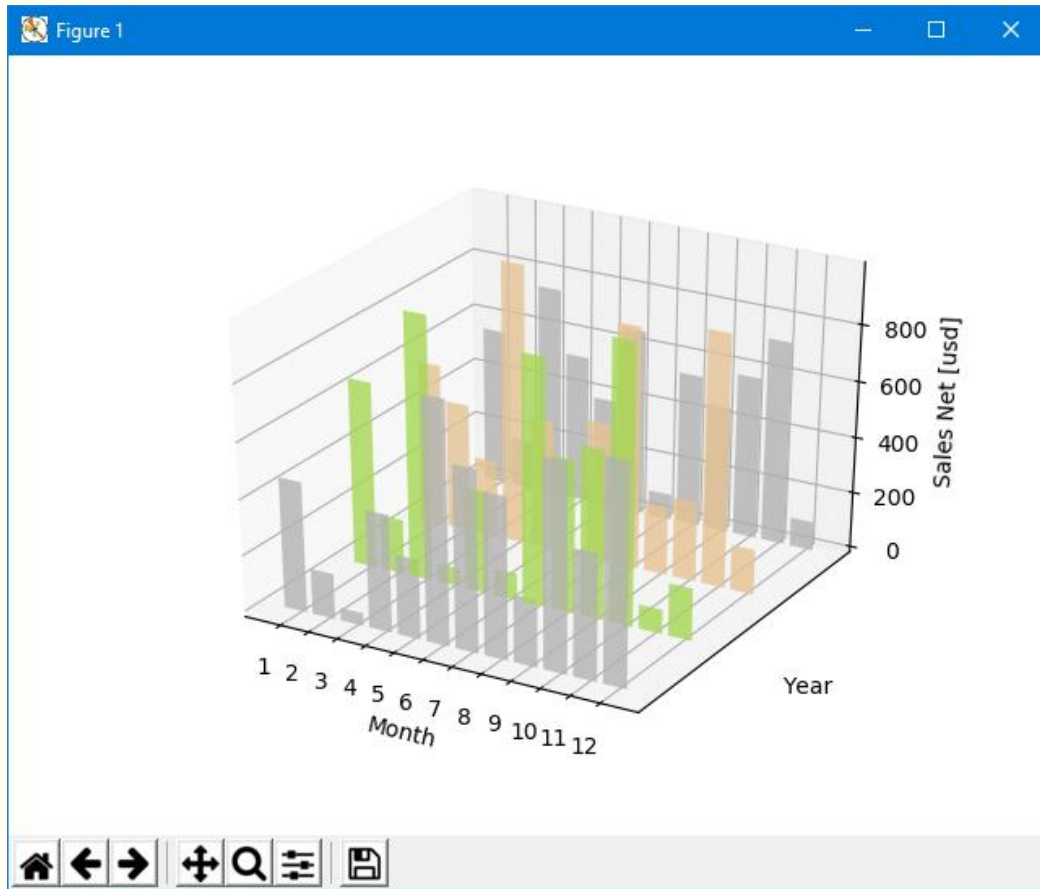
- Також необхідно вказати `matplotlib`, що осі повинні бути в полярній системі координат.
  - Виконується за допомогою аргументу `polar=True` для функцій `add_axes()` або `add_subplot()`.
  - Додатково для визначення інших властивостей рисунку, зокрема радіальної сітки чи кутів, потрібно використовувати `matplotlib.pyplot.rgrids()`, щоб перемикає видимість сітки чи накладати написи.
  - Аналогічно застосовуємо `matplotlib.pyplot.thetagrid()`, щоб налаштувати кутові насічки та написи.
- Потім генеруємо випадкові значення для набору кутів (`theta`) та набору довжин радіус-векторів (`radii`).
  - Оскільки ми рисували стовпці, також було потрібно згенерувати набір ширин кожного стовпчика.
  - Оскільки `matplotlib.axes.bar` приймає масив значень, не потрібно обходити циклом даний згенерований датасет; необхідно лише викликати `bar()` один раз з усіма переданими в нього аргументами.

# Створення тривимірних стовпців (bars)

---

- Існують різні розширення (extensions), які дозволяють будувати візуалізації на географічних картах, тісніше інтегруватись з Excel чи рисувати в 3D.
  - У контексті matplotlib такі розширення називають тулкітами (toolkits).
  - Тулкіт – це колекція функцій, сфокусованих на одній функціональності, наприклад, у 3D-рисунні.
  - Популярні тулкіти: Basemap, GTK Tools, Excel Tools, Natgrid, AxesGrid, mplot3d.
- Для прикладу розглянемо mplot3d, який забезпечує базову 3D-рисовку.
  - Підтримуються графіки виду scatter, surf, line та mesh.
  - Хоч це не найкраща бібліотека 3D-рисуння, вона йде в комплекті з matplotlib.

# Загалом, все ще потрібно створювати рисунок (figure) та додавати на нього потрібні осі



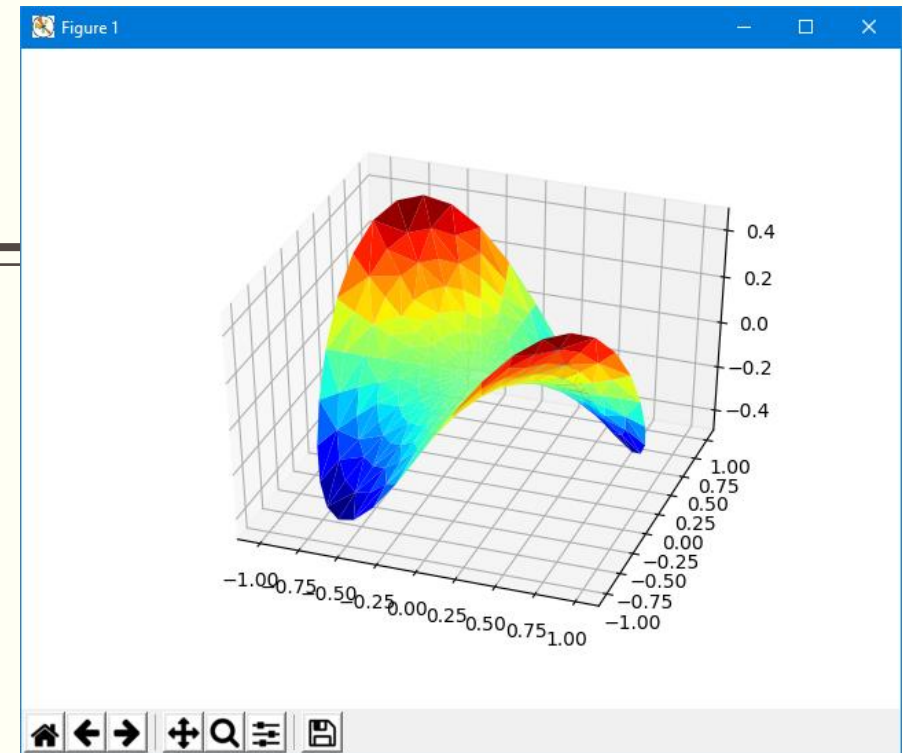
Відмінність: ми задаємо 3D-проекцію для графіка, а також додаємо осі типу `Axes3D`.

- Після цього можна використовувати практично ті ж функції для рисування графіків, проте з іншими аргументами.
- Наприклад, функція `mpl_toolkits.mplot3d.Axes3D.plot()` задає аргументи:
  - `xs` та `ys`: координати для осей `x` та `y`
  - `zs`: значення для вісі `z`. Одне значення для всіх точок або одне значення для кожної точки.
  - `zdir`: обирає вимір `z`-вісі (зазвичай це `zs`, проте може бути `xs` чи `ys`)
- Решта аргументів переносяться напряму в `matplotlib.axes.Axes.plot`.

```

1 import random
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 import matplotlib.dates as mdates
6 from mpl_toolkits.mplot3d import Axes3D
7
8 mpl.rcParams['font.size'] = 10
9 fig = plt.figure()
10 ax = fig.add_subplot(111, projection='3d')
11 for z in [2011, 2012, 2013, 2014]:
12     xs = range(1, 13)
13     ys = 1000 * np.random.rand(12)
14     color = plt.cm.Set2(random.choice(range(plt.cm.Set2.N)))
15     ax.bar(xs, ys, zs=z, zdir='y', color=color, alpha=0.8)
16
17 ax.xaxis.set_major_locator(mpl.ticker.FixedLocator(xs))
18 ax.yaxis.set_major_locator(mpl.ticker.FixedLocator(ys))
19 ax.set_xlabel('Month')
20 ax.set_ylabel('Year')
21 ax.set_zlabel('Sales Net [usd]')
22 plt.show()

```



- Також доступні інші способи побудови графіків з 2D matplotlib; наприклад, `scatter()` з подібним до `plot()` інтерфейсом взаємодії, проте з доданим розміром маркера.
  - Також наявні `contour()`, `contourf()` та `bar()`.
  - The new types that are available just in 3D are wireframe, surface, and tri-surface plots.
  - Код буде поверхню (tri-surface plot) для гіперболічного параболоїда



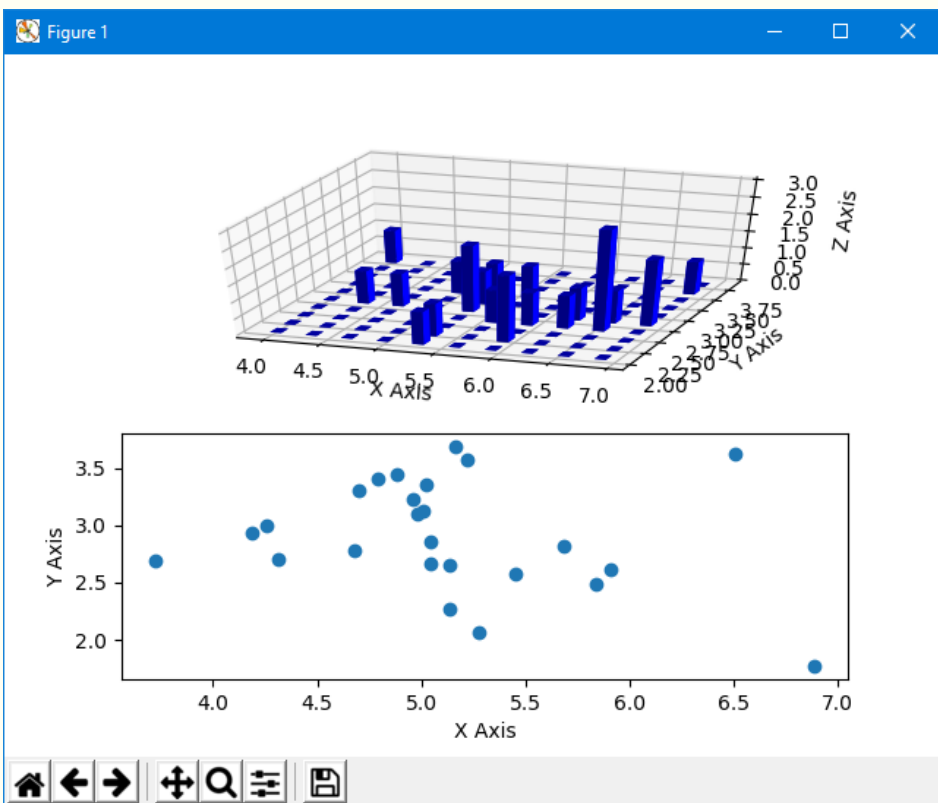
```

1  from mpl_toolkits.mplot3d import Axes3D
2  from matplotlib import cm
3  import matplotlib.pyplot as plt
4  import numpy as np
5  n_angles = 36
6  n_radii = 8
7  # An array of radii
8  # Does not include radius r=0, this is to eliminate duplicate points
9  radii = np.linspace(0.125, 1.0, n_radii)
10 # An array of angles
11 angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)
12 # Repeat all angles for each radius
13 angles = np.repeat(angles[..., np.newaxis], n_radii, axis=1)
14 # Convert polar (radii, angles) coords to cartesian (x, y) coords
15 # (0, 0) is added here. There are no duplicate points in the (x, y) plane
16 x = np.append(0, (radii*np.cos(angles)).flatten())
17 y = np.append(0, (radii*np.sin(angles)).flatten())
18 # Pringle surface
19 z = np.sin(-x*y)
20 fig = plt.figure()
21 ax = fig.gca(projection='3d')
22 ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
23 plt.show()

```



# Створення 3D-гістограм



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib as mpl
4 from mpl_toolkits.mplot3d import Axes3D
5
6 mpl.rcParams['font.size'] = 10
7 samples = 25
8 x = np.random.normal(5, 1, samples)
9 y = np.random.normal(3, .5, samples)
10 fig = plt.figure()
11 ax1 = fig.add_subplot(211, projection='3d')
12 # compute two-dimensional histogram
13 hist, xedges, yedges = np.histogram2d(x, y, bins=10)
14 # compute location of the x,y bar positions
15 elements = (len(xedges) - 1) * (len(yedges) - 1)
16 xpos, ypos = np.meshgrid(xedges[:-1]+.25, yedges[:-1]+.25)
17 xpos = xpos.flatten()
18 ypos = ypos.flatten()
19 zpos = np.zeros(elements)
20 # make every bar the same width in base
21 dx = .1 * np.ones_like(zpos)
22 dy = dx.copy()
```

## ■ Для даного прикладу:

- 1. Використовуємо NumPy для обчислення гістограми двох змінних.
- 2. Генеруємо x та y з нормальних розподілів, проте з різними параметрами.
- 3. Будуємо діаграму розсіювання того ж датасету, щоб продемонструвати, які відрізняється діаграма від 3D-гістограми.

```

24     # this defines the height of the bar
25     dz = hist.flatten()
26     ax1.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', alpha=0.4)
27     ax1.set_xlabel('X Axis')
28     ax1.set_ylabel('Y Axis')
29     ax1.set_zlabel('Z Axis')
30     # plot the same x,y correlation in scatter plot
31     # for comparison
32     ax2 = fig.add_subplot(212)
33     ax2.scatter(x, y)
34     ax2.set_xlabel('X Axis')
35     ax2.set_ylabel('Y Axis')
36     plt.show()

```

- Готуємо гістограму за допомогою `np.histogram2d()`, яка повертає our histogram (`hist`) and the `x` and `y` bin edges.
  - Because for the `bar3d` function we need coordinates in `x, y` space, so we need to compute the common matrix coordinates, and for that we use `np.meshgrid` that combines the `x` and `y` positional vectors into the 2D space grid (matrix). We can use this to plot bars in the `xy` plane locations.
  - The variables `dx` and `dy` represent width of the base of each bar and we want to make this constant, hence we give it a 0.1 point value for every position in the `xy` plane.
  - The value in the `z` axis (`dz`) is actually our computer histogram (in variable `hist`) that represents the count of common `x` and `y` samples at a particular bin.



# ДЯКУЮ ЗА УВАГУ!

Зустрінемося на навчальній практиці!

# Анімації в matplotlib

---

- A framework for animation is added to the standard matplotlib from version 1.1 and its main class is `matplotlib.animation.Animation`.
  - This class is a base class; it is to be subclassed for specific behavior as is the case with the already provided classes: `TimedAnimation`, `ArtistAnimation`, and `FuncAnimation`

Class name (parent class)	Description
<code>Animation(object)</code>	This class wraps the creation of an animation using matplotlib. It is only a base class that should be subclassed to provide the needed behavior.
<code>TimedAnimation(Animation)</code>	This animation subclass supports time-based animation, and drawing a new frame every interval* milliseconds.
<code>ArtistAnimation(TimedAnimation)</code>	Before calling this function, all plotting should have taken place and the relevant artists saved.
<code>FuncAnimation(TimedAnimation)</code>	This makes an animation by repeatedly calling a function, passing in (optional) arguments.