

# 机器学习与深度学习 ——Python编程基础



Personal Website: <https://www.miaopeng.info/>



Email: [miaopeng@stu.scu.edu.cn](mailto:miaopeng@stu.scu.edu.cn)



Github: <https://github.com/MMeowhite>



Youtube: <https://www.youtube.com/@pengmiao-bmm>

# 目录章节

---

CONTENTS

01 语法基础与数据类型

02 Numpy: 数值计算

03 Pandas: 数据处理

04 Matplotlib: 数据可视化

05 Scikit-learn: 机器学习

# ► 什么是Python？

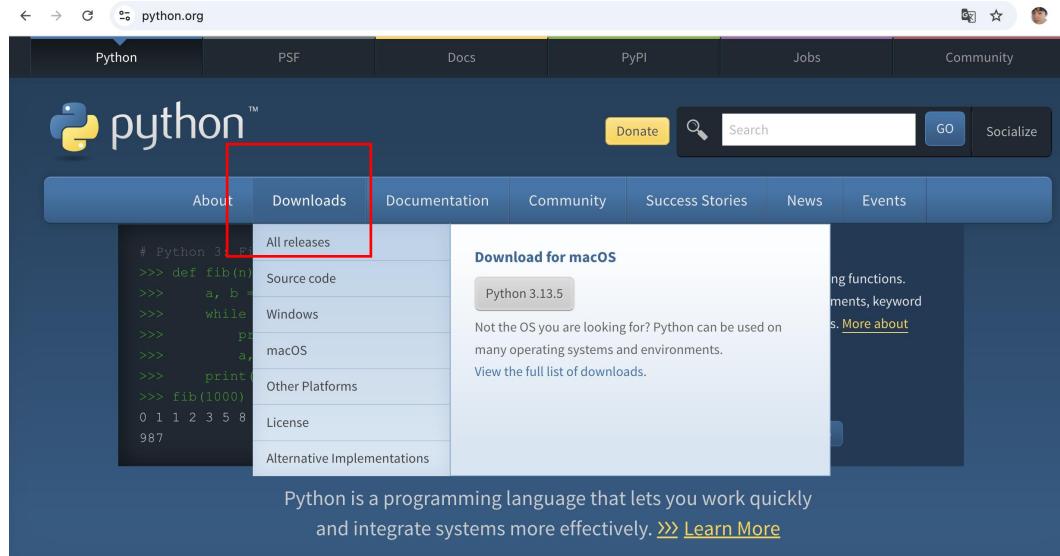
- Python 是一种简单易学的编程语言，适合初学者和专业开发者，广泛应用于数据分析、人工智能、Web开发、自动化脚本等领域。
- Python 拥有众多高性能的第三方库，如 NumPy、Pandas、Matplotlib、Scikit-learn 和 PyTorch 等，这些工具在 AI 模型的构建与训练过程中发挥着关键作用，为开发者提供了高效且直观的编程支持。



Python语法简洁、可读性强，是许多人学习编程的首选语言

# ▶ 怎么安装Python？

- 访问 Python 官网 [python.org](https://python.org)，点击Downloads，下载与你系统匹配的安装包。

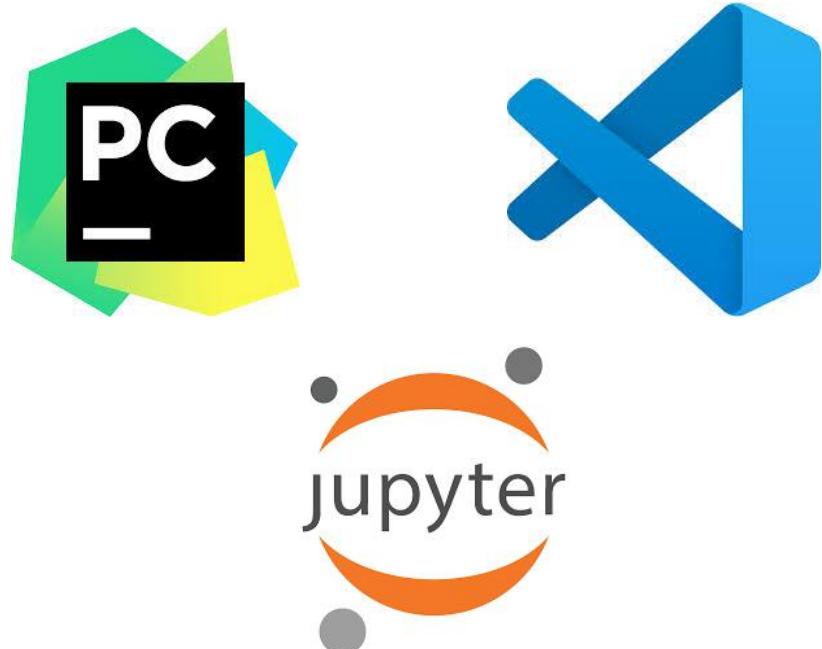


- 运行安装程序包（勾选“Add Python to PATH”，一般默认），然后点击 Install。
- 安装完成后，打开终端（或命令提示符）输入 python，看到版本号即安装成功。

```
# miaopeng @ MMeowhite in ~/code/GithubProject/Ml-Dl-Fullstack-Guide on git:main o [10:40:35]
$ python
Python 3.11.0 | packaged by conda-forge | (main, Jan 14 2023, 12:26:40) [Clang 14.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

## ► 集成开发环境（IDE）

- IDE（Integrated Development Environment，集成开发环境）是一个用于**编写、调试、运行和管理代码**的一站式工具。
- IDE的核心功能：代码编辑、调试工具、运行环境、项目管理、插件扩展



名称	特点	安装步骤
Pychar m	专业强大、功能全面、 适合大型项目	官网下载即可
VS Code	轻量级、插件丰富、 跨语言支持	官网下载即可
Jupyter Notebo ok	适合数据分析与教学， 支持图表与 Markdown 交互式展 示	终端运行命令`pip install notebook`

IDE 集成了代码编辑、调试和运行功能，帮助程序员更高效地编写和管理代码。

# ► 扩展：Jupyter Notebook

- Jupyter Notebook是一种“交互式编程环境”，支持代码+文字+图标的组合，同时文档区域支持markdown语法，特别适合数据分析、可视化以及机器学习。
- 安装方式：1) Anaconda；2) pip工具：pip install notebook；3) 直接使用网页端平台如Kaggle、Google Colab
- 启动方式：打开终端（terminal），输入jupyter notebook，会自动打开浏览器，进入本地Jupyter 管理页面（以当前的文件夹路径为根目录）。

```
# miaopeng @ MMeowhite in ~ [13:58:15] C:127
$ jupyter notebook
[I 2025-07-20 13:58:27.967 ServerApp] Package notebook took 0.000s to import
[I 2025-07-20 13:58:27.982 ServerApp] Package jupyter_lsp took 0.0152s to import
[W 2025-07-20 13:58:27.982 ServerApp] A `__jupyter_server_extension_points` function was not found in jupyter_lsp. Instead, a `__jupyter_server_extension_paths` function was found and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[I 2025-07-20 13:58:27.998 ServerApp] Package jupyter_server_terminals took 0.007s to import
[I 2025-07-20 13:58:27.998 ServerApp] Package jupyter_lab took 0.000s to import
[I 2025-07-20 13:58:28.379 ServerApp] Package notebook_shim took 0.000s to import
[W 2025-07-20 13:58:28.379 ServerApp] A `__jupyter_server_extension_points` function was not found in notebook_shim. Instead, a `__jupyter_server_extension_paths` function was found and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[I 2025-07-20 13:58:28.881 ServerApp] Package panel.io.jupyter_server_extension took 0.5014s to import
[I 2025-07-20 13:58:28.881 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2025-07-20 13:58:28.883 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2025-07-20 13:58:28.884 ServerApp] jupyterlab | extension was successfully linked.
[I 2025-07-20 13:58:28.886 ServerApp] notebook | extension was successfully linked.
[I 2025-07-20 13:58:29.131 ServerApp] notebook_shim | extension was successfully linked.
[I 2025-07-20 13:58:29.131 ServerApp] panel.io.jupyter_server_extension | extension was successfully linked.
[I 2025-07-20 13:58:29.163 ServerApp] notebook_shim | extension was successfully loaded.
[I 2025-07-20 13:58:29.164 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2025-07-20 13:58:29.165 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2025-07-20 13:58:29.167 LabApp] JupyterLab extension loaded from /opt/homebrew/anaconda3/lib/python3.11/site-packages/jupyterlab
[I 2025-07-20 13:58:29.167 LabApp] JupyterLab application directory is /opt/homebrew/anaconda3/share/jupyter/lab
[I 2025-07-20 13:58:29.168 LabApp] Extension Manager is 'pypi'.
[I 2025-07-20 13:58:29.169 ServerApp] jupyterlab | extension was successfully loaded.
```

Name	Last Modified	File Size
Applications	9 months ago	
code	3 months ago	
Desktop	29 days ago	
Documents	2 days ago	
Downloads	21 hours ago	
igv	last year	
Movies	2 days ago	
Music	last year	
Pictures	last year	
Public	last year	

Jupyter Notebook就像是程序员的练习本 + 实验台，能一边写代码一边看到结果。

## ► 扩展：Jupyter Notebook

- 基本使用操作，包括创建新文件、创建单元格（Code/Markdown）、运行代码（Shift+Enter）、插入新单元格、保存与导出等。

### Python的语法基础与数据类型

#### Python的语法与缩进

#### Markdown

- Python 是一种解释型语言，不需要编译
- 缩进代表代码结构，缩进错误 = 程序错误
- 每一行代码无需写 ;，以换行作为结束
- 语句末尾常以 : 结尾（如 if、for）

```
print("Hello World!")
```

[26]

... Hello World!

#### Code

Python

#### Output

用 Jupyter，就像边写笔记边调试魔法，做数据分析从此不再黑箱。

# ▶ Python 语法与缩进

- Python 是一种**解释型**语言，不需要编译。
- 每4个空格代表一个**缩进等级**，缩进代表代码结构，缩进错误 = 程序错误。
- 每一行代码无需写`;`，以换行作为结束。
- **一些特殊关键字**（如if、for）语句末尾常以**：结尾**。

记住每4个空格代码一个缩进，特殊关键字如 `if` 等后面必须加:

```
if True:  
    print("Hello World!")  
[3]  ✓  0.0s  
...  Hello World!
```

Python

如果这里 `if` 后面的下面一行没有相应的缩进等级的话就会报错: `IndentationError`

```
if True:  
    print("Hello World!")  
[4]  ✗  0.0s  
...  Cell In[4], line 2  
      print("Hello World!")  
      ^  
      IndentationError: expected an indented block after 'if' statement on line 1
```

Python

# ▶ Python 变量

- 什么是变量：变量就像一个“盒子”，用来**存储数据或信息**。变量有名字（标识符），通过名字可以访问盒子里的内容。在 Python 中，赋值就是创建或改变变量内容的过程。

```
x = 10      # 创建变量 x, 存储整数 10
name = "Tom"  # 创建变量 name, 存储字符串 "Tom"
print(x) # 打印变量 x
print(name) # 打印变量 name
```

[5] ✓ 0.0s

Python

```
... 10
Tom
```

- 变量命名规则：

- 只能包含字母、数字、下划线（\_），且不能以数字开头。
- 不能使用 Python 保留字（如 if、for、class 等）。
- 区分大小写，age 和 Age 是不同变量。
- 尽量用有意义的名字，便于理解代码。

```
# 合法的变量名示例
my_var = 10          # 可以包含字母、数字、下划线，且不能以数字开头
age = 25             # 纯字母变量名
user_name1 = "Alice" # 可以包含数字，但不能开头是数字
_underscore = True    # 变量名可以以下划线开头

print(my_var, age, user_name1, _underscore)

# 以下是非法的变量名，Python 语法会报错，注释中说明原因，运行取消注释符号 # 即可。
# 2nd_place = "second"   # 错误，变量名不能以数字开头
# user-name = "Bob"      # 错误，变量名不能有减号 (-)
# if = 100                # 错误，if 是 Python 保留字，不能用作变量名
```

[6] ✓ 0.0s

Python

```
... 10 25 Alice True
```

# ▶ Python 数据类型

➤ Python 是动态类型语言，**变量赋值时自动确定类型**，常见类型包括：

类型	说明	示例
整数 (int)	整数数字	10, -3, 0
浮点数 (float)	带小数点的数字	3.14, -0.001, 2.0
字符串 (str)	一串文本	"Hello", 'Python'
布尔值 (bool)	True/False	True, False
列表 (list)	有序可变的元素集合	[1, 2, 3], ["a", "b"]
元组 (tuple)	有序不可变的元素集合	(1, 2, 3), ("x", "y")
字典 (dict)	无序键值对集合	{"name": "Tom", "age": 18}
集合 (set)	不重复元素集合	{1, 2, 3}, {"a", "b"}

```
# 整数 (int)
a = 10
print("整数 a =", a, "类型:", type(a))
# 浮点数 (float)
b = 3.14
print("浮点数 b =", b, "类型:", type(b))
# 字符串 (str)
c = "Hello, Python!"
print("字符串 c =", c, "类型:", type(c))
# 布尔值 (bool)
d = True
print("布尔值 d =", d, "类型:", type(d))
# 列表 (list)
e = [1, 2, 3, 4, 5]
print("列表 e =", e, "类型:", type(e))
# 元组 (tuple)
f = (10, 20, 30)
print("元组 f =", f, "类型:", type(f))
# 字典 (dict)
g = {"name": "Alice", "age": 25}
print("字典 g =", g, "类型:", type(g))
# 集合 (set)
h = {1, 2, 3, 4, 4}
print("集合 h =", h, "类型:", type(h)) # 注意集合会自动去重
✓ 0.0s
```

[7] ...  
整数 a = 10 类型: <class 'int'>  
浮点数 b = 3.14 类型: <class 'float'>  
字符串 c = Hello, Python! 类型: <class 'str'>  
布尔值 d = True 类型: <class 'bool'>  
列表 e = [1, 2, 3, 4, 5] 类型: <class 'list'>  
元组 f = (10, 20, 30) 类型: <class 'tuple'>  
字典 g = {'name': 'Alice', 'age': 25} 类型: <class 'dict'>  
集合 h = {1, 2, 3, 4} 类型: <class 'set'>

Python

## ▶ Python 数据类型转换

➤ Python 支持数据类型间转换，转换规则如下：

- 数字与字符串： `int()` 可以将字符串转换为整数，但字符串必须是数字格式； `float()` 可以将字符串转换为浮点数，支持带小数点的数字字符串； `str()` 可以把整数或浮点数转换为字符串。
- 整数与浮点数互转： `float()` 把整数转为浮点数（如  $10 \rightarrow 10.0$ ）； `int()` 把浮点数转为整数，会截断小数部分（如  $3.99 \rightarrow 3$ ）。
- 布尔值与数字互转： `int(True)` 转为 1， `int(False)` 转为 0； `bool()` 将数字转换为布尔值，非零为 `True`，零为 `False`。
- 布尔值与自负互转： 空字符串 `" "` 转为 `False`，非空字符串（如 `"hello"`）转为 `True`； 使用 `str()` 可以将布尔值转换成字符串 `"True"` 或 `"False"`。
- 注意： 转换字符串为数字时，字符串必须符合数字格式，否则会报错； 浮点转整数会舍弃小数部分，不进行四舍五入。

# ▶ Python 运算: 数学运算 (Arithmetic)

- Python 的数学运算符用于执行加减乘除等基本数值计算，是最基础的计算工具。数学运算符号整理如下：

运算符	含义	示例	输出结果
+	加	$2 + 3$	5
-	减	$5 - 2$	3
*	乘	$3 * 4$	12
/	除	$10 / 4$	2.5
//	取整除	$10 // 4$	2
%	取余	$10 \% 4$	2
**	幂 (次方)	$2 ** 3$	8

```
a = 10
b = 3

print("加法:", a + b)
print("减法:", a - b)
print("乘法:", a * b)
print("除法:", a / b)
print("整除:", a // b)
print("取余:", a % b)
print("幂运算:", a ** b)

✓ 0.0s
```

加法: 13  
减法: 7  
乘法: 30  
除法: 3.3333333333333335  
整除: 3  
取余: 1  
幂运算: 1000

# ▶ Python 运算: 关系运算 (Comparison)

- Python 的关系运算符用来比较两个值，判断它们是否相等、大于、小于等，返回布尔值 (True/False)。关系运算符号整理如下：

运算符	含义	示例	输出结果
==	等于	<code>3 == 3</code>	True
!=	不等于	<code>3 != 4</code>	True
>	大于	<code>5 &gt; 2</code>	True
<	小于	<code>2 &lt; 1</code>	False
>=	大于等于	<code>5 &gt;= 5</code>	True
<=	小于等于	<code>3 &lt;= 2</code>	False

```
x = 5
y = 7

print("等于 (==): ", x == y)      # False
print("不等于 (!=): ", x != y)    # True
print("大于 (>): ", x > y)       # False
print("小于 (<): ", x < y)       # True
print("大于等于 (>=): ", x >= y)  # False
print("小于等于 (<=): ", x <= y)  # True

✓ 0.0s
```

等于 (==) : False  
不等于 (!=) : True  
大于 (>) : False  
小于 (<) : True  
大于等于 (>=) : False  
小于等于 (<=) : True

# ▶ Python 运算: 逻辑运算 (Logical)

- Python 的逻辑算符用于多个条件组合判断，常用于控制流程或判断复杂条件。逻辑运算符号整理如下：

运算符	含义	示例	输出结果
and	与	True and False	False
or	或	True and False	True
not	非 (取反)	not True	False

```
x = 5
print("初始值 x = ", x)

x += 2
print("x += 2 ->", x)

x *= 3
print("x *= 3 ->", x)

x -= 4
print("x -= 4 ->", x)

x /= 2
print("x /= 2 ->", x)

✓ 0.0s
```

初始值 x = 5  
x += 2 -> 7  
x \*= 3 -> 21  
x -= 4 -> 17  
x /= 2 -> 8.5

# ▶ Python 运算: 赋值运算 (Assignment)

- Python 的赋值运算符把右边的值或表达式结果赋给变量，还能进行加减乘除的“快捷更新”。赋值运算符号整理如下：

运算符	含义	示例	输出结果
=	赋值	x = 5	
+=	加后赋值	x += 1	x = x + 1
-=	减后赋值	x -= 1	x = x - 1
*=	乘后赋值	x *= 2	x = x * 2
/=	除后赋值	x /= 2	x = x / 2

```
a = 10  
b = 3  
  
print("加法:", a + b)  
print("减法:", a - b)  
print("乘法:", a * b)  
print("除法:", a / b)  
print("整除:", a // b)  
print("取余:", a % b)  
print("幂运算:", a ** b)
```

✓ 0.0s

加法: 13  
减法: 7  
乘法: 30  
除法: 3.333333333333335  
整除: 3  
取余: 1  
幂运算: 1000

# ▶ Python 数据类型转换

```
# 字符串转整数
s = "123"
num = int(s)
print(f"字符串'{s}'转整数:", num, type(num))
# 字符串转浮点数
s_float = "3.1415"
f = float(s_float)
print(f"字符串'{s_float}'转浮点数:", f, type(f))
# 整数转浮点数
i = 10
f2 = float(i)
print(f"整数{i}转浮点数:", f2, type(f2))
# 浮点数转整数（会舍弃小数部分）
f3 = 9.99
i2 = int(f3)
print(f"浮点数{f3}转整数:", i2, type(i2))
# 数字转字符串
num2 = 456
s2 = str(num2)
print(f"数字{num2}转字符串:", s2, type(s2))
# 布尔值转整数
b = True
i3 = int(b)
print(f"布尔值{b}转整数:", i3, type(i3))
# 布尔值转字符串
s3 = str(b)
print(f"布尔值{b}转字符串:", s3, type(s3))
# 字符串转布尔值（空字符串为False，非空字符串为True）
s4 = ""
b2 = bool(s4)
print(f"字符串'{s4}'转布尔值:", b2, type(b2))
s5 = "hello"
b3 = bool(s5)
print(f"字符串'{s5}'转布尔值:", b3, type(b3))
```

[8] ✓ 0.0s

字符串'123'转整数: 123 <class 'int'>  
字符串'3.1415'转浮点数: 3.1415 <class 'float'>  
整数10转浮点数: 10.0 <class 'float'>  
浮点数9.99转整数: 9 <class 'int'>  
数字456转字符串: 456 <class 'str'>  
布尔值True转整数: 1 <class 'int'>  
布尔值True转字符串: True <class 'str'>  
字符串''转布尔值: False <class 'bool'>  
字符串'hello'转布尔值: True <class 'bool'>

## ➤ 总结:

- Python 支持使用内置函数如 `int()`、`float()`、`str()`、`bool()` 进行不同数据类型之间的转换。
- 数值与字符串、布尔值之间可灵活转换，但需确保格式正确，否则会报错。
- 浮点转整数会舍弃小数，空字符串转布尔值为 `False`，非空字符串为 `True`。

## ▶ Python 动态语言的特点

- Python在赋值的时候**无需声明变量类型**， 变量在赋值时自动决定类型， 例如 `x = 10`， Python 会自动将 `x` 识别为整数类型。
- Python的**数据类型可随时改变**， 例如同一个变量可以先是整数， 后变成字符串。
- Python在**运行时进行类型检查**， 类型是在程序运行过程中动态确定的， 出错通常发生在运行阶段而非编译阶段。

```
# Python 是动态类型语言 —— 不需要预先声明变量类型
x = 10          # x 是整数类型
print("x =", x, "类型:", type(x))
x = "Hello, world!"  # 同一个变量 x 被重新赋值为字符串类型
print("x =", x, "类型:", type(x))
x = [1, 2, 3]      # 同一个变量 x 又变成了列表类型
print("x =", x, "类型:", type(x))

# 动态类型的灵活性也容易带来潜在问题:
x = 5
y = "3"
# 下面这行会报错, 因为试图把 int 和 str 相加
# print(x + y) # TypeError: unsupported operand type(s) for +: 'int' and 'str'

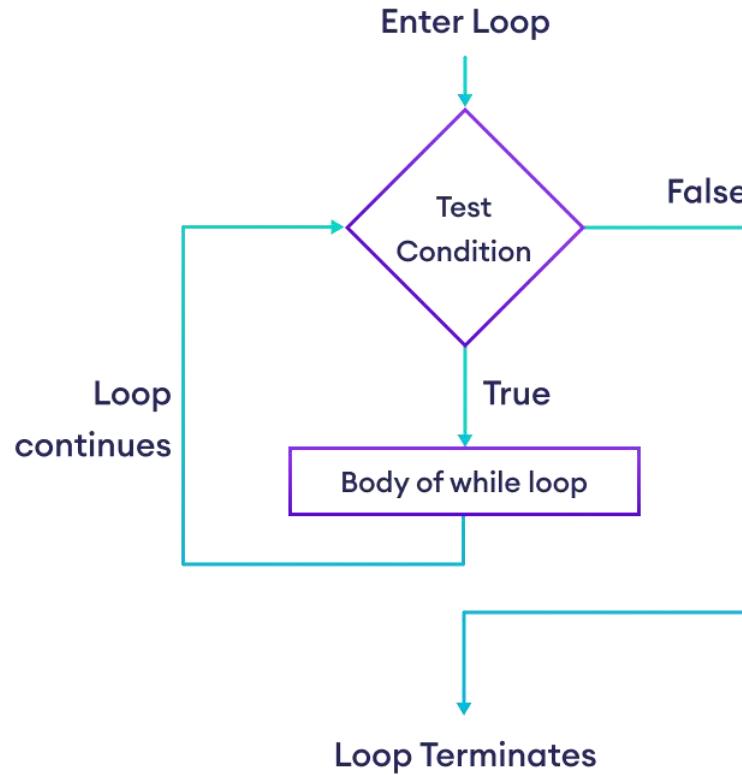
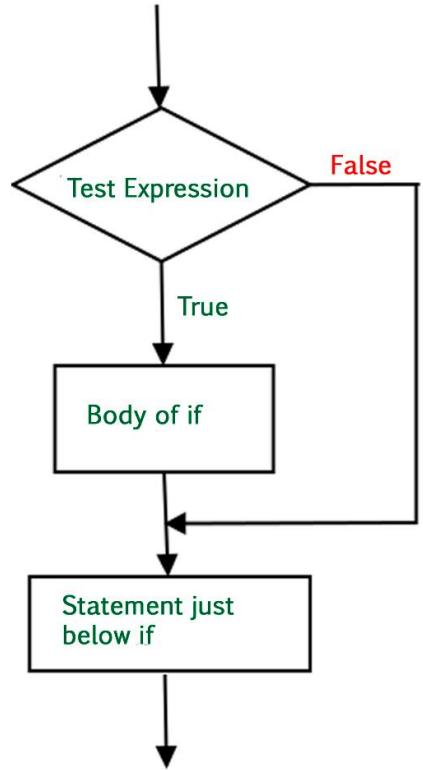
# 正确的做法是先转换类型
print("x + int(y) =", x + int(y))

[9]    ✓  0.0s                                         Python
...
...   x = 10 类型: <class 'int'>
...   x = Hello, world! 类型: <class 'str'>
...   x = [1, 2, 3] 类型: <class 'list'>
...   x + int(y) = 8
```

- 优点：代码简洁、开发效率高；
- 缺点：类型错误可能在运行时才发现，不利于大型项目的类型安全。

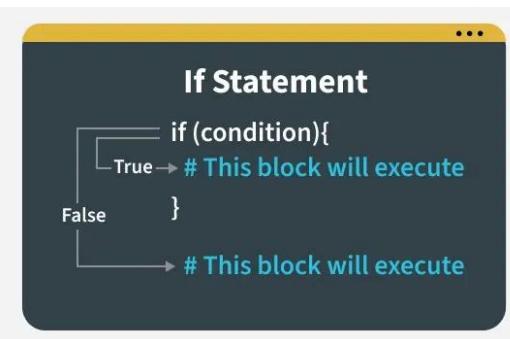
# ▶ Python 条件与循环语句

- 条件语句：用于根据不同的条件决定程序执行哪一部分代码（如：if, if-else, if-elif-else）。
- 循环语句：用于重复执行某段代码，直到满足特定条件（如：for 循环、while 循环）。



它们是控制程序“逻辑流程”的核心工具，让代码更智能、更灵活。

# ► Python 条件与循环语句: if, if-else, if-elif-else

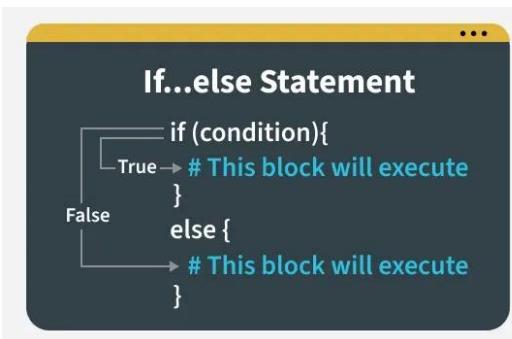


[12]

```
x = 10  
if x > 5:  
    print("x 大于 5")
```

✓ 0.0s

... x 大于 5



[13]

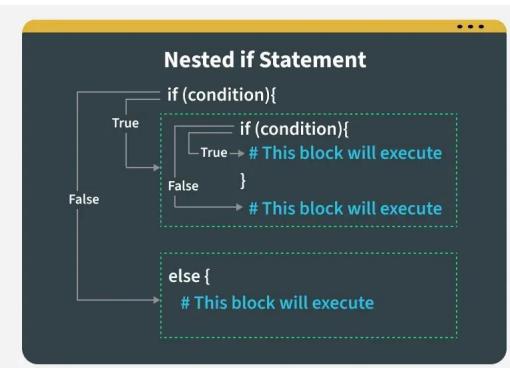
```
x = 3  
if x > 5:  
    print("x 大于 5")  
else:  
    print("x 小于或等于 5")
```

✓ 0.0s

... x 小于或等于 5

if 用于判断一个条件是否为真，条件成立时执行对应代码。

if-else 结构在条件不成立时提供备选的执行路径。

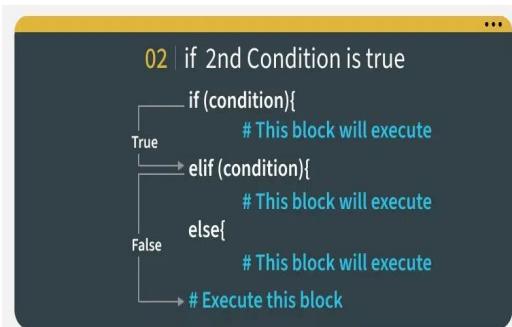


[15]

```
x = 8  
if x > 0:  
    if x < 10:  
        print("x 是一个正的一位数")  
    else:  
        print("x 是一个正的多位数")  
else:  
    print("x 不是正数")
```

✓ 0.0s

... x 是一个正的一位数



[14]

```
score = 75  
  
if score >= 90:  
    print("优秀")  
elif score >= 60:  
    print("及格")  
else:  
    print("不及格")
```

✓ 0.0s

... 及格

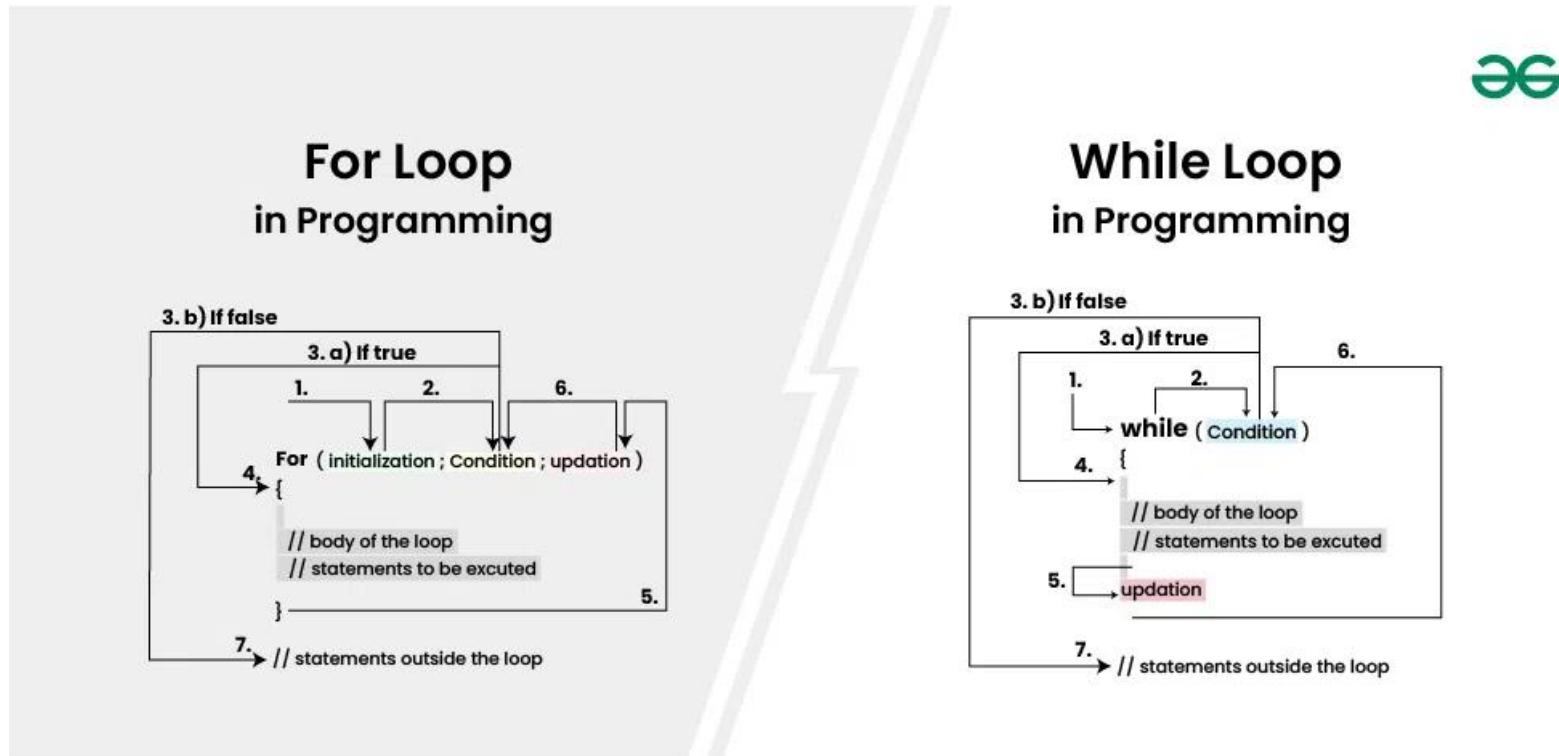
if-elif-else 用于多个条件判断，依次匹配并执行第一个成立的分支。

嵌套 if 让条件判断更灵活，可以在一个判断中再细分逻辑。

**编程的核心在于根据具体任务灵活设计逻辑，if 语句正是实现需求的灵魂工具。**

# ► Python 条件与循环语句: For-Loop, While-Loop

- 编程的精妙在于重复中的控制，for 循环让我们优雅地遍历数据，按部就班完成每一步。
- 程序的灵活源于判断驱动，while 循环让代码在条件满足时持续运转，直到目标达成。



循环结构是编程的动力引擎，for 负责遍历，while 把握时机，助力任务高效完成。

# ▶ Python 函数

- 函数是将一组相关操作封装在一起的“工具”，可以反复调用，避免重复写代码。它接收输入（参数），执行代码逻辑，并返回结果（可选）。Python 使用 **def** 关键字来定义函数。
- 函数的作用：封装一段可复用的代码逻辑，提高代码模块化和可读性
- 函数的基本语法：关键字**def**/函数名/**文档说明**/函数体/**参数**/**返回值....**

```
def 函数名(参数1, 参数2, ...):  
    """可选的文档说明"""  
    函数体  
    return 返回值 (可选)
```

```
def greet(name):  
    print(f"Hello, {name}!")  
greet("Alice")  
[10] ✓ 0.0s  
... Hello, Alice!  
Python
```

函数可以没有返回值

```
def add(a, b=5):  
    return a + b  
print(add(3))  
print(add(3, 6))  
[11] ✓ 0.0s  
... 8  
9  
Python
```

函数有位置参数和关键字参数

## ▶ Python 小结

- 变量：变量是存储数据的“容器”，用名字访问，变量命名需遵守相应规则。
- 各种运算是编程中的基础工具，帮助我们实现计算、判断、赋值与逻辑控制，让代码具备思考与行动能力。
- 数据类型转换：Python 支持通过内置函数在不同数据类型之间进行显式转换，使变量在不同上下文中灵活使用。
- Python 动态语言：Python 自动推断变量类型，支持多种基本和复杂数据类型。Python 在**运行时进行类型检查**，类型是在程序运行过程中动态确定的，出错通常发生在运行阶段而非编译阶段。
- 条件语句：通过 if、elif 和 else 判断不同条件是否成立，从而控制程序在不同情况下执行不同的代码块，实现分支逻辑。
- 循环语句：使用 for 或 while 重复执行某段代码，直到满足结束条件，可结合 break 和 continue 控制循环流程，提高效率和灵活性。
- 函数：将具有特定功能的代码封装成一个模块，通过函数名和参数调用，提高代码复用性、可读性和逻辑组织能力，是结构化编程的核心。

# 目录章节

---

CONTENTS

01

语法基础与数据类型

02

Numpy：数值计算

03

Pandas：数据处理

04

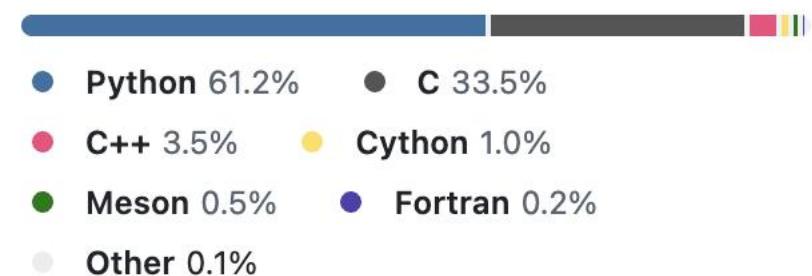
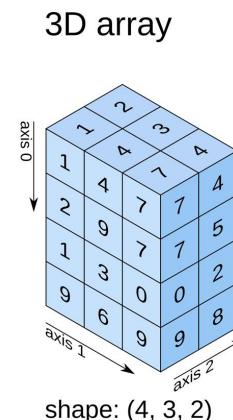
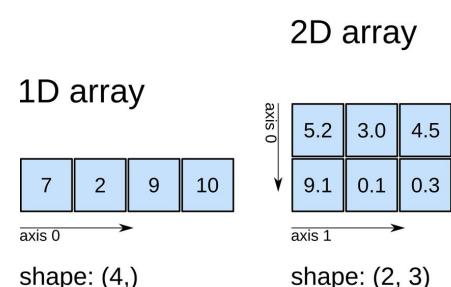
Matplotlib：数据可视化

05

Scikit-learn：机器学习

# ▶ Numpy是什么？

- NumPy 是 Python 中进行科学计算和数值处理的基础库，提供了高性能**多维数组对象（ndarray）**，以及**丰富的数学函数**。是许多机器学习和数据科学库（如 Pandas、Pytorch、Scikit-learn）的底层依赖。
- NumPy的优势：1) 效率高，使用**底层 C 实现**，处理大规模数据比原生 Python 快数十倍；2) 功能强：支持广播（broadcasting）、**矩阵运算**、线性代数、随机数生成等；3) 操作简洁：支持**向量化操作**，避免写复杂循环。

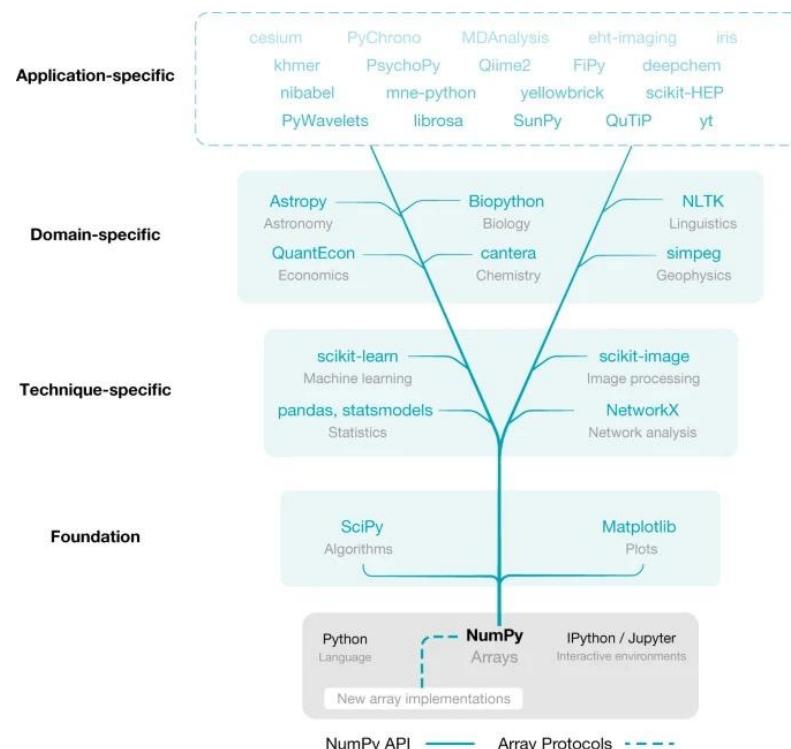


C/C++底层支持占比高达38%

**NumPy：程序员的“数值外挂”，搞定数组像切菜，没它，数据都懒得理你！**

# ▶ Numpy的应用场景

- NumPy 广泛应用于数据科学、机器学习、图像处理、金融分析与科学计算等领域；通过高效的**多维数组和向量化操作**，为大规模数值计算提供了**基础支撑**。



## 主要应用场景：

- 数据科学：批量数据预处理、数值分析、数据清洗（如归一化、缺失值填充）。
- 机器学习：特征矩阵构建、模型输入张量处理（如输入数据标准化）。
- 图像处理：图像像素是二维/三维数组，NumPy 用于像素级操作。
- 金融分析：股票价格矩阵、收益率计算、协方差矩阵构建。
- 科学计算：仿真模拟、微积分、矩阵求解、线性代数计算。
- 深度学习底层：PyTorch / TensorFlow 背后的张量操作类似 NumPy。

**NumPy：程序员的“数值外挂”，搞定数组像切菜，没它，数据都懒得理你！**

## ► Numpy基础：核心数据结构ndarray

- ndarray是NumPy的核心数据结构，全称是 N-dimensional array（多维数组）。
- NumPy 数据结构专为**高效数值计算**而设计，支持**向量化操作**（速度快），所有元素类型一致（更节省内存）
- ndarray 的特点：1) 支持一维、二维、三维甚至更多维度；2) 可快速进行切片、索引、运算、聚合；3) 常见属性：.ndim（维度）、.shape（形状）、.dtype（元素类型）

```
import numpy as np  
  
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(f"arr:{arr}")  
print(f"arr数组的维度:{arr.ndim}")  
print(f"arr数组的形状:{arr.shape}")  
print(f"arr数组元素的类型:{arr.dtype}")  
  
[50] ✓ 0.0s
```

```
... [[1 2 3]  
 [4 5 6]  
 [7 8 9]]  
arr数组的维度: 2  
arr数组的形状: (3, 3)  
arr数组元素的类型: int64
```

```
print(arr[0, 1])      # 第一行第二列: 2  
print(arr[1])        # 第二行: [4 5 6]  
print(arr[:, 1])     # 第二列: [2 5 8]  
print(arr[0:2, 1:])   # 子矩阵: 取前两行, 从第2列开始  
  
[51] ✓ 0.0s
```

```
... 2  
[4 5 6]  
[2 5 8]  
[[2 3]  
 [5 6]]
```

# ► Numpy基础：核心数据结构 ndarray

```
print(arr + 10)      # 每个元素加 10  
print(arr * 2)      # 每个元素乘以 2
```

[52]

✓ 0.0s

```
... [[11 12 13]  
 [14 15 16]  
 [17 18 19]]  
[[ 2  4  6]  
 [ 8 10 12]  
 [14 16 18]]
```

➤ ndarray 是 NumPy 的核心数据结构，用于**高效存储和操作同类型的多维数组数据**。

```
print(f"数组的总和: {arr.sum()}")      # 总和: 45  
print(f"数组的平均值: {arr.mean()}")    # 平均值: 5.0  
print(f"数组中元素的最大值为: {arr.max()}, 最小值为: {arr.min()}") # 最大值与最小值: 9 1  
print(f"数组按列求和: {arr.sum(axis=0)}") # 按列求和: [12 15 18]  
print(f"数组按行求和: {arr.sum(axis=1)}") # 按行求和: [ 6 15 24]
```

[53]

✓ 0.0s

```
... 数组的总和: 45  
数组的平均值: 5.0  
数组中元素的最大值为: 9, 最小值为: 1  
数组按列求和: [12 15 18]  
数组按行求和: [ 6 15 24]
```

# ▶ Numpy操作： 数组创建方式

方法	示例
np.array()	np.array([1,2,3])
np.zeros((2, 3))	全0, 形状为2行3列
np.ones((2,3))	全1, 形状为2行3列
np.full((2,2), 5)	指定填充值5, 形状为2行2列
np.arange(0, 10, 2)	步长生成数组
np.linspace(0, 1, 5)	从范围为0, 1均分生成 5 个点
np.random.rand(2,3)	随机生成 0~1 之间
np.eye(3)	单位矩阵, 形状为3行3列

```
# 1. 从列表创建
a = np.array([1, 2, 3])
print("\nnp.array([1, 2, 3]) =", a)

# 2. 全 0 数组
b = np.zeros((2, 3))
print("\nnp.zeros((2, 3)) =", b)

# 3. 全 1 数组
c = np.ones((2, 3))
print("\nnp.ones((2, 3)) =", c)

# 4. 指定值填充
d = np.full((2, 2), 5)
print("\nnp.full((2, 2), 5) =", d)

# 5. 步长生成
e = np.arange(0, 10, 2)
print("\nnp.arange(0, 10, 2) =", e)

# 6. 均分生成
f = np.linspace(0, 1, 5)
print("\nnp.linspace(0, 1, 5) =", f)

# 7. 随机生成 (0~1 之间)
g = np.random.rand(2, 3)
print("\nnp.random.rand(2, 3) =", g)

# 8. 单位矩阵
h = np.eye(3)
print("\nnp.eye(3) =", h)

[60]   ✓ 1.2s
...
np.array([1, 2, 3]) = [1 2 3]

np.zeros((2, 3)) =
[[0. 0. 0.]
 [0. 0. 0.]]

np.ones((2, 3)) =
[[1. 1. 1.]
 [1. 1. 1.]]

np.full((2, 2), 5) =
[[5 5]
 [5 5]]

np.arange(0, 10, 2) = [0 2 4 6 8]

np.linspace(0, 1, 5) = [0. 0.25 0.5 0.75 1. ]

np.random.rand(2, 3) =
[[0.89521793 0.54845126 0.40721121]
 [0.77804314 0.02779394 0.47902974]]

np.eye(3) =
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

# ▶ Numpy操作： 数组索引与切片

方法	示例代码	说明
基本切片	arr[0:2, 1:]	行从0到1，列从1到最后，二维切片
获取单个元素	arr[2, 1]	获取第3行第2列的元素（索引从0开始）
步长切片	arr[::-2, ::2]	每隔一行/一列取一个值
整数索引 (花式索引)	arr[[0, 2], [1, 2]]	获取位置(0,1)和(2,2)的元素
列切片	arr[:, 1]	获取第2列所有行的数据
行切片	arr[1, :]	获取第2行所有列的数据
布尔索引	arr[arr > 50]	获取所有大于50的元素
修改部分值	arr[0:2, 0:2] = 99	将左上角2×2区域的值全部修改为99

```
arr = np.array([[10, 20, 30],  
               [40, 50, 60],  
               [70, 80, 90]])  
  
print("原始数组: \n", arr)  
  
# 1. 基本切片  
print("\n切片 arr[0:2, 1:] :\n", arr[0:2, 1:])  
  
# 2. 获取单个元素  
print("\n获取 arr[2, 1] :\n", arr[2, 1])  
  
# 3. 步长切片  
print("\n步长切片 arr[::-2, ::2] :\n", arr[::-2, ::2])  
  
# 4. 整数索引 (花式索引)  
print("\n花式索引 arr[[0, 2], [1, 2]] :\n", arr[[0, 2], [1, 2]])  
  
# 5. 获取某一列  
print("\n第二列 arr[:, 1] :\n", arr[:, 1])  
  
# 6. 布尔索引  
print("\n布尔索引 arr[arr > 50] :\n", arr[arr > 50])  
  
# 7. 修改部分值 (切片赋值)  
arr_copy = arr.copy() # 保留原数组  
arr_copy[0:2, 0:2] = 99  
print("\n修改 arr[0:2, 0:2] = 99 :\n", arr_copy)  
✓ 0.0s  
...  
原始数组:  
[[10 20 30]  
 [40 50 60]  
 [70 80 90]]  
  
切片 arr[0:2, 1:]:  
[[20 30]  
 [50 60]]  
  
获取 arr[2, 1]: 80  
  
步长切片 arr[::-2, ::2]:  
[[10 30]  
 [70 90]]  
  
花式索引 arr[[0, 2], [1, 2]]: [20 90]  
  
第二列 arr[:, 1]: [20 50 80]  
  
布尔索引 arr[arr > 50]: [60 70 80 90]  
  
修改 arr[0:2, 0:2] = 99:  
[[99 99 30]  
 [99 99 60]  
 [70 80 90]]
```

# ▶ Numpy操作：数组变形与拼接

操作类型	方法/函数	示例代码	说明
变形	reshape()	arr.reshape(2, 6)	改变形状为 2 行 6 列
	ravel()	arr.ravel()	将数组展开为一维（返回拷贝）
	flatten()	arr.flatten()	同上，但始终返回拷贝
	transpose()	arr.T 或 arr.transpose()	转置数组（行列互换）
拼接	concatenate()	np.concatenate([a, b], axis=0)	沿指定轴连接数组
	vstack()	np.vstack([a, b])	垂直方向拼接
	hstack()	np.hstack([a, b])	水平方向拼接
	stack()	np.stack([a, b], axis=0)	增加新维度后拼接

```
# 创建一个基础数组
arr = np.arange(12) # [0, 1, 2, ..., 11]
print("原始数组: ", arr)

# 1. reshape 变形
reshaped = arr.reshape(2, 6)
print("\nreshape(2,6):\n", reshaped)

# 2. ravel 与 flatten 展开
print("\nravel():", reshaped.ravel())      # 返回视图
print("flatten():", reshaped.flatten())   # 返回副本

# 3. transpose 转置
transposed = reshaped.T
print("\n转置 reshape(2,6).T:\n", transposed)

# 4. 拼接操作
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

print("\na:\n", a)
print("\nb:\n", b)

# concatenate
print("\nconcatenate axis=0:\n", np.concatenate([a, b], axis=0))
print("concatenate axis=1:\n", np.concatenate([a, b], axis=1))

# vstack / hstack
print("\nvstack:\n", np.vstack([a, b]))
print("hstack:\n", np.hstack([a, b]))

# stack
print("\nstack axis=0:\n", np.stack([a, b], axis=0)) # shape: (2,2,2)
print("stack axis=1:\n", np.stack([a, b], axis=1)) # shape: (2,2,2)
```

[62]

Python

```
原始数组: [ 0  1  2  3  4  5  6  7  8  9 10 11]

reshape(2,6):
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]

ravel():
[ 0  1  2  3  4  5  6  7  8  9 10 11]
flatten():
[ 0  1  2  3  4  5  6  7  8  9 10 11]

转置 reshape(2,6).T:
[[ 0  6]
 [ 1  7]
 [ 2  8]
 [ 3  9]
 [ 4 10]
 [ 5 11]]
```

```
a:
[[1 2]
 [3 4]]
b:
[[5 6]
 [7 8]]
```

```
concatenate axis=0:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
concatenate axis=1:
[[1 2 5 6]
 [3 4 7 8]]

vstack:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
hstack:
[[1 2 5 6]
 [3 4 7 8]]

stack axis=0:
[[[1 2]
 [3 4]]]
```

```
[[5 6]
 [7 8]]]
stack axis=1:
[[[1 2]
 [5 6]
 [3 4]
 [7 8]]]
```

# ▶ Numpy操作：基本计算（元素级）

## 操作类型

### 示例代码

### 说明

加法

$a + b$

对应元素相加

减法

$a - b$

对应元素相减

乘法

$a * b$

对应元素相乘

除法

$a / b$

对应元素相除

幂运算

$a ** 2$

所有元素平方

取余

$a \% b$

对应元素取余

比较运算

$a > b, a == b$

返回布尔数组

广播机制

$a + 10, a * [1,2,3]$

支持形状不同数组运算

通用函数

`np.sqrt(a), np.sin(a)`

NumPy 提供的 ufunc,  
元素级操作

```
# 创建两个相同形状的数组
a = np.array([[1, 2, 3],
              [4, 5, 6]])
b = np.array([[6, 5, 4],
              [3, 2, 1]])

# 基本算术运算
print("a + b:\n", a + b)
print("a - b:\n", a - b)
print("a * b:\n", a * b) # 元素相乘
print("a / b:\n", a / b)

# 幂运算 & 取余
print("a ** 2:\n", a ** 2)
print("a % 2:\n", a % 2)

# 比较运算
print("a > b:\n", a > b)
print("a == b:\n", a == b)

# 广播机制
print("a + 10:\n", a + 10)
print("a * np.array([1,2,3]):\n", a * np.array([1, 2, 3]))

# 通用函数 (ufunc)
print("np.sqrt(a):\n", np.sqrt(a))
print("np.sin(a):\n", np.sin(a))

[63] ✓ 0.0s
```

```
a + b:
[[7 7 7]
 [7 7 7]]
a - b:
[[-5 -3 -1]
 [ 1  3  5]]
a * b:
[[ 6 10 12]
 [12 10  6]]
a / b:
[[0.16666667 0.4           0.75      ]
 [1.33333333 2.5           6.         ]]
a ** 2:
[[ 1  4  9]
 [16 25 36]]
a % 2:
[[1 0 1]
 [0 1 0]]
a > b:
[[False False False]
 [ True  True  True]]
a == b:
[[False False False]
 [False False False]]
a + 10:
[[11 12 13]
 [14 15 16]]
a * np.array([1,2,3]):
[[ 1  4  9]
 [ 4 10 18]]
np.sqrt(a):
[[1.          1.41421356 1.73205081]
 [2.          2.23606798 2.44948974]]
np.sin(a):
[[ 0.84147098  0.90929743  0.14112001]
 [-0.7568025   -0.95892427 -0.2794155 ]]
```

# ▶ Numpy操作：聚合函数与统计分析

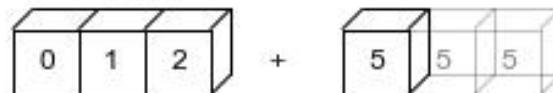
函数	示例代码	说明
np.sum()	np.sum(a)	所有元素求和
np.mean()	np.mean(a)	求平均值
np.median()	np.median(a)	求中位数
np.std()	np.std(a)	标准差
np.var()	np.var(a)	方差
np.min() / np.max()	np.min(a) / np.max(a)	最小值 / 最大值
np.argmin() / np.argmax()	np.argmin(a)	最小值 / 最大值索引
np.percentil e()	np.percentile(a, 50)	百分位数
np.cumsum ( )	np.cumsum(a)	累加和
np.cumpro d()	np.cumprod(a)	累加积

```
a = np.array([[1, 2, 3],  
             [4, 5, 6]])  
  
print("数组:\n", a)  
  
# 总和、均值、中位数  
print("总和:", np.sum(a))  
print("均值:", np.mean(a))  
print("中位数:", np.median(a))  
  
# 最大、最小与对应索引  
print("最大值:", np.max(a))  
print("最小值:", np.min(a))  
print("最大值索引 (展平后):", np.argmax(a))  
  
# 标准差、方差  
print("标准差:", np.std(a))  
print("方差:", np.var(a))  
  
# 百分位  
print("50%分位数 (中位数):", np.percentile(a, 50))  
  
# 累加 & 累乘  
print("累加和:", np.cumsum(a))  
print("累积积:", np.cumprod(a))  
  
# 指定轴进行聚合  
print("每列的和 [axis=0]:", np.sum(a, axis=0))  
print("每行的平均 [axis=1]:", np.mean(a, axis=1))  
  
[64] ✓ 0.0s  
...  
数组:  
[[1 2 3]  
 [4 5 6]]  
总和: 21  
均值: 3.5  
中位数: 3.5  
最大值: 6  
最小值: 1  
最大值索引 (展平后): 5  
标准差: 1.707825127659933  
方差: 2.9166666666666665  
50%分位数 (中位数): 3.5  
累加和: [ 1 3 6 10 15 21]  
累积积: [ 1 2 6 24 120 720]  
每列的和 (axis=0): [5 7 9]  
每行的平均 (axis=1): [2. 5.]
```

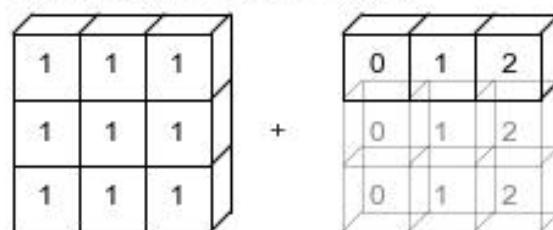
## ► Numpy操作：广播机制（Broadcasting）

- 广播（Broadcasting）是 NumPy 在执行不同形状数组之间的算术运算时的一种自动扩展机制，使数组在维度不一致时也能进行计算，无需显式复制数据。
- 核心规则：1) 从尾部对齐维度（右对齐）；2) 某一维度相同或其中一个为1，可以广播；3) 不能满足规则就报错（ValueError）

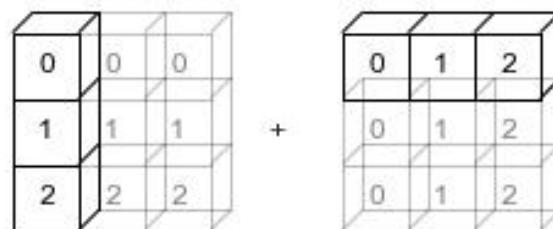
`np.arange(3)+5`



`np.ones((3, 3))+np.arange(3)`



`np.arange(3).reshape((3, 1))+np.arange(3)`



```
a = np.array([[1, 2, 3],  
             [4, 5, 6]]) # shape: (2, 3)  
b = np.array([10, 20, 30]) # shape: (3, )  
  
print("a + b = \n", a + b)
```

[65]

✓ 0.0s

```
... a + b =  
[[11 22 33]  
[14 25 36]]
```

# ► Numpy案例： 数据标准化与图像灰度处理

数据标准化：标准化是机器学习中常见的预处理操作，将特征缩放到均值为 0，标准差为 1。

```
data = np.array([5.0, 10.0, 15.0])
print(f"数组标准化之前: {data}")

# 数据标准化: (数据值-平均值) / 标准差
mean = data.mean() # 计算数组的平均值
std = data.std() # 计算数组的标准差
standardized = (data - mean) / std # 标准化

print(f"数组标准化之后: {standardized}")
```

[54] ✓ 0.0s

```
... 数组标准化之前: [ 5. 10. 15.]
数组标准化之后: [-1.22474487  0.           1.22474487]
```

```
from PIL import Image
import numpy as np
import requests
from io import BytesIO
import matplotlib.pyplot as plt

# 1. 加载图像
url = "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRHAZgpQV2hjpKQczT_mQdDxigzJkv1HwX9BA&q=tbo"
response = requests.get(url)
img = Image.open(BytesIO(response.content)).convert("RGB") # 确保是RGB图像

# 2. 转为 NumPy 数组并灰度化 (加权平均法)
img_np = np.array(img)
# 灰度计算原理: gray = 0.299 * R + 0.587 * G + 0.114 * B
gray_np = np.dot(img_np[:, :, :3], [0.299, 0.587, 0.114]).astype(np.uint8)

# 3. 显示原图和灰度图
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(img_np)
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Grayscale")
plt.imshow(gray_np, cmap="gray")
plt.axis("off")

plt.tight_layout()
plt.show()
```

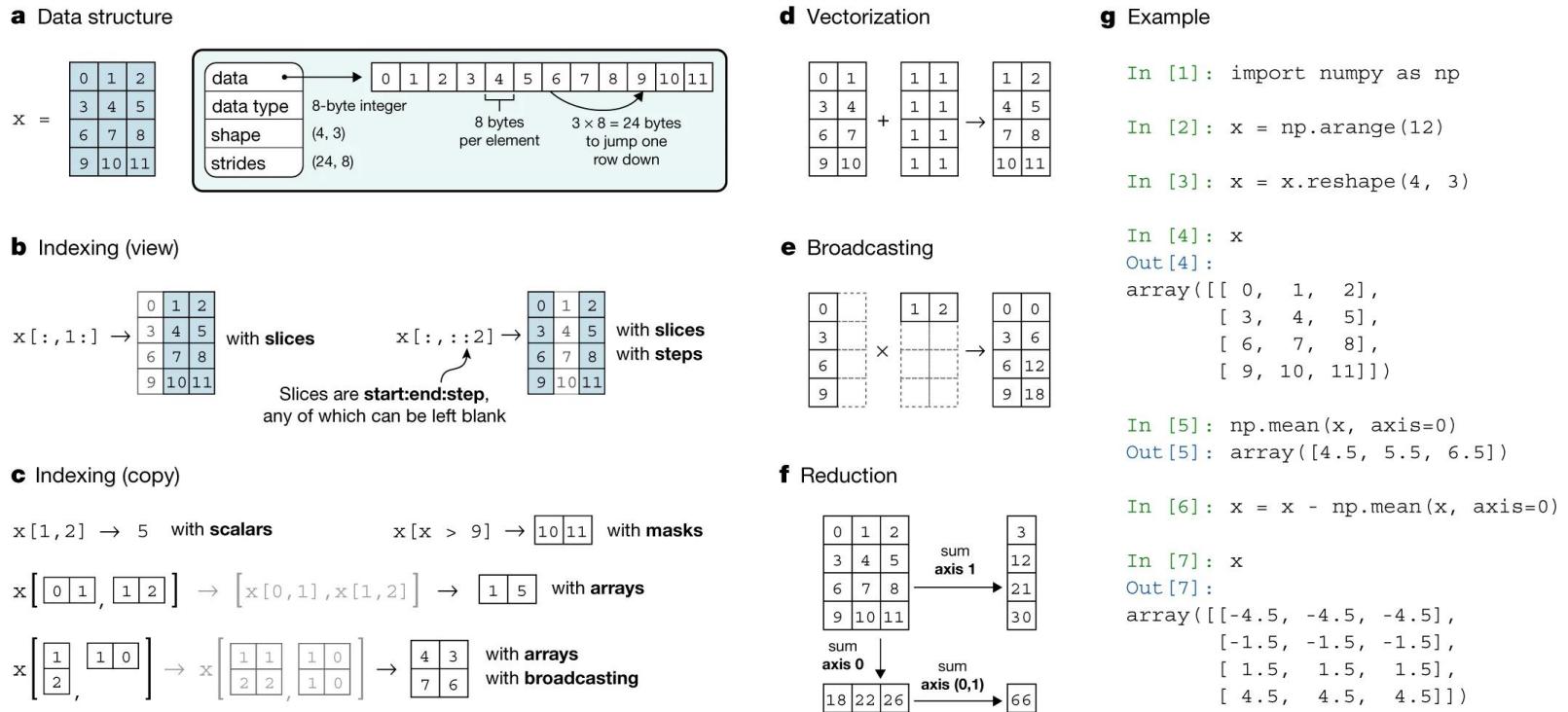
[59] ✓ 1.6s Python



➤ NumPy 可高效实现数据标准化与图像灰度处理，是数值计算与图像分析的利器。

# ▶ 小结

- NumPy 是 Python 科学计算的核心库，提供高性能的 多维数组对象 ndarray。
- 支持向量化操作（更快更简洁），广泛用于数据处理、机器学习、图像分析等领域。常用的操作：数组创建、访问与切片、数据变形与拼接、基本计算、聚合分析、广播机制



NumPy 是高效科学计算的利器，掌握它，等于打开数据分析和 AI 的大门！

# 目录章节

---

## CONTENTS

01

语法基础与数据类型

02

Numpy：数值计算

03

Pandas：数据处理

04

Matplotlib：数据可视化

05

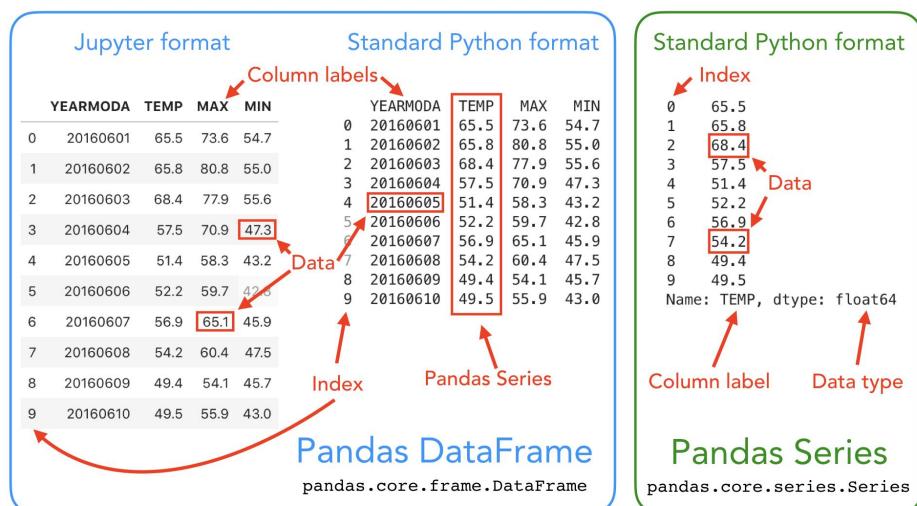
Scikit-learn：机器学习

# ► Pandas 是什么？

- Pandas 是基于 NumPy 的高级数据处理库，专为结构化数据（如表格）而设计，它提供了两个核心数据结构：**Series（一维）** 和 **DataFrame（二维）**，让数据处理像操作 Excel 一样简单。
- Pandas 的优势：1) **数据结构灵活**：提供 Series 和 DataFrame，适合处理一维、二维表格型数据；2) **功能丰富**：支持数据清洗、筛选、合并、分组统计、透视表、时间序列等高级操作；3) **使用便捷**：语法直观，集成性强，能与 NumPy、Matplotlib 等无缝对接。



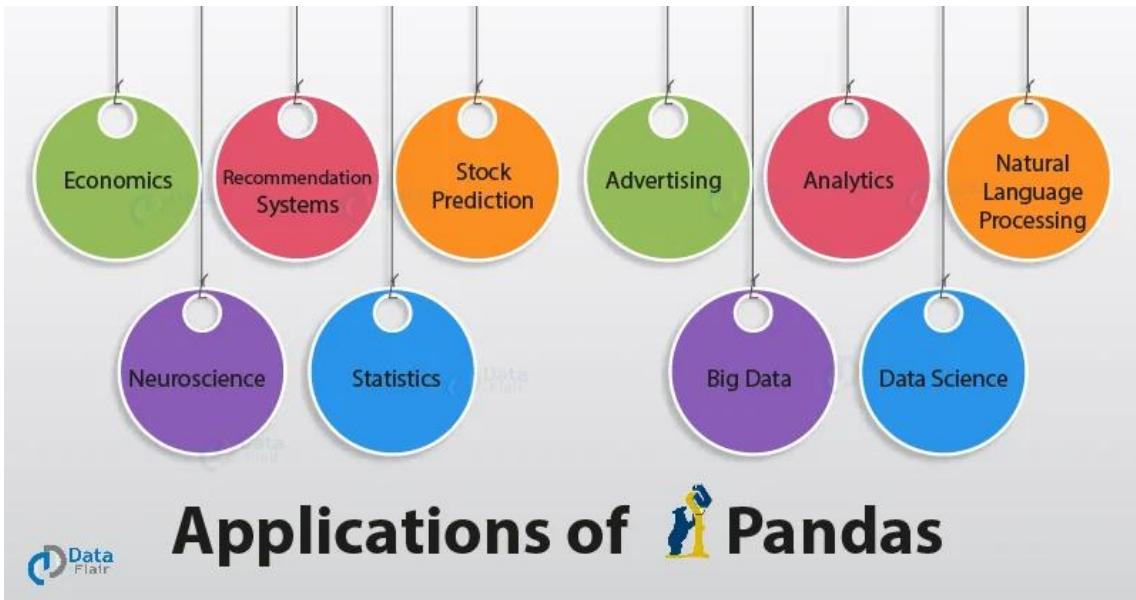
Notes: Pandas = “Panel Data” + “Python”



**Pandas：程序员的“数据保姆”，表格、统计样样通，数据整活全靠它！**

# ► Pandas 的应用场景

- Pandas 是基于 Python 的强大数据处理工具，支持高效的**表格型数据读取、清洗、分析与可视化**，是数据分析和科学计算中的核心组件。

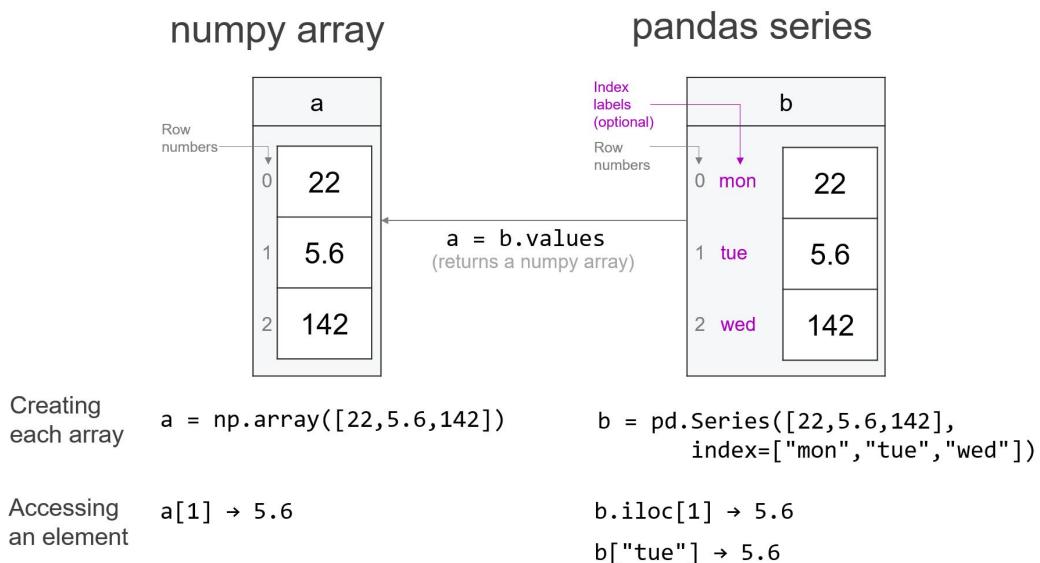
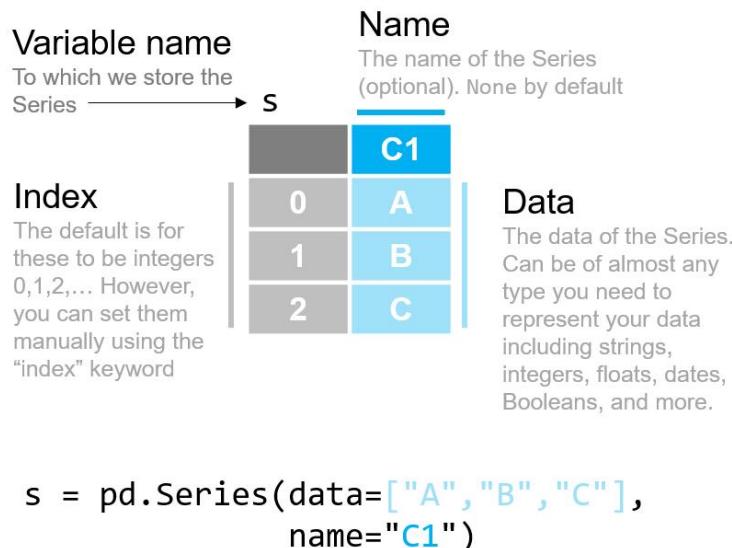


场景	描述
数据清洗	缺失值填充、重复值处理、格式统一、数据标准化等
数据分析	分组统计、聚合分析、时间序列处理
数据可视化	与Matplotlib/Seaborn配合快速绘图
数据读取与导出	支持Excel、CSV、JSON、SQL等多种格式
探索性数据分析(EDA)	快速了解数据结构、分布、关系
报表生成与导出	结合样式和分组功能生成漂亮的输出

Pandas 提供了类似 Excel 的强大数据操作能力，是数据处理与分析的核心利器。

# ▶ Pandas基础：核心数据结构Series

- Series 是 Pandas 中的“一维数据容器”，带标签的数组结构，既能像数组一样操作，也能像字典一样索引。支持 NumPy 运算、切片、索引、自动对齐等操作
  - Series 的特点：1) 本质是**带标签（索引）的一维数组**；2) 支持切片、索引、数学运算、逻辑判断等操作；3) 常见属性：`.index`（索引）、`.values`（值）、`.dtype`（元素类型）

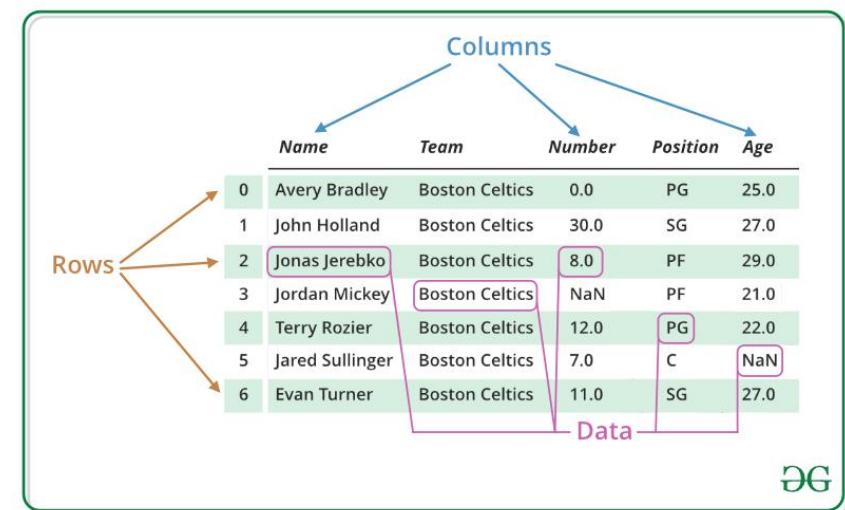


`Series` 是 Pandas 中用于表示一维标记数组的基本对象，常用于表示单列数据。

# ► Pandas基础：核心数据结构DataFrame

- DataFrame 是 Pandas 中的“二维表格数据结构”，类似 Excel 表格，支持行列标签，能高效处理结构化数据。
- DataFrame 的特点：1) 本质是由**多个 Series 构成的二维表格**；2) 支持按行列访问、统计分析、缺失值处理、合并分组等操作；3) 常见属性：`.shape`（形状）、`.columns`（列标签）、`.index`（行索引）、`.dtypes`（各列数据类型）

Series 1		Series 2		Series 3		Dataframe			
INDEX	DATA	INDEX	DATA	INDEX	DATA	INDEX	SERIES 1	SERIES 2	SERIES 3
0	A	0	1	0	[1, 2]	0	A	1	[1, 2]
1	B	1	2	1	A	1	B	2	A
2	C	2	3	2	1	2	C	3	1
3	D	3	4	3	(4, 5)	3	D	4	(4, 5)
4	E	4	5	4	{"a": 1}	4	E	5	{"a": 1}
5	F	5	6	5	6	5	F	6	6



DataFrame是Pandas最常用的数据结构，适用于处理**表格型数据**，功能强大。

# ► Pandas操作：创建Series

方法	示例代码	说明
从列表创建	<code>pd.Series([10, 20, 30])</code>	默认整数索引
自定义索引	<code>pd.Series([10, 20, 30], index=['a', 'b', 'c'])</code>	指定标签索引
从Numpy数组 创建	<code>pd.Series(np.array([1, 2, 3]))</code>	支持 ndarray 数据
从字典创建	<code>pd.Series({'a': 1, 'b': 2})</code>	自动使用字典键作为索引
指定数据类型	<code>pd.Series([1, 2, 3], dtype='float32')</code>	显式设定数据类型
单一标量扩展	<code>pd.Series(5, index=['a', 'b', 'c'])</code>	标量值会被复制到每个索引
从已有Series 创建	<code>pd.Series(existing_series)</code>	创建副本或变体

```
import pandas as pd
import numpy as np

# 1. 从列表创建
s1 = pd.Series([10, 20, 30])
print("默认索引:\n", s1)

# 2. 自定义索引
s2 = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print("\n自定义索引:\n", s2)

# 3. 从字典创建
s3 = pd.Series({'x': 100, 'y': 200})
print("\n从字典创建:\n", s3)

# 4. 标量扩展
s4 = pd.Series(7, index=['a', 'b', 'c'])
print("\n标量扩展:\n", s4)

# 5. 修改数据类型
s5 = pd.Series([1, 2, 3], dtype='float64')
print("\n修改数据类型:\n", s5)

[67] ✓ 0.0s

Python
... 默认索引:
0    10
1    20
2    30
dtype: int64

... 自定义索引:
a    10
b    20
c    30
dtype: int64

... 从字典创建:
x    100
y    200
dtype: int64

... 标量扩展:
a    7
b    7
c    7
dtype: int64

... 修改数据类型:
0    1.0
1    2.0
2    3.0
dtype: float64
```

# ► Pandas操作：创建DataFrame

方法	示例代码	说明
从字典（列表值）	<code>pd.DataFrame({'A': [1, 2], 'B': [3, 4]})</code>	最常见方式，列名为键，值为列表
从列表（嵌套列表）	<code>pd.DataFrame([[1, 2], [3, 4]], columns=['A', 'B'])</code>	数据为二维列表，需显式指定列明
从字典（Series值）	<code>pd.DataFrame({'A': pd.Series([1, 2]), 'B': pd.Series([3, 4])})</code>	自动对齐索引
从 NumPy ndarray	<code>pd.DataFrame(np.array([[1, 2], [3, 4]]), columns=['A', 'B'])</code>	从Numpy数组转换，需指定列明
从字典（字典值）	<code>pd.DataFrame({'row1': {'A': 1}, 'row2': {'A': 2}})</code>	字典嵌套字典，常用于构建行索引结构
从列表字典	<code>pd.DataFrame([{'A': 1}, {'A': 2}, {'B': 3}])</code>	列不齐也可构建，缺失值填 NaN
从结构化数组	<code>pd.DataFrame(np.array([(1,2)]), dtype=[('A','i4'),('B','i4')]))</code>	从 Numpy 结构化数组创建
从已有 Series 组合	<code>pd.DataFrame({'A': s1, 'B': s2})</code>	组合多个 Series 组成 DataFrame

# ► Pandas操作：创建DataFrame

```
# 1. 从字典 (列表为值)
df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print("1. 字典 (列表值):\n", df1, '\n')

# 2. 从嵌套列表 (需指定列名)
df2 = pd.DataFrame([[1, 2], [3, 4]], columns=['A', 'B'])
print("2. 嵌套列表:\n", df2, '\n')

# 3. 从字典 (值为 Series)
s1 = pd.Series([1, 2], index=[0, 1])
s2 = pd.Series([3, 4], index=[0, 1])
df3 = pd.DataFrame({'A': s1, 'B': s2})
print("3. 字典[Series值]:\n", df3, '\n')

# 4. 从 NumPy ndarray
array = np.array([[1, 2], [3, 4]])
df4 = pd.DataFrame(array, columns=['A', 'B'])
print("4. Numpy ndarray:\n", df4, '\n')

# 5. 从字典 (字典为值) —行为键
df5 = pd.DataFrame({'row1': {'A': 1, 'B': 2}, 'row2': {'A': 3, 'B': 4}})
print("5. 字典[字典值, orient='index'):\n", df5, '\n')

# 6. 从列表字典 (列不一定齐全)
df6 = pd.DataFrame([{'A': 1}, {'A': 2, 'B': 3}])
print("6. 列表字典:\n", df6, '\n')

# 7. 从结构化 NumPy 数组
structured_array = np.array([(1, 2), (3, 4)], dtype=[('A', 'i4'), ('B', 'i4')])
df7 = pd.DataFrame(structured_array)
print("7. 结构化数组:\n", df7, '\n')

# 8. 从已有 Series 组合
df8 = pd.DataFrame({'A': pd.Series([10, 20]), 'B': pd.Series([30, 40])})
print("8. Series组合:\n", df8, '\n')

# 9. 创建空 DataFrame
df9 = pd.DataFrame()
print("9. 空 DataFrame:\n", df9, '\n')

✓ 0.0s
```

1. 字典 (列表值) :

A	B
0	1 3
1	2 4

2. 嵌套列表:

A	B
0	1 2
1	3 4

3. 字典 (Series值) :

A	B
0	1 3
1	2 4

4. Numpy ndarray:

A	B
0	1 2
1	3 4

5. 字典 (字典值, orient='index'):

row1	row2
A	1 3
B	2 4

6. 列表字典:

A	B
0	1 NaN
1	2 3.0

7. 结构化数组:

A	B
0	1 2
1	3 4

8. Series组合:

A	B
0	10 30
1	20 40

9. 空 DataFrame:

Empty DataFrame
Columns: []
Index: []

# ► Pandas操作：数据查看与基本信息

方法	示例代码	说明
df.head(n)	df.head(3)	查看前n行（默认n=5）
df.tail(n)	df.tail(2)	查看后n行（默认=5）
df.info()	df.info()	查看数值列的统计摘要 (均值、标准差等)
df.describe()	df.describe()	输出每列数据类型、非空数等基本信息
df.shape	df.shape	返回（列数, 行数）的元祖
df.columns	df.columns	返回所有列名
df.index	df.index	返回行索引范围
df.dtypes	df.dtypes	查看各列的数据类型
df.values	df.values	将DataFrame转换为二维ndarray

```
◆ df.head(3):
   Name  Age  Score
0 Alice  25  88.5
1 Bob   30  92.0
2 Charlie 35  85.0

◆ df.tail(2):
   Name  Age  Score
3 David  40  89.5
4 Eva   28  95.0

◆ df.shape:
(5, 3)

◆ df.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   Name    5 non-null      object 
 1   Age     5 non-null      int64  
 2   Score   5 non-null      float64 
dtypes: float64(1), int64(1), object(1)
memory usage: 252.0+ bytes

◆ df.describe():
   Age  Score
count 5.00000 5.00000
mean 31.60000 90.00000
std 5.94138 3.758324
min 25.00000 85.00000
25% 28.00000 88.50000
50% 30.00000 89.50000
75% 35.00000 92.00000
max 40.00000 95.00000

◆ df.columns:
Index(['Name', 'Age', 'Score'], dtype='object')

◆ df.index:
RangeIndex(start=0, stop=5, step=1)

◆ df.dtypes:
Name    object
Age     int64
Score   float64
dtype: object

◆ df.values:
[['Alice' 25 88.5]
 ['Bob' 30 92.0]
 ['Charlie' 35 85.0]
 ['David' 40 89.5]
 ['Eva' 28 95.0]]
```

# ► Pandas操作：数据选择与索引

方法	示例代码	说明
<code>df['列名']</code>	<code>df['Age']</code>	选取单列（返回 Series）
<code>df[['列1', '列2']]</code>	<code>df[['Name', 'Score']]</code>	选取多列（返回 DataFrame）
<code>df.iloc[行号]</code>	<code>df.iloc[1]</code>	按行号选取（位置索引）
<code>df.loc[标签]</code>	<code>df.loc[0]</code>	按索引标签选取
<code>df.iloc[行, 列]</code>	<code>df.iloc[1, 2]</code>	行列同时按位置索引
<code>df.loc[行, 列]</code>	<code>df.loc[1, 'Score']</code>	行列同时按标签索引
<code>df[切片]</code>	<code>df[1:4, 1:4]</code>	行/列切片（类似列表）
布尔索引	<code>df[df['Age'] &gt; 30]</code>	筛选符合条件的行

```
# 创建示例 DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, 40, 28],
    'Score': [88.5, 92.0, 85.0, 89.5, 95.0]
}
df = pd.DataFrame(data)

print("\n◆ 示例 DataFrame:")
print(df)

# 选取单列
print("◆ 单列 (Series):")
print(df['Age'])

# 选取多列
print("◆ 多列 (DataFrame):")
print(df[['Name', 'Score']])

# 使用 iloc 进行位置索引
print("◆ 第 2 行 (iloc):")
print(df.iloc[1])

print("\n◆ 第 2 行第 3 列 (iloc):")
print(df.iloc[1, 2])

# 使用 loc 进行标签索引
print("◆ 第 2 行 (loc):")
print(df.loc[1])

print("\n◆ 第 2 行 Score 列 (loc):")
print(df.loc[1, 'Score'])

# 行切片
print("\n◆ 行切片 1 到 3:")
print(df[1:4])

# 布尔索引
print("\n◆ Age > 30 的行:")
print(df[df['Age'] > 30])

◆ 示例 DataFrame:
   Name  Age  Score
0  Alice   25  88.5
1    Bob   30  92.0
2 Charlie   35  85.0
3  David   40  89.5
4    Eva   28  95.0

◆ 单列 (Series):
0    25
1    30
2    35
3    40
4    28
Name: Age, dtype: int64

◆ 多列 (DataFrame):
   Name  Score
0  Alice  88.5
1    Bob  92.0
2 Charlie  85.0
3  David  89.5
4    Eva  95.0

◆ 第 2 行 (iloc):
Name      Bob
Age       30
Score    92.0
Name: 1, dtype: object

◆ 第 2 行第 3 列 (iloc):
92.0

◆ 第 2 行 (loc):
Name      Bob
Age       30
Score    92.0
Name: 1, dtype: object

◆ 第 2 行 Score 列 (loc):
92.0

◆ 行切片 1 到 3:
   Name  Age  Score
1    Bob   30  92.0
2 Charlie   35  85.0
3  David   40  89.5

◆ Age > 30 的行:
   Name  Age  Score
2 Charlie   35  85.0
3  David   40  89.5
```

# ► Pandas操作：缺失值处理与数据清洗

方法	示例代码	说明
df[条件表达式]	df[df['Age'] > 30]	按条件筛选行
多条件：与 (&)	df[(df['Age'] > 30) & (df['Score'] > 90)]	同时满足多个条件
多条件：或 ( )	df[(df['Age'] < 30)   (df['Score'] > 90)]	满足条件即可，非同时
df.query()	df.query("Age > 30 and Score > 90")	使用字符串表达式筛选
isin()	df[df['Name'].isin(['Alice', 'Eva'])]	筛选指定值列表中的行
str.contains()	df[df['Name'].str.contains("a")]	文本模糊匹配（正则）

```
# 创建示例 DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, 40, 28],
    'Score': [88.5, 92.0, 85.0, 89.5, 95.0]
}
df = pd.DataFrame(data)

print("\n◆ 示例 DataFrame:")
print(df)

# 筛选 Age > 30
print("◆ Age > 30:")
print(df[df['Age'] > 30])

# 筛选 Age > 30 且 Score > 90
print("\n◆ Age > 30 且 Score > 90:")
print(df[(df['Age'] > 30) & (df['Score'] > 90)])

# 筛选 Age < 30 或 Score > 90
print("\n◆ Age < 30 或 Score > 90:")
print(df[(df['Age'] < 30) | (df['Score'] > 90)])

# 使用 query 方法
print("\n◆ 使用 query 筛选:")
print(df.query("Age > 30 and Score > 90"))

# 使用 isin
print("\n◆ Name 是 Alice 或 Eva:")
print(df[df['Name'].isin(['Alice', 'Eva'])])

# 字符串匹配
print("\n◆ Name 中包含 'a':")
print(df[df['Name'].str.contains("a", case=False)])
```

◆ 示例 DataFrame:

	Name	Age	Score
0	Alice	25	88.5
1	Bob	30	92.0
2	Charlie	35	85.0
3	David	40	89.5
4	Eva	28	95.0

◆ Age > 30:

	Name	Age	Score
2	Charlie	35	85.0
3	David	40	89.5

◆ Age > 30 且 Score > 90:

	Name	Age	Score

◆ Age < 30 或 Score > 90:

	Name	Age	Score
0	Alice	25	88.5
1	Bob	30	92.0
4	Eva	28	95.0

◆ 使用 query 筛选:

	Name	Age	Score

◆ Name 是 Alice 或 Eva:

	Name	Age	Score
0	Alice	25	88.5
4	Eva	28	95.0

◆ Name 中包含 'a':

	Name	Age	Score
0	Alice	25	88.5
2	Charlie	35	85.0
3	David	40	89.5
4	Eva	28	95.0

# ► Pandas操作：数据修改与新增列/行

方法	示例代码	说明
修改某列的值	<code>df['col'] = df['col'] * 2</code>	将某列整体乘以2（支持广播）
修改满足条件的值	<code>df.loc[df['age'] &lt; 18, 'group'] = 'child'</code>	条件筛选并修改对应列的值
使用apply修改列	<code>df['name_len'] = df['name'].apply(len)</code>	对列中每个值应用函数生成新列
新增列（常数）	<code>df['flag'] = 1</code>	添加新列，所有值为1
新增列（表达式）	<code>df['bmi'] = df['weight'] / (df['height'] / 100)**2</code>	根据已有列计算并添加新列
新增列（使用assign）	<code>df = df.assign(score2 = df['score'] * 1.5)</code>	链式添加新列，推荐用于函数式写法
新增多列	<code>df[['x', 'y']] = some_array</code>	批量添加列，来自 ndarray 或其他结构
新增一行（字典）	<code>df.loc[len(df)] = {'name': 'Tom', 'age': 20}</code>	向末尾添加一行数据
新增多行（concat）	<code>df = pd.concat([df, new_df], ignore_index=True)</code>	合并两个DataFrame以添加多行
修改单元格	<code>df.at[3, 'name'] = 'Alice'</code>	定位行列并修改（高效）
批量修改多列	<code>df[['col1', 'col2']] = df[['col1', 'col2']] + 10</code>	多列同时修改（向量化）
重命名列名	<code>df.rename(columns={'old': 'new'}, inplace=True)</code>	修改列名（也可用于行索引）
插入列到指定位置	<code>df.insert(2, 'new_col', values)</code>	在第2列插入名为 new_col 的列

# Pandas操作：数据修改与新增列/行

```

# 1. 创建示例数据
df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie', 'David'],
    'age': [25, 17, 35, 29],
    'weight': [55, 68, 70, 80],
    'height': [165, 175, 180, 170]
})

print("原始数据:")
print(df)

# 2. 修改某列的值 (*2)
df['age'] = df['age'] * 2
print("\n年龄列 *2 后:")
print(df)

# 3. 修改满足条件的值
df.loc[df['age'] < 40, 'group'] = 'young'
df.loc[df['age'] >= 40, 'group'] = 'adult'
print("\n根据年龄条件添加 group 列:")
print(df)

# 4. 使用 apply 新增列
df['name_len'] = df['name'].apply(len)
print("\n添加 name_len 列:")
print(df)

# 5. 新增常数列
df['flag'] = 1
print("\n添加 flag 列:")
print(df)

# 6. 新增表达式列
df['bmi'] = df['weight'] / (df['height'] / 100)**2
print("\n计算 BMI 并添加列:")
print(df)

# 7. 使用 assign 新增列
df = df.assign(score2 = df['age'] * 1.5)
print("\n使用 assign 添加 score2 列:")
print(df)

# 8. 新增一行 (字典方式)
df.loc[len(df)] = {'name': 'Tom', 'age': 32, 'weight': 75, 'height': 172, 'group': 'adult',
                   'name_len': 3, 'flag': 1, 'bmi': 25.3, 'score2': 48.0}
print("\n新增一行:")
print(df)

# 9. 新增多行 (使用 concat)
new_rows = pd.DataFrame([
    {'name': 'Eva', 'age': 28, 'weight': 60, 'height': 165, 'group': 'young',
     'name_len': 3, 'flag': 1, 'bmi': 22.0, 'score2': 42.0},
    {'name': 'Frank', 'age': 38, 'weight': 82, 'height': 178, 'group': 'adult',
     'name_len': 5, 'flag': 1, 'bmi': 26.0, 'score2': 57.0}
])
df = pd.concat([df, new_rows], ignore_index=True)
print("\n新增多行 (concat):")
print(df)

# 10. 修改单元格
df.at[0, 'name'] = 'Alicia'
print("\n修改单元格 第0行 name 改 Alicia:")
print(df)

# 11. 批量修改多列
df[['age', 'score2']] = df[['age', 'score2']] + 10
print("\n批量修改多列 (age 和 score2 +10):")
print(df)

# 12. 修改列名
df.rename(columns={'name': 'Name', 'age': 'Age'}, inplace=True)
print("\n修改列名:")
print(df)

# 13. 插入列到指定位置
df.insert(2, 'status', 'active')
print("\n插入 status 列到第2列位置:")
print(df)

```

原始数据:

	name	age	weight	height
0	Alice	25	55	165
1	Bob	17	68	175
2	Charlie	35	70	180
3	David	29	80	170

年龄列 \*2 后:

	name	age	weight	height
0	Alice	50	55	165
1	Bob	34	68	175
2	Charlie	70	70	180
3	David	58	80	170

根据年龄条件添加 group 列:

	name	age	weight	height	group
0	Alice	50	55	165	adult
1	Bob	34	68	175	young
2	Charlie	70	70	180	adult
3	David	58	80	170	adult

添加 name\_len 列:

	name	age	weight	height	group	name_len
0	Alice	50	55	165	adult	5
1	Bob	34	68	175	young	3
2	Charlie	70	70	180	adult	7
3	David	58	80	170	adult	5

添加 flag 列:

	name	age	weight	height	group	name_len	flag
0	Alice	50	55	165	adult	5	1
1	Bob	34	68	175	young	3	1
2	Charlie	70	70	180	adult	7	1
3	David	58	80	170	adult	5	1

计算 BMI 并添加列:

	name	age	weight	height	group	name_len	flag	bmi
0	Alice	50	55	165	adult	5	1	20.202020
1	Bob	34	68	175	young	3	1	22.204082
2	Charlie	70	70	180	adult	7	1	21.604938
3	David	58	80	170	adult	5	1	27.681661

使用 assign 添加 score2 列:

	name	age	weight	height	group	name_len	flag	bmi	score2
0	Alice	50	55	165	adult	5	1	20.202020	75.0
1	Bob	34	68	175	young	3	1	22.204082	51.0
2	Charlie	70	70	180	adult	7	1	21.604938	105.0
3	David	58	80	170	adult	5	1	27.681661	87.0
4	Tom	32	75	172	adult	3	1	25.300000	48.0

新增一行:

	name	age	weight	height	group	name_len	flag	bmi	score2
0	Alice	50	55	165	adult	5	1	20.202020	75.0
1	Bob	34	68	175	young	3	1	22.204082	51.0
2	Charlie	70	70	180	adult	7	1	21.604938	105.0
3	David	58	80	170	adult	5	1	27.681661	87.0
4	Tom	32	75	172	adult	3	1	25.300000	48.0
5	Eva	28	60	165	young	3	1	22.000000	42.0
6	Frank	38	82	178	adult	5	1	26.000000	57.0

新增多行 (concat):

	name	age	weight	height	group	name_len	flag	bmi	score2
0	Alice	50	55	165	adult	5	1	20.202020	75.0
1	Bob	34	68	175	young	3	1	22.204082	51.0
2	Charlie	70	70	180	adult	7	1	21.604938	105.0
3	David	58	80	170	adult	5	1	27.681661	87.0
4	Tom	32	75	172	adult	3	1	25.300000	48.0
5	Eva	28	60	165	young	3	1	22.000000	42.0
6	Frank	38	82	178	adult	5	1	26.000000	57.0

修改单元格 (第0行 name 改 Alicia):

	name	age	weight	height	group	name_len	flag	bmi	score2
0	Alicia	50	55	165	adult	5	1	20.202020	75.0
1	Bob	34	68	175	young	3	1	22.204082	51.0
2	Charlie	70	70	180	adult	7	1	21.604938	105.0
3	David	58	80	170	adult	5	1	27.681661	87.0
4	Tom	32	75	172	adult	3	1	25.300000	48.0
5	Eva	28	60	165	young	3	1	22.000000	42.0
6	Frank	38	82	178	adult	5	1	26.000000	57.0

批量修改多列 (age 和 score2 +10):

	name	age	weight	height	group	name_len	flag	bmi	score2
0	Alicia	60	55	165	adult	5	1	20.202020	85.0
1	Bob	44	68	175	young	3	1	22.204082	61.0
2	Charlie	80	70	180	adult	7	1	21.604938	115.0
3	David	68	80	170	adult	5	1	27.681661	97.0
4	Tom	42	75	172	adult	3	1	25.300000	58.0
5	Eva	38	60	165	young	3	1	22.000000	52.0
6	Frank	48	82	178	adult	5	1	26.000000	67.0

修改列名:

	Name	Age	weight	height	group	name_len	flag	bmi	score2
0	Alicia	60	55	165	adult	5	1	20.202020	85.0
1	Bob	44	68	175	young	3	1	22.204082	61.0
2	Charlie	80	70	180	adult	7	1	21.604938	115.0
3	David	68	80	170	adult	5	1	27.681661	97.0
4	Tom	42	75	172	adult	3	1	25.300000	58.0
5	Eva	38	60	165	young	3	1	22.000000	52.0
6	Frank	48	82	178	adult	5	1	26.000000	67.0

插入 status 列到第2列位置:

	Name	Age	status	weight	height	group	name_len	flag	bmi	score2
0	Alicia	60	active	55	165	adult	5	1	20.202020	85.0
1	Bob	44	active	68	175	young	3	1	22.204082	61.0
2	Charlie	80	active	70	180	adult	7	1	21.604938	115.0
3	David	68	active	80	170	adult	5	1	27.681661	97.0
4	Tom	42	active	75	172	adult	3	1	25.300000	58.0
5	Eva	38	active	60	165	young	3	1	22.000000	52.0
6	Frank	48	active	82	178	adult	5	1	26.000000	67.0

score2

	0	1	2	3	4	5	6
0	85.0						
1	61.0						
2	115.0						
3	97.0						
4	58.0						
5	52.0						
6	67.0						

# ► Pandas操作：分组与统计分析

方法	示例代码	说明
分组	<code>df.groupby('col')</code>	按单列分组，返回 GroupBy 对象
多列分组	<code>df.groupby(['col1', 'col2'])</code>	多层分组
分组后计数	<code>df.groupby('group')['score'].count()</code>	每组数量统计
分组后求和	<code>df.groupby('group')['score'].sum()</code>	每组求和
分组后均值	<code>df.groupby('group')['score'].mean()</code>	每组平均值
多个聚合	<code>df.groupby('group')['score'].agg(['mean', 'sum'])</code>	同时计算多个统计量
自定义聚合函数	<code>df.groupby('group')['score'].agg(lambda x: x.max() - x.min())</code>	聚合时使用自定义函数
分组排序	<code>df.groupby('group')['score'].mean().sort_values(ascending=False)</code>	按分组平均值排序
重置索引	<code>grouped.reset_index()</code>	聚合后将 groupby 索引还原为列
分组取前几	<code>df.groupby('group').head(2)</code>	每组取前 N 行数据
分组应用函数	<code>df.groupby('group').apply(lambda g: g[g['score'] &gt; 60])</code>	对每组分别应用函数

# ► Pandas操作：分组与统计分析

```
# 构造示例数据
df = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank', 'Grace', 'Helen'],
    'group': ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B'],
    'score': [85, 76, 90, 88, 79, 70, 95, 83],
    'age': [23, 21, 25, 24, 22, 26, 28, 23]
})

print("原始数据: ")
print(df)

# 1. 按组统计平均成绩
print("\n按组统计平均成绩:")
print(df.groupby('group')['score'].mean())

# 2. 多统计量聚合（均值与最大值）
print("\n每组 score 的均值和最大值: ")
print(df.groupby('group')['score'].agg(['mean', 'max']))

# 3. 每组人数统计
print("\n每组人数:")
print(df.groupby('group').size())

# 4. 每组最大年龄
print("\n每组最大年龄:")
print(df.groupby('group')['age'].max())

# 5. 自定义函数: score 极差
print("\n每组 score 的极差(max - min): ")
print(df.groupby('group')['score'].agg(lambda x: x.max() - x.min()))

# 6. 每组按 score 降序排序后取前 1
print("\n每组 score 最高的人: ")
print(df.sort_values('score', ascending=False).groupby('group').head(1))

# 7. 使用 apply 筛选每组分数 >80 的人
print("\n每组中 score > 80 的人: ")
print(df.groupby('group').apply(lambda g: g[g['score'] > 80].reset_index(drop=True)))

# 8. 多列分组统计
print("\n多列分组(group + age)统计 score 平均: ")
print(df.groupby(['group', 'age'])['score'].mean())
```

```
原始数据:
   name group  score  age
0  Alice     A      85  23
1    Bob     B      76  21
2  Charlie    A      90  25
3  David     B      88  24
4    Eva     A      79  22
5  Frank     B      70  26
6  Grace     A      95  28
7  Helen     B      83  23
```

```
按组统计平均成绩:
group
A  87.25
B  79.25
Name: score, dtype: float64
```

```
每组 score 的均值和最大值:
group | mean  max
A     87.25  95
B     79.25  88
```

```
每组人数:
group
A  4
B  4
dtype: int64
```

```
每组最大年龄:
group
A  28
B  26
Name: age, dtype: int64
```

```
每组 score 的极差 (max - min):
group
A  16
B  18
Name: score, dtype: int64
```

```
每组 score 最高的人:
   name group  score  age
6  Grace     A      95  28
3  David     B      88  24
```

```
每组中 score > 80 的人:
   name group  score  age
0  Alice     A      85  23
1  Charlie    A      90  25
2  Grace     A      95  28
3  David     B      88  24
4  Helen     B      83  23
```

```
多列分组(group + age)统计 score 平均:
group  age
A     22  79.0
      23  85.0
      25  90.0
      28  95.0
B     21  76.0
      23  83.0
      24  88.0
      26  70.0
Name: score, dtype: float64
```

## ► Pandas操作：数据导入与导出

操作	示例代码	说明
读取CSV文件	<code>pd.read_csv('data.csv')</code>	默认读取为 DataFrame
读取Excel文件	<code>pd.read_excel('file.xlsx', sheet_name='Sheet1')</code>	指定sheet
读取JSON文件	<code>pd.read_json('data.json')</code>	从JSON文件读取
写出为CSV	<code>df.to_csv('out.csv', index=False)</code>	保存为CSV，不带索引
写出为Excel	<code>df.to_excel('out.xlsx', index=False)</code>	保存为Excel
写出为JSON	<code>df.to_json('out.json', orient='records')</code>	保存为JSON
读取剪贴板数据	<code>pd.read_clipboard()</code>	直接读取复制的表格数据
写入剪贴板	<code>df.to_clipboard(index=False)</code>	复制DataFrame到剪贴板

# ► Pandas操作：数据合并与拼接

操作	示例代码	说明
行拼接（纵向）	<code>pd.concat([df1, df2])</code>	类似 SQL 的 UNION
列拼接（横向）	<code>pd.concat([df1, df2], axis=1)</code>	按列拼接
重设索引	<code>pd.concat(..., ignore_index=True)</code>	拼接后自动编号
按key合并	<code>pd.merge(df1, df2, on='id')</code>	类似 SQL 的 inner join
左连接	<code>pd.merge(df1, df2, on='id', how='left')</code>	保留左表全部记录
外连接	<code>pd.merge(df1, df2, on='id', how='outer')</code>	全连接，缺失填 NaN
多键合并	<code>pd.merge(df1, df2, on=['key1', 'key2'])</code>	指定多个字段 join
索引对齐拼接	<code>df1.join(df2)</code>	按索引自动对齐拼接

# ► Pandas操作：数据合并与拼接

```
# 示例 DataFrame
df1 = pd.DataFrame({
    'id': [1, 2, 3],
    'name': ['Alice', 'Bob', 'Charlie']
})

df2 = pd.DataFrame({
    'id': [2, 3, 4],
    'score': [88, 92, 75]
})

df3 = pd.DataFrame({
    'id': [5, 6],
    'name': ['David', 'Eva']
})

# 查看原始DataFrame
print("示例 DataFrame df1:")
print(df1)
print("\n示例 DataFrame df2:")
print(df2)
print("\n示例 DataFrame df3:")
print(df3)

# 1. 合并 (内连接)
merged_inner = pd.merge(df1, df2, on='id')
print("\n内连接 (id 匹配): ")
print(merged_inner)

# 2. 合并 (左连接)
merged_left = pd.merge(df1, df2, on='id', how='left')
print("\n左连接 (保留 df1 所有行): ")
print(merged_left)

# 3. 合并 (外连接)
merged_outer = pd.merge(df1, df2, on='id', how='outer')
print("\n外连接 (所有 id): ")
print(merged_outer)

# 4. 行拼接 (concat)
concat_rows = pd.concat([df1, df3], ignore_index=True)
print("\n行拼接 (concat): ")
print(concat_rows)

# 5. 列拼接 (需要行数相同)
df_col1 = pd.DataFrame({'A': [1, 2, 3]})
df_col2 = pd.DataFrame({'B': ['x', 'y', 'z']})
concat_cols = pd.concat([df_col1, df_col2], axis=1)
print("\n列拼接 (concat axis=1): ")
print(concat_cols)
```

示例 DataFrame df1:

	id	name
0	1	Alice
1	2	Bob
2	3	Charlie

示例 DataFrame df2:

	id	score
0	2	88
1	3	92
2	4	75

示例 DataFrame df3:

	id	name
0	5	David
1	6	Eva

内连接 (id 匹配):

	id	name	score
0	2	Bob	88
1	3	Charlie	92

左连接 (保留 df1 所有行):

	id	name	score
0	1	Alice	NaN
1	2	Bob	88.0
2	3	Charlie	92.0

外连接 (所有 id):

	id	name	score
0	1	Alice	NaN
1	2	Bob	88.0
2	3	Charlie	92.0
3	4	NaN	75.0

行拼接 (concat):

	id	name
0	1	Alice
1	2	Bob
2	3	Charlie
3	5	David
4	6	Eva

列拼接 (concat axis=1):

	A	B
0	1	x
1	2	y
2	3	z

# ► Pandas案例：在线课程平台用户行为分析

- 目标：分析某在线教育平台上学生的学习行为数据，以评估课程质量、活跃度以及用户特征。

```
import pandas as pd
import numpy as np

np.random.seed(42)

# 学生表 (用户基础信息)
students = pd.DataFrame({
    'user_id': range(1, 11),
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Frank', 'Grace', 'Helen', 'Ian', 'Judy'],
    'age': np.random.randint(18, 35, size=10),
    'gender': np.random.choice(['M', 'F'], size=10)
})

# 课程表
courses = pd.DataFrame({
    'course_id': [101, 102, 103],
    'course_name': ['Python基础', '数据分析', '机器学习'],
    'teacher': ['李老师', '王老师', '赵老师']
})

# 行为日志表 (多条记录)
logs = pd.DataFrame({
    'log_id': range(1, 31),
    'user_id': np.random.choice(students['user_id'], size=30),
    'course_id': np.random.choice(courses['course_id'], size=30),
    'action': np.random.choice(['进入课程', '观看视频', '提交作业', '讨论交流'], size=30),
    'duration_min': np.random.randint(1, 60, size=30), # 行为时长
    'timestamp': pd.date_range('2025-07-01', periods=30, freq='H')
})

# 查看原始数据
print("◆ 学生表: ")
print(students)
print("\n◆ 课程表: ")
print(courses)
print("\n◆ 行为日志表: ")
print(logs)
```

◆ 学生表:

	user_id	name	age	gender
0	1	Alice	24	F
1	2	Bob	32	M
2	3	Charlie	28	F
3	4	David	25	F
4	5	Eva	24	F
5	6	Frank	28	F
6	7	Grace	28	F
7	8	Helen	21	F
8	9	Ian	25	F
9	10	Judy	20	F

◆ 课程表:

	course_id	course_name	teacher
0	101	Python基础	李老师
1	102	数据分析	王老师
2	103	机器学习	赵老师

◆ 行为日志表:

	log_id	user_id	course_id	action	duration_min	timestamp
0	1	5	103	讨论交流	42	2025-07-01 00:00:00
1	2	1	103	观看视频	39	2025-07-01 01:00:00
2	3	10	102	进入课程	58	2025-07-01 02:00:00
3	4	6	101	讨论交流	41	2025-07-01 03:00:00
	...					
26	27	2	101	提交作业	14	2025-07-02 02:00:00
27	28	4	102	进入课程	3	2025-07-02 03:00:00
28	29	7	103	提交作业	1	2025-07-02 04:00:00
29	30	8	101	进入课程	5	2025-07-02 05:00:00

# ► Pandas案例：在线课程平台用户行为分析

- 流程：通过构建学生、课程与学习日志数据，运用 Pandas 对用户行为进行合并、分组统计和可视化准备，以评估在线课程的学习情况与用户活跃度。

```
# 合并学生与行为日志，补充用户信息
merged = pd.merge(logs, students, on='user_id', how='left')
merged = pd.merge(merged, courses, on='course_id', how='left')

# 查看合并后的数据
print("\n◆ 合并后的数据: ")
print(merged.head())

# 每位学生的总学习时长
print("\n每位学生总学习时长:")
print(merged.groupby('name')['duration_min'].sum().sort_values(ascending=False))

# 每门课程的学习人数
print("\n每门课程的学习人数:")
print(merged.groupby('course_name')['user_id'].nunique())

# 每个行为类型的次数统计
print("\n用户行为频率统计:")
print(merged['action'].value_counts())

# 每门课程每天的平均学习时长
merged['date'] = merged['timestamp'].dt.date
print("\n每日每门课程平均学习时长:")
print(merged.groupby(['course_name', 'date'])['duration_min'].mean())

# 活跃学生（学习行为 > 5 次）
active_students = merged['user_id'].value_counts()
active_students = active_students[active_students > 3].index.tolist()
print("\n活跃用户列表(行为记录 > 3):")
print(students[students['user_id'].isin(active_students)])
```

◆ 合并后的数据:

log_id	user_id	course_id	action	duration_min	timestamp	name	
0	1	5	103	讨论交流	42	2025-07-01 00:00:00	Eva
1	2	1	103	观看视频	39	2025-07-01 01:00:00	Alice
2	3	10	102	进入课程	58	2025-07-01 02:00:00	Judy
3	4	6	101	讨论交流	41	2025-07-01 03:00:00	Frank
4	5	9	102	讨论交流	28	2025-07-01 04:00:00	Ian

age gender course\_name teacher

0	24	F	机器学习	赵老师
1	24	F	机器学习	赵老师
2	20	F	数据分析	王老师
3	28	F	Python基础	李老师
4	25	F	数据分析	王老师

每位学生总学习时长:

name	duration_min
Eva	181
Ian	158
Judy	105
Bob	89
Grace	81
Charlie	79
David	59
Alice	46
Frank	41
Helen	5

Name: duration\_min, dtype: int64

每门课程的学习人数:

course_name	user_id
Python基础	6
数据分析	8
机器学习	6

Name: user\_id, dtype: int64

用户行为频率统计:

action	count
进入课程	12
讨论交流	9
提交作业	8
观看视频	1

Name: count, dtype: int64

每日每门课程平均学习时长:

course_name	date	duration_min
Python基础	2025-07-01	41.333333
	2025-07-02	22.000000
数据分析	2025-07-01	28.466667
	2025-07-02	20.000000
机器学习	2025-07-01	31.000000
	2025-07-02	1.000000

Name: duration\_min, dtype: float64

活跃用户列表 (行为记录 > 3):

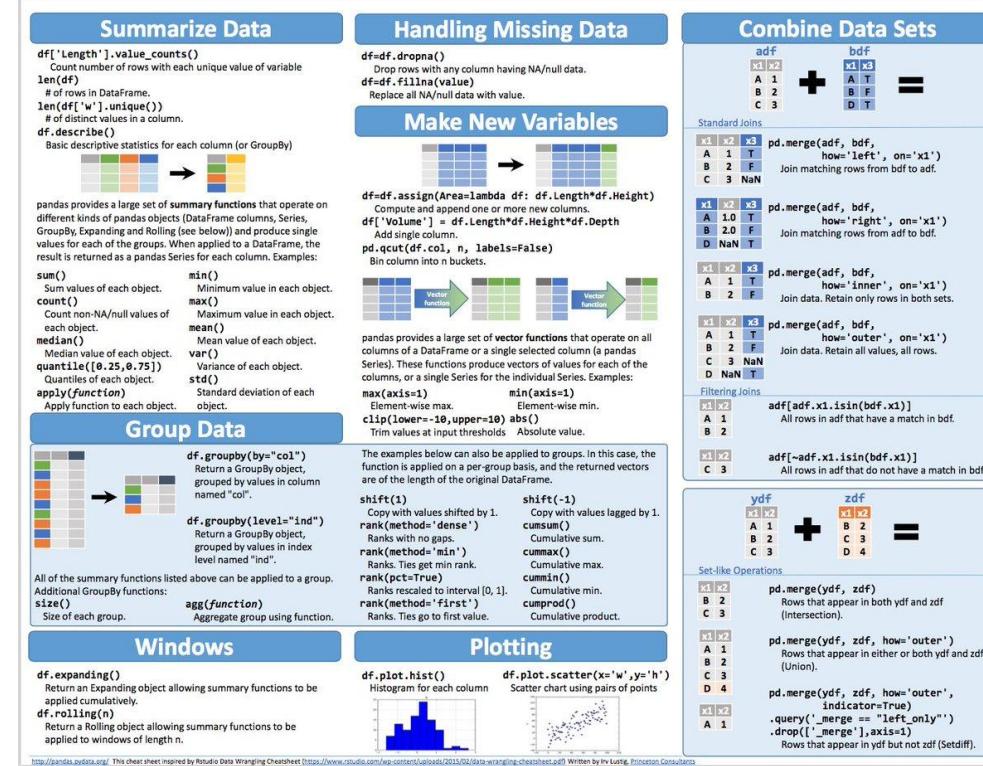
user_id	name	age	gender
4	Eva	24	F
6	Grace	28	F
8	Ian	25	F
9	Judy	20	F

## ▶ 小结

- Pandas 是一个基于 Python 的强大数据分析工具，提供了灵活高效的 DataFrame 数据结构，适合进行清洗、转换、分组、聚合等操作。
  - 它广泛应用于数据科学、金融分析和机器学习前的数据处理，是数据分析工作流程中的核心组件。



source: <https://github.com/randylosat/Python-Tutorials/blob/master/Pandas%20Reference%20Guide.md>



用 Pandas 操作数据，就像用魔法整理房间，一行代码，乱七八糟立马整整齐齐！

# 目录章节

---

CONTENTS

01

语法基础与数据类型

02

Numpy：数值计算

03

Pandas：数据处理

04

Matplotlib：数据可视化

05

Scikit-learn：机器学习

## ► Matplotlib 是什么？

- Matplotlib 是 Python 中最常用的**二维**绘图库，支持绘制线图、柱状图、散点图、饼图等多种静态、动态与交互式图表，它为科学计算和数据可视化提供了强大支持。
- Matplotlib 的优势：1) **语法灵活**：既可快速绘图（pyplot），又能精细控制图层；2) **图表类型丰富**：支持多种图形、坐标系与样式定制；3) **兼容性好**：可与 NumPy、Pandas、Seaborn 等生态完美协作。

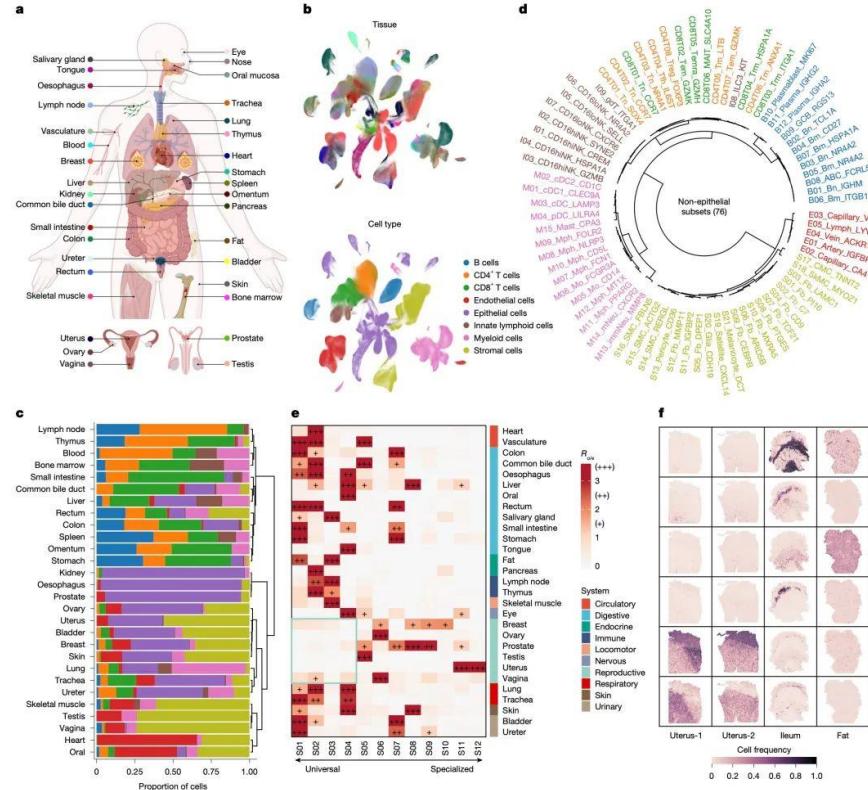
matplotlib



**Matplotlib**: 程序员的数据画板，一行代码让表格变图，数字说话有图有真相！

# ► Pandas 的应用场景

- Matplotlib 是 Python 中最基础且功能强大的可视化工具，适用于绘制线图、柱状图、散点图、饼图等多种图表，广泛用于科研绘图、数据分析展示和报告图像生成。

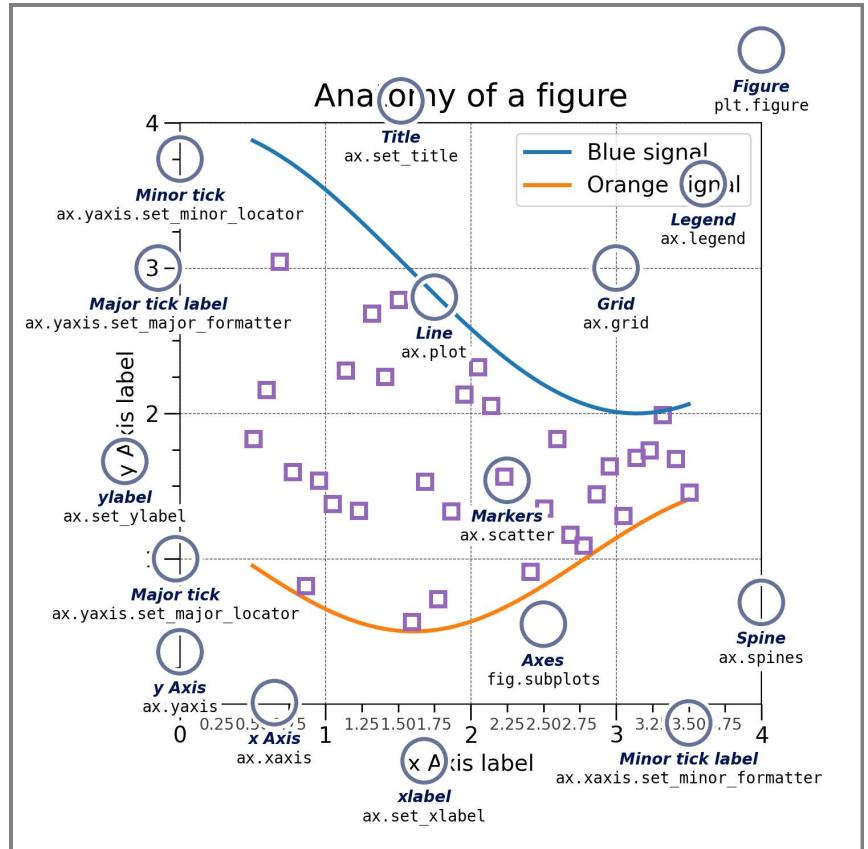


场景	描述
数据探索	通过折线图、散点图等快速观察数据分布、趋势与异常值
结果展示	用图表直观呈现分析结果，便于交流、报告和汇报
科研绘图	支持高质量图像输出，适合学术论文和技术报告中的图表制作
教学演示	在教学中动态展示数据变化过程，提升可视化教学效果
可视化调试	将中间数据或模型输出可视化，辅助模型开发与调参
自定义图形绘制	提供底层 API，支持自定义坐标轴、图例、颜色、线型、注释等图形细节控制

Matplotlib 提供灵活的图形控制能力，是数据探索、结果展示和科研绘图的强大工具。

# ► Matplotlib 基础：分层架构

➤ Matplotlib 采用**分层架构**: Figure -> Axes -> Axis -> Artist



层次架构递进：

- Figure: 整个**图像** (画布)
- Axes: 一个**子图** (包含坐标系)
- Axis: **坐标轴** (X轴、Y轴)
- Artist: 所有**可见元素** (如线、文本、图例等)

熟练掌握这四层结构，才能真正把握 Matplotlib!

# ► Matplotlib 的基础：基本绘图操作

- Matplotlib 是 Python 中常用的可视化库，基本操作流程包括：创建数据、调用绘图函数、显示图表。
- 最常用的绘图函数是 plt.plot(x, y)，用于绘制二维折线图；使用 plt.title()、plt.xlabel()、plt.ylabel() 和 plt.show() 可以完成图表的标题、轴标签和展示。

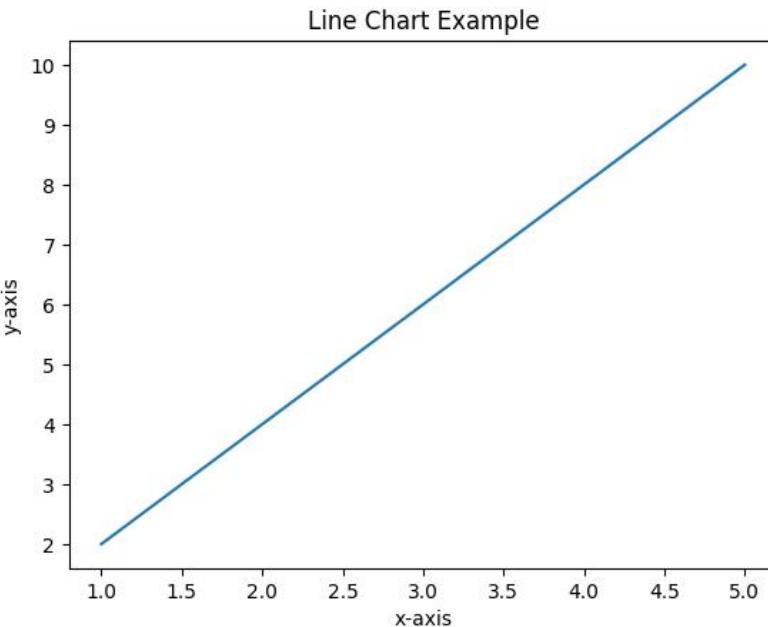
```
import matplotlib.pyplot as plt

# ===== 1. 创建数据 =====
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# ===== 2. 绘图 =====
plt.plot(x, y) # 折线图

# ===== 3. 添加标题与轴标签（支持中文）=====
plt.title("Line Chart Example") # 图表标题
plt.xlabel("x-axis") # X 轴标签
plt.ylabel("y-axis") # Y 轴标签

# ===== 4. 显示图形 =====
plt.show()
```



Matplotlib 的基本绘图操作是：创建数据，调用绘图函数，设置图表元素并显示图形。

## ► Matplotlib 的基础：图形类型操作

- Matplotlib 支持多种图形类型，如折线图、散点图、柱状图、直方图和饼图等，满足不同数据可视化需求。

图类型	函数	示例
折线图	plot()	plt.plot(x, y)
散点图	scatter()	plt.scatter(x, y)
柱状图	bar()	plt.bar(x, y)
水平柱状图	barh()	plt.barh(x, y)
直方图	hist()	plt.hist(data)
饼图	pie()	plt.pie(data)
箱线图	boxplot()	plt.boxplot(data)
面积图	fill_between()	plt.fill_between(x, y)
多子图	subplot()	plt.subplot(2, 1, 1)

每种图形通过不同的函数实现，如 `plot()`、`scatter()`、`bar()` 等，调用方式简单灵活。

# ► Matplotlib 的基础：图形类型操作

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

# 设置中文字体
plt.rcParams['font.family'] = 'Arial Unicode MS'

# 1. Line Plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.figure(figsize=(10, 6))
plt.subplot(231) # 创建一个 2x3 的子图网格, 当前绘制在第一个位置
plt.plot(x, y)
plt.title("线图")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

# 2. Scatter Plot
x2 = [1, 2, 3, 4, 5]
y2 = [5, 4, 3, 2, 1]
plt.subplot(232) # 创建在第二个位置
plt.scatter(x2, y2, color='green')
plt.title("散点图")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

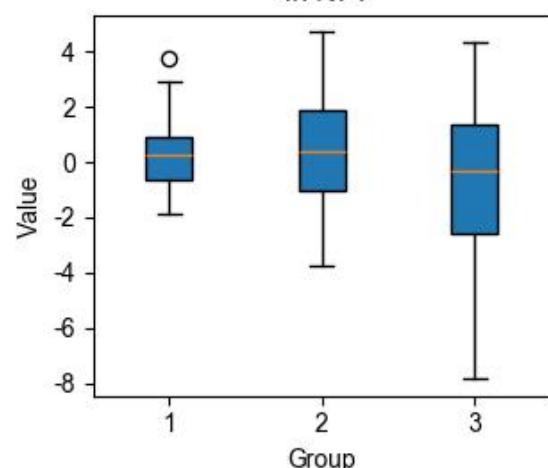
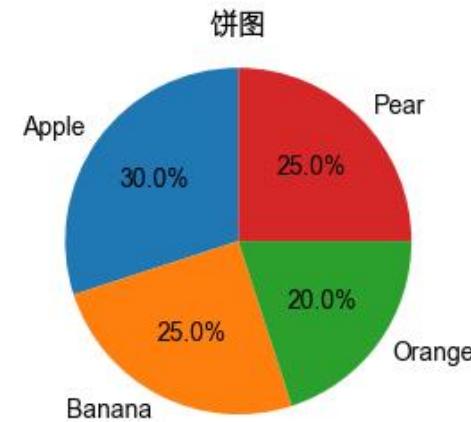
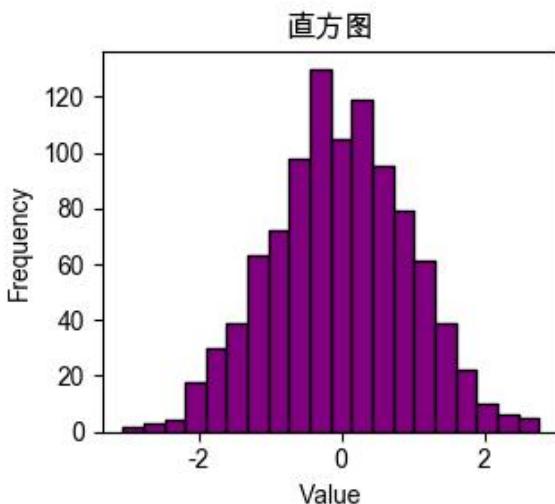
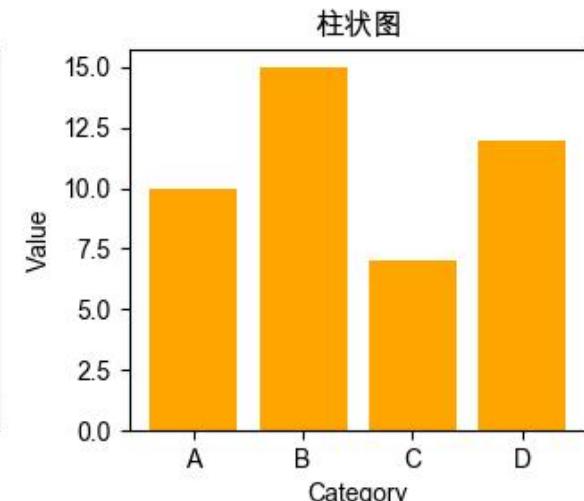
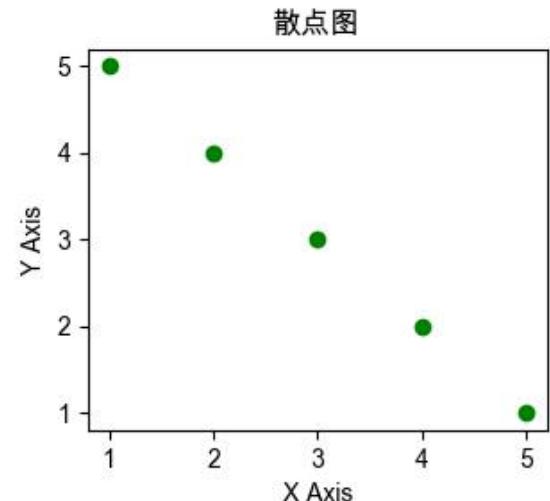
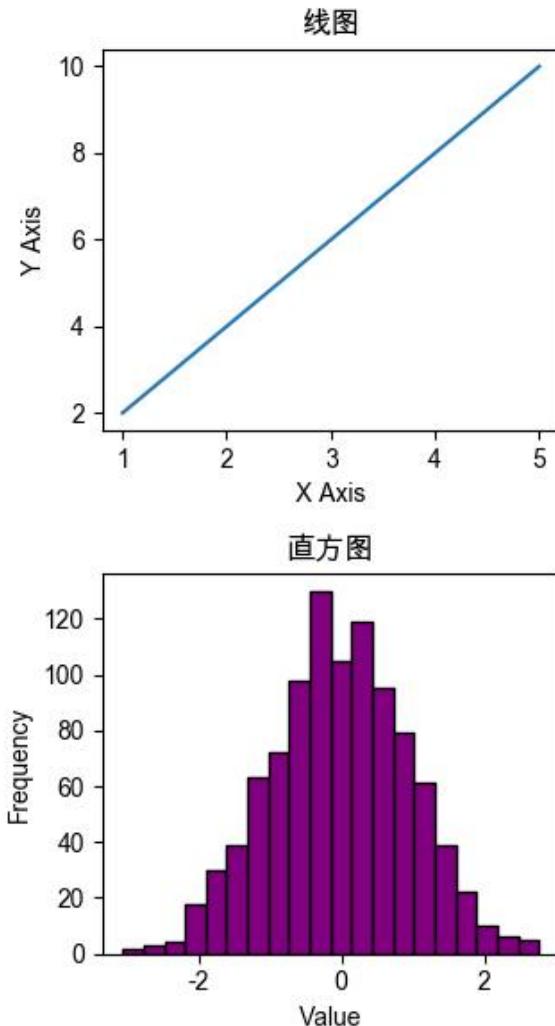
# 3. Bar Chart
categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 12]
plt.subplot(233) # 创建在第三个位置
plt.bar(categories, values, color='orange')
plt.title("柱状图")
plt.xlabel("Category")
plt.ylabel("Value")

# 4. Histogram
data = np.random.randn(1000)
plt.subplot(234) # 创建在第四个位置
plt.hist(data, bins=20, color='purple', edgecolor='black')
plt.title("直方图")
plt.xlabel("Value")
plt.ylabel("Frequency")

# 5. Pie Chart
labels = ['Apple', 'Banana', 'Orange', 'Pear']
sizes = [30, 25, 20, 25]
plt.subplot(235) # 创建在第五个位置
# autopct 显示百分比 startangle 旋转起始角度
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("饼图")
plt.axis('equal')

# 6. Box Plot
data2 = [np.random.normal(0, std, 100) for std in range(1, 4)]
plt.subplot(236) # 创建在第六个位置
plt.boxplot(data2, vert=True, patch_artist=True)
plt.title("箱线图")
plt.xlabel("Group")
plt.ylabel("Value")

plt.tight_layout()
plt.show()
```



## ► Matplotlib 的基础：标题与标签设置

- Matplotlib 可以通过各种函数修改标题与标签设置，例如Matplotlib 中通过 plt.title() 设置标题，plt.xlabel() 和 plt.ylabel() 设置坐标轴标签，简单易用且支持丰富的文本属性。
- 标题和标签是图表的重要元素，用于说明图形内容和坐标轴含义，提升图表的可读性。

操作	函数	示例
图标标题	plt.title()	plt.title("销售趋势")
X轴标签	plt.xlabel()	plt.xlabel("月份")
Y轴标签	plt.ylabel()	plt.ylabel("销售额")
图例	plt.legend()	plt.legend(loc="upper left")
图例标签设置	label=	plt.plot(x, y, label="一季度")
注释文本	plt.annotate()	plt.annotate("高点", xy=(3,25), xytext=(2,28))

# ► Matplotlib 基础：标题与标签设置

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y1 = [2, 4, 6, 8, 10]
y2 = [1, 3, 5, 7, 9]

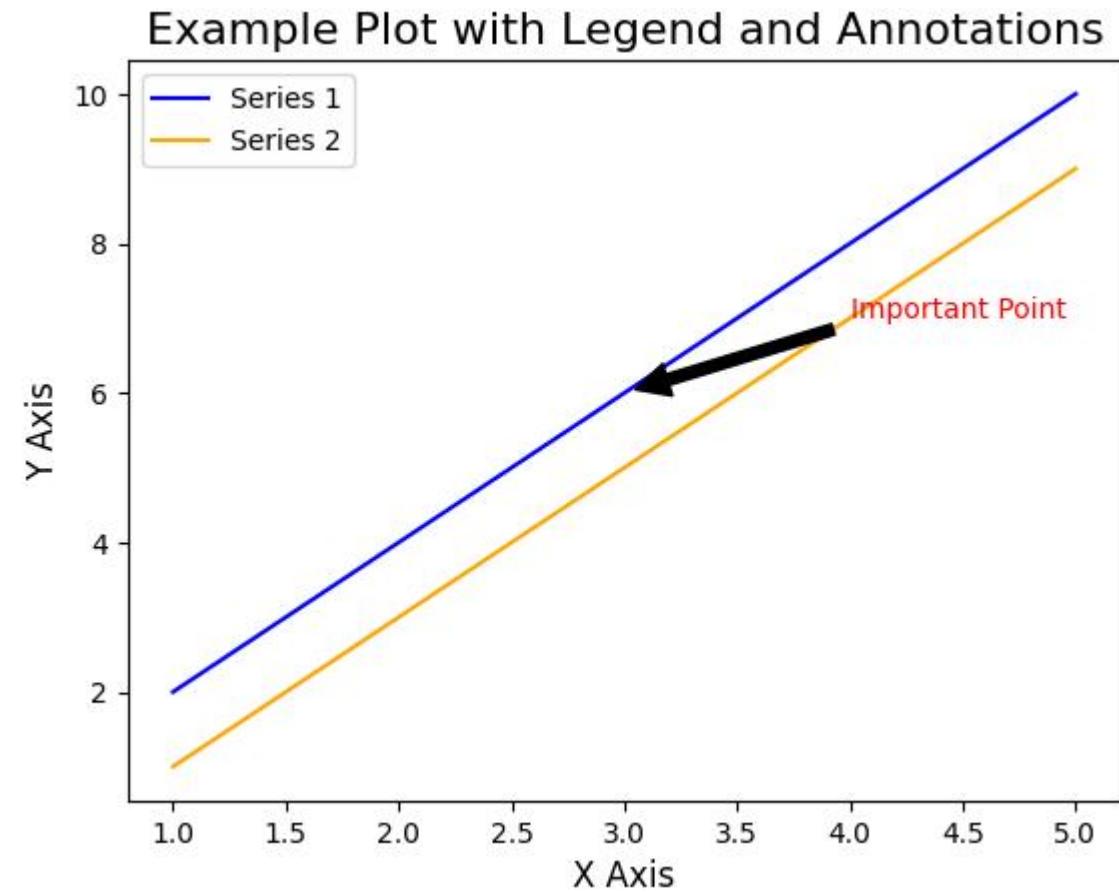
plt.plot(x, y1, label='Series 1', color='blue') # 绘制第一条线
plt.plot(x, y2, label='Series 2', color='orange') # 绘制第二条线

# 设置标题和坐标轴标签
plt.title("Example Plot with Legend and Annotations", fontsize=16)
plt.xlabel("X Axis", fontsize=12)
plt.ylabel("Y Axis", fontsize=12)

# 显示图例
plt.legend(loc='upper left')

# 添加注释，箭头指向点 (3, 6)
plt.annotate('Important Point', xy=(3, 6), xytext=(4, 7),
             arrowprops=dict(facecolor='black', shrink=0.05),
             fontsize=10, color='red')

plt.show()
```



## ► Matplotlib 基础：样式与配色设置

- Matplotlib 支持通过 color、linestyle、marker、linewidth 等参数灵活设置线条和标记样式，提升图表美观性与识别度。
- 同时可结合 plt.style.use()、内置调色板、RGB/十六进制颜色等方式实现高级配色控制。

样式项	参数名	示例值	示例
线条颜色	color	'red', '#00FF00'	plt.plot(x, y, color='red')
线条样式	linestyle	'-', '--', ':'	plt.plot(..., linestyle='--')
标记形状	marker	'o', 's', 'x', '^'	plt.plot(..., marker='o')
线宽	linewidth	1.5, 2	plt.plot(..., linewidth=2)
标记大小	markersize	6, 10	plt.plot(..., markersize=8)

# ► Matplotlib 基础：样式与配色设置

```
import matplotlib.pyplot as plt

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

plt.figure(figsize=(12, 8))

# 1. 线条颜色
plt.subplot(231)
plt.plot(x, y, color='red')
plt.title("Color = 'red'")

# 2. 线条样式
plt.subplot(232)
plt.plot(x, y, linestyle='--')
plt.title("Linestyle = '--'")

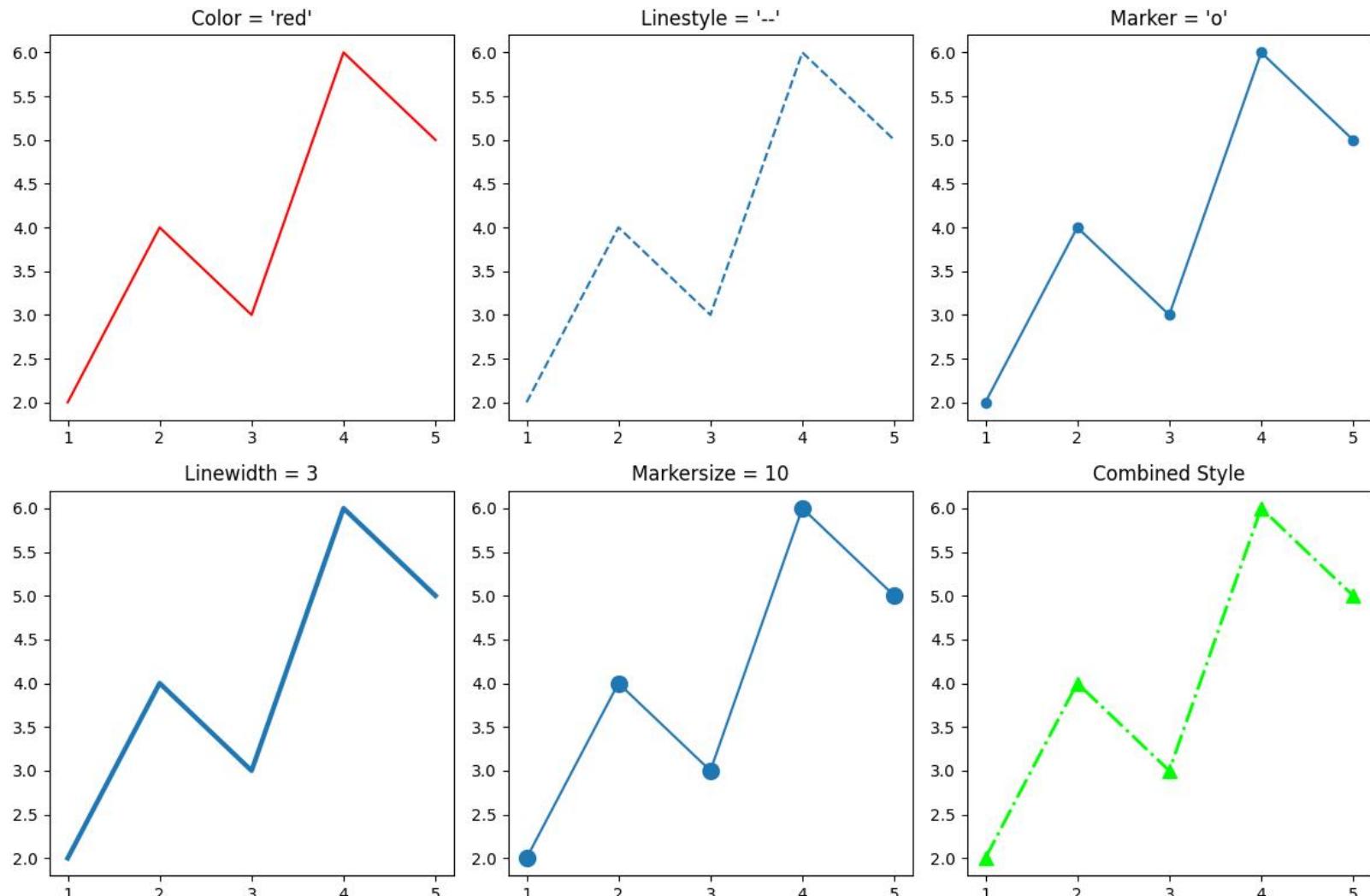
# 3. 标记形状
plt.subplot(233)
plt.plot(x, y, marker='o')
plt.title("Marker = 'o'")


# 4. 线宽 linewidth
plt.subplot(234)
plt.plot(x, y, linewidth=3)
plt.title("Linewidth = 3")

# 5. 标记大小 markersize
plt.subplot(235)
plt.plot(x, y, marker='o', markersize=10)
plt.title("Markersize = 10")


# 6. 综合样式 (全部一起展示)
plt.subplot(236)
plt.plot(
    x, y,
    color="#00FF00",
    linestyle='-.',
    marker='^',
    linewidth=2,
    markersize=8
)
plt.title("Combined Style")

plt.tight_layout()
plt.show()
```



## ► Matplotlib 的基础：网格与坐标轴控制

- Matplotlib 提供灵活的函数如 `grid()`、`xlim()`、`ylim()`、`xticks()`、`yticks()` 以及 `set_aspect()` 来精细控制坐标轴样式与范围。
- 主要作用：设置对于强调关键数据、调整图形比例、美化图表展示等。

操作	函数	示例
显示网格线	<code>plt.grid(True)</code>	<code>plt.grid(True)</code>
设置X轴范围	<code>plt.xlim()</code>	<code>plt.xlim(0, 10)</code>
设置Y轴范围	<code>plt.ylim()</code>	<code>plt.ylim(0, 100)</code>
设置刻度值	<code>plt.xticks() / yticks()</code>	<code>plt.xticks([1,2,3], ["A", "B", "C"])</code>
坐标轴对齐	<code>ax.set_aspect('equal')</code>	<code>ax.set_aspect('auto')</code>

# ► Matplotlib 的基础：网格与坐标轴控制

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 40, 30, 80]

fig, ax = plt.subplots(figsize=(8, 6))

# 绘图
ax.plot(x, y, marker='o')

# 1. 显示网格线
plt.grid(True)

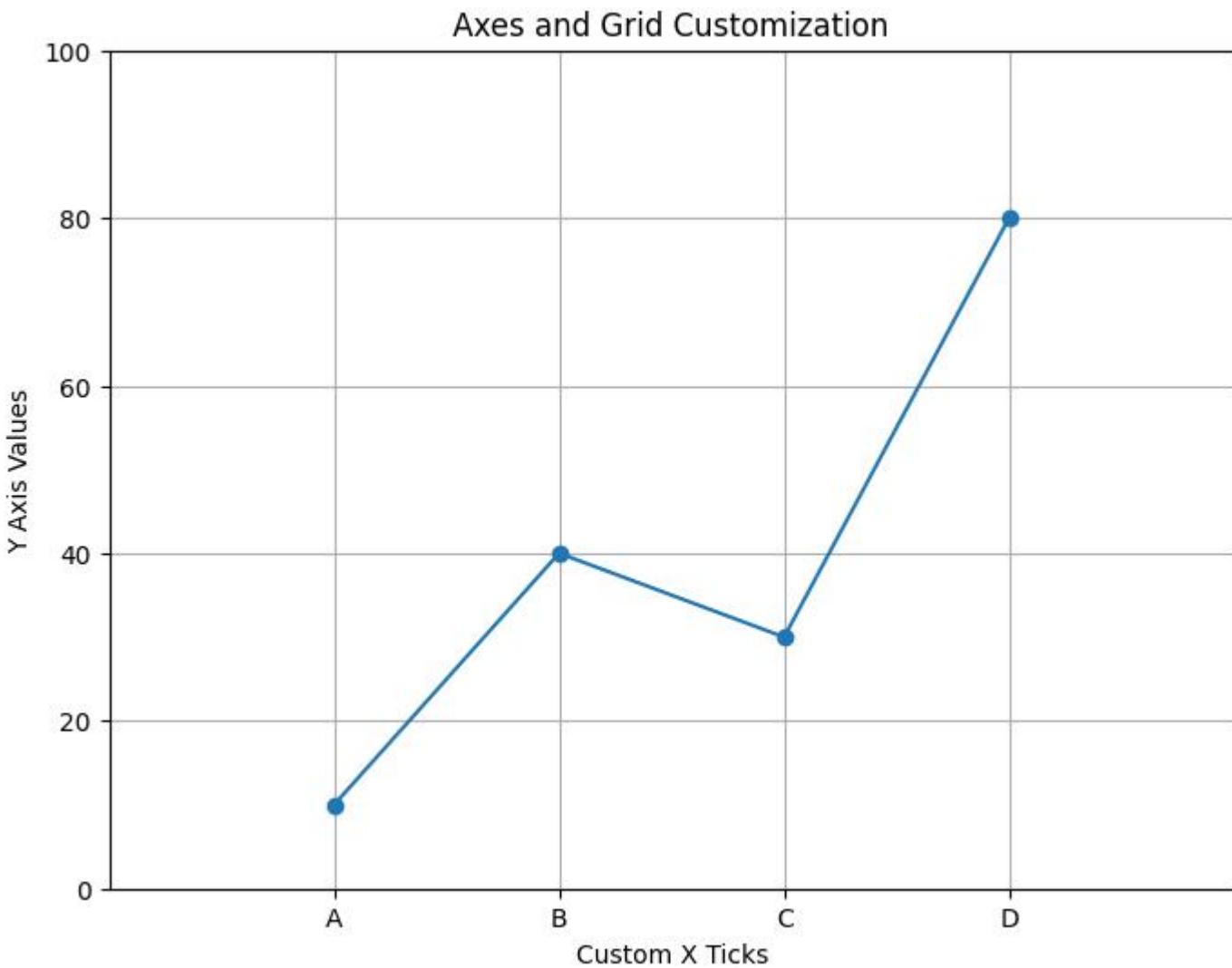
# 2. 设置坐标轴范围
plt.xlim(0, 5)
plt.ylim(0, 100)

# 3. 设置刻度值及刻度标签
plt.xticks([1, 2, 3, 4], ["A", "B", "C", "D"])
plt.yticks([0, 20, 40, 60, 80, 100])

# 4. 设置坐标轴比例（对齐方式）
ax.set_aspect('auto') # 可改为 'equal' 试试看

# 标题和标签
plt.title("Axes and Grid Customization")
plt.xlabel("Custom X Ticks")
plt.ylabel("Y Axis Values")

# 显示图形
plt.show()
```



## ► Matplotlib 的基础：多子图布局

- Matplotlib 提供 subplot() 和 subplots() 两种方式创建多子图，适合展示多个图表在同一画布中的排列效果。
- 通过参数如 nrows, ncols, sharex, tight\_layout() 等可以灵活控制子图数量、位置和布局紧凑性。

布局方式	函数/方法	示例
简单子图	subplot(nrows, ncols, index)	plt.subplot(2, 1, 1)
自动布局	subplots()	fig, ax = plt.subplots(2, 2)
紧凑布局	plt.tight_layout()	plt.tight_layout()
共享坐标轴	sharex, sharey	plt.subplots(2, sharex=True)

# ► Matplotlib 的基础：多子图布局

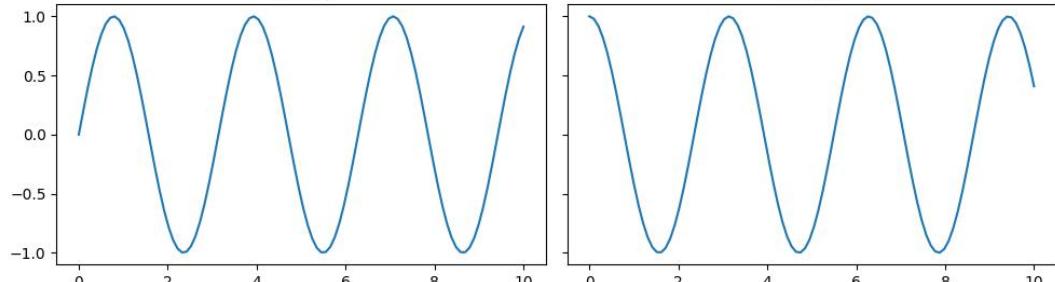
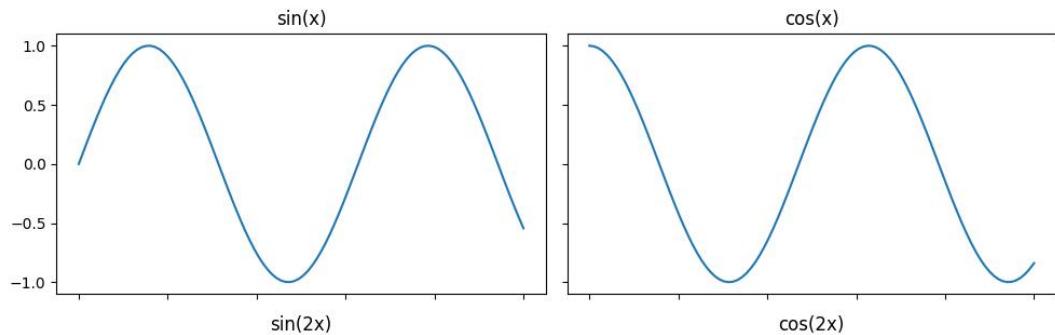
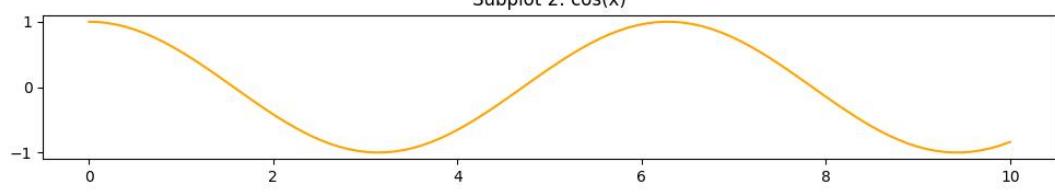
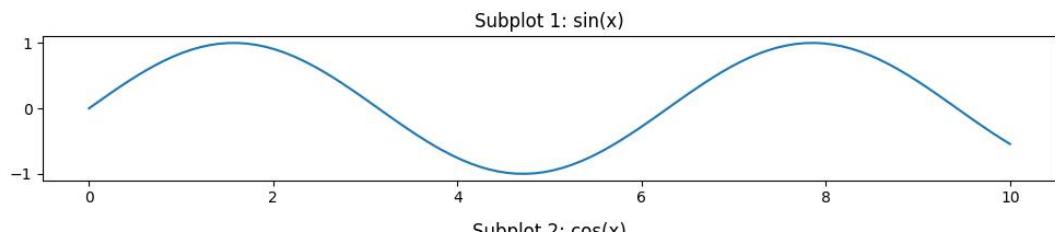
```
import matplotlib.pyplot as plt
import numpy as np

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

# 1. 使用 subplot(nrows, ncols, index)
plt.figure(figsize=(10, 4))
plt.subplot(2, 1, 1) # 两行一列, 第一个子图
plt.plot(x, y1, label='sin(x)')
plt.title("Subplot 1: sin(x)")
plt.subplot(2, 1, 2) # 第二个子图
plt.plot(x, y2, label='cos(x)', color='orange')
plt.title("Subplot 2: cos(x)")
plt.tight_layout() # 紧凑布局, 避免重叠
plt.show()

# 2. 使用 subplots() 自动布局 + 共享坐标轴
fig, axs = plt.subplots(2, 2, figsize=(10, 6), sharex=True, sharey=True)
axs[0, 0].plot(x, np.sin(x))
axs[0, 0].set_title("sin(x)")
axs[0, 1].plot(x, np.cos(x))
axs[0, 1].set_title("cos(x)")
axs[1, 0].plot(x, np.sin(2*x))
axs[1, 0].set_title("sin(2x)")
axs[1, 1].plot(x, np.cos(2*x))
axs[1, 1].set_title("cos(2x)")

# 设置整体布局紧凑
plt.tight_layout()
plt.show()
```



## ► Matplotlib 的基础：图像保存

- Matplotlib 使用 `plt.savefig()` 方法可以将图表保存为 PNG、PDF、SVG 等多种格式的图像文件。
- 通过参数如 `dpi` 设置分辨率、`bbox_inches='tight'` 去除空白边距，使图像更适合用于展示或出版。

操作	函数	示例
保存图像	<code>plt.savefig()</code>	<code>plt.savefig("图表.png")</code>
设置分辨率	<code>dpi=参数</code>	<code>plt.savefig("图.png", dpi=300)</code>
去除边距	<code>bbox_inches='tight'</code>	<code>plt.savefig("图.png", bbox_inches='tight')</code>
多参数组合	一起设置多个参数	<code>plt.savefig("plot.png", dpi=300, bbox_inches='tight')</code>

# ► Matplotlib 的基础：图像保存

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

# 生成数据
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# 创建图像
plt.figure(figsize=(6, 4))
plt.plot(x, y, label='sin(x)', color='blue')
plt.title("Sine Wave")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.legend()
plt.grid(True)

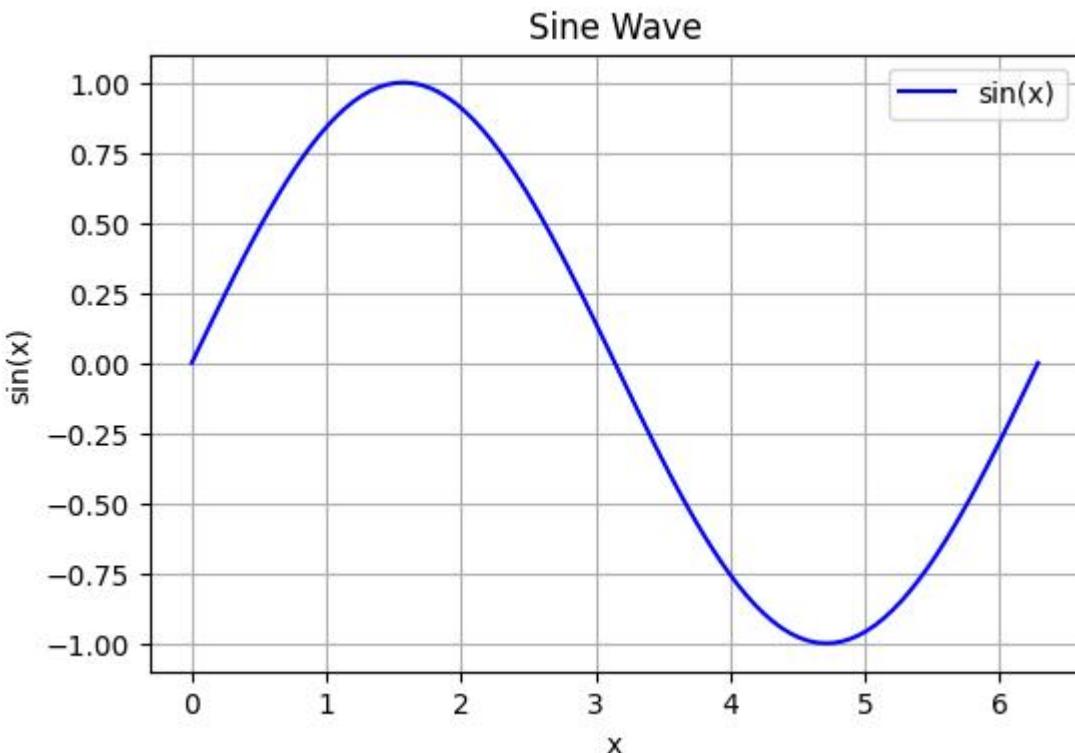
# 保存图像（同时展示三种设置）
if not os.path.exists("output"):
    os.makedirs("output")

# 1. 基本保存
plt.savefig("output/basic_plot.png")

# 2. 设置高分辨率
plt.savefig("output/high_res_plot.png", dpi=300)

# 3. 去除多余边距
plt.savefig("output/tight_plot.png", bbox_inches='tight')

# 可选：也可以展示图像
plt.show()
```



## ► Matplotlib 的基础：联合Pandas快速绘表

- Matplotlib 可与 Pandas 紧密配合，直接通过 DataFrame.plot() 快速生成常见图表。详细可见官方文档。

操作	函数	示例
折线图	df.plot()	df.plot(x="月份", y="销售额")
柱状图	kind="bar"	df.plot(kind="bar")
饼图	kind="pie"	df["类型"].value_counts().plot(kind="pie")
多列绘图	多列字段	df.plot(y=["A", "B"])

这种方式简单高效，适用于数据分析中对折线图、柱状图、饼图等的快速可视化需求。

# ► Matplotlib 的基础：联合Pandas快速绘表

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

# 设置中文字体
plt.rcParams['font.family'] = 'Arial Unicode MS' # 确保你的系统中有这个字体

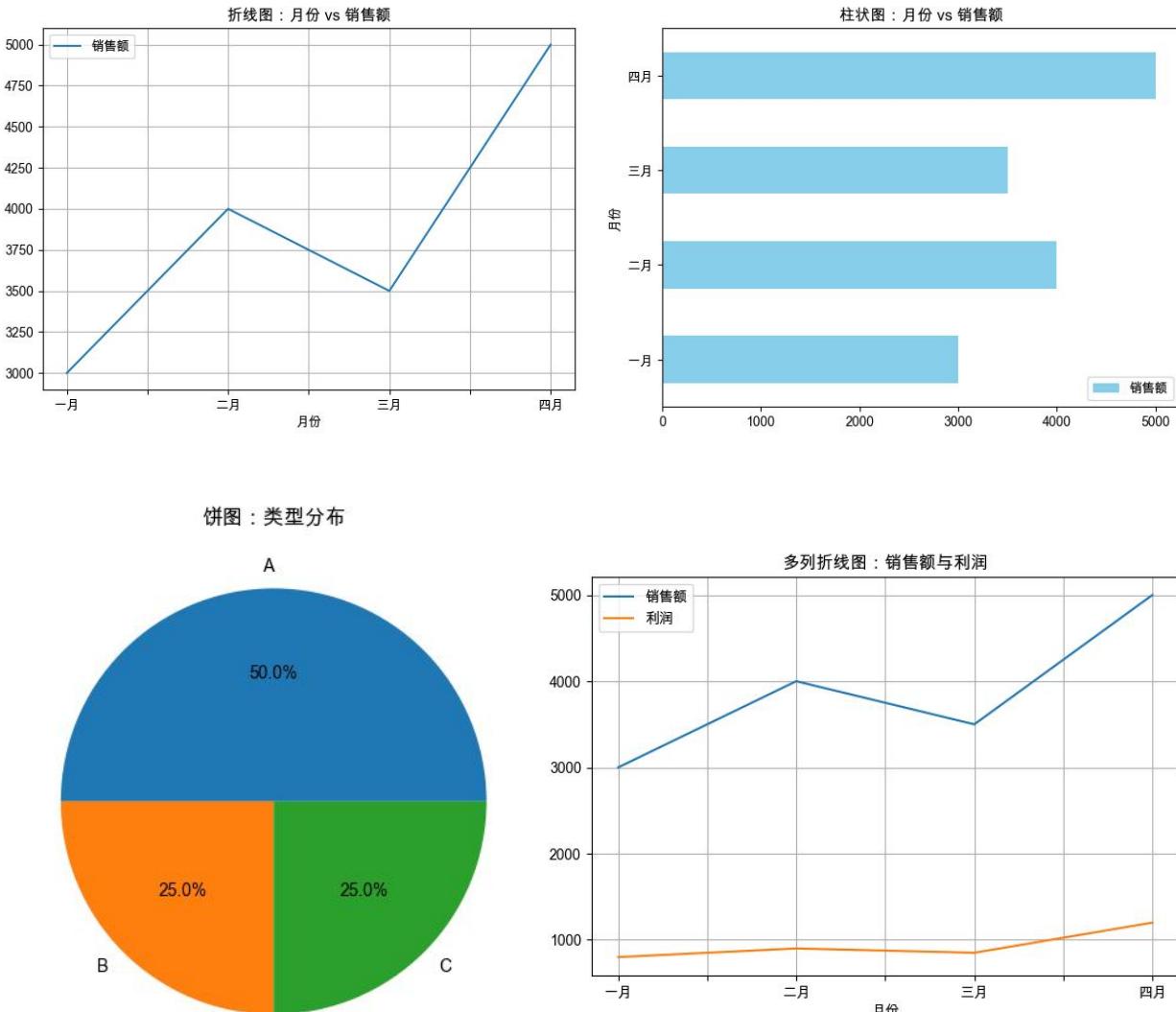
# 构造示例数据
data = {
    "月份": ["一月", "二月", "三月", "四月"],
    "销售额": [3000, 4000, 3500, 5000],
    "利润": [800, 900, 850, 1200],
    "类型": ["A", "B", "A", "C"]
}
df = pd.DataFrame(data)

# 折线图 (以“月份”为横轴)
df.plot(x="月份", y="销售额", title="折线图：月份 vs 销售额")
plt.grid(True)
plt.tight_layout()
plt.show()

# 柱状图
df.plot(x="月份", y="销售额", kind="barh", title="柱状图：月份 vs 销售额", color='skyblue')
plt.tight_layout()
plt.show()

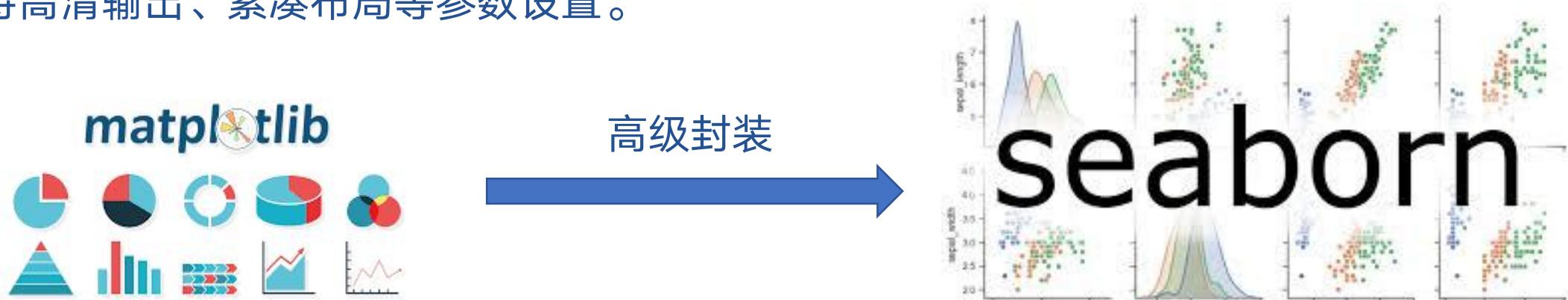
# 饼图 (类型分布)
df["类型"].value_counts().plot(kind="pie", autopct='%1.1f%%', title="饼图：类型分布")
plt.ylabel("") # 去除默认的 y 轴标签
plt.tight_layout()
plt.show()

# 多列绘图 (销售额 & 利润)
df.plot(x="月份", y=["销售额", "利润"], title="多列折线图：销售额与利润")
plt.grid(True)
plt.tight_layout()
plt.show()
```



## ► 小结

- Matplotlib 是 Python 中最基础、最强大的绘图库之一，支持绘制各种静态、动态和交互式图表。常用图形包括：折线图、散点图、柱状图、饼图、直方图等。
- 核心结构包括：Figure（画布）、Axes（子图区域）、Axis（坐标轴）、Artist（图形元素）。同时支持样式自定义（颜色、线型、标记、字体）和多子图布局，适合用于数据分析、科研展示和教学。
- 可与 Pandas 无缝集成，实现 DataFrame.plot() 快速可视化。图表可保存为多种格式，支持高清输出、紧凑布局等参数设置。



用 Matplotlib 画图，就像给数据穿上华丽外衣，一行代码，瞬间变得赏心悦目！

# 目录章节

---

## CONTENTS

01

语法基础与数据类型

02

Numpy：数值计算

03

Pandas：数据处理

04

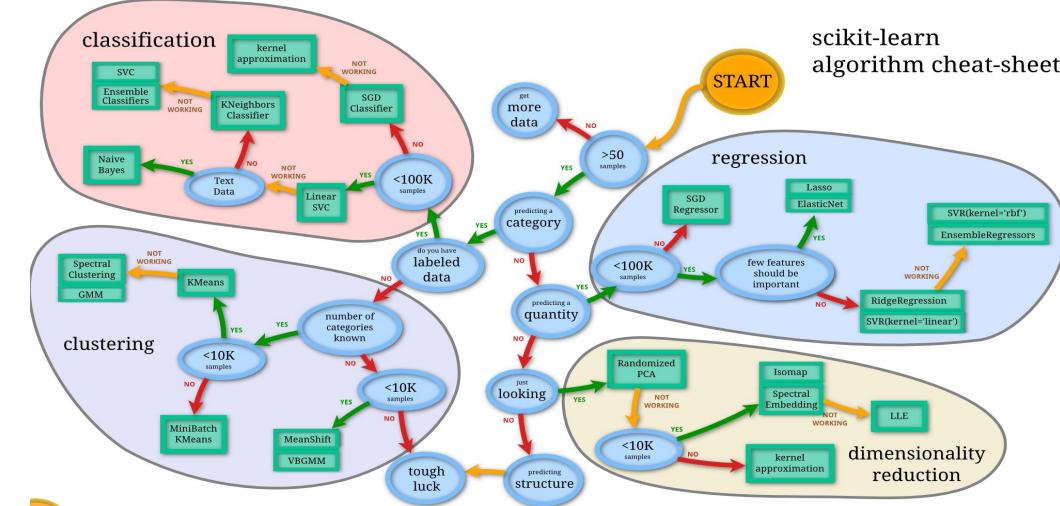
Matplotlib：数据可视化

05

Scikit-learn：机器学习

# ► Scikit-learn 是什么？

- Scikit-learn (sklearn) 是一个基于 Python 的开源机器学习库，提供了统一且简洁的 API 来处理数据预处理、模型训练、评估与调优任务。
- Scikit-learn 的特点：1) 基于 NumPy、SciPy 和 matplotlib；2) 支持 监督学习（分类、回归）与 非监督学习（聚类、降维）；3) 提供大量成熟的算法：如 SVM、决策树、KNN、随机森林、PCA 等；4) 模块化设计、接口统一，适合教学与工程项目。



**Scikit-learn：机器学习工具箱，一行代码搞定训练预测调参，算法上阵不求人！**

# ▶ Scikit-learn 的应用场景

- Scikit-learn 适用于金融风控、商业智能、医学预测、文本分析等多种实际场景，助力数据驱动决策。



场景	描述	使用模型
科研分析	数据探索、回归建模、统计建模	LinearRegression, PCA
商业智能	用户分类、购买预测、推荐系统	KMeans, RandomForest
金融风控	信用评分、欺诈检测	LogisticRegression, SVC
医疗诊断	疾病预测、图像分析	DecisionTree, KNN
产品功能	文本分类、情感分析、广告点击预测	NaiveBayes, SGDClassifier
制造与质量控制	设备异常检测、故障预测	IsolationForest, SVM

Scikit-learn 广泛应用于分类、回归、聚类、降维等机器学习任务。

## ► Scikit-learn 基础：分层架构

- Scikit-learn 的基础是统一的模型接口 + 标准的数据结构 + 流水线式的机器学习流程。



统一的API接口设计

所有模型都使用`.fit()`来训练，`.predict()`来预测，`.score()`或评估指标函数以评价结果



数据格式：Numpy或  
DataFrame

输入数据通常是二维的 X  
(特征) 和一维的 y (标签)

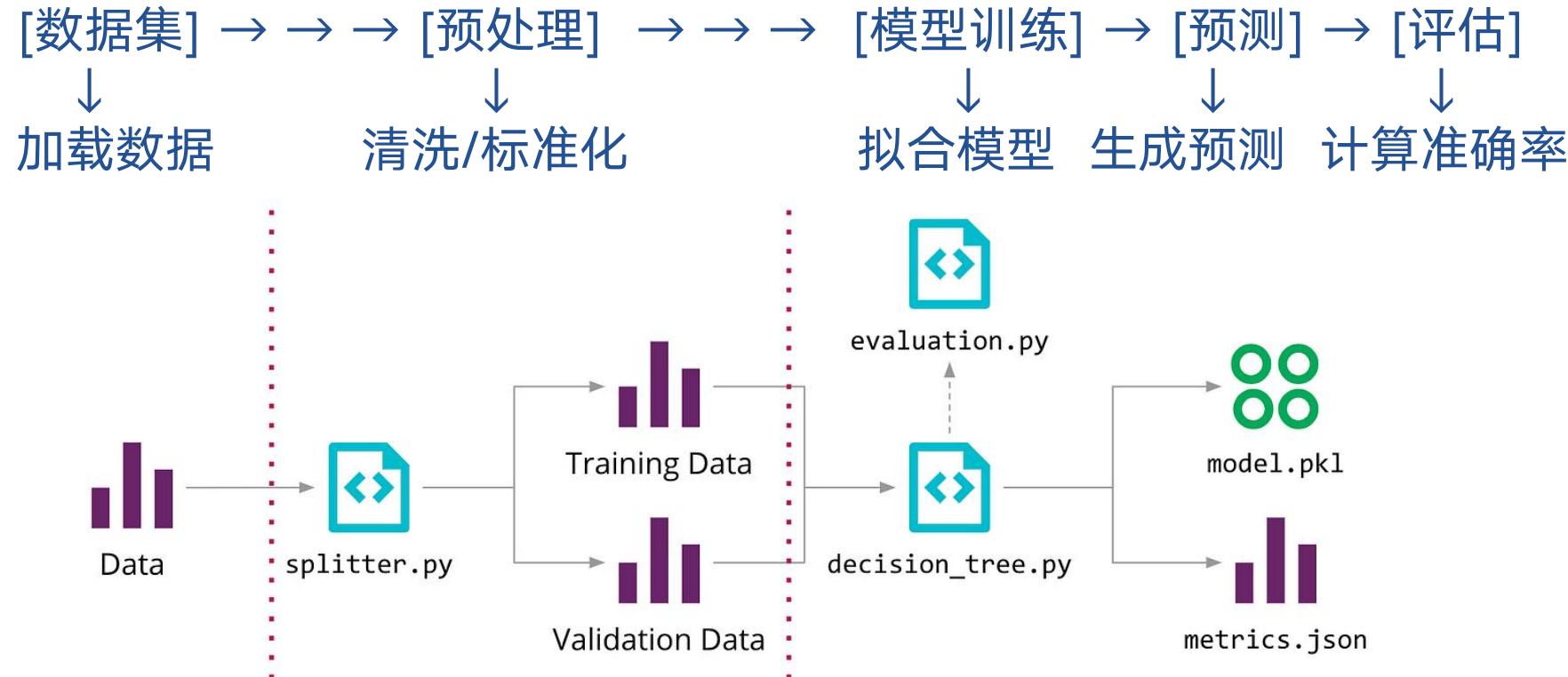


机器学习流程

数据预处理 -> 训练/测试集划分 -> 模型选择与训练 -> 模型评估与调参

## ► Scikit-learn操作：基本使用流程

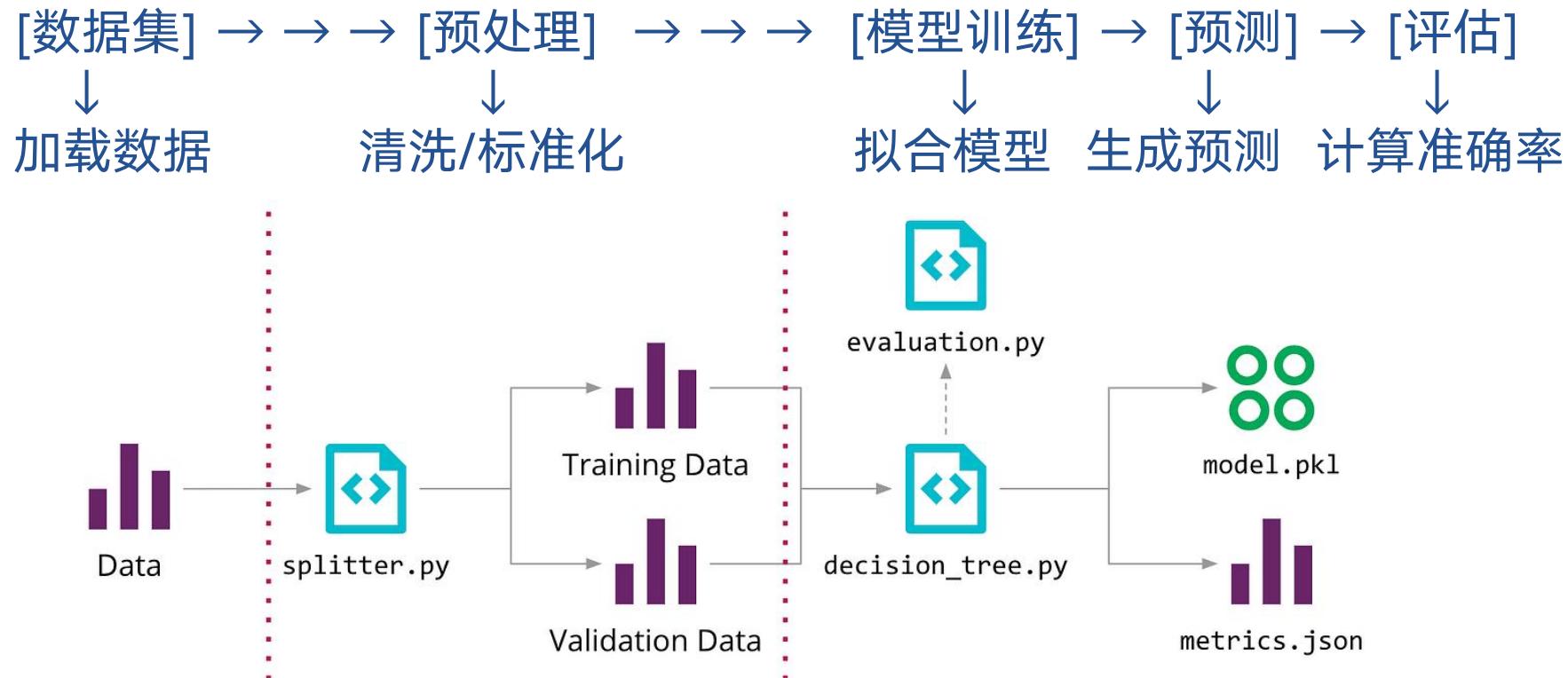
- Scikit-learn的使用流程归纳如下：**导入数据 -> 拆分训练/测试集 -> 模型选择与训练 -> 模型预测 -> 模型评估**



**载数据，切两半，挑模型，学一遍，猜得准，点个赞！**

## ► Scikit-learn操作：基本使用流程

- Scikit-learn的使用流程归纳如下：导入数据 -> 拆分训练/测试集 -> 模型选择与训练 -> 模型预测 -> 模型评估



载数据，切两半，挑模型，学一遍，猜得准，点个赞！

# ► Scikit-learn操作：基本使用流程

- Scikit-learn的使用流程归纳如下：**导入数据 -> 拆分训练/测试集 -> 模型选择与训练 -> 模型预测 -> 模型评估**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 1. 加载数据
X, y = load_iris(return_X_y=True)

# 查看数据
print("特征数据形状:", X.shape)
print("标签数据形状:", y.shape)
print("特征数据示例:\n", X[:5])
print("标签数据示例:", y[:5])

# 2. 划分数据
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. 建模训练
model = LogisticRegression()
model.fit(X_train, y_train)

# 4. 预测
y_pred = model.predict(X_test)

# 5. 评估
print("准确率:", accuracy_score(y_test, y_pred))
```

特征数据形状: (150, 4)

标签数据形状: (150, )

特征数据示例:

[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]]

标签数据示例: [0 0 0 0 0]

准确率: 1.0

**载数据，切两半，挑模型，学一遍，猜得准，点个赞！**

## ► Scikit-learn操作：常用模块和工具箱

- Scikit-learn 提供丰富的模块工具，覆盖数据加载、预处理、建模、评估与调参全流程。通过统一的 API，用户可以轻松切换模型、组合流程，高效完成机器学习任务。

功能	模块	示例
数据预处理	preprocessing	标准化、独热编码
数据集加载	datasets	<code>load_iris()</code> 、 <code>fetch_20newsgroups()</code>
模型选择	model_selection	<code>train_test_split()</code> 、交叉验证
模型训练	各类 Classifier/Regressor	决策树、逻辑回归、SVM 等
模型评估	metrics	<code>accuracy_score()</code> 、 <code>confusion_matrix()</code>
管道与调参	Pipline, GridSearchCV	组合预处理与模型，调最优参数

这些模块就像积木，预处理、建模、评估各司其职，灵活组合可构建完整机器学习管道。

## ► Scikit-learn操作：延伸与实践建议

- 实战技巧：实战中，数据预处理是基础，标准化和特征选择能有效提升模型效果；构建 Pipeline 可以把预处理与建模流程封装在一起，简洁又稳定；通过参数调优与交叉验证，可找到更优解法，避免模型过拟合或欠拟合。

技巧	工具/方法	作用
数据标准化	StandardScaler()	让特征均衡，提升模型表现
特征选择	SelectKBest, PCA	降维去噪，提高效率和准确率
模型保存加载	joblib, pickle	模型训练一次，多次复用
管道构建	Pipeline	将预处理 + 模型封装成一步
参数调优	GridSearchCV	自动搜索最优参数组合
交叉验证	cross_val_score()	更稳定的模型评估方式

不同的技巧需要根据数据，模型，训练效果等灵活选择。

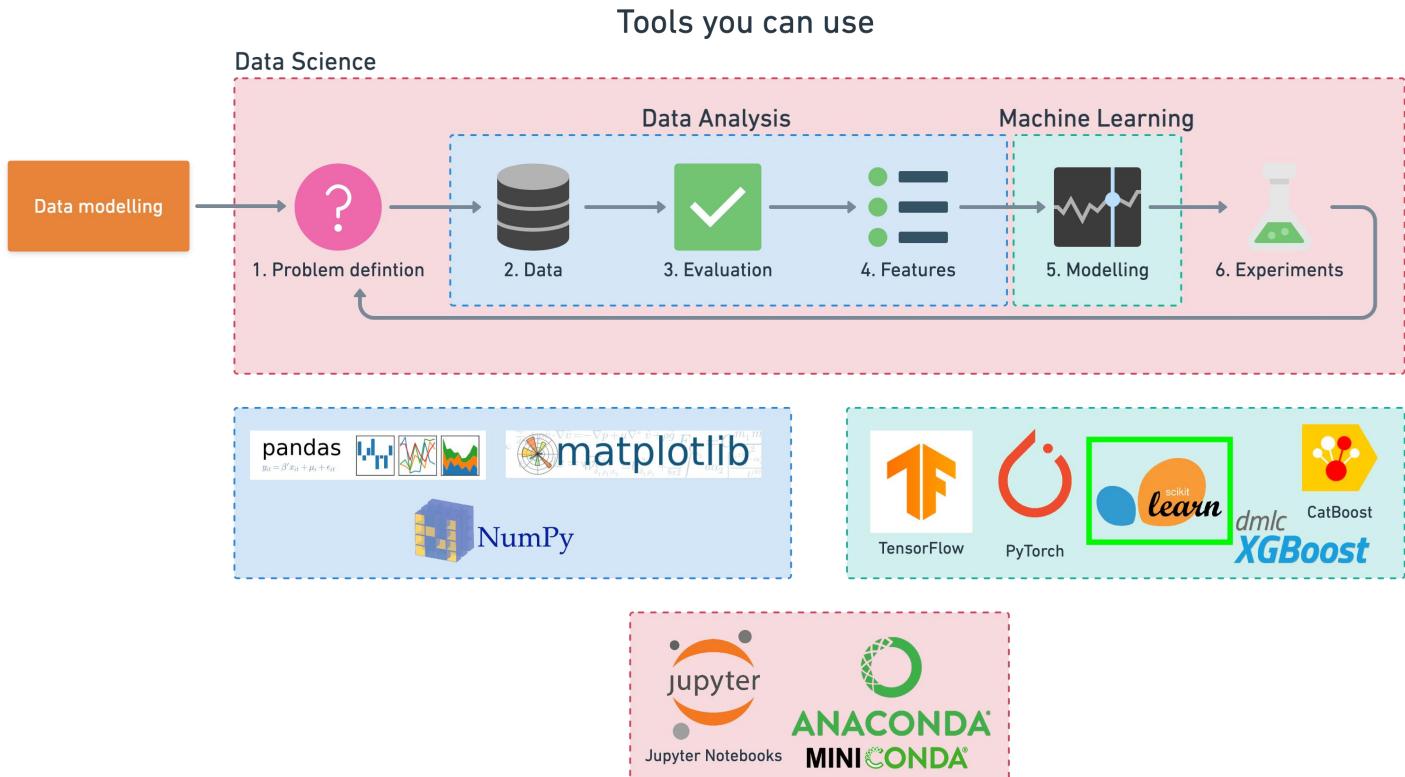
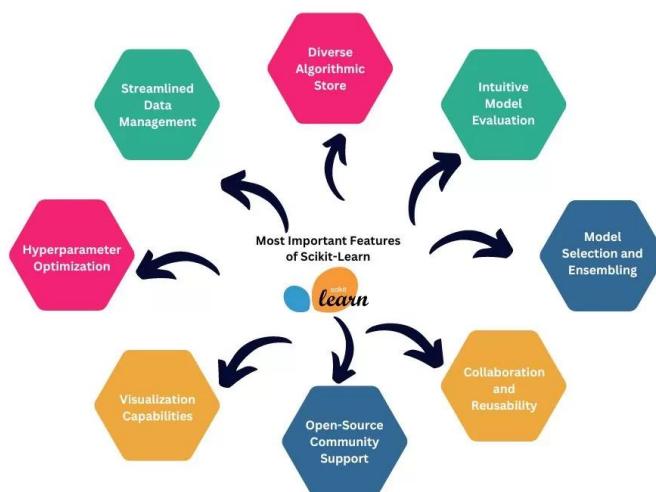
## ► Scikit-learn操作：延伸与实践建议

- 进阶技巧包括类别特征编码、样本不平衡处理与集成学习，能显著提升模型实战效果与泛化能力。

技巧/方法	用途/优势
类别特征编码	使用 OneHotEncoder 或 OrdinalEncoder 处理文本类特征
自定义特征工程	利用 FunctionTransformer 或 ColumnTransformer 做列处理
样本不平衡处理	使用 class_weight='balanced' 或结合 SMOTE 等方法
模型融合与集成学习	VotingClassifier, Bagging, Boosting 提高模型鲁棒性
目标编码	特别适用于类别特征与目标强相关的情况（需注意泄漏风险）
学习曲线分析	用 learning_curve 评估模型是否过拟合/欠拟合
模型持久化	使用 joblib.dump() 保存训练好的模型，便于部署或复用
降维可视化	使用 PCA, t-SNE 等方式帮助理解高维数据
自动化构建流程	使用 Pipeline + GridSearchCV 实现全流程自动调参

# ► 小结

- Scikit-learn 是 Python 最常用的机器学习库，封装了从数据预处理到模型评估的全流程工具。
- 它语法统一、上手简单，几行代码即可完成训练与预测任务。



**掌握 Sklearn，是迈入数据科学与 AI 应用的第一扇门。**

# ► 扩展：进阶学习

## Data 100: Principles and Techniques of Data Science

UC Berkeley, Summer 2025

[Ed](#) [Datahub](#) [Pensieve](#) [Additional Accommodations](#) [Office Hours Queue](#)

 Josh Grossman HE/HIM  
jdgg AT berkeley DOT edu  
data100.instructors@berkeley.edu

 Michael Xiao HE/HM  
michaelxiao1999 AT berkeley DOT edu  
data100.instructors@berkeley.edu

Welcome to Week 4 of Data 100!  
Lectures will be webcast at: <https://berkeley.zoom.us/j/91772046015>.

Mon June 23:	<a href="#">LECTURE 1</a> Introduction <a href="#">LECTURE PARTICIPATION 1</a> Slido <a href="#">LAB 1</a> Prerequisite Coding (due Thu 6/26) <a href="#">HOMEWORK 1</a> Prerequisite Math and Coding (due Fri 6/27)
Tue June 24:	<a href="#">LECTURE 2</a> Pandas I <a href="#">LECTURE PARTICIPATION 2</a> Slido
Wed June 25:	<a href="#">LECTURE 3</a> Pandas II <a href="#">LECTURE PARTICIPATION 3</a> Slido <a href="#">DISCUSSION 1</a> Prerequisites
Thu June 26:	<a href="#">LECTURE 4</a> Pandas III <a href="#">LECTURE PARTICIPATION 4</a> Slido <a href="#">LAB 2A</a> Pandas (due Mon 6/30)
Fri June 27:	<a href="#">HOMEWORK 2A</a> Food Safety (due Wed 7/2)
Week 2	
Mon June 30:	<a href="#">LECTURE 5</a> Data Cleaning & EDA <a href="#">LECTURE PARTICIPATION 5</a> Slido <a href="#">DISCUSSION 2</a> Pandas I
Tue July 1:	<a href="#">LAB 2B</a> Data Cleaning & EDA (due Thu 7/3) <a href="#">LECTURE 6</a> Regex <a href="#">LECTURE PARTICIPATION 6</a> Slido

 **rougier / numpy-100**

≡ kaggle

- + Create
- ⟳ Home
- 🏆 Competitions
- 📁 Datasets
- 📊 Models
- 📊 Benchmarks
- <> Code
- 💬 Discussions
- 🎓 Learn
- ⌄ More
- ⌚ Your Work
- ⌄ VIEWED
- 👤 Meet The Grandmasters
- 📺 Titanic - Machine Lear...
- 📺 KNN Classification mo...
- 📺 House Prices | AECin...

[View Active Events](#)  
<https://www.kaggle.com/competitions>

 **guipsamora / pandas\_exercises**

Host a Competition Your Work

Search competitions

All Competitions Everything, past & present

Featured Premier challenges with prizes

Getting Started Approachable ML fundamentals

Research Scientific and scholarly challenges

Community Created by Kagglers

See all

Getting Started Competitions with approachable ML fundamentals.

 Titanic - Machine Learning from... Start here! Predict survival ... Getting Started 15405 Teams  Knowledge Ongoing	 House Prices - Advanced... Predict sales prices and pr... Getting Started 4675 Teams  Knowledge Ongoing	 Spaceship Titanic : Advanced... Predict which passengers ... Getting Started 2021 Teams  Knowledge Ongoing	 LLM Classification : Finetuning Finetune LLMs to Predict H... Getting Started - Code Co... 330 Teams  Knowledge Ongoing
			

Hotness

# 感谢聆听



Personal Website: <https://www.miaopeng.info/>



Email: [miaopeng@stu.scu.edu.cn](mailto:miaopeng@stu.scu.edu.cn)



Github: <https://github.com/MMeowhite>



Youtube: <https://www.youtube.com/@pengmiao-bmm>