Національний університет біоресурсів і
природокористування України

Візуалізація даних за допомогою Matplotlib

Лектор: к.ф.-м.н., доцент Кириченко Віктор Вікторович

Вступ до Matplotlib

Що таке Matplotlib?

- Бібліотека для візуалізації даних у Python
- Створена Джоном Хантером у 2003 році
- Найпопулярніший інструмент для побудови графіків

Переваги

- Гнучкість та повний контроль над візуалізацією
- Підтримка різних форматів виводу
- Інтеграція з NumPy, Pandas та іншими бібліотеками

Основні компоненти

- `pyplot` - інтерфейс для швидкого створення графіків
- `Figure` - контейнер для всіх елементів графіка
- `Axes` - область побудови графіка
- `Artist` - всі видимі елементи на графіку

Типи візуалізацій

- Лінійні графіки та точкові діаграми
- Стовпчикові діаграми та гістограми
- Кругові діаграми та контурні графіки
- Тривимірні графіки та зображення

Встановлення та налаштування Matplotlib

Встановлення

Через pip (рекомендовано):

```
pip install matplotlib
```

Через conda:

```
conda install matplotlib
```

З вихідного коду:

```
git clone https://github.com/matplotlib/matplotlib.git
cd matplotlib
python -m pip install -e .
```

Перевірка встановлення

- Перевірка версії Matplotlib

```
import matplotlib
print(matplotlib.__version__)
```

Імпорт модулів

- Стандартний спосіб імпорту

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

- Імпорт з NumPy для роботи з даними

```
import numpy as np
```

Базове налаштування

- Налаштування стилю графіків

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

- Налаштування відображення графіків

```
%matplotlib inline # для Jupyter Notebook
plt.show() # для скриптів
```

Основні функції для побудови двовимірних графіків

Функція plot()

- Створює лінійний графік
- Базовий синтаксис: `plt.plot(x, y)`

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

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

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

Параметри plot()

- `color` - колір лінії
- `linestyle` - стиль лінії
- `linewidth` - товщина лінії
- `marker` - маркери точок

Функція scatter()

- Створює точкову діаграму
- Базовий синтаксис: `plt.scatter(x, y)`

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

np.random.seed(42)
x = np.random.randn(100)
y = np.random.randn(100)

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

Параметри scatter()

- `s` - розмір точок
- `c` - колір точок
- `marker` - форма маркера
- `alpha` - прозорість точок

Лінійні графіки

Створення лінійних графіків

- Базовий синтаксис: `plt.plot(x, y)`
- Короткий формат: `plt.plot(y)` (x автоматично)

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

x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

Маркери та їх налаштування

- Типи маркерів: `'o'` (коло), `'s'` (квадрат), `'^'` (трикутник)
- Розмір маркера: `markersize`

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

x = np.linspace(0, 10, 20)
y = np.sin(x)

plt.plot(x, y, 'bo-',
         markersize=8,
         linewidth=2,
         markerfacecolor='white')
plt.show()
```

Кольори та стилі ліній

- Кольори: `'r'` (червоний), `'g'` (зелений), `'b'` (синій)
- Стилі ліній: `'-'` (суцільна), `'--'` (пунктир), `':'` (крапка)
- Комбінація: `'r--'` (червона пунктирна)

Приклад комбінованого графіка

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

x = np.linspace(0, 10, 50)

plt.plot(x, np.sin(x), 'r-', label='sin(x)')
plt.plot(x, np.cos(x), 'b--', label='cos(x)')
plt.plot(x, np.sin(x)+np.cos(x), 'g:', label='sin(x)+cos(x)')

plt.legend()
plt.show()
```

Точкові діаграми

Основи точкових діаграм

- Базовий синтаксис: `plt.scatter(x, y)`
- Відображення залежності між двома змінними

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

np.random.seed(42)
x = np.random.randn(100)
y = x * 0.8 + np.random.randn(100) * 0.2

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

Кольори та розміри точок

- `c` - колір точок (можна задавати масивом)
- `s` - розмір точок (можна задавати масивом)
- `alpha` - прозорість точок (0-1)

Форми маркерів

- `marker` - форма маркера
- Популярні маркери: `'o'`, `'s'`, `'^'`, `'D'`

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

np.random.seed(42)
x = np.random.randn(50)
y = np.random.randn(50)
colors = np.random.rand(50)
sizes = 100 * np.random.rand(50)

plt.scatter(x, y, c=colors, s=sizes,
        alpha=0.7, cmap='viridis')
plt.colorbar()
plt.show()
```

Бульбашкові діаграми

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

np.random.seed(42)
x = np.random.rand(30)
y = np.random.rand(30)
z = np.random.rand(30) * 1000

plt.scatter(x, y, s=z, alpha=0.6,
        c=z, cmap='plasma')
plt.colorbar(label='Значення Z')
plt.show()
```

Налаштування оформлення графіків

Заголовки та підписи

- `plt.title()` - заголовок графіка
- `plt.xlabel()` - підпис осі X
- `plt.ylabel()` - підпис осі Y

```
plt.title('Залежність температури від часу')
plt.xlabel('Час (години)')
plt.ylabel('Температура (°C)')
```

Кольори та стилі

- `plt.plot(..., color='name')` - колір лінії
- `plt.plot(..., linestyle='style')` - стиль лінії
- `plt.plot(..., linewidth=value)` - товщина лінії

```
plt.plot(x, y, color='#1A365D',
         linestyle='-.',
         linewidth=2)
```

Сітка та масштабування

- `plt.grid()` - увімкнення сітки
- `plt.xlim()` - межі осі X
- `plt.ylim()` - межі осі Y

```
plt.grid(True, linestyle='--', alpha=0.7)
plt.xlim(0, 10)
plt.ylim(-2, 2)
```

Легенди

- `plt.plot(..., label='name')` - мітка для легенди
- `plt.legend()` - відображення легенди
- `plt.legend(loc='position')` - позиція легенди

```
plt.plot(x, y1, label='Синусоїда')
plt.plot(x, y2, label='Косинусоїда')
plt.legend(loc='upper right')
```

Підписи осей та заголовки

Заголовки графіків

- `plt.title()` - основний заголовок
- `plt.suptitle()` - надзаголовок

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

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

plt.plot(x, y)
plt.title('Графік синусоїди')
plt.suptitle('Тригонометричні функції')
plt.show()
```

Форматування заголовків

- `fontsize` - розмір шрифту
- `fontweight` - товщина шрифту
- `color` - колір тексту

```
plt.title('Заголовок',
          fontsize=16,
          fontweight='bold',
          color='#1A365D')
```

Підписи осей

- `plt.xlabel()` - підпис осі X
- `plt.ylabel()` - підпис осі Y

```
plt.plot(x, y)
plt.xlabel('Час (с)')
plt.ylabel('Амплітуда')
plt.title('Коливання')
plt.show()
```

Розміщення та форматування

- `labelpad` - відступ підпису
- `rotation` - поворот тексту

```
plt.xlabel('Дні', labelpad=10)
plt.ylabel('Температура (°C)',
           rotation=0,
           labelpad=20)
```

Легенди

Створення легенди

- `label` параметр у функціях побудови
- `plt.legend()` для відображення легенди

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

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x), label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')
plt.legend()
plt.show()
```

Розташування легенди

- `loc` параметр для позиціонування
- Можливі значення: `'best'`, `'upper right'`, `'lower left'`

```
plt.legend(loc='upper right')
plt.legend(loc='lower left')
plt.legend(loc='best')
```

Оформлення легенди

- `title` - заголовок легенди
- `fontsize` - розмір шрифту
- `framealpha` - прозорість рамки

```
plt.legend(title='Функції',
           fontsize=10,
           framealpha=0.8)
```

Кількість стовпців

- `ncol` параметр для кількості стовпців

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

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x), label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')
plt.plot(x, np.tan(x), label='tan(x)')
plt.legend(ncol=3, loc='upper center')
```

Стовпчикові діаграми

Основи стовпчикових діаграм

- Базовий синтаксис: `plt.bar(x, height)`
- Відображення категоріальних даних

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

categories = ['A', 'B', 'C', 'D']
values = [15, 30, 45, 10]

plt.bar(categories, values)
plt.show()
```

Параметри стовпчиків

- `width` - ширина стовпчиків
- `color` - колір стовпчиків
- `align` - вирівнювання (`'center'`, `'edge'`)

Груповані та складені діаграми

- Груповані: зміщення позиції стовпчиків
- Складені: параметр `bottom`

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

x = np.arange(4)
y1 = [15, 30, 45, 10]
y2 = [25, 20, 35, 15]

width = 0.35

plt.bar(x - width/2, y1, width)
plt.bar(x + width/2, y2, width)
plt.show()
```

Горизонтальні стовпчикові діаграми

- Функція `plt.barh()`

```
import matplotlib.pyplot as plt

categories = ['A', 'B', 'C', 'D']
values = [15, 30, 45, 10]

plt.barh(categories, values,
        color='skyblue')
plt.show()
```

Кругові діаграми

Основи кругових діаграм

- Базовий синтаксис: `plt.pie(sizes)`
- Відображення пропорційних даних

```
import matplotlib.pyplot as plt

sizes = [30, 20, 25, 15, 10]
labels = ['A', 'B', 'C', 'D', 'E']

plt.pie(sizes, labels=labels)
plt.show()
```

Параметри кругових діаграм

- `colors` - кольори секторів
- `explode` - відстань секторів від центру
- `autopct` - формат відображення відсотків

Кільцеві діаграми

- Створення кільцевої діаграми за допомогою `wedgeprops`

```
import matplotlib.pyplot as plt

sizes = [30, 20, 25, 15, 10]
labels = ['A', 'B', 'C', 'D', 'E']

plt.pie(sizes, labels=labels,
        wedgeprops={'width': 0.4})
plt.show()
```

Приклад кастомізації

```
import matplotlib.pyplot as plt

sizes = [30, 20, 25, 15, 10]
labels = ['A', 'B', 'C', 'D', 'E']
explode = [0.1, 0, 0, 0, 0]

plt.pie(sizes, labels=labels,
        explode=explode,
        autopct='%1.1f%%',
        shadow=True,
        startangle=90)
plt.show()
```

Гістограми

Основи гістограм

- Базовий синтаксис: `plt.hist(data)`
- Відображення розподілу числових даних

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

np.random.seed(42)
data = np.random.normal(0, 1, 1000)

plt.hist(data)
plt.show()
```

Параметри гістограм

- `bins` - кількість інтервалів
- `density` - нормалізація
- `color` - колір стовпців
- `alpha` - прозорість

Порівняння розподілів

- Накладення кількох гістограм

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

np.random.seed(42)
data1 = np.random.normal(0, 1, 1000)
data2 = np.random.normal(2, 1, 1000)

plt.hist(data1, alpha=0.5, label='Дані 1')
plt.hist(data2, alpha=0.5, label='Дані 2')
plt.legend()
plt.show()
```

Кумулятивні гістограми

- Параметр `cumulative=True`

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

np.random.seed(42)
data = np.random.normal(0, 1, 1000)

plt.hist(data, bins=20,
         cumulative=True,
         density=True)
plt.show()
```

Збереження графіків у різних форматах

Базове збереження

- Функція `plt.savefig()`
- Формат визначається розширенням файлу

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

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

plt.plot(x, y)
plt.savefig('my_plot.png')
plt.savefig('my_plot.pdf')
```

Параметри якості

- `dpi` - роздільна здатність (крапки на дюйм)
- `quality` - якість для JPEG (1-100)
- `optimize` - оптимізація для PNG

Розміри та орієнтація

- `figsize` - розмір фігури в дюймах
- `orientation` - орієнтація сторінки

```
plt.figure(figsize=(10, 6))
plt.plot(x, y)
plt.savefig('large_plot.png',
        orientation='landscape')
```

Популярні формати

- `PNG` - растровий, підтримка прозорості
- `PDF` - векторний, для друку та публікацій
- `SVG` - векторний, для веб та редагування
- `JPG/JPEG` - растровий, стиснення з втратами

Розширені можливості налаштування

Робота з підграфіками

- `plt.subplot()` - створення сітки підграфіків
- `plt.subplots()` - створення фігури з підграфіками

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

fig, axes = plt.subplots(2, 2, figsize=(10, 8))

x = np.linspace(0, 10, 100)
axes[0, 0].plot(x, np.sin(x))
axes[0, 1].plot(x, np.cos(x))
axes[1, 0].plot(x, np.tan(x))
axes[1, 1].plot(x, np.exp(x))
plt.tight_layout()
```

Налаштування розміру фігури

- `plt.figure(figsize=(w, h))` - розмір у дюймах
- `plt.tight_layout()` - автоматичне розміщення
- `plt.subplots_adjust()` - ручне налаштування відступів

Кастомізація сітки

- `plt.grid()` - увімкнення сітки
- `axis='both'` - сітка на обох осях

```
plt.grid(True, linestyle='--',
         alpha=0.7,
         color='gray',
         linewidth=0.5)

plt.grid(True, which='minor',
         linestyle=':')
```

Стилі та теми

- `plt.style.use()` - застосування стилю
- `plt.style.available` - доступні стилі

```
import matplotlib.pyplot as plt

# Перегляд доступних стилів
print(plt.style.available)

# Застосування стилю
plt.style.use('seaborn')
# або 'ggplot', 'classic', 'bmh'...
```

Робота з підграфіками

Створення сітки підграфіків

- `plt.subplot(nrows, ncols, index)`
- `plt.subplots(nrows, ncols)`

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

# Метод 1: subplot
plt.figure(figsize=(10, 6))
plt.subplot(2, 2, 1)
plt.plot([1, 2, 3], [4, 5, 6])
plt.subplot(2, 2, 2)
plt.plot([1, 2, 3], [6, 5, 4])
```

Робота з об'єктом Figure

- Створення фігури з підграфіками

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

# Метод 2: subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

x = np.linspace(0, 10, 100)
axes[0, 0].plot(x, np.sin(x))
axes[0, 1].plot(x, np.cos(x))
axes[1, 0].plot(x, np.tan(x))
axes[1, 1].plot(x, np.exp(x))
plt.tight_layout()
```

Розміщення підграфіків

- `plt.tight_layout()` - автоматичне вирівнювання
- `fig.subplots_adjust()` - ручне налаштування відступів
- `sharex, sharey` - спільні осі

Складне розміщення

- `GridSpec` - гнучке розміщення

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig = plt.figure(figsize=(10, 8))
gs = GridSpec(3, 3)

ax1 = fig.add_subplot(gs[0, :])
ax2 = fig.add_subplot(gs[1, :2])
ax3 = fig.add_subplot(gs[1:, 2])
ax4 = fig.add_subplot(gs[2, :2])
```

Практичні приклади використання

Візуалізація часових рядів

- Аналіз даних про продажі
- Відстеження температурних змін

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

dates = pd.date_range('20230101', periods=12)
sales = [120, 135, 148, 165, 180, 210,
        225, 240, 255, 270, 290, 310]

plt.plot(dates, sales, 'o-', color='#1A365D')
plt.xticks(rotation=45)
plt.title('Динаміка продажів')
```

Наукові дослідження

- Візуалізація експериментальних даних
- Порівняння теоретичних та практичних результатів

Аналіз даних

- Виявлення закономірностей у даних
- Статистичний аналіз

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

np.random.seed(42)
data1 = np.random.normal(0, 1, 1000)
data2 = np.random.normal(2, 1.5, 1000)

plt.figure(figsize=(10, 6))
plt.hist(data1, bins=30, alpha=0.5,
        label='Група А')
plt.hist(data2, bins=30, alpha=0.5,
        label='Група Б')
plt.legend()
```

Бізнес-звіти

- Візуалізація фінансових показників
- Порівняння продуктивності відділів

Кращі практики та поради

Чіткість та інформативність

- Використовуйте зрозумілі заголовки та підписи осей
- Обирайте відповідний тип графіка для ваших даних
- Уникайте перевантаження графіка зайвими елементами
- Додавайте легенди, коли це необхідно

Ефективне представлення даних

- Налаштовуйте масштаб осей для кращої видимості
- Використовуйте логарифмічну шкалу для великих діапазонів
- Додавайте анотації для ключових точок даних
- Групуйте пов'язані дані на одному графіку

Використання кольорів

- Використовуйте контрастні кольори для різних даних
- Уникайте яскравих кольорів, що викликають напругу
- Враховуйте кольорову сліпоту при виборі палітри
- Використовуйте тематичні колірні схеми (colormaps)

Оптимізація коду

- Створюйте функції для повторюваних графіків
- Використовуйте стилі для узгодженості
- Зберігайте параметри оформлення у словниках
- Використовуйте об'єктно-орієнтований підхід для складних візуалізацій

Висновки

Основні можливості Matplotlib

- Matplotlib є стандартним інструментом для візуалізації даних у Python
- Бібліотека надає гнучкі можливості для створення різноманітних графіків
- Підтримує широкий спектр типів візуалізацій: від простих лінійних графіків до складних 3D-зображень

Практичне застосування

- Matplotlib ефективно використовується для наукової візуалізації та аналізу даних
- Можливість збереження графіків у різних форматах дозволяє інтегрувати їх у звіти та публікації

Ключові навички

- Вміння створювати та налаштовувати основні типи графіків є фундаментальною навичкою для аналітика даних
- Розуміння принципів оформлення графіків дозволяє створювати інформативні та візуально привабливі візуалізації
- Володіння технікою роботи з підграфіками розширює можливості комплексного представлення даних

Перспективи розвитку

- Matplotlib продовжує розвиватися, розширюючи функціональність та покращуючи продуктивність
- Інтеграція з іншими бібліотеками (Pandas, Seaborn) робить його ще потужнішим інструментом