


Matplotlib

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

Matplotlib — Visualization with Python

 <https://matplotlib.org/>

Intro

Matplotlib - це бібліотека графіків низького рівня в Python, яка служить утилітою візуалізації.

Matplotlib Pyplot є модулем бібліотеки Matplotlib, який надає функціональність для створення графіків та візуалізації даних у Python.

Для початку роботи з Matplotlib Pyplot, спочатку потрібно імпортувати модуль. Зазвичай це робиться так:

Після імпортування модуля `matplotlib.pyplot` можна створювати графіки та додавати до них дані за допомогою функцій та методів, доступних у цьому модулі.

Основні функції Matplotlib Pyplot включають:

1. `plt.plot(x, y, [fmt], [kwargs])`: Створює лінійний графік на основі набору точок `x` та `y`. Можна вказати формат (`fmt`) для відображення стилю лінії та кольору.
2. `plt.scatter(x, y, [fmt], [kwargs])`: Створює точковий графік, де кожна точка задається парами координат `x` та `y`. Також можна вказати формат (`fmt`) для відображення стилю точок.
3. `plt.bar(x, height, [width], [align], [kwargs])`: Створює стовпчикову діаграму на основі набору значень `height` для відповідних категорій `x`.
4. `plt.pie(x, [labels], [colors], [startangle], [shadow], [explode], [autopct], [kwargs])`: Створює кругову діаграму на основі набору значень `x`. Можна надати мітки (`labels`), кольори (`colors`), кут початку (`startangle`) та інші параметри.

5. `plt.xlabel(text)`, `plt.ylabel(text)`: Встановлюють підписи для вісей `x` та `y` відповідно.
6. `plt.title(text)`: Встановлює заголовок графіку.
7. `plt.legend()`: Відображає легенду графіку, яка пояснює значення або типи ліній, точок або стовпчиків.

Це лише декілька основних функцій Matplotlib Pyplot. Бібліотека надає багато інших можливостей для настройки графіків, таких як зміна кольорів, стилів ліній, додавання тексту та багато іншого.

Нижче наведений приклад коду, який демонструє створення простого лінійного графіку з використанням Matplotlib Pyplot:

```
import matplotlib.pyplot as plt

# Дані для графіку
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Створення графіку
plt.plot(x, y, 'ro-')

# Настройка вісей та заголовка
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Графік функції Y = X^2')

# Відображення графіку
plt.show()
```

Цей код створює графік квадратичної функції $Y = X^2$ з використанням червоних кружочків (`'ro-'`).

Plottin

Matplotlib Plotting (Малювання з використанням Matplotlib) є одним з ключових функціональних компонентів бібліотеки Matplotlib, яка дозволяє створювати різноманітні типи графіків та візуалізаційних елементів у Python.

Для початку роботи з Matplotlib Plotting, спочатку потрібно імпортувати модуль `matplotlib.pyplot`. Зазвичай це робиться так:

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

Після імпортування модуля, можна використовувати різні функції та методи, доступні у `plt`, для створення графіків та налаштування їх параметрів.

Основні функції Matplotlib Plotting включають:

1. `plt.plot(x, y, [fmt], [kwargs])`: Створює лінійний графік на основі набору точок `x` та `y`. Опціонально можна вказати формат (`fmt`) для відображення стилю лінії та кольору.
2. `plt.scatter(x, y, [fmt], [kwargs])`: Створює точковий графік, де кожна точка задається парами координат `x` та `y`. Опціонально можна вказати формат (`fmt`) для відображення стилю точок.
3. `plt.bar(x, height, [width], [align], [kwargs])`: Створює стовпчикову діаграму на основі набору значень `height` для відповідних категорій `x`.
4. `plt.pie(x, [labels], [colors], [startangle], [shadow], [explode], [autopct], [kwargs])`: Створює кругову діаграму на основі набору значень `x`. Опціонально можна вказати мітки (`labels`), кольори (`colors`), кут початку (`startangle`) та інші параметри.
5. `plt.xlabel(text)`, `plt.ylabel(text)`: Встановлюють підписи для вісей `x` та `y` відповідно.
6. `plt.title(text)`: Встановлює заголовок графіку.
7. `plt.legend()`: Відображає легенду графіку, яка пояснює значення або типи ліній, точок або стовпців.

Це лише к

ілька основних функцій Matplotlib Plotting. Бібліотека надає безліч інших можливостей для настройки графіків, таких як зміна кольорів, стилів ліній, додавання тексту, настройка шрифтів та багато іншого.

Важливо пам'ятати, що після налаштування параметрів графіка, потрібно викликати `plt.show()` для відображення самого графіка.

Ось приклад коду, який демонструє створення лінійного графіка з використанням Matplotlib Plotting:

```
import matplotlib.pyplot as plt

# Дані для графіку
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Створення лінійного графіка
plt.plot(x, y)

# Налаштування вісей та заголовка
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Графік функції  $Y = X^2$ ')

# Відображення графіка
plt.show()
```

У цьому прикладі ми спочатку імпортуємо модуль `matplotlib.pyplot` як `plt`. Потім ми визначаємо дані `x` і `y` для графіка.

За допомогою функції `plt.plot(x, y)` ми створюємо лінійний графік на основі цих даних.

Далі ми налаштовуємо підписи для вісей `x` та `y` за допомогою функцій `plt.xlabel('X')` та `plt.ylabel('Y')`. Також ми встановлюємо заголовок графіка за допомогою `plt.title('Графік функції $Y = X^2$ ')`.

Нарешті, за допомогою `plt.show()` ми відображаємо графік на екрані.

Цей код створить лінійний графік для функції $Y = X^2$ з використанням наданих даних `x` та `y`.

Markers

Маркери Matplotlib дозволяють позначати окремі точки на графіках. Ось декілька прикладів різних маркерів, які ви можете використовувати:

- `'.'` - крапка
- `'r'` - піксель
- `'o'` - круг

- 'v' - трикутник вниз
- '^' - трикутник вгору
- '<' - трикутник вліво
- '>' - трикутник вправо
- 's' - квадрат
- '+' - плюс
- 'x' - хрестик

Ці маркери можна використовувати у функціях `plt.plot()` та `plt.scatter()`.
Ось приклад коду, де демонструється використання різних маркерів:

```
import matplotlib.pyplot as plt

# Дані для графіків
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Створення графіка з різними маркерами
plt.plot(x, y, marker='o', label='Круг')
plt.plot(x, [i**2 for i in x], marker='s', label='Квадрат')
plt.plot(x, [i**3 for i in x], marker='^', label='Трикутник')

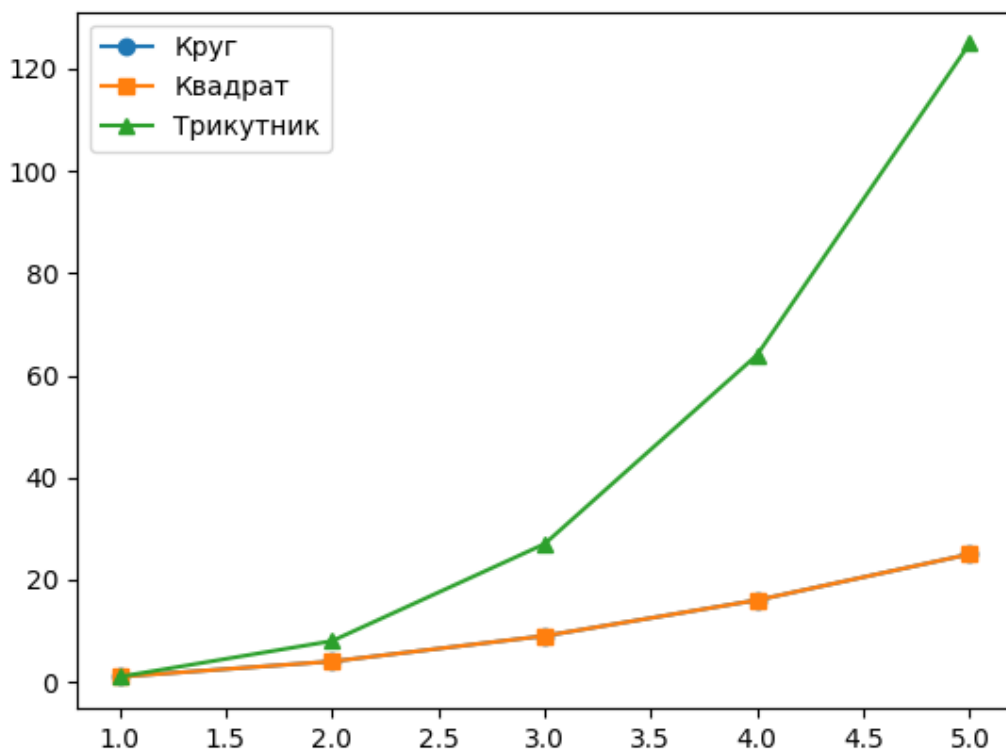
# Налаштування легенди
plt.legend()

# Відображення графіка
plt.show()
```

У цьому прикладі ми використовуємо різні маркери для позначення точок на графіку. Кожен `plt.plot()` має вказане значення параметра `marker`, яке визначає тип маркера.

В результаті ми отримуємо графік з трьома лініями, кожна з яких має свій власний маркер – круг, квадрат та трикутник. Легенда вказує на кожен тип

маркера. Ви можете експериментувати з різними маркерами та їх комбінаціями для отримання бажаного ефекту на графіку.



Matplotlib Line

Модуль Matplotlib надає можливість малювати лінії на графіках. Ви можете використовувати функцію `plt.plot()` для створення лінійних графіків. Ось приклад коду, де демонструється використання ліній:

```
import matplotlib.pyplot as plt

# Дані для графіку
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Створення лінійного графіка
plt.plot(x, y)

# Налаштування вісей та заголовка
```

```
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Лінійний графік')

# Відображення графіка
plt.show()
```

У цьому прикладі ми використовуємо функцію `plt.plot()` для створення лінійного графіка на основі набору даних `x` та `y`. Виклик цієї функції створить лінію, яка проходить через послідовні точки, визначені координатами `(x[0], y[0])`, `(x[1], y[1])`, `(x[2], y[2])` і так далі.

Ви також можете налаштувати зовнішній вигляд лінії, передавши додаткові параметри до функції `plt.plot()`. Наприклад, ви можете вказати колір лінії, товщину лінії та тип лінії. Наприклад:

```
plt.plot(x, y, color='red', linewidth=2, linestyle='--')
```

У цьому прикладі лінія матиме червоний колір, товщиною 2 та пунктирний тип.

Ви можете експериментувати з різними параметрами для досягнення бажаного вигляду лінії на вашому графіку.

В Matplotlib є кілька стилів ліній, які ви можете використовувати при малюванні графіків. Ви можете встановити бажаний стиль лінії, використовуючи параметр `linestyle` у функції `plt.plot()`. Ось декілька прикладів різних стилів ліній:

- `'-'` - неперервна лінія (за замовчуванням)
- `'--'` - штрихова лінія
- `'.'` - пунктирна лінія
- `'-.'` - штрихпунктирна лінія

Ось приклад коду, де показано використання різних стилів ліній:

```
import matplotlib.pyplot as plt

# Дані для графіка
```

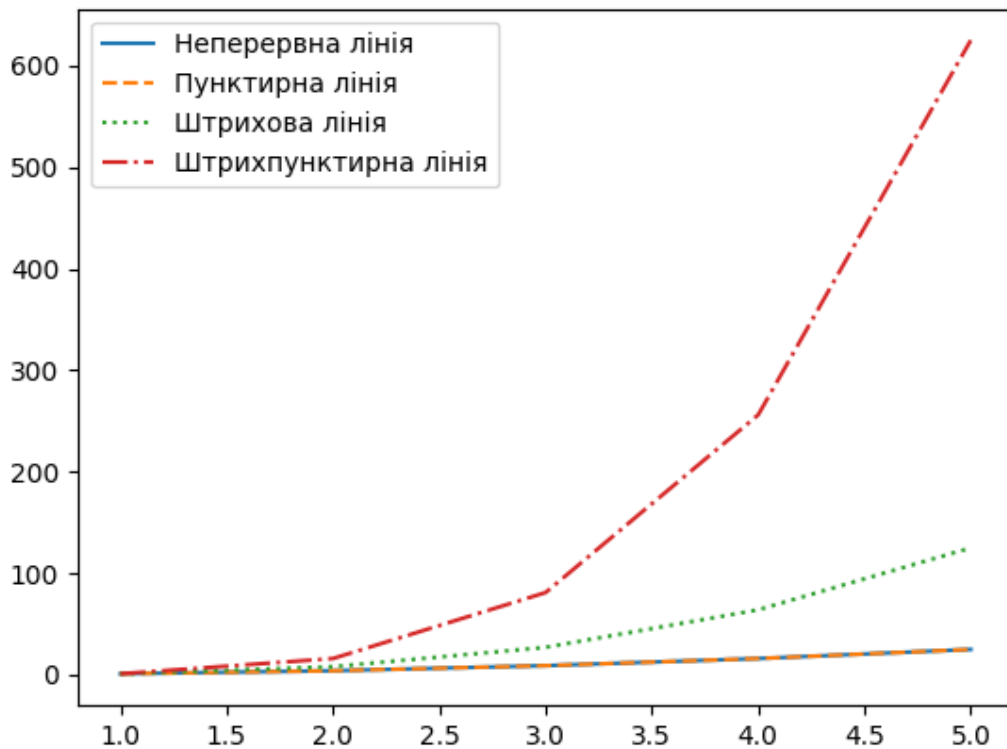
```
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# Створення графіка з різними стилями ліній
plt.plot(x, y, linestyle='-', label='Неперервна лінія')
plt.plot(x, [i**2 for i in x], linestyle='--', label='Пунктирна лінія')
plt.plot(x, [i**3 for i in x], linestyle=':', label='Штрихова лінія')
plt.plot(x, [i**4 for i in x], linestyle='-.', label='Штрихпунктирна лінія')

# Налаштування легенди
plt.legend()

# Відображення графіка
plt.show()
```

У цьому прикладі ми використовуємо параметр `linestyle` у функції `plt.plot()` для встановлення різних стилів ліній. Кожен `plt.plot()` має вказане значення параметра `linestyle`, яке визначає тип лінії.



В результаті ми отримуємо графік з чотирма лініями, кожна з яких має свій власний стиль - неперервну лінію, пунктирну лінію, штрихову лінію та штрихпунктирну лінію. Легенда вказує на кожен тип лінії. Ви можете експериментувати з різними стилями ліній для отримання бажаного ефекту на графіку.

Matplotlib Labels and Title

В Matplotlib ви можете додати підписи до вісей і заголовки до графіка для покращення його зрозумілості і пояснення даних. Для цього використовуються функції `plt.xlabel()`, `plt.ylabel()` та `plt.title()`. Ось приклад коду, де демонструється використання підписів і заголовка:

```
import matplotlib.pyplot as plt

# Дані для графіка
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
```

```
# Створення лінійного графіка
plt.plot(x, y)

# Додавання підписів до вісей
plt.xlabel('Ось X')
plt.ylabel('Ось Y')

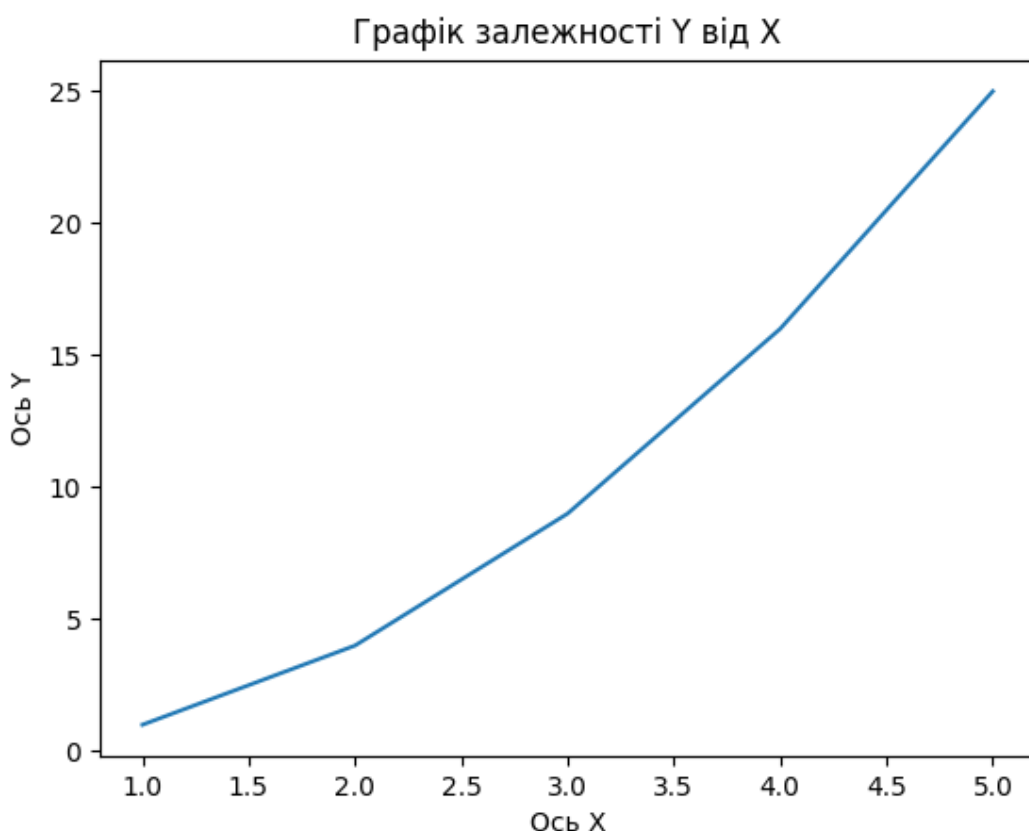
# Додавання заголовка графіка
plt.title('Графік залежності Y від X')

# Відображення графіка
plt.show()
```

У цьому прикладі ми використовуємо функції `plt.xlabel()` і `plt.ylabel()` для додавання підписів до вісей X і Y відповідно. Функція `plt.title()` використовується для додавання заголовка до графіка.

Ви можете замінити рядки `'Ось X'`, `'Ось Y'` і `'Графік залежності Y від X'` на власні назви, які краще відповідають вашим даним і контексту.

Після налаштування підписів і заголовка викличте `plt.show()` для відображення графіка з оновленими підписами і заголовком.



Ви також можете використовувати різні параметри, такі як шрифт, розмір і колір тексту, щоб налаштувати вигляд підписів і заголовка. Для цього використовуйте додаткові аргументи в функціях `plt.xlabel()`, `plt.ylabel()` та `plt.title()`.

Grid

В Matplotlib ви можете додати сітку (grid lines) до графіка, що допомагає візуально орієнтуватися в даних і полегшує їх інтерпретацію. Для цього використовується функція `plt.grid()`. Ось приклад коду, де демонструється використання сітки:

```
import matplotlib.pyplot as plt

# Дані для графіка
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
```

```
# Створення лінійного графіка
plt.plot(x, y)

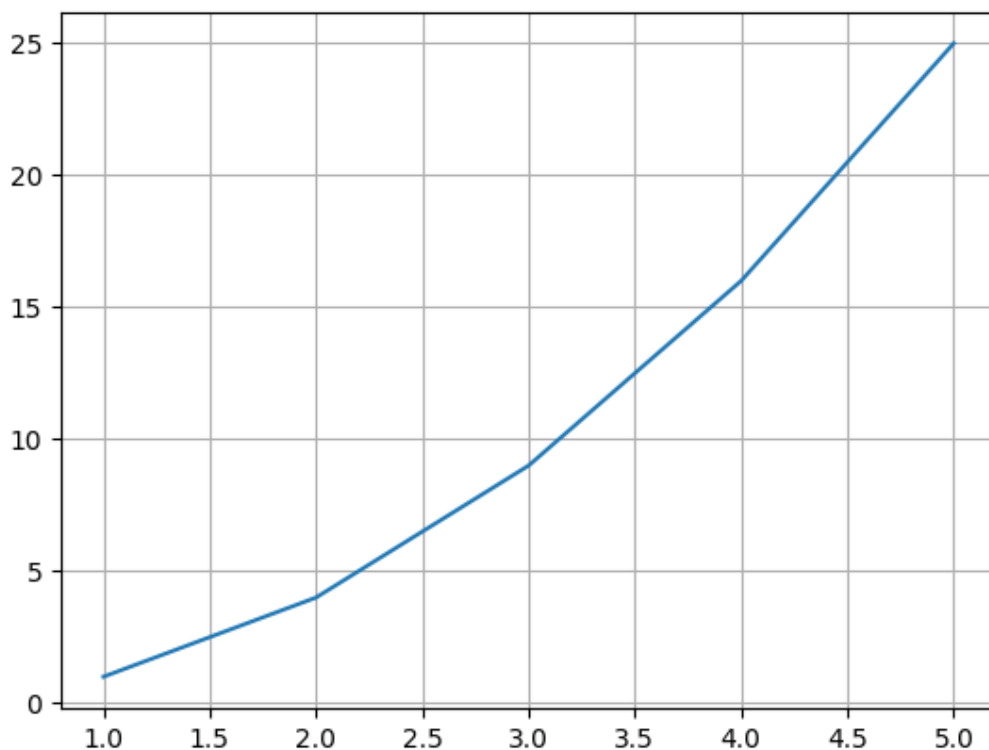
# Додавання сітки
plt.grid(True)

# Відображення графіка
plt.show()
```

У цьому прикладі ми використовуємо функцію `plt.grid(True)`, щоб додати сітку до графіка. Параметр `True` вказує, що сітка повинна бути включена.

Ви також можете налаштувати стиль і вигляд сітки, використовуючи додаткові параметри функції `plt.grid()`. Наприклад, ви можете встановити колір сітки, що видно тільки на головних лініях сітки, за допомогою

```
plt.grid(color='gray', linestyle='-', linewidth=0.5).
```



Додавання сітки допомагає візуально вирівняти дані і зробити графік більш зрозумілим, особливо при роботі з великими наборами даних або при

порівнянні різних серій даних.

subplots

В Matplotlib ви можете створювати графіки у вигляді підграфіків (subplots), що дозволяє відображати кілька графіків на одному полотні. Для цього використовуються функції `plt.subplots()` та `plt.subplot()`. Ось приклад коду, де демонструється використання підграфіків:

```
import matplotlib.pyplot as plt

# Дані для графіків
x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [1, 8, 27, 64, 125]

# Створення полотна та підграфіків
fig, axes = plt.subplots(nrows=1, ncols=2)

# Перший підграфік
axes[0].plot(x, y1)
axes[0].set_title('Графік 1')

# Другий підграфік
axes[1].plot(x, y2)
axes[1].set_title('Графік 2')

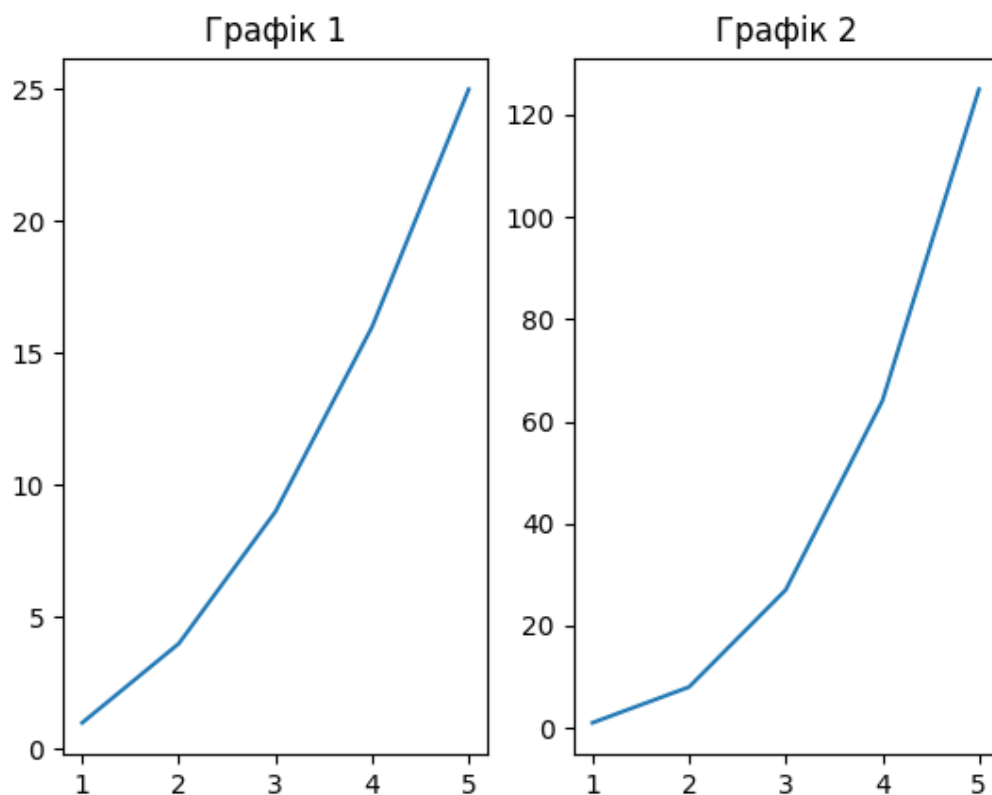
# Відображення графіків
fig.show()
```

У цьому прикладі ми використовуємо функцію `plt.subplots(nrows=1, ncols=2)`, щоб створити полотно з одним рядком і двома стовпцями, що дозволяє розмістити два підграфіки поруч.

Отримані об'єкти `fig` і `axes` представляють полотно та підграфіки відповідно. Ми використовуємо індекси `axes[0]` і `axes[1]` для доступу до окремих підграфіків і задання їх властивостей.

У прикладі ми створюємо два графіки `Графік 1` і `Графік 2` з використанням функції `plot()`, а також встановлюємо заголовки для кожного підграфіка за допомогою `set_title()`.

Після налаштування підграфіків викличте `plt.show()` для відображення полотна з усіма підграфіками.



Ви можете налаштовувати розміри, розташування і спільні властивості підграфіків за допомогою додаткових параметрів функції

`plt.subplots()` та `plt.subplot()`.

Scatter

`scatter()` - це функція Matplotlib, яка використовується для створення точкових діаграм або діаграм розсіювання. Вона відображає окремі точки на графіку, де кожна точка може мати свої власні координати та характеристики.

Ось приклад коду, який демонструє використання функції `scatter()`:

```
import matplotlib.pyplot as plt

# Дані для точкової діаграми
x = [1, 2, 3, 4, 5]
y = [3, 5, 2, 6, 1]
colors = ['red', 'green', 'blue', 'orange', 'purple']
sizes = [30, 50, 80, 20, 40]

# Створення точкової діаграми
plt.scatter(x, y, c=colors, s=sizes)

# Налаштування вісей і заголовка
plt.xlabel('X-ось')
plt.ylabel('Y-ось')
plt.title('Точкова діаграма')

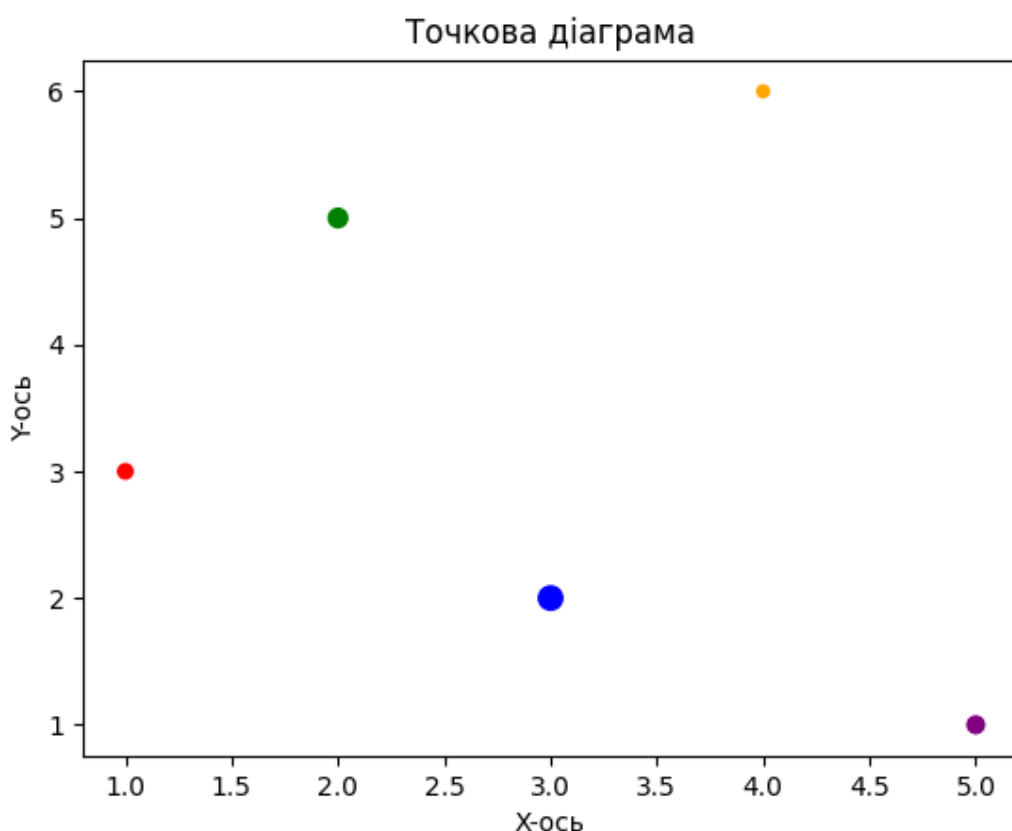
# Відображення діаграми
plt.show()
```

У цьому прикладі ми використовуємо функцію `scatter()` для створення точкової діаграми. Параметр `x` - це список значень по осі X, `y` - список значень по осі Y. Крім того, ми використовуємо параметр `c` для вказання кольорів точок та параметр `s` для вказання розмірів точок.

Після налаштування точкової діаграми, ми можемо налаштувати вісі (`xlabel()` та `ylabel()`) та заголовок (`title()`) графіку.

Нарешті, викликається `show()` для відображення діаграми.

Ви можете налаштовувати різні параметри, такі як розмір точок, колір, маркери, прозорість тощо, щоб налаштувати точкову діаграму за своїми потребами.



Barchart

`bar()` - це функція Matplotlib, яка використовується для створення стовпчикових діаграм. Вона дозволяє візуалізувати категоріальні дані, де кожна категорія має свої значення.

Ось приклад коду, який демонструє використання функції `bar()`:

```
import matplotlib.pyplot as plt

# Дані для стовпчикової діаграми
categories = ['A', 'B', 'C', 'D', 'E']
values = [15, 10, 7, 12, 9]

# Створення стовпчикової діаграми
plt.bar(categories, values)

# Налаштування вісей і заголовка
plt.xlabel('Категорії')
plt.ylabel('Значення')
```



```
plt.title('Стовпчикова діаграма')

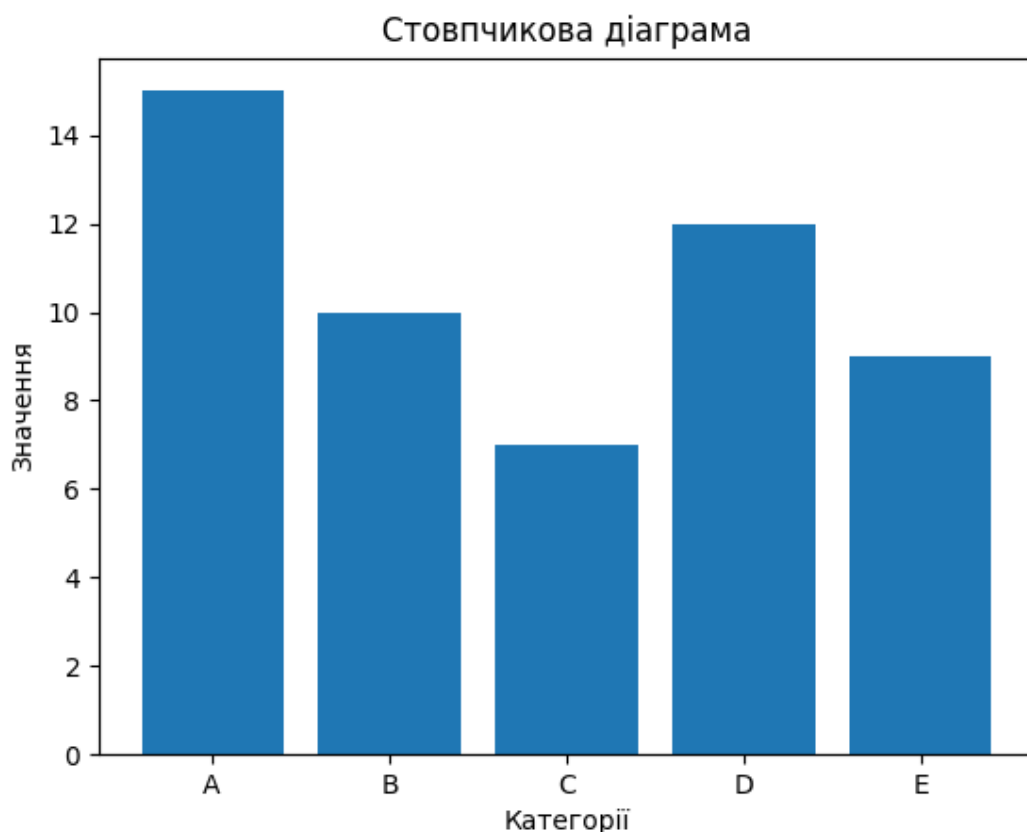
# Відображення діаграми
plt.show()
```

У цьому прикладі ми використовуємо функцію `bar()` для створення стовпчикової діаграми. Параметр `categories` - це список категорій, а `values` - список значень для кожної категорії.

Після створення стовпчикової діаграми, ми можемо налаштувати вісі (`xlabel()` та `ylabel()`) та заголовок (`title()`) графіку.

Нарешті, викликається `show()` для відображення діаграми.

Ви можете налаштовувати різні параметри, такі як кольори стовпчиків, ширина стовпчиків, відступи між стовпчиками, заголовки категорій тощо, щоб налаштувати стовпчикову діаграму за своїми потребами.



Histograms

Гістограма - це графік, що відображає частотні розподіли.

Це графік, що показує кількість спостережень у кожному відповідному інтервалі.

`hist()` - це функція Matplotlib, яка використовується для створення гістограм. Гістограма дозволяє візуалізувати розподіл значень в діапазоні.

Ось приклад коду, який демонструє використання функції `hist()`:

```
import matplotlib.pyplot as plt

# Дані для гістограми
data = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5]

# Створення гістограми
plt.hist(data)

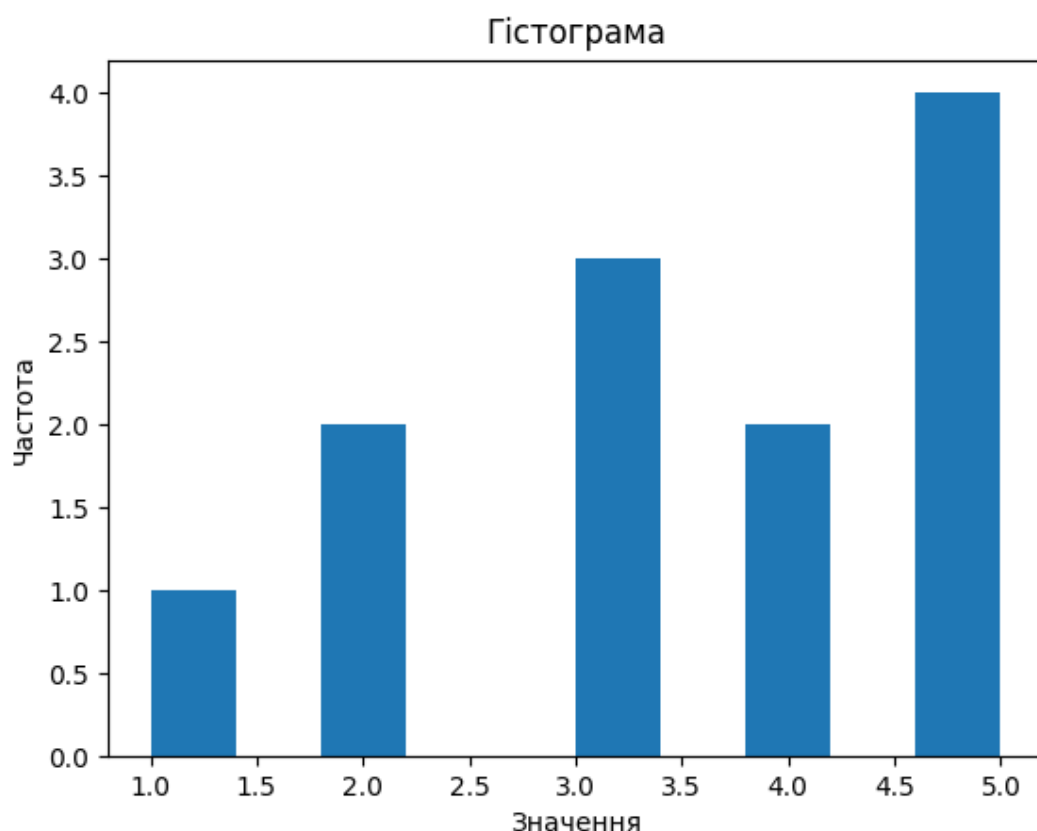
# Налаштування вісей і заголовка
plt.xlabel('Значення')
plt.ylabel('Частота')
plt.title('Гістограма')

# Відображення гістограми
plt.show()
```

У цьому прикладі ми використовуємо функцію `hist()` для створення гістограми. Параметр `data` - це список значень, для яких ми хочемо побудувати гістограму.

Після створення гістограми, ми можемо налаштувати вісі (`xlabel()` та `ylabel()`) та заголовок (`title()`) графіку.

Нарешті, викликається `show()` для відображення гістограми.



Ви можете налаштовувати різні параметри, такі як кількість бінів (інтервалів), кольори стовпчиків, прозорість, додаткові графічні елементи тощо, щоб налаштувати гістограму за своїми потребами.

Matplotlib Pie Charts

`pie()` - це функція Matplotlib, яка використовується для створення кругових діаграм. Кругова діаграма дозволяє візуалізувати частки частин у відношенні до цілого.

Ось приклад коду, який демонструє використання функції `pie()`:

```
import matplotlib.pyplot as plt

# Дані для кругової діаграми
labels = ['Частка 1', 'Частка 2', 'Частка 3']
sizes = [45, 30, 25]
```

```
# Створення кругової діаграми
plt.pie(sizes, labels=labels)

# Налаштування заголовка
plt.title('Кругова діаграма')

# Відображення діаграми
plt.show()
```

У цьому прикладі ми використовуємо функцію `pie()` для створення кругової діаграми. Параметр `sizes` - це список значень, які представляють частки. Параметр `labels` - це список міток для кожної частки.

Після створення кругової діаграми, ми можемо налаштувати заголовок (`title()`) графіку.

Нарешті, викликається `show()` для відображення діаграми.



Ви можете налаштовувати різні параметри, такі як кольори секторів, відстані між секторами, висування сектора, товщину обводки тощо, щоб

налаштувати кругову діаграму за своїми потребами.

Як обрати графік?

Кожен тип графіку має свої унікальні властивості і застосування. Ось деякі приклади, коли краще використовувати різні типи графіків:

1. Лінійний графік:

- Відстеження змін величини в часі (наприклад, ціни акцій, температура, виробництво тощо).
- Показ тенденцій, трендів або залежностей між змінними.
- Приклад: Графік зміни температури протягом дня на годинниковому графіку.

2. Точковий графік:

- Візуалізація кореляції між двома змінними.
- Показ розподілу точок на графіку.
- Приклад: Залежність між віком людей та їхнім доходом.

3. Гістограма:

- Відображення розподілу частоти або величини в певних інтервалах.
- Аналіз розподілу даних та виявлення викидів або аномалій.
- Приклад: Розподіл висоти учнів у школі.

4. Стовпчикова діаграма:

- Порівняння значень між категоріями.
- Відображення дискретних даних для різних груп або категорій.
- Приклад: Порівняння продажів різних товарів у магазині.

5. Кругова діаграма:

- Показ відсоткового співвідношення складових частин до цілого.
- Візуалізація розподілу категорій або часток на основі відсотків.
- Приклад: Поділ розподілу бюджету на різні види витрат.

Вибір типу графіку залежить від природи даних, завдання візуалізації та орієнтації аудиторії.

Savefig

Функція `savefig` у бібліотеці Matplotlib використовується для збереження графіків або фігур у вигляді зображень. Ця функція дозволяє зберігати візуалізації у різних форматах файлів, таких як PNG, JPEG, PDF, SVG та інші.

Ось короткий огляд того, як використовувати `savefig` у бібліотеці Matplotlib:

1. Імпортуйте необхідні бібліотеки:

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

1. Створіть свій графік або фігуру, використовуючи функції побудови графіків у Matplotlib.
2. Після створення графіку викличте функцію `savefig`, щоб зберегти його як зображення:

```
plt.savefig('шлях/до/збереження/зображення.png')
```

У цьому прикладі файл буде збережений як 'зображення.png' за вказаним шляхом. Ви також можете вказати відносний або абсолютний шлях для збереження зображення у певному каталозі.

1. Опціонально, ви можете вказати додаткові параметри при використанні `savefig`. Наприклад, ви можете налаштувати роздільну здатність (DPI) зображення за допомогою параметра `dpi`:

```
pythonCopy code
plt.savefig('шлях/до/збереження/зображення.png', dpi=300)
```

Більші значення DPI дають зображення вищої роздільної здатності.

Отже, це базовий огляд використання `savefig` у бібліотеці Matplotlib. Це корисна функція для збереження ваших графіків або фігур для подальшого використання або надсилання іншим користувачам. Пам'ятайте, що слід викликати `savefig` після завершення побудови графіка, але перед викликом `plt.show()`, оскільки `savefig` зберігає поточну активну фігуру.

Розмір графіку

У бібліотеці Matplotlib є кілька способів задати розмір графіка. Ось декілька прикладів:

1. Використання параметра `figsize` при створенні фігури:

```
pythonCopy code
import matplotlib.pyplot as plt

# Задати розмір фігури в дюймах (ширина, висота)
plt.figure(figsize=(6, 4))
# Побудова графіку
plt.plot(x, y)
# Відображення графіка
plt.show()
```

У цьому прикладі функція `figure` приймає параметр `figsize`, в якому вказується бажаний розмір фігури. Значення вказуються у дюймах (ширина, висота).

1. Використання методу `set_size_inches` для налаштування розміру фігури:

```
pythonCopy code
import matplotlib.pyplot as plt

# Створення фігури
fig = plt.figure()
# Побудова графіку
plt.plot(x, y)
# Задати розмір фігури в дюймах (ширина, висота)
fig.set_size_inches(6, 4)
# Відображення графіка
plt.show()
```

У цьому прикладі ми створюємо фігуру, побудовуємо графік, а потім використовуємо метод `set_size_inches` об'єкта фігури для задання бажаного розміру у дюймах.

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

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

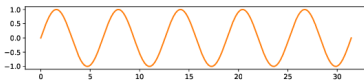
2 Prepare

```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

4 Observe



Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```



```
Z = np.random.uniform(0, 1, (8, 8))
ax.imshow(Z)
```



```
Z = np.random.uniform(0, 1, (8, 8))
ax.contourf(Z)
```



```
Z = np.random.uniform(0, 1, 4)
ax.pie(Z)
```



```
Z = np.random.normal(0, 1, 100)
ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



```
Z = np.random.normal(0, 1, (100, 3))
ax.boxplot(Z)
```



Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



Organize

You can plot several data on the same figure, but you can also split a figure in several subplots (named Axes):

```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```



```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```



Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```



```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

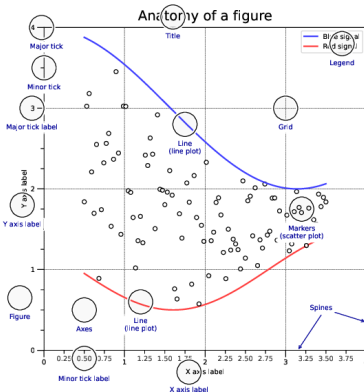
Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib 3.7.4 handout for beginners. Copyright (c) 2021 Matplotlib Development Team. Released under a CC-BY 4.0 International License. Supported by NumFOCUS.

Matplotlib for intermediate users

A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.



Figure, axes & spines

```
fig, axes = plt.subplots(3, 3)
axes[0, 0].set_facecolor("#dddfdf")
axes[2, 2].set_facecolor("#ffffdf")
```



```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#dddfdf")
```

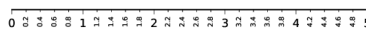


```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```



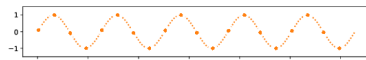
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x', which='minor', rotation=90)
```



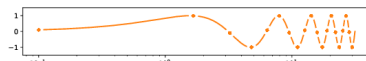
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=50, mec="1.0")
```



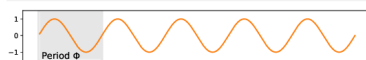
Scales & projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o:", markevery=50, mec="1.0")
```



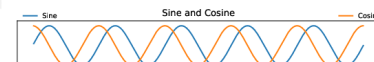
Text & ornaments

```
ax.fill_between([-1, 1], [0], [2*np.pi])
ax.text(0, -1, r"Period $\Phi$")
```



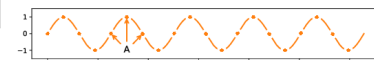
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0, 1, 1, .1), ncol=2,
         mode="expand", loc="lower left")
```



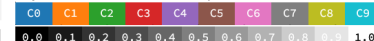
Annotation

```
ax.annotate("A", (X[250], Y[250]), (X[250], -1),
         ha="center", va="center", arrowprops={
         "arrowstyle": "->", "color": "C1"})
```



Colors

Any color can be used, but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-column A4 paper with 2 cm margins on each side and a column separation of 1 cm. The width of a figure is (21 - 2*2 - 1)/2 = 8 cm. One inch being 2.54 cm, figure size should be 3.15x3.15 in.

```
fig = plt.figure(figsize=(3.15, 3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Matplotlib 3.7.4 handout for intermediate users. Copyright (c) 2021 Matplotlib Development Team. Released under a CC-BY 4.0 International License. Supported by NumFOCUS.

Matplotlib tips & tricks

Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density. Multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1, 1, 500)
Y = np.random.normal(-1, 1, 500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1", lw=0, alpha=0.1)
```



Rasterization

If your figure has many graphical elements, such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

Offline rendering

Use the Agg backend to render a figure directly in an array.

```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure())
... # draw some stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

Range of continuous colors

You can use colormap to pick from a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Oranges")
colors = cmap([0.2, 0.4, 0.6, 0.8])
ax.hist(X, 2, histtype='bar', color=colors)
```



Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
    fx.Stroke(linewidth=3, foreground='1.0'),
    fx.Normal()])
```



Multiline plot

You can plot several lines at once using None as separator.

```
X, Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
    X.append([x, x, None]), Y.append([0, sin(x), None])
ax.plot(X, Y, "black")
```



Dotted lines

To have rounded dotted lines, use a custom lines tyle and modify dash_capstyle.

```
ax.plot([0, 1], [0, 0], "C1",
        linestyle=(0, (0.01, 1)), dash_capstyle="round")
ax.plot([0, 1], [1, 1], "C1",
        linestyle=(0, (0.01, 2)), dash_capstyle="round")
```



Combining axes

You can use overlaid axes with different projections.

```
ax1 = fig.add_axes([0, 0, 1, 1],
                    label="cartesian")
ax2 = fig.add_axes([0, 0, 1, 1],
                    label="polar",
                    projection="polar")
```



Colorbar adjustment

You can adjust a colorbar's size when adding it.

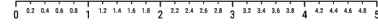
```
im = ax.imshow(Z)
cb = plt.colorbar(im,
                  fraction=0.046, pad=0.04)
cb.set_ticks([])
```



Taking advantage of typography

You can use a condensed font such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname("Roboto Condensed")
```



Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

Hatching

You can achieve a nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/")
```



Read the documentation

Matplotlib comes with an extensive documentation explaining the details of each command and is generally accompanied by examples. Together with the huge online gallery, this documentation is a gold-mine.

Matplotlib 3.7.4 handbook for tips & tricks. Copyright (c) 2021 Matplotlib Development Team. Released under a CC-BY 4.0 International License. Supported by NumFOCUS.