Python 101

1 聊聊“巨蟒”

有用废话多是 ChatGPT 生成，安装测试 Anaconda 是关键

悟知天地纵横交错，万物相生相联之网。而你我也系其中一环，一念一动皆牵动周身。

There is urgency in coming to see the world as a web of interrelated processes of which we are integral parts, so that all of our choices and actions have consequences for the world around us.

—— 阿尔弗雷德·怀特海 (Alfred Whitehead) | 英国数学家、哲学家 | 1861 ~ 1947

1.1 ChatGPT：用 Python 写成的人工智能工具

本节很多问题都采用 ChatGPT 答案，作者对回答文本略加编辑。本书中，ChatGPT 的答案用 标识。建议大家在学习时，不管是概念、程序、数学上遇到问题，都可以使用类似 ChatGPT 工具作为助手。值得注意的是，ChatGPT 经常犯错，大家注意甄别。

什么是 ChatGPT?

ChatGPT 是一种基于自然语言处理 (Natural Language Processing, NLP) 技术的人工智能（AI）应用程序，它是由 OpenAI 公司开发的一种大规模预训练语言模型。ChatGPT 使用深度神经网络来模拟人类的对话过程，它可以理解和生成人类语言，可以用于实现聊天机器人、智能客服、智能助手等应用。ChatGPT 还可以帮助用户进行文本自动生成、文本摘要、文本分类、情感分析等任务。ChatGPT 使用 Python 编程语言进行搭建。在搭建 ChatGPT 时，OpenAI 使用了 Python 的深度学习框架 TensorFlow 和 PyTorch，以及一些其他的 Python 库和工具来完成。

1.2 Python? 巨蟒?

Python 的用途极为广泛，特别是在机器学习、深度学习领域。Python 免费开源，语言语法友好，而且社区活跃。这就是为什么“鸢尾花书”系列会选择 Python 作为编程语言的原因。

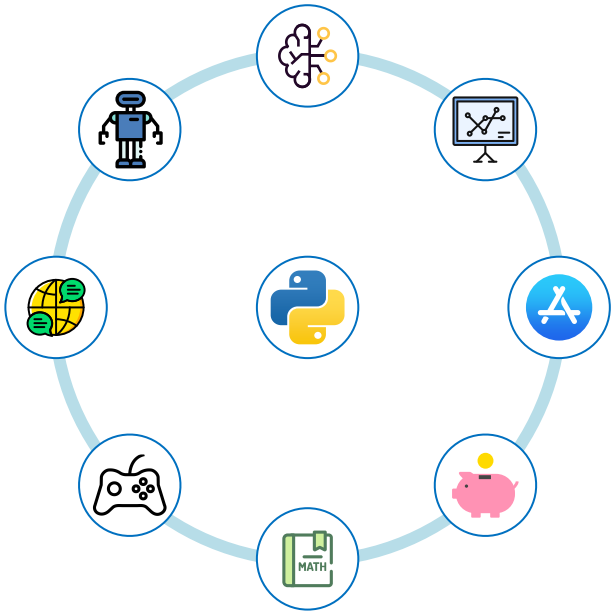


图 1. Python 应用场景

我们为什么要学 Python?

如图 1 所示，Python 具有广泛的用途，比如：

- 机器学习：Python 在数据科学和机器学习领域非常受欢迎，因为它有很多强大的库和工具，如 NumPy、Pandas、Scikit-learn 等。
- 深度学习：Python 的深度学习工具，比如 PyTorch、TensorFlow，常用来开发各种人工智能应用，比如智能设备、无人驾驶、自然语言处理工具等。
- Web 开发：Python 可以用于 Web 开发，有许多流行的 Web 框架，如 Django、Flask 等。
- 自动化脚本：Python 可以用于自动化任务，例如自动备份、自动化测试、爬虫等。
- 游戏开发：Python 可以用于游戏开发，如 Pygame 等库和工具。
- 系统管理和网络编程：Python 可以用于系统管理和网络编程，例如网络爬虫、服务器开发、安全工具等。

什么是 Python?

Python 是一种高级编程语言，由 Guido van Rossum 于 1991 年首次发布。它是一种解释型语言，使用动态类型系统和自动内存管理。Python 具有简单易学、易于阅读和编写、可移植性强等特点，广泛应用于 Web 开发、数据分析、人工智能、科学计算、自动化等领域。Python 语言具有丰富的标准库和第三方库，可支持各种编程任务，例如文件处理、网络编程、GUI 开发、图像处理、数据可视化、机器学习等。Python 的语法简洁清晰，易于阅读和理解，因此也被广泛应用于教育和科研领域。Python 语言的解释器可用于不同的操作系统，例如 Windows、MacOS、Linux 等，因此 Python 具有很好的跨平台性。

图 2 所示为本书涉及到的 9 个重要的 Python 库。

图 2.《编程不难》涉及到的 Python 库

Python 中，什么是模块、包、库?

在 Python 中，模块、包、库是三个常见的概念。它们的含义如下：

模块 (Module)：是一个 Python 程序文件，包含了一组相关的函数、类、变量和常量等，可供其他程序引用。Python 中的模块是一种可重用的代码组件，可用于将相关的代码组织到一起，以便更好地管理和维护代码。一个模块可以包含多个函数、类、变量和常量等，可以被其他模块或程序引用和调用。

包 (Package)：是一组相关的模块的集合，用于组织 Python 代码的层次结构。一个包是一个文件夹，其中包含其他模块或子包。包是一种通过模块命名空间进行模块组织的方式，可用于更好地组织和管理大型代码库。

库 (Library)：是由一组模块和包组成的软件组件，提供了一系列函数、类、变量和常量等，用于解决特定问题。Python 标准库是 Python 官方提供的一组库，包含了大量的模块和功能，可以直接使用。此外，还有第三方库，如 NumPy、Pandas、Matplotlib 等，用于数据处理、科学计算、可视化等领域。

需要注意的是，模块是最小的可重用代码单元，而包和库是由多个模块组成的更大的结构。在 Python 中，通常使用 import 语句来引入需要使用的包、库或模块。

我们为什么要学 Python?

作者认为，面向人工智能时代的教育，特别是数学教育，必须结合编程、可视化、应用。而 Python 既是编程工具，也拥有大量可视化工具，同时可以用来完成各种数据科学、机器学习任务。“编程 + 可视化 + 数学 + 机器学习”正是“鸢尾花书”创作的内核，贯穿整套图册始终。

对于初高中生、大学生，学习 Python 有很多好处，比如：

- 培养编程思维：Python 作为一种编程语言，可以帮助大家培养编程思维能力。大家可以通过编写简单的程序和解决各种问题，锻炼逻辑思考、问题解决和创造力等能力。
- 高效地学习数学及其他学科：将公式、模型写成 Python 代码的过程，本身就是一种“习题”。而且这类习题比传统课本习题更能激发大家的兴趣。
- 图形化强化记忆：公式、定理、定义、解题技巧 … 大家考完试也就忘记了。但是利用 Python 编程，把公式、定理、定义变成一幅幅活生生的图形之后，这些概念将会深深地刻在大家脑中，甚至一辈子不会忘记。
- 提高学习效率：Python 可以用于自动化各种重复性的任务，如数据处理、文本处理等。大家可以通过编写 Python 程序来自动化这些任务，从而节省时间和精力，提高学习效率。
- 为未来的学习和职业做准备：Python 是一种非常流行的编程语言，它在数据科学、机器学习、人工智能等领域有很多应用，大家可以通过学习 Python，为未来的学习和职业做准备，提前掌握一些必要的技能和知识。
- 增强实践能力：学习 Python 可以让大家更容易地将学到的知识应用到实践中，例如编写简单应用程序、游戏、网站等，这有助于大家增强实践能力和探索新领域的能力。
- 培养团队合作意识：学习 Python 可以让大家更容易地与他人合作，例如在编写程序时可以分工合作、交流想法等，这有助于培养团队合作意识和沟通能力。

未来已来，你来不来？

怎么学 Python?

和汉语、英语一样，Python 也是语言。只不过，Python 是人和机器交互的语言。Python 的语法有绝对的对错，不能模棱两可。错误的命令，要么出 bug，要么得出错误的结果。

大家学 Python 时，一定要吸取英语教育的教训。学了十几年的哑巴英语，原因就是只关注词汇、语法、应试，而不是利用语言进行口头、书面交流。这是学 Python 必须避免。建议大家在学 Python 编程的时候，一定要抱着极强的“功利心 ”——立刻开始 coding、debugging!

学习 Python 和学习英语有什么相似之处?

学习 Python 和学习英语有一些相似之处，以下是其中一些：

都需要掌握基础知识：学习 Python 和学习英语都需要掌握基础知识，Python 的基础语法、数据类型、流程控制语句、函数等，英语的基本词汇、语法、发音等。

都需要不断练习：学习 Python 和学习英语都需要不断地练习，Python 需要编写程序来实践，英语需要口语练习和写作练习。

都需要实践和应用：学习 Python 和学习英语都需要不断地实践和应用，Python 可以应用到数据处理、人工智能、游戏开发等领域，英语可以应用到国际交流、留学、工作等方面。

都需要耐心和坚持：学习 Python 和学习英语都需要耐心和坚持，需要花费大量时间和精力来学习和练习，才能达到良好的掌握和应用水平。

总之，学习 Python 和学习英语都需要掌握基础知识、不断练习、实践和应用，同时也需要耐心和坚持。虽然二者是不同的领域，但都是对自己未来发展非常有帮助的技能。

鸢尾花书有编程、可视化、数学、机器学习这几个重要的元素，而这几个元素都离不开 Python。下面我们逐一聊一下。

1.3 Python 和可视化有什么关系？

Python 和可视化有很密切的关系。Python 中有很多强大的可视化库和工具，可以帮助用户对数据进行可视化呈现。

以下是 Python 和可视化的一些关系：

- 数据可视化：Python 中有许多数据可视化的库，例如 Matplotlib、Seaborn、Plotly 等，可以帮助用户将数据可视化呈现出来，从而更好地理解数据的分布、趋势等信息。本书的绘图部分将蜻蜓点水地讲解 Matplotlib、Seaborn、Plotly 常用绘图命令。“鸢尾花书”的《可视之美》一册将专门讲解数据可视化这一话题。
- 图像处理：Python 中有许多图像处理的库，例如 OpenCV 等，可以帮助用户进行图像处理和分析，同时也可以将处理后的图像进行可视化呈现。
- 交互式可视化：Python 中也有许多用于交互式可视化的库，例如 Bokeh、Altair 等，可以帮助用户建立交互式的数据可视化应用程序。
- 3D 可视化：Python 中也有许多用于 3D 可视化的库，例如 Mayavi、VisPy 等，可以帮助用户对三维数据进行可视化呈现。

1.4 Python 和数学有什么关系？

Python 和数学有着密切的关系。Python 是一种非常适合数学建模和数据分析的编程语言，拥有大量的数学计算库和工具。

以下是 Python 和数学的一些关系：

- 数学计算：Python 中有很多用于数学计算的库和工具，例如 NumPy、SciPy 等，可以帮助用户进行矩阵运算、微积分、最优化、统计分析等数学计算任务。
- 数据分析：Python 中有很多用于数据分析的库和工具，例如 Pandas、Matplotlib、Seaborn 等，可以帮助用户对数据进行统计分析、可视化呈现等。
- 数学建模：Python 中还有很多用于数学建模的库和工具，例如 SymPy 等，可以帮助用户进行数学建模和优化任务。
- 教学和研究：Python 也被广泛应用于数学教学和研究领域，例如用 Python 实现数学实验、数学模型的探索、算法的实现等。

1.5 Python 和机器学习有什么关系？

Python 与机器学习有非常密切的关系。Python 是一种简单易学、可读性强的编程语言，同时也拥有丰富的第三方库和工具，这使得 Python 成为机器学习领域的重要工具之一。

机器学习是一种应用人工智能的技术，通过让计算机从数据中学习并改善性能，来实现对未知数据的预测和决策。

Python 在机器学习领域的应用非常广泛，主要有以下几个方面：

- 数据处理和分析：Python 中有许多用于数据处理和分析的库，例如 Pandas、NumPy 和 SciPy，这些库能够帮助用户轻松地处理和分析数据。
- 机器学习框架：Python 中也有许多用于机器学习的框架，例如 TensorFlow、PyTorch 和 Scikit-Learn 等，这些框架可以帮助用户更加高效地进行机器学习建模和预测。
- 可视化工具：Python 中的 Matplotlib 和 Seaborn 等可视化库，可以帮助用户更加清晰地理解数据和模型，以及呈现结果。
- 自然语言处理：Python 中的自然语言处理库，例如 NLTK 和 Spacy 等，可以帮助用户进行文本数据的处理、分析和预测。

什么是机器学习？

机器学习是一种人工智能技术，它使计算机系统能够通过数据和经验自主学习和改进，而无需显式地编程指令。简单来说，机器学习是通过训练算法从数据中学习模式和规律，然后利用这些模式和规律来进行预测或决策。在机器学习中，模型是通过训练算法从大量数据中学习而来的，这些数据被称为训练数据集。训练数据集包含已知结果的输入输出对，这些输入输出对用于训练模型来预测未知数据的输出。训练数据集中的数据越多，训练时间越长，模型就越准确。机器学习可以应用于各种领域，例如语音识别、图像识别、自然语言处理、推荐系统和金融分析等。它已成为当今科技领域中最热门和最具前途的领域之一。

1.6 集成开发环境

Python 有很多常用的 IDE (集成开发环境)，比如：

- JupyterLab：基于 Web 的交互式开发环境，支持多种编程语言，包括 Python，可以快速编写、测试和共享代码，非常适合数据科学和机器学习领域。作者认为，JupyterLab 和 Jupyter Notebook 非常适合大家做探究式学习。目前，《编程不难》、《可视之美》两册的配套的代码多是 Jupyter 笔记。这个话题后文将详细介绍如何使用 JupyterLab。
- Spyder：基于 Qt 开发的 Python IDE，提供了一个集成的开发环境，包括编辑器、调试器和控制台，非常适合科学计算和数据分析。虽然“鸢尾花书”剩余几册的代码都是在 Spyder 中完成，建议初学者还是在 JupyterLab 中分段运行代码。对于 MATLAB 转 Python 的读者来说，Spyder 可能是最容易上手的 IDE。在所有的 Python IDE 中，Spyder 最像 MATLAB。
- PyCharm：JetBrains 公司开发的跨平台 Python IDE，提供了许多功能，包括代码智能提示、代码自动完成、调试和单元测试等。建议有 Python 开发经验的读者使用 PyCharm 运行本书代码。

什么是集成开发环境？

集成开发环境 (Integrated Development Environment，简称 IDE) 是一种用于软件开发的工具。它通常包括一个代码编辑器、一个调试器和一个构建工具，以及其他功能，例如自动补全、语法高亮、代码重构等。IDE 的目的是提供一个集成的工作环境，使开发人员能够更高效地编写、调试和测试代码。使用 IDE 可以极大地提高开发效率。例如，它可以帮助开发人员在编写代码时自动补全函数名称、参数等，减少打错代码的风险；它可以提供一些调试工具来检测和修复代码中的错误，使得开发人员更容易发现问题；它可以通过自动构建工具来编译和构建代码，减少手动操作的繁琐过程。总之，IDE 是一种开发人员必备的工具，可以让开发人员更加专注于编写高质量的代码。

表 1. 比较三个常用的 IDE

维度	JupyterLab	Spyder	PyCharm
适用场景	数据科学、机器学习、交互式	科学计算、数据分析	通用编程、开发
编辑器	基于 Web 的文本编辑器	Qt 构建的文本编辑器	IntelliJ IDEA 编辑器
调试器	内置的交互式调试器	内置的调试器	内置的调试器
插件支持	丰富的插件生态系统	插件支持较少	丰富的插件生态系统
社区支持	由 Jupyter 项目支持	由 Spyder 社区支持	由 JetBrains 公司支持
扩展性	支持自定义和扩展	可以自定义外观和行为	支持自定义和扩展
学习曲线	平缓	友好	稍微陡峭
收费?	免费	免费	有免费和付费版本
平台支持	支持 Windows、Mac 和 Linux	支持 Windows、Mac 和 Linux	支持 Windows、Mac 和 Linux

Anaconda

Anaconda 可谓“科学计算全家桶”，包含科学计算领域可能用到的大部分 Python 工具，包括 Python 解释器、常用的第三方库、包管理器、IDE 等。前文提到的 JupyterLab、Spyder、PyCharm 这三个 IDE 都在 Anaconda 中。

什么是 Anaconda?

Anaconda 是一个流行的 Python 发行版，由 Anaconda, Inc.开发和维护，旨在为数据科学、机器学习和科学计算提供一个全面的工具包。Anaconda 集成了许多常用的 Python 库和工具，如 NumPy、SciPy、Pandas、Matplotlib、Scikit-learn、Jupyter Notebook 等。它还包括一个名为 conda 的软件包管理器，可以帮助用户安装、更新和管理 Python 库和依赖项。Anaconda 还提供了一个名为 Anaconda Navigator 的图形用户界面，用户可以通过这个界面轻松地管理他们的 Python 环境、安装和卸载库、启动 Jupyter Notebook 等操作。除了 Python 环境和库之外，Anaconda 还包括许多其他工具和应用程序，如 Spyder、PyCharm、VS Code、R 语言环境等等，使得它成为数据科学家和研究人员的首选工具之一。Anaconda 可以安装在多个平台上，包括 Windows、Linux 和 Mac OS X。

安装 Anaconda

下文手把手教大家如何在 Windows 上安装、测试 Anaconda，有经验的读者可以跳过。

对于 Mac 用户，大家可以参考如下链接安装 Anaconda：

https://docs.anaconda.com/anaconda/install/mac-os/

要是想特别安装某个版本的 Python，请参考：

https://pythonhowto.readthedocs.io/zh_CN/latest/install.html

注意，Anaconda 安装后大概占用 5G 空间。有 Python 开发经验的读者，可以根据需求自行分别安装 JupyterLab、Spyder、PyCharm。

在 Windows 上安装 Anaconda 可以按照以下步骤进行：

a) 下载。在 Anaconda 官网 (https://www.anaconda.com/) 下载适合大家操作系统的 Anaconda 版本，选择对应的 Python 版本 (一般建议选择最新版 Python3.x)，并下载对应的安装程序。注意，Anaconda 不断推出新版本，大家下载的版本号肯定和下图的版本号不同。建议大家从官网下载最新版本安装程序。

图 3. 安装程序图标

b) 运行安装程序：下载完毕后，双击下载文件运行安装程序。在安装程序打开后，点击“Next”进入下一步。

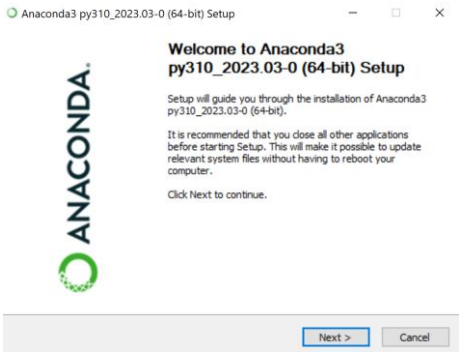


图 4. 运行安装程序

c) 阅读协议：阅读协议并同意“I Agree”，然后点击“Next”。

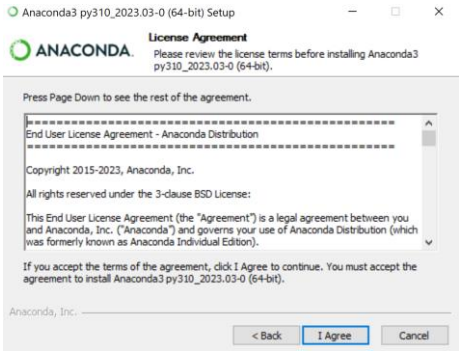


图 5. 阅读协议

d) 安装类型：推荐默认“Just Me”；对于多用户 PC，可以选择“All Users”；然后点击“Next”。

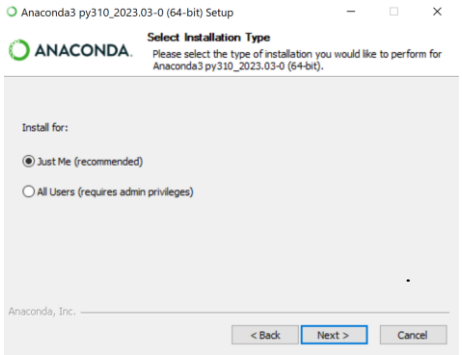


图 6. 安装类型

e) 安装路径：可以指定 Anaconda 的安装路径 (建议零基础读者选择默认路径)，然后点击“Next”。

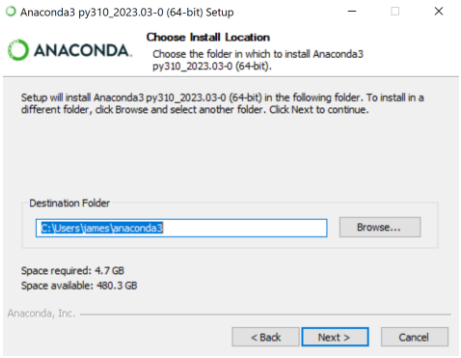


图 7. 安装路径

f) 配置环境变量：选择是否将 Anaconda 添加到系统环境变量中，建议勾选该选项，这样就可以在命令行中使用 Anaconda 的工具了。然后点击“Install”进行安装。

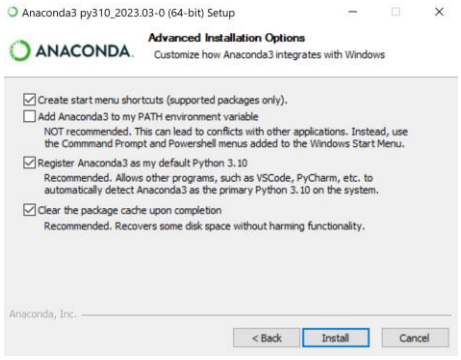


图 8. 安装选择

g) 等待安装完成：安装过程可能持续 10 分钟左右。等待安装完成后，会弹出“Installation Complete”对话框，点击“Next”。如果这步持续时间过长 (超过一小时)，建议强制停止安装，删除安装包。关机再开机，重新下载安装包从头开始再尝试安装。

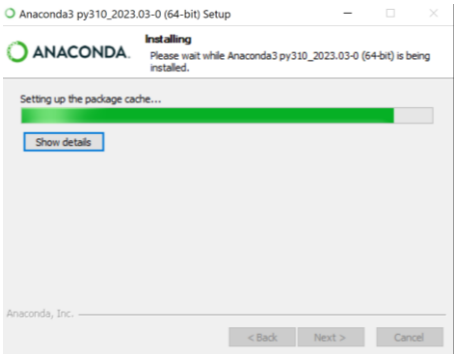


图 9. 等待安装完成

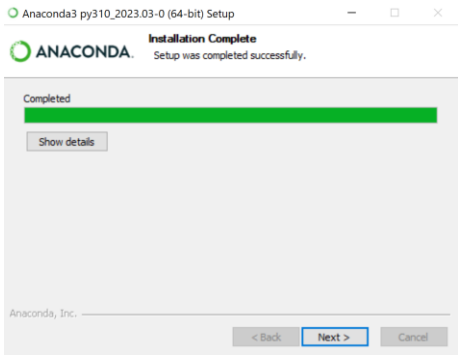


图 10. 安装完成

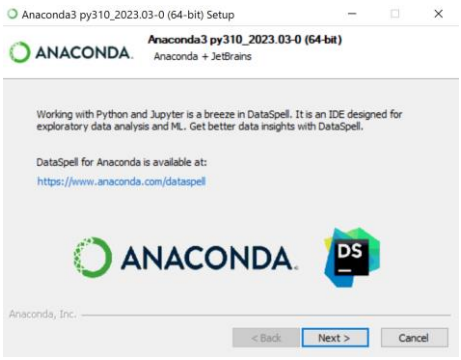


图 11. 广告时间，点 Next

h) 完成安装：点击“Finish"完成 Anaconda 的安装。之后会跳出两个网页，不需要理会，关闭即可。

图 12. 确认完成

安装完成后，可以在“开始菜单”中找到 Anaconda 的安装目录，并启动“Anaconda Navigator”来使用 Anaconda 的工具和功能。同时，也可以在命令行中使用 Anaconda 的工具和命令，例如使用“conda”命令来管理 Python 的虚拟环境和安装依赖包等。

测试 JupyterLab

这是本节最后，也是最关键的一个任务。

要打开并测试 JupyterLab，可以按照以下步骤进行：

a) 找到并打开 Anaconda Navigator (需要 1 分钟左右，稍安勿躁)，点击 JupyterLab 对应的 Launch。马上一个网页将会跳出来，建议大家默认使用 Chrome 浏览器，Firefox 或 Edge 也都可以。

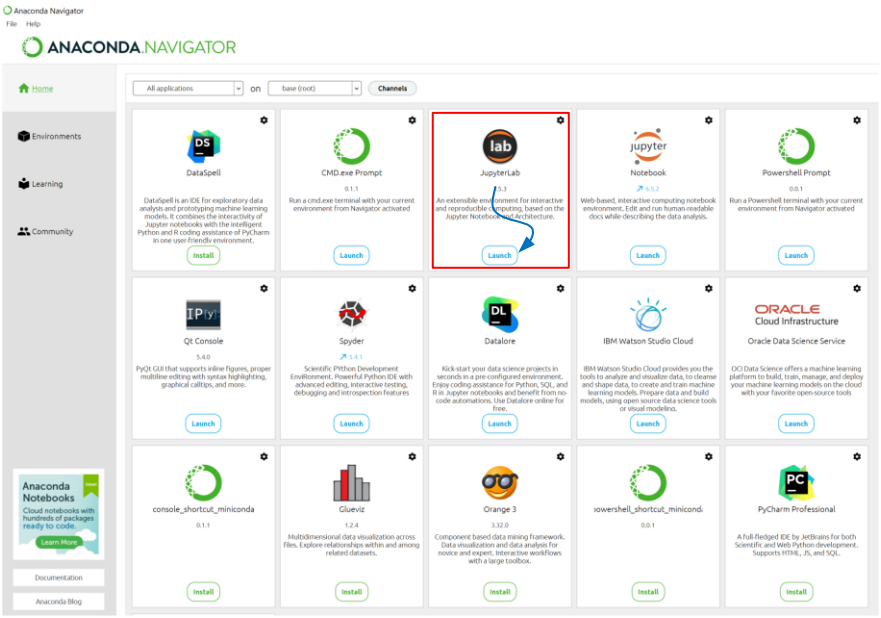


图 13. Anaconda Navigator 界面

b) 进入 JupyterLab 界面，点击 Notebook (Python 3)，创建 Jupyter Notebook。

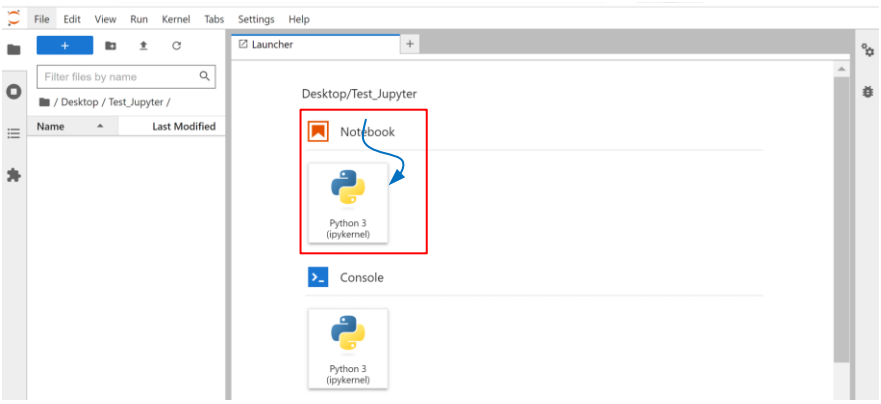


图 14. JupyterLab 界面

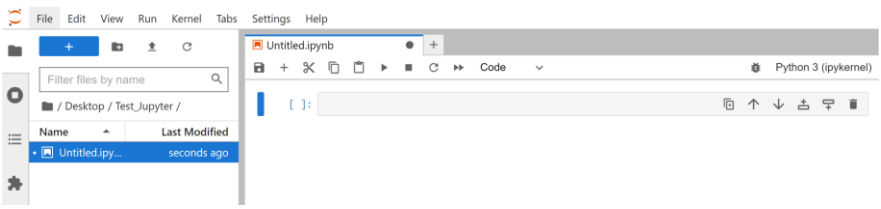


图 15. 创建 Jupyter Notebook

c) 在下面窗口中输入，1 + 2，然后点击“Ctrl + Enter”快捷键，运行并得到 3 这个结果。大家也可以尝试“Shift + Enter”快捷键，运行代码同时生成新区块，大家自己可以先玩一会。下一节将专门讲解如何使用 JupyterLab。

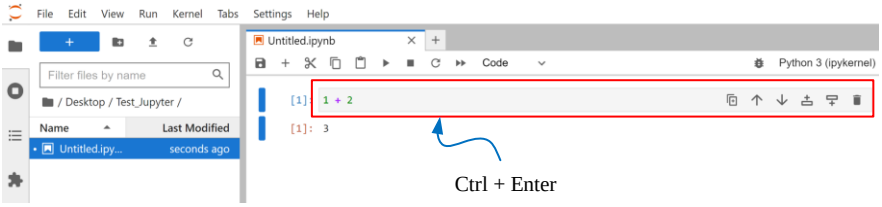


图 16. 运算

这一节的习题只需要大家完成 Anaconda 安装，并测试 JupyterLab。

* 这道题目很基础，本书不给答案。

Using JupyterLab

2 JupyterLab，用起来!

特别适合探究式学习，代码、绘图、脚本、公式 ...

> 教育不是为生活做准备；教育就是生活本身。
>
> ***Education is not a preparation for life; education is life itself.***
>
> —— 约翰·杜威 (John Dewey) | 美国著名哲学家、教育家、心理学家 | 1859 ~ 1952

2.1 啥是 JupyterLab?

JupyterLab 集合“浏览器 + 编程 + 文档 + 绘图 + 多媒体 + 发布”众多功能与一身。“鸢尾花书”不同场合反复提过，对于初学者，哪怕是有开发经验的读者来说，JupyterLab 都特别适合探究式学习。目前《数学要素》、《可视之美》中，几乎所有的代码都是用 JupyterLab 写的。如果大家对 JupyterLab 反馈正面，其他分册也考虑提供 Jupyter Notebook 配套文件。

这一话题将和大家聊一聊如何使用 JupyterLab。注意，本节不求“事无巨细”地介绍 JupyterLab，而是要全景地浏览 JupyterLab 的主要功能，保证“够用就好”，以便大家轻装上阵。

对于 JupyterLab 的外观、窗口布局等细节问题，这个话题就不展开了，大家如果有需要可以很容易搜索到结果。当大家对 JupyterLab 熟悉之后，建议大家了解如何用 JupyterLab 的 debug 功能。此外，很多开发者专门针对 JupyterLab 开发各种小插件，很多插件的确能提高工作效率，也建议大家自行了解。

大家 JupyterLab 用熟之后，会发现这一节最重要的内容只有——快捷键。

什么是 JupyterLab?

JupyterLab 是一个交互式开发环境，可以让用户创建和共享 Jupyter 笔记本、代码、数据和文档。它是 Jupyter Notebook 的升级版本，提供了更强大的功能和更直观的用户界面。JupyterLab 支持多种语言，包括 Python、R、Julia 和 Scala 等。它还提供了多个面向数据科学的扩展，如 JupyterLab Git、JupyterLab LaTeX 和 JupyterLab Debugger 等，使得数据科学家和开发人员可以更加高效地进行数据分析、机器学习和模型开发等工作。JupyterLab 的主要特点包括：基于 web 的用户界面，可以让用户同时在一个界面中管理多个笔记本和文件。支持多种文件格式，包括 Jupyter 笔记本、Markdown 文档、Python 脚本和 CSV 文件等。可以通过拖放和分栏等方式来组织和管理笔记本和文件。提供了一组内置的编辑器、终端、文件浏览器和输出查看器等工具。可以通过扩展系统来扩展和定制 JupyterLab 的功能。

2.2 使用 JupyterLab：立刻用起来

新建 Notebook

大家首先通过 Anaconda Navigator (上一节内容) 打开 JupyterLab。

如图 1 所示，不管点击 A 或 B 都会看到 C 这个图标，点击 C 就会生成一个 Notebook。此外，新建 Notebook 前，点击图 1 中 D，我们可以改变文件路径。

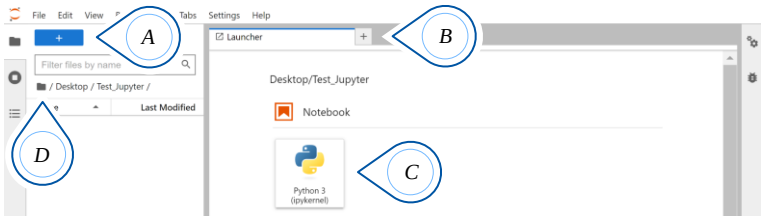


图 1. 新建 Notebook

如图 2 所示，Notebook 界面的有很多板块。

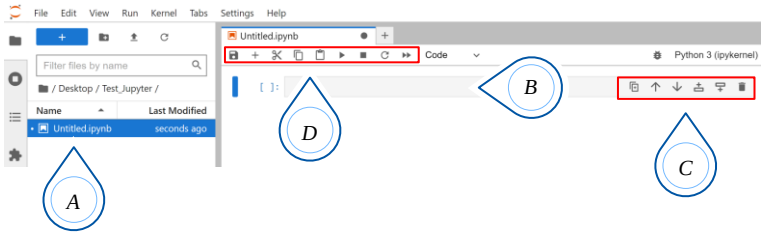


图 2. JupyterLab 中新建 Notebook 界面

JupyterLab 中的 cell 是什么？

在 JupyterLab 中，Cell (单元格) 是指一个包含代码或文本的矩形区域，它是用户编写和执行代码、编写文本和渲染 Markdown 的基本单位。Cell 可以包含多种类型的内容，包括代码、Markdown、LaTeX 公式等。JupyterLab 中的 Cell 可以通过交互式的方式进行编辑和执行。例如，在 Code Cell 中，用户可以编写 Python 代码，并使用 Shift+Enter 快捷键执行代码并显示结果；在 Markdown Cell 中，用户可以使用 Markdown 语法编写文本，并使用 Shift+Enter 快捷键渲染 Markdown 文本。JupyterLab 中的 Cell 还支持多种交互式扩展，例如使用 IPython Magic 命令、使用自动完成、代码补全和代码调试等。Cell 也可以被复制、剪切、粘贴、移动和删除，使得用户可以轻松地组织和管理笔记本中的内容。

对于初学者，大家先注意 4 点：

- 图 2 中的 A 对应的是 Notebook 默认的名字。右键可以对文件进行各种操作，比如重命名、剪切、复制、粘贴、删除等等。
- 图 2 中的 B 是 Notebook 中第一个 cell。在 Notebook 里，一个基本的代码块被称作一个 cell。注意，一个 Notebook 可以有若干 cell；而一个 cell 理论上可以有无数行代码。
- 图 2 中的 C 对应的是 cell 的几个常见操作——复制并向下粘贴、向上、向下、向上加 cell、向下加 cell、删除 cell。
- 图 2 中的 D 对应的操作——保存文件、向下加 cell、剪切 cell、复制 cell、粘贴 cell、运行当前 cell 后移动 (或创建) 到下一个 cell、停止运行、重启 kernel、重启重跑所有 cell、code/markdown 转换。

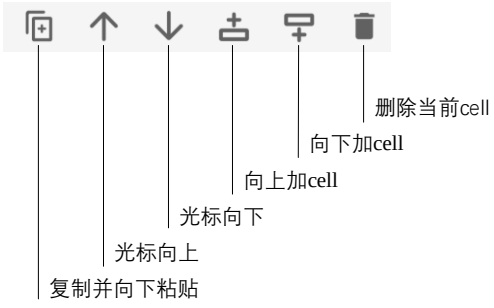


图 3. C 对应的是 cell 的几个常见操作

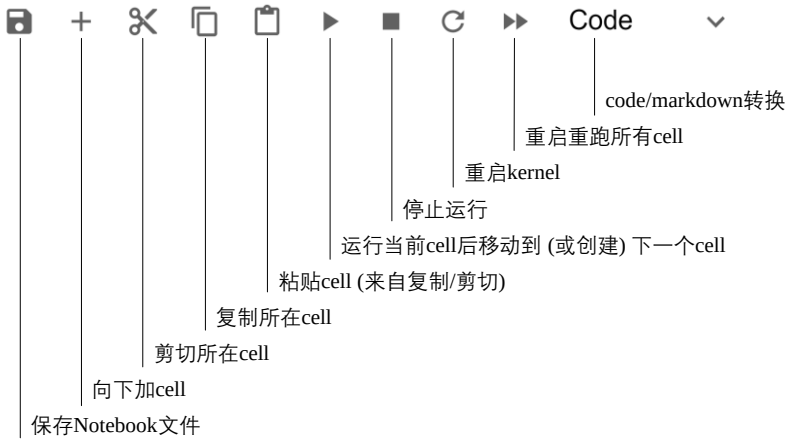


图 4. D 对应的是 cell 的几个常见操作

JupyterLab 中的 kernel 是什么？

JupyterLab 中，内核 (kernel) 是指与特定编程语言交互的后台进程，它负责编译和执行用户在 JupyterLab 中编写的代码，并返回执行结果。内核与 JupyterLab 之间通过一种称为“Jupyter 协议”的通信协议进行交互。打开一个新的 notebook 或 console 时，JupyterLab 会自动启动一个内核，这个内核将与该 notebook 或 console 中编写的代码进行交互。在 notebook 或 console 中编写代码，并使用内核来执行它们。内核还可以保存笔记本中的变量和状态，使得大家可以在多个代码单元格之间共享变量和状态。JupyterLab 支持多种编程语言的内核，可以在启动 notebook 或 console 时选择要使用的内核。例如，如果想使用 Python 内核，可以选择“Python 3”内核。一旦选择了内核，JupyterLab 将与该内核建立连接，并使用它来执行该 notebook 或 console 中编写的代码。如果希望在 notebook 或 console 中使用其他语言的内核，需要先安装并配置这些内核。

代码 vs 文本

Jupyter 的 cell 常用两种状态——代码、文本。文本也叫 markdown。两种状态之间可以相互转换。

顾名思义，代码状态的 cell 中的内容会被视为“代码”，# 开头的部分会被视作为“注释”

文本 markdown 状态下，整个 cell 的内容可以是文本/Latex 公式/超链接/图片等等，这个 cell 不会被当成代码执行。图 4 中的“code/markdown”选项可以帮助我们在两种 cell 状态切换。

我们常在 JupyterLab 中敲入各种 Latex 公式，本书后续将会见缝插针地讲解如何用 Latex 写各种公式。

多数时候为了提高切换效率，我们通常使用快捷键。下面介绍 JupyterLab 中常用的快捷键。

本节配套的 Jupyter Notebook 文件 BK_2_Topic_1.02_1.ipynb 向大家展示如何在 Jupyter Notebook 中进行探究式学习。本节配套的微课视频会逐 cell 讲解这个 Notebook 文件。

JupyterLab 中的 markdown 是什么？

在 JupyterLab 中，Markdown 是一种轻量级标记语言，可以用于编写文档、笔记和报告等。通过使用 Markdown 语法，用户可以在 JupyterLab 中轻松地创建格式化文本、插入图片、添加链接、创建列表等。Markdown 语法非常简单，易于学习和使用。例如，使用 Markdown 语法，用户可以使用井号 (#) 来创建标题，使用“-”或“*”符号加上空格来创建 bullet list，使用双星号(**)来加粗文本，使用单星号 (*) 来斜体文本等。用户可以在 Markdown 单元格中编写 Markdown 语法，然后使用 Shift+Enter 键来渲染 Markdown 文本。JupyterLab 中的 Markdown 支持 LaTeX 语法，用户可以使用 LaTeX 语法来插入数学公式，从而方便地创建数学笔记和报告。

2.3 快捷键：这一章最有用的内容

建议大家使用快捷键完成常见 cell 操作。JupyterLab 的快捷键分成两种状态：a) 编辑模式；b) 命令模式。

编辑模式，允许大家向 cell 中敲入代码或 markdown 文本。表 1 总结编辑模式下常用快捷键。为了帮助大家识别这些快捷键组合，图 5 给出标准键盘主键盘上各个按键的位置。

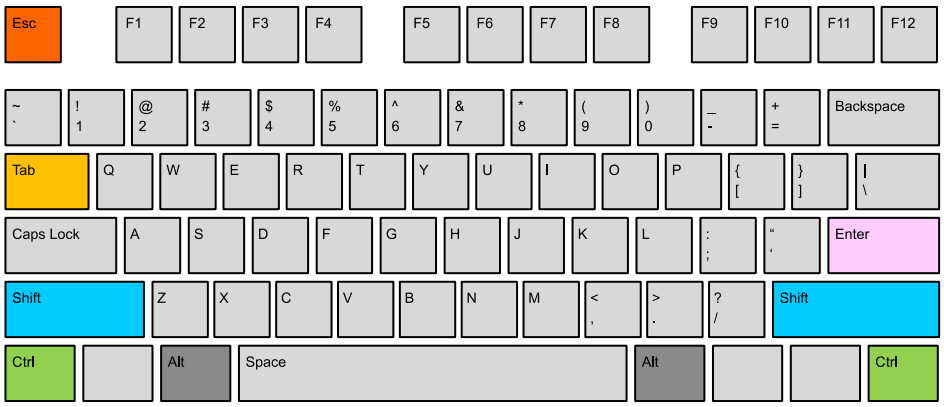


图 5. 标准键盘，Mac 的 command 对应 ctrl

命令模式，单击 esc 进入命令模式，这时可以通过键盘键入命令快捷键。表 2 总结命令模式下常用快捷键。

注意，表格中的加号 + 表示“一起按下”，不是让大家按加号键。加号 + 前后的按键没有先后顺序。

此外，本书 GitHub 中还给出 JupyterLab 快捷键的 cheat sheet，建议大家专门将其打印出来，编程的时候放在一边参考。

表 1 和表 2 两个表格中都是常用默认快捷键。如果大家对某个快捷键组合不满意，可以自行修改。特别是需要在多个 IDE 之间转换时，由于不同 IDE 的默认快捷键不同，一般都会将常用快捷

键统一设置成自己习惯的组合。JupyterLab 中修改快捷键的路径为 Settings → Advanced Settings Editor (或 esc → ctrl + ,) → 搜索 Keyboard Shortcuts。注意，不建议初学者修改默认快捷键。

表 1. 编辑模式，常用快捷键

快捷键组合	功能
esc	进入“命令”模式；鼠标左键单击任何 cell 返回，或单击 enter 返回编辑模式
ctrl + M	进入“命令”模式
ctrl + S	保存；尽管 JupyterLab 会自动保存，建议大家还是要养成边写边存的好习惯
shift + enter	执行 + 跳转；运行当前 cell 中的代码，光标跳转到下一 cell
ctrl + enter	执行；运行当前 cell 中的代码
alt + enter	执行 + 创建 cell；运行当前 cell 中的代码，并在下方创建一个新 cell
ctrl + shift + -	分割；在光标所在位置将代码/文本分割成两个 cells
ctrl + /	注释/撤销注释；对所在行，或选中行进行注释/撤销注释操作
ctrl + [	向右缩进；行首加四个空格
ctrl +]	向左缩进；行首减四个空格
ctrl + A	全选；全选当前 cell 内容
ctrl + Z	撤销；撤销上一个键盘操作
ctrl + shift + Z	重做：恢复刚才撤销命令对应操作，相当于“撤销撤销”
ctrl + C	复制；复制选中的代码或文本
ctrl + X	剪切；剪切选中的代码或文本
ctrl + V	粘贴；粘贴复制/剪切的代码或文本
ctrl + F	查询；实际上就是浏览器的搜索
home	跳到某一行开头
end	跳到某一行结尾
ctrl + home	跳到多行 cell 第一行开头
ctrl + end	跳到多行 cell 最后一行结尾
tab	代码补齐；忘记函数拼写时，可以给出前一两个字母，按 tab 键得到提示
shift + tab	对键入的函数提供帮助文档
ctrl + B	展开/关闭左侧 sidebar

表 2. 命令模式，常用快捷键

快捷键组合	功能
esc	编辑模式下，进入“命令”模式；鼠标左键单击任何 cell 返回，或单击 enter 返回编辑模式
esc → M	在按下 esc 进入编辑模式后，将当前 cell 从代码 markdown 转成文本
esc → Y	将当前 cell 从文本 markdown 转成代码
enter	从命令模式进入编辑模式，或者鼠标左键单击任何 cell

esc → Ⓐ	插入；在当前 cell 上方插入新 cell
esc → Ⓑ	插入；在当前 cell 下方插入新 cell
esc → Ⓓ → Ⓓ	删除；在按下 esc 进入编辑模式后，连续按两下 D，删除当前 cell
esc → ⓪ → ⓪	重启 kernel；在按下 esc 进入编辑模式后，连续按两下零 0，重启 kernel
esc → ctrl + Ⓑ	展开/关闭左侧 sidebar
esc → ctrl + Ⓐ	选中所有 cells
esc → shift + ▲	选中当前和上方 cell，不断按 ctrl + ▲不断选中更上一层 cell
esc → shift + ▼	选中当前和下方 cell，不断按 ctrl + ▼不断选中更下一层 cell
shift + Ⓜ	合并；将所有选中的 cells 合并；如果没有多选 cell，则将当前 cell 和下方 cell 合并
shift + enter	执行 + 跳转；运行当前 cell 中的代码，光标跳转到下一 cell；和编辑模式一致
ctrl + enter	执行；运行当前 cell 中的代码；和编辑模式一致
alt + enter	执行 + 创建 cell；运行当前 cell 中的代码，并在下方创建一个新 cell；和边际模式一致
esc → ①	一级标题，等同于 markdown 状态下 #
esc → ②	二级标题，等同于 markdown 状态下 ##
esc → ③	三级标题，等同于 markdown 状态下 ###，以此类推

这一章的习题很简单，请大家从零开始复刻 Bk1_Ch2_01.ipynb，并在创建 Jupyter Notebook 文档的过程使用快捷键。

* 这道题目很基础，本书不给答案。

Fundamentals of Grammar in Python

4 Python 语法，边学边用

吸取英语学习失败的教训，不能死磕语法

当你建造空中楼阁时，它不会倒塌；空中楼阁本应属于高处。现在，撸起袖子把地基夯实。

If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them.

—— 亨利·戴维·梭罗 (Henry David Thoreau) | 作家、诗人 | 1817 ~ 1862

4.1 Python 也有语法?

和汉语、英语、法语等人类语言一样，Python 也是语言。只不过 Python 是编程语言，是人和计算机交互语言。凡是语言就有语法——一套约定交流规则。

有了类似 ChatGPT 这样的自然语言处理工具，人类的确可以直接使用人类语言和机器交流。但是，考虑到 ChatGPT 也是用 Python 开发而成，Python 不过是退隐幕后罢了。

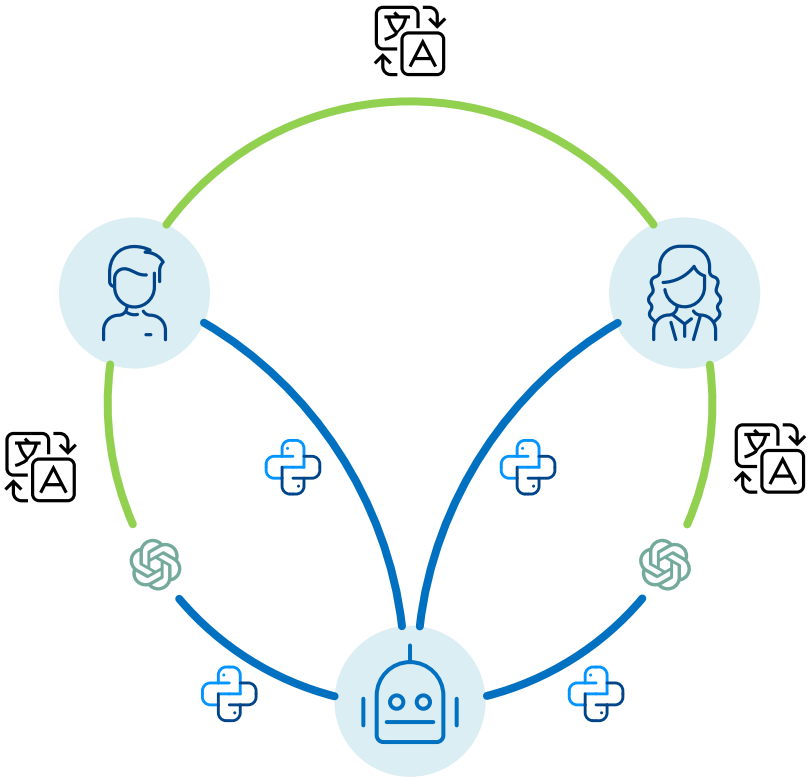


图 1. Python 也是语言

Python 语法使用数量极少的英文词汇，而且都是很基本的词汇。

Python 和英语都有一些关键词，例如 Python 中的 if、else、for、while 等关键词，和英语中的 if、else、for、while 等单词是一样的。

Python 和英语都有语法结构，例如 Python 中的 if 语句和英语中的条件句都是用来表示条件语句的结构。

Python 和英语都有一些语法规则，例如 Python 中的缩进规则和英语中的句子结构规则都是用来规范语法的。

Python 语法相对来说比英语语法容易掌握，因为 Python 语法的规则和规范性更强。

表 1 总结 Python 中常用英文关键词。

⚠ 请大家注意大小写，特别是 True、False、None 需要首字母大写。

表 1. Python 中常用英语关键词

英语	汉语	介绍
and	和	逻辑操作符，要求两个条件都满足时才返回 True。
argument	参数	print('Hey you!') 中的 'Hey you' 是函数 print() 的输入参数
as	作为	用于别名，可以给模块、函数或类指定另一个名称，比如 import numpy as np。
assert	断言	用于测试代码的正确性，如果条件不成立则会引发异常。
boolean	布尔值	True 和 False 是两个布尔值
break	中断	用于跳出循环语句。
class	类	定义一个类，包含属性和方法。
complex	复数	3 + 4j 是一个复数
condition	条件	if x > 0: 是一个条件语句
continue	继续	用于跳过当前循环的剩余部分，继续执行下一次循环。
def	定义	定义一个函数。
del	删除	用于删除变量或对象。
dictionary	字典	{'name': 'James', 'age': 18} 是一个字典
elif	否则如果	用于在 if 语句中添加多个条件判断。
else	否则	用于 if 语句中，当所有条件都不满足时执行。
except	除外	用于捕获异常。
False	假	表示布尔值为假。
finally	最后	用于定义无论是否发生异常都要执行的代码块。
float	浮点数	3.14 是一个浮点数
for	循环	用于迭代遍历序列、集合或其他可迭代对象。
from	来自	用于从模块中导入特定函数、类或变量，比如 from numpy import random。
function	函数	print() 是一个函数
global	全局	用于在函数中引用全局变量。
if	如果	用于条件判断，比如 if x > 0:。
import	导入	用于导入模块，比如 import numpy。
in	在	用于检查元素是否存在于序列、集合或其他可迭代对象中。
integer	整数	3 是一个整数。
is	是	用于检查两个对象是否相同。
lambda	匿名	定义一个匿名函数。
list	列表	[1, 2, 3] 是一个列表。
loop	循环	for i in range(10): 是一个循环语句。
module	模块	import math 导入了 Python 的 math 模块。
None	空	表示一个空值或缺少值。
not	非	逻辑操作符，将 True 变为 False，将 False 变为 True。

object	对象	my_object = MyClass() 中 my_object 是一个 MyClass 类的对象
or	或	逻辑操作符，只要一个条件满足就返回 True。
package	包	import numpy 导入了 Python 的 numpy 包
pass	跳过	用于占位符，不执行任何操作。
raise	引发异常	用于引发异常，比如，raise ValueError("Invalid value.")
return	返回	用于从函数中返回值。
set	集合	{1, 2, 3} 是一个集合
statement	语句	x = 5 是一个赋值语句
string	字符串	Hey you!' 是一个字符串
True	真	表示布尔值为真。
try	尝试	用于包含可能引发异常的代码块，比如 try: except ValueError:。
tuple	元组	(1, 2, 3) 是一个元组
variable	变量	x = 5 中 x 是一个变量
while	当	用于创建循环，只要条件为真就重复执行代码块。
with	使用	用于自动管理资源，例如文件句柄或数据库连接。
yield	产生	用于生成器函数，暂停函数执行并返回一个值。

Python vs C 语言

Python 是一种高级的面向对象编程语言。C 语言是一种编译型语言，非常适合编写底层的系统软件，例如操作系统、编译器和设备驱动程序等。C 语言的优势在于其对硬件和操作系统的底层控制，而这也是 Python 所缺乏的。Python 在处理复杂的数据结构和算法时，通常比 C 语言慢得多。

Python 的优势主要是其强大的第三方库和工具生态系统，使得 Python 可以用于更高层次的机器控制和自动化任务，例如数据处理、机器学习和自然语言处理等。

什么是面向对象编程语言？什么是编译型语言？

面向对象编程语言是一种编程范式，它将现实世界中的概念和模型转化为计算机程序中的类和对象。面向对象编程中的核心概念包括封装、继承和多态性。

编译型语言是指需要先通过编译器将源代码转换成可执行代码的编程语言。在编译过程中，编译器会对代码进行语法分析、词法分析、语义分析、优化等操作，将源代码转换成二进制可执行文件。编译型语言的执行速度更快，但开发效率较低，因为需要编写和编译源代码。

学习板块

本书有关 Python 语法主要包括以下几个板块：

- 基础语法 (本章)：注释、缩进、变量、包、代码风格等。
- 数据类型 (第 5 章)：数字、字符串、列表、元组、字典等。
- 运算符 (第 6 章)：算术运算符、比较运算符、逻辑运算符、位运算符等。
- 控制结构 (第 7 章)：条件语句、循环语句、异常处理语句等。

- 函数和模块 (第 8 章)：函数和模块的定义和使用。
- 面向对象编程 (第 9 章)：定义类、对象、方法、属性等。

4.2 注释

Python 注释 (comment) 就是在写 Python 代码时，为了方便自己和别人理解代码，添加的文字说明。这些文字说明不会被 Python 解释器 (interpreter) 执行，只是为了让代码更易读懂和更易维护。

在 Python 代码中，我们可以使用 # (hash, hashtag, hashmark) 符号来添加注释。

当 Python 解释器读取代码时，如果遇到 # 符号，它就会将 # 所在行后面的内容视为注释，而不是代码的一部分。

⚠ 注意，# 后面的字符开始直到该行的结尾都被认为是注释。

#注释：整行、单行尾部

如图 2 所示，可以把注释 (图中高亮部分) 看作是给代码添加的“贴纸”，用来解释代码的用途、原理、变量的含义等等。机器遇到图中高亮部分文字就自然跳过。

图 2 展示了两种注释：1) 整行注释；2) 单行尾部注释。

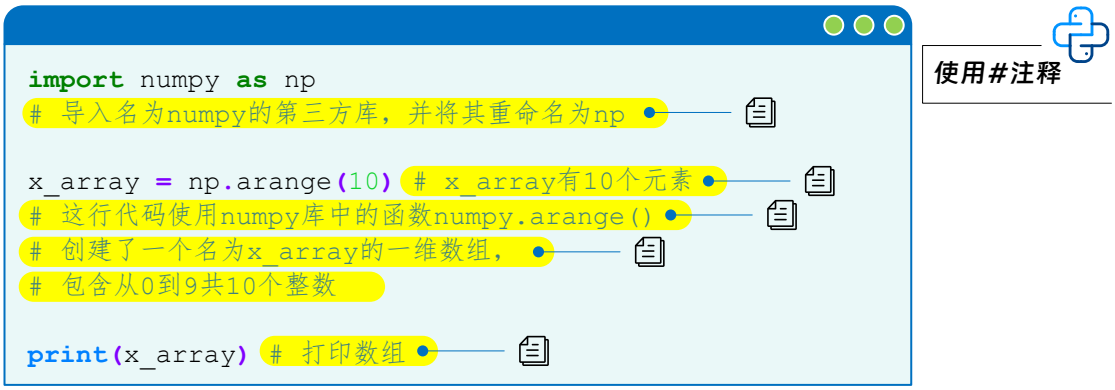


图 2. 举例说明 Python 代码中的注释

在 Jupyter Notebook 中，markdown 的功能和 comment 显然不同。Markdown 相当于笔记，可以是标题、文本段落、列表、图片、链接等等。而 comment 是在代码块中添加对具体代码的说明和解释。

⚠ 再次提醒，JupyterLab 中 comment 和 uncomment 默认快捷键为 ctrl + /。

'''或"""注释：多行

此外，我们还可以用三个引号 ('''或""") 来添加多行注释。

比如，要在 Python 代码中添加一段多行注释，来描述一个函数的功能和用法，那么可以使用三个引号来实现。以下是一个例子，我们定义了一个名为“my_function”的函数，使用三个引号来添加多行注释。

⚠注意，中文输入法下的单、双引号都是“全角引号”，Python 解释器会抛出语法错误。在 Python 中，只有半角引号（'）和双半角引号（"）才可以用来定义字符串，而全角引号则不能用于字符串的定义。此外，使用圆括号、中括号等符号时也需要注意全角、半角问题，避免语法错误。

```
def my_function(x, y):
    """
    这个自定义函数计算两个数值x和y的和
    函数输入为：
        x: The first number to be added.
        y: The second number to be added.
    函数输出为：
        The sum of x and y.
    """
    return x + y

# 打印结果
print(my_function(1.5, 2))
```

使用"""注释

图 3. 用三个引号来添加多行注释

在上面的例子中，我们使用了三个引号来包裹函数的注释文字，这个注释可以跨越多行，并且被 Python 解释器忽略掉，不会被当作代码执行。这样，其他程序员在阅读我们的代码时，就可以清晰地了解这个函数的作用、输入和输出参数、以及函数的返回值。

本书第 8 章将介绍自定义函数。

4.3 缩进

相信大家已经在图 3 发现了缩进 (indentation)。

在 Python 中，缩进是非常重要的。缩进是指在代码行前面留出的空格 (space) 或制表符 (tab →)，它们用于表示代码块的开始和结束。换句话说，缩进用于指示哪些代码行属于同一个代码块。

在其他编程语言中，通常使用花括号或关键字来表示代码块的开始和结束 (比如 MATLAB 用 end 表示代码块结束)。但在 Python 中，使用缩进来代替。

注意，缩进的大小没有严格规定，一般情况下建议使用四个空格作为缩进，并不鼓励用制表符 tab 缩进。特别反对混用四个空格、tab 缩进。

Python 中常见的需要缩进的场合包括 for 循环，while … 循环，if … else … 判断语句，函数定义以及类的定义等。同一缩进级别里的代码属于同一逻辑块。这些需要使用缩进的场合往往都是需要使用冒号 : 来表示下一行需要使用缩进。

注意，为了保证字母、数字、符号等显示时宽度一致，本书正文示例代码采用的字体为 `Courier New`。

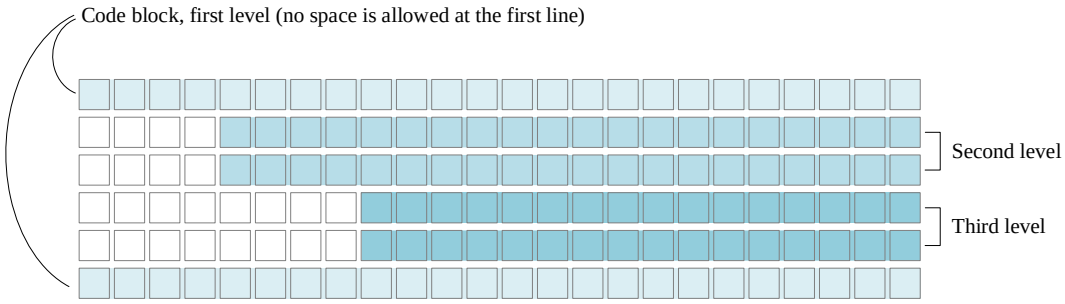


图 4. 缩进形成不同的代码级别

条件语句

在 if … elif … else … 语句中，它们所控制的代码块需要缩进，以表示它们属于条件语句。

注意，如果缩进有误编译器会报错，报错内容为 `IndentationError: unindent does not match any outer indentation level`。

```python
# 定义变量x，从用户输入中获取数值
x = float(input("请输入一个数值："))
# 定义变量abs_x，用来存放绝对值

abs_x = x
# 如果x为正数
if x > 0:
    print("x is positive")

# 如果x为零
elif x == 0:
    print("x is zero")

# 如果x为负数
else:
    print("x is negative")

    # 计算负数绝对值
    abs_x = -x

print("该数值的绝对值为：", abs_x)
```

条件语句中使用缩进

图 5. 条件语句中使用缩进

循环语句

在 for、while 等循环语句中，循环体内的代码块需要缩进，以表示它们属于循环语句。

```python
x_string = 'Python 3.X is easy!'

# 利用for循环打印每个字符
for i_str in x_string:
    print(i_str)
```

for循环语句中使用缩进

图 6. for 循环语句中使用缩进

函数定义

在函数定义时，函数体内的代码块需要缩进，以表示该代码块属于函数体。

在这个例子中，我们定义 `fibonacci()` 函数生成斐波那契数列 (Fibonacci sequence) 接受一个整数 `n`，它返回 `Fibonacci` 数列的第 `n` 项。如果 `n` 小于或等于 `1`，它将直接返回 `n`。否则，它将调用两次自己，并将 `n-1` 和 `n-2` 作为参数传递给它们。最终，当 `n` 达到 `0` 或 `1` 时，递归将停止，返回相应的值。

通过使用 for 循环来输出 Fibonacci 数列的前 10 项，可以看到这个函数在工作时是如何递归调用自己的。

《数学要素》将专门介绍斐波那契数列，《矩阵力量》讲解如何用线性代数工具求解斐波那契数列通项公式。

```python
# 使用递归函数生成 Fibonacci 数列
def fibonacci(n):
    # 如果 n 小于或等于 1，它将直接返回 n
    if n <= 1:
        return n
    # 否则，它将调用两次自己
    # 并将 n-1 和 n-2 作为参数传递给它们
    else:
        return fibonacci(n-1) + fibonacci(n-2)

# 通过使用 for 循环来输出 Fibonacci 数列的前 10 项
for i in range(10):
    print(fibonacci(i))
```

定义函数时使用缩进

图 7. 定义函数时使用缩进

什么是斐波那契数列？

斐波那契数列是一组数字，其中每个数字都是前两个数字的和。斐波那契数列的前几个数字是 0、1、1、2、3、5、8、13、21、34 等等。斐波那契数列是计算机科学中常用的例子，用于介绍递归和动态规划等概念。在植物学中，叶子、花瓣和果实的排列顺序可以遵循斐波那契数列。许多音乐家和作曲家使用斐波那契数列的规律来创建旋律和和弦。

4.4 变量

在 Python 中，变量 (variable) 是用于存储数据值的标识符。它们用于引用内存中的值，这些值可以是数字、字符串、列表、字典、函数等各种类型的数据。如图 8 所示，简单来说，变量就是个“箱子”。下表为 Python 中常见数据类型。本书第 5 章专门介绍数据类型。

数据类型	type()	特点	举例
数字 (Number)	int float	包括整数、浮点数等	x = 10 y = 3.14
字符串 (String)	str	一系列字符的序列	s = 'hello world'
列表 (List)	list	一组有序的元素，可以修改	a = [1, 2, 3, 4] b = ['apple', 'banana', 'orange']
元组 (Tuple)	tuple	一组有序的元素，不能修改	c = (1, 2, 3, 4) d = ('apple', 'banana', 'orange')

集合 (Set)	set	一组无序的元素，不允许重复	e = {1, 2, 3, 4} f = {'apple', 'banana', 'orange'}
字典 (Dictionary)	dict	一组键-值对，键必须唯一	g = {'name': 'Tom', 'age': 18}
布尔 (Boolean)	bool	代表 True 和 False 两个值	x = True y = False
None 类型	NoneType	代表空值或缺失值	z = None

在 Python 中，变量是动态类型的，这意味着我们可以在运行时为变量分配不同类型的值。不需要提前声明变量的类型，Python 会根据所赋予的值自动确定其类型。也就是说，这个 Python 中的箱子什么都能装。在 Python 中，可以使用内置的 type() 函数来判定数据的类型。type() 函数返回一个表示对象类型的值。

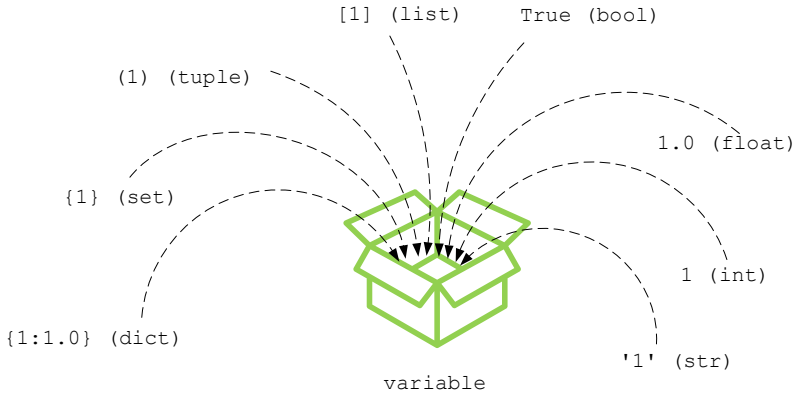


图 8. Python 变量就是个“箱子”，什么都能装

什么是动态类型语言？

动态类型语言是指在运行时可以自动判断变量的数据类型的编程语言。动态类型语言不需要在编写代码时显式地指定变量的数据类型，而是在程序运行时自动进行类型检查。

与之相对的是静态类型语言。静态类型语言中，每个变量都必须在声明时指定其数据类型，编译器会在编译时检查变量是否被正确使用。比如，C 语言是一种静态类型语言。int x = 10; int y = 20; x 和 y 都被声明为整数类型 (int)，编译器会在编译时检查它们是否被正确使用。

变量命名规则

Python 中的变量命名规则和建议如下：

- 变量名必须是一个合法的标识符，即由字母、数字和下划线组成，且不能以数字开头。例如，x、my_var、var_1 等都是合法的标识符。注意，变量名不能以数字开头，比如 1_variable 作为变量名不合法。
- 变量名区分大小写。例如，my_var 和 My_var 是不同的变量名。

- 变量名应该具有描述性，能够清晰地表达其所代表的内容。例如，name 可以代表人名，age 可以代表年龄等。
- 变量名应该尽量简洁明了，但不要过于简短或过于复杂。避免使用单个字母或缩写作为变量名，除非上下文明确。
- 变量名不应该与 Python 中的保留函数 (关键字) 重名，否则会导致语法错误。例如，不能使用 if、else、while 等关键字作为变量名。
- 在特定的上下文中，可以使用特定的命名约定。例如，类名应该使用驼峰命名法 (camelCase)，函数名和变量名应该使用下划线分隔法 (snake_case) 等。

有关 Python 内置函数用法，请参考：

https://docs.python.org/zh-cn/3/library/functions.html

驼峰、蛇形命名法

常见有两种变量命名法——camel case、snake case。下面简单比较两者。

- 驼峰命名法 (camel case) 得名于其类似于骆驼背部的形状，其中单词之间的空格被移除，而每个单词首字母一般大写。在驼峰命名法中，通常有两种常见的变体：小驼峰命名法 (lower camel case)，比如 firstName、totalAmount，和大驼峰命名法 (upper camel case)，比如 FirstNames、TotalAmount。大驼峰命名法也叫帕斯卡命名法 (Pascal case)。Pascal case 在 C#中应用更多。
- 蛇形命名法 (snake case) 以其类似于蛇的形状而得名，其中单词之间用下划线 _ 分隔，而且所有字母都是小写，例如 first_name 或 total_amount。Python 社区普遍采用蛇形命名法，而 Java 和 JavaScript 等语言则更常使用驼峰命名法。

变量赋值

大家都已经起初，我们可以使用等号 = 将一个值赋给一个变量。

可以同时给多个变量赋值，用逗号分隔每个变量，并使用等号将值分配给变量。例如：

```
x, y, z = 1, 2, 3
```

可以使用链式赋值的方式给多个变量赋相同的值。例如：

```
x = y = z = 0
```

可以使用增量赋值的方式对变量进行递增或递减。例如：

```
x += 1  # 等价于 x = x + 1
```

4.5 导入包

Python 包是一组相关的模块和函数的集合，用于实现特定的功能或解决特定的问题。包通常由一个顶层目录和一些子目录和文件组成，其中包含了实现特定功能的模块和函数。Python 中有很多常用的包，包括数据处理和可视化、机器学习和深度学习、网络编程、Web 开发等。其中，常用的可视化包包括 Matplotlib、Seaborn、Plotly 等，机器学习常用的包包括 NumPy、Pandas、Statsmodels、Scikit-learn、TensorFlow 等。

Matplotlib 是 Python 中最流行的绘图库之一，可用于创建各种类型的静态图形，如线图、散点图、柱状图、等高线图等。

Seaborn 是基于 Matplotlib 的高级绘图库，提供了更美观、更丰富的图形元素和绘图样式。

Plotly 是一款交互式绘图库，可用于创建各种类型的交互式图形，如散点图、热力图、面积图、气泡图等，支持数据可视化的各个方面，包括统计学可视化、科学可视化、金融可视化等。

NumPy 是 Python 中常用的数值计算库，提供了数组对象和各种数学函数，用于高效地进行数值计算和科学计算。

Pandas 是 Python 中常用的数据处理库，提供了高效的数据结构和数据分析工具，可用于数据清洗、数据处理和数据可视化。

Scikit-learn 是 Python 中常用的机器学习库，提供了各种常见的机器学习算法和模型，包括分类、回归、聚类、降维等。

TensorFlow 是谷歌开发的机器学习框架，提供了各种深度学习模型和算法，可用于构建神经网络、卷积神经网络、循环神经网络等深度学习模型。

本书前文介绍过如何安装、更新、删除某个具体包，下面我们聊一聊如何在 Python 中导入包。

导入包

下面以 NumPy 为例介绍几种常用的导入包的方式。

第一种，直接导入整个 NumPy 包：

```
import numpy
```

这种方式会将整个 NumPy 包导入到当前的命名空间中，需要使用完整的包名进行调用，例如：

```
a = numpy.array([1, 2, 3])
```

第二种，导入 NumPy 包并指定别名：

```
import numpy as np
```

这种方式会将 NumPy 包导入到当前的命名空间中，并使用别名 np 来代替 NumPy，例如：

```
a = np.array([1, 2, 3])
```

第三种，导入 NumPy 包中的部分模块或函数：

```
from numpy import array
```

这种方式会将 NumPy 包中的 `array` 函数导入到当前的命名空间中，可以直接调用该函数，例如：

```
a = array([1, 2, 3])
```

第四种，导入 NumPy 包中的所有模块或函数：

```
from numpy import *
```

这种方式会将 NumPy 包中的所有函数和模块导入到当前的命名空间中，可以直接调用任意函数或模块，例如：

```
a = array([1, 2, 3])

b = random.rand(3, 3)
```

在实际应用中，可以根据需要选择和使用适当的导入方式。一般来说，建议使用第二种 (导入 NumPy 包并指定别名) 或第三种方式 (导入部分模块或函数)，这样既可以简化代码，又不会导入太多无用的函数或模块，从而提高代码的可读性和性能。

4.6 Pythonic：Python 风格

"Pythonic" 翻译成中文可以是 "符合 Python 风格的"、"Python 风格的" 等。让 Python 代码 Pythonic 是指遵循 Python 社区的最佳实践和代码风格，使代码更加易读、易维护、易扩展和高效。

以下是一些让 Python 代码 Pythonic 的方法：

- 遵循 PEP8 规范：PEP8 是 Python 社区的代码风格指南，包括缩进、命名、代码结构、注释等。编写符合 PEP8 规范的代码可以提高代码的可读性和可维护性。
- 使用 Python 内置函数和数据结构：Python 提供了许多内置函数和数据结构，如列表、字典、集合、生成器、装饰器、lambda 表达式等。使用这些功能可以使代码更加简洁、高效和易于理解。
- 使用异常处理机制：Python 的异常处理机制可以使代码更加健壮和容错。在编写代码时应该预见到可能的异常情况，并使用 try/except 块来处理这些异常情况。
- 避免使用全局变量：全局变量可以使代码更加难以理解和维护，因为它们可能会被其他代码意外修改。应该尽量避免使用全局变量，而是使用函数或类来封装状态和行为。
- 使用函数式编程风格：函数式编程风格强调函数的不可变性和无状态性，使得代码更加简洁、高效和易于测试。应该尽可能使用纯函数，避免使用副作用和可变状态。

- 使用面向对象编程风格：面向对象编程风格可以使代码更加模块化和易于扩展。使用类和对象可以封装状态和行为，使代码更加结构化和易于维护。
- 编写文档和测试：编写文档和测试可以使代码更加易读、易于理解和易于维护。

有关 PEP8，请参考：

https://peps.python.org/pep-0008/

如果在 Python 编程中遇到问题或者 bug，可以去以下几个地方寻求帮助：

- 官方文档：Python 官方文档提供了丰富的资源，包括语言参考手册、标准库参考手册、教程、示例代码等。可以先在官方文档中查找相关信息，寻找解决问题的方法。
- https://stackoverflow.com/：这是一个广泛使用的程序员问答社区，拥有庞大的用户群体和丰富的问题解答资源。可以在这里提出你的问题，或者搜索其他人遇到的类似问题的解决方法。
- 此外，ChatGPT 之类的助手工具也可以帮助我们解决编程中遇到的问题。

本章题目仅是请大家在 JupyterLab 中复刻所有示例代码，并逐行注释加强理解。

* 题目不提供答案。

希望大家学习 Python 时，一定要吸取英语学习失败的教训，千万不能死磕 Python 语法。要用为主、学为辅，边学边用，活学活用。

5 Python 数据类型

Data Types in Python

字符串、列表、元组、字典...蜻蜓点水，了解就好

每个人都是天才。但是，如果您以爬树的能力来判断一条鱼，那么那条鱼终其一生都会相信自己是愚蠢的。

Everybody is a genius. But if you judge a fish by its ability to climb a tree, it will live its whole life believing that it is stupid.

—— 阿尔伯特·爱因斯坦 (Albert Einstein) | 理论物理学家 | 1879 ~ 1955

5.1 数据类型有哪些?

通过上一章学习，我们知道 Python 是一种动态类型语言，它支持多种数据类型。以下是 Python 中常见的数据类型：

- 数字 (number) 类型：整数、浮点数、复数等。
- 字符串 (string) 类型：表示文本的一系列字符。
- 列表 (list) 类型：表示一组有序的元素，可以修改。
- 元组 (tuple) 类型：表示一组有序的元素，不能修改。
- 集合 (set) 类型：表示一组无序的元素，不允许重复。
- 字典 (dictionary) 类型：表示键-值对，其中键必须是唯一的。
- 布尔 (Boolean) 类型：表示 True 和 False 两个值。
- None 类型：表示空值或缺失值。

注意，大小写问题，比如 True、False、None 都是首字母大写。此外，注意 Python 代码都是半角字符，只有注释、Markdown 才能出现全角字符。

Python 还支持一些高级数据类型，如生成器 (Generator)、迭代器 (Iterator)、函数 (Function)、类 (Class) 等。

注意，对于 Python 初学者，请大家切记完全没有必要熟练掌握每一种数据类型。对于数据类型等 Python 语法细节，希望大家蜻蜓点水，轻装上阵，边用边学。

5.2 数字

Python 有三种内置数字类型：

- 整数 (int)：表示整数值，没有小数部分。例如，`42`、`-123`、`0` 等。
- 浮点数 (float)：表示实数值，可以有小数部分。例如，`3.14`、`-0.5`、`2.0` 等。
- 复数 (complex)：表示由实数和虚数构成的数字。

什么是复数?

复数是数学中的一个概念，由实部和虚部组成。它可以表示为 $a + bi$ 的形式，其中 a 是实部，b 是虚部，而 i 是虚数单位，满足 $i^2 = -1$。复数在数学和物理等领域中有广泛的应用。

复数扩展了实数域，使得可以处理平面上的向量运算、波动和振荡等问题。它在电路分析、信号处理、量子力学、调频通信等领域具有重要作用。复数还能用于描述周期性事件、解析函数和几何形状等。

通过复数的运算，我们可以进行加法、减法、乘法和除法等操作，同时也可以求解方程、解析函数和变换等数学问题。复数的使用使得我们能够更好地描述和理解许多实际问题，扩展了数学的应用范围。

图 1 是一些示例，请大家在 JupyterLab 中自行练习。

```
x = 42      # 整数
y = -3.14   # 浮点数
z = 2 + 3j  # 虚数
print(type(x))
# <class 'int'>
print(type(y))
# <class 'float'>
print(type(z))
# <class 'complex'>
```

三类数值

图 1. Python 中三类数值

在 Python 中，数字类型可以进行基本的算术操作，例如加法 (+)、减法 (-)、乘法 (*)、除法 (/)、取余数 (%)、乘幂 (**) 等。数字类型还支持比较运算符，如等于 (==)、不等于 (!=)、大于 (>)、小于 (<)、大于等于 (>=)、小于等于 (<=)。此外，本书前文还介绍过的自加运算 (+=)、自减运算 (-=)、自乘运算 (*=)、自除运算 (/=) 等。

本书第 6 章将专门介绍 Python 常见运算符。

类型转换

在 Python 中，可以使用内置函数将一个数字类型转换为另一个类型。下面是常用的数字类型转换函数：

- `int(x)`：将 `x` 转换为整数类型。如果 `x` 是浮点数，则会向下取整；如果 `x` 是字符串，则字符串必须表示一个整数。
- `float(x)`：将 `x` 转换为浮点数类型。如果 `x` 是整数，则会转换为相应的浮点数；如果 `x` 是字符串，则字符串必须表示一个浮点数。
- `complex(x)`：将 `x` 转换为复数类型。如果 `x` 是数字，则表示实部，虚部为 `0`；如果 `x` 是字符串，则字符串必须表示一个复数；如果 `x` 是两个参数，则分别表示实部和虚部。
- `str(x)`：将 `x` 转换为字符串类型。如果 `x` 是数字，则表示为字符串；如果 `x` 是布尔类型，则返回`'True'`或`'False'`字符串。

图 2 是一些示例，请大家在 JupyterLab 中自行练习。

需要注意的是，如果在类型转换过程中出现了不合理的转换，例如将一个非数字字符串转换为数字类型，就会导致 ValueError 异常。

本书第 7 章将专门介绍如何处理异常。

数值转化

```python
x = 42.0
y = 3
# 将浮点数转换为整数
x_to_int = int(x)
print(x_to_int)  # 42

# 将整数转换为浮点数
y_to_float = float(y)
print(y_to_float)  # 3.0

# 将整数转换为复数
y_to_complex = complex(y)
print(y_to_complex)  # (3+0j)

# 将数字转换为字符串
x_to_str = str(x)
print(x_to_str)  # '42.0'
```

图 2. Python 中数值转换

什么是异常？

在 Python 中，异常 (exception) 是指在程序执行期间出现的错误或异常情况。当出现异常时，程序的正常流程被中断，转而执行异常处理的代码块，以避免程序崩溃或产生不可预知的结果。

Python 中有许多不同类型的异常，每种异常都代表了特定类型的错误。以下是一些常见的异常类型：ValueError (数值错误)：当函数接收到一个不合法的参数值时引发。TypeError (类型错误)：当使用不兼容的类型进行操作或函数调用时引发。IndexError (索引错误)：当尝试访问列表、元组或字符串中不存在的索引时引发。FileNotFoundError (文件未找到错误)：当尝试打开不存在的文件时引发。ZeroDivisionError (零除错误)：当尝试将一个数除以零时引发。

可以使用 try-except 语句来捕获并处理这些异常，以便在程序出现问题时执行适当的操作或提供错误信息。

特殊数值

有很很多场合还需要用到特殊数值，比如圆周率 pi、自然对数底数 e 等等。在 Python 中，可以使用 math 模块来引入这些特殊值，请大家在 JupyterLab 中练习。

```
import math

print(math.pi)       # 输出π的值
print(math.e)        # 输出e的值
print(math.sqrt(2))  # 输出根号2的值
```

Math模块

图 3. Math 模块中的特殊数值

除了这些特殊数值外，Math 模块还提供了许多其他数学函数，比如四舍五入 round()、上入取整数 ceil()、下舍取整数 floor()、乘幂运算 pow()、指数函数 exp()、以 e 为底数的对数 log()、以 10 为底数的对数 log10()等等。

注意，大家日后会发现我们一般很少用到 Math 模块，会直接采用 NumPy、Pandas 中的运算函数。

5.3 字符串

Python 中字符串 (string) 是一个常见的数据类型，常常用于表示文本信息。本节介绍一些常用的字符串用法。

字符串定义

使用单引号'、双引号"、三引号'''或"""将字符串内容括起来即可定义字符串。请大家在 JupyerLab 中练习图 4 代码。三引号'''或"""一般用来创建多行字符串。

注意，空格、标点符号都是字符串的一部分。使用加号 + 将多个字符串连接起来，使用乘号 * 复制字符串。数字字符串仅仅是文本，不能直接完成算数运算，需要转化成整数、浮点数之后才能进行算数运算。

请大家用 len() 函数获得图 4 每个字符串的长度，即字符串中字符个数。

注意，Python 中单字符也是字符串类型。

字符串定义

```
str1 = 'I am learning Python 101!'
print(str1)
# 打印

str2 = "Python is fun. Machine learning can
be fun too."
print(str2)
# 打印

# 使用加号 + 将多个字符串连接起来
str4 = 'Hey, ' + 'James!'
print(str4)
# 'Hey, James!'

# 使用乘号*将一个字符串复制多次
str5 = 'Scikit-Learn is fun! '  #
字符串最后有一个空格
str6 = str5 * 3
print(str6)
# 'Python is fun! Python is fun! Python is
fun!'

# 字符串中的数字仅仅是字符
str7 = '123'
str8 = str7 * 3
print(str8)

str9 = '456'
str10 = str9 + str7
print(str10)
print(type(str10))
```

图 4. 字符串定义

索引、切片

在 Python 中，可以通过索引 (indexing) 和切片 (slicing) 来访问和操作字符串中的单个字符、部分字符。

如图 5 所示，字符串中的每个字符都有一个对应的索引位置，索引从 0 开始递增。可以使用方括号 [] 来访问指定索引位置的字符。

可以使用负数索引来从字符串的末尾开始计算位置。例如，-1 表示倒数第一个字符，-2 表示倒数第二个字符，依此类推。请大家自行在 JupyterLab 中练习图 7。

图 7 代码中使用了 for 循环来遍历字符串中的每个字符，并打印出字符及其对应的序号。enumerate() 函数来同时获取字符和它们的索引位置。enumerate() 函数会返回一个迭代器，包含每个字符及其对应的索引。然后，通过 for 循环遍历迭代器，依次打印出每个字符和它们的序号。

本书第 7 章将专门介绍 for 循环。

在代码中，f-字符串 (formatted string) 是一种用于格式化字符串的语法。它以字母 "f" 开头，并使用花括号（{}）来插入变量或表达式的值。在这个特定的例子中，f-字符串用于构建一个带

有变量值的字符串。通过在字符串中使用花括号和变量名，可以在字符串中插入变量的值。在这种情况下，使用了两个变量 {char} 和 {index}。当代码执行时，{char} 会被替换为当前循环迭代的字符，{index} 会被替换为对应字符的索引值。这样就创建了一个字符串，包含了字符及其对应的序号信息。

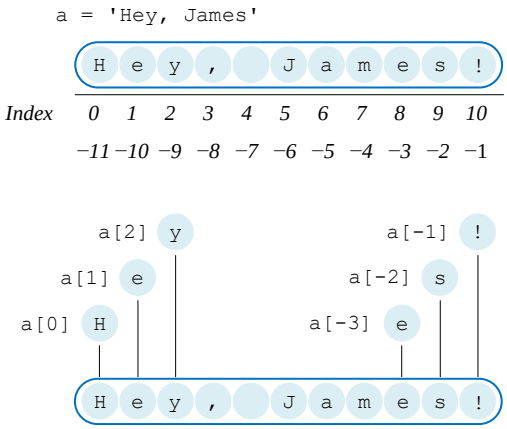


图 5. 字符串的索引

切片是指从字符串中提取出一部分子字符串。可以使用半角冒号 : 来指定切片的起始位置 start 和结束 end 位置。语法为 string[start:end]，包括 start 序号对应的字符，但是不包括 end 位置的字符，相当于“左闭右开”区间。

切片还可以指定步长 (step)，用于跳过指定数量的字符。语法为 string[start:end:step]。

注意，复制字符串可以采用 string_name[:] 实现。

Python 中还有很多字符串“花式”切片方法，大家没有必要花大力气去“精雕细琢”。大概知道字符串有哪些常见的索引、切片方法就足够了，等到用到时再去特别学习。还是那句话，别死磕 Python 语法！

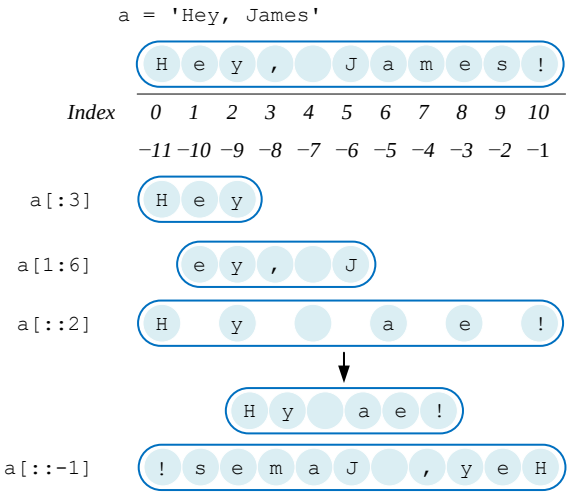


图 6. 字符串的切片

需要注意的是，索引和切片操作不会改变原始字符串，而是返回一个新的字符串。

字符串索引、切片

```
greeting_str = 'Hey, James!'
# 打印字符串长度
print('字符串的长度为：')
print(len(greeting_str))

# 打印每个字符和对应的序号
for index, char in enumerate(greeting_str):
    print(f"字符：{char}，序号：{index}")

# 单个字符索引
print(greeting_str[0])
print(greeting_str[1])

print(greeting_str[-1])
print(greeting_str[-2])

# 切片
# 取出前3个字符，序号为0、1、2
print(greeting_str[:3])

# 取出序号1、2、3、4、5，不含0，不含6
print(greeting_str[1:6])

# 指定步长2，取出第0、2、4、6
print(greeting_str[::2])

# 指定步长-1，倒序
print(greeting_str[::-1])
```

图 7. 字符串索引和切片

从 0 计数 vs 从 1 计数

从 0 计数和从 1 计数是在数学和编程中常见的计数方式。

从 0 计数 (zero-based counting) 将第一个元素的索引或位置标记为 0，即从 0 开始计数。例如，对于一个包含 n 个元素的序列，它们的索引分别为 0、1、2、...、$n-1$。在计算机科学和编程中，Python 使用从 0 计数的方式。

从 1 计数 (one-based counting) 将第一个元素的索引或位置标记为 1，即从 1 开始计数。例如，对于一个包含 n 个元素的序列，它们的索引分别为 1、2、3、...、n。MATLAB 使用从 1 计数方式；统计学 (样本)、线性代数 (矩阵、向量) 等通常使用从 1 计数的方式。

相比来看，从 1 计数更符合人类直观理解的习惯。从 1 计数在数学、统计学、数值计算等领域中较为常见。编程角度来看，从 0 计数在计算机科学中更常见，因为它与计算机内存和数据结构的底层表示方式相匹配。它使得处理数组、列表和字符串等数据结构更加高效和一致。

在实际编程中，理解和适应使用不同的计数方式是重要的。需要根据具体情况选择适当的计数方式，以确保正确地处理索引、循环和算法等操作。同时，注意在不同的领域和语境中遵循相应的计数习惯和规则。

字符串方法

Python 提供了许多用于字符串处理的常见方法。下面是一些常见的字符串方法及其示例。

len() 返回字符串的长度，比如下例。

```
string = "Hello, James!"
length = len(string)
print(length)
```

lower() 和 upper() 将字符串转换为小写或大写，比如下例。

```
string = "Hello, James!"
lower_string = string.lower()
upper_string = string.upper()
print(lower_string)  # 输出 "hello, james!"
print(upper_string)  # 输出 "HELLO, JAMES!"
```

以下是一些常见的 Python 字符串方法及其作用：capitalize():将字符串的第一个字符转换为大写，其他字符转换为小写。count() 统计字符串中指定子字符串的出现次数。find() 在字符串中查找指定子字符串的第一次出现，并返回索引值。isalnum() 检查字符串是否只包含字母和数字。isalpha() 检查字符串是否只包含字母。isdigit() 检查字符串是否只包含数字。join() 将字符串列表或可迭代对象中的元素连接为一个字符串。replace() 将字符串中的指定子字符串替换为另一个字符串。split() 将字符串按照指定分隔符分割成子字符串，并返回一个列表。

注意，这些方法大家也不需要死记硬背！了解就好，轻装上阵。数据分析、机器学习中更常用的 NumPy 数组、Pandas 数据帧，这都是本书后续要重点介绍的内容。

5.4 列表

在 Python 中，列表 (list) 是一种非常常用的数据类型，可以存储多个元素，并且可以进行增删改查等多种操作。

图 10 代码生成的是一个特殊的列表，我们称之为混合列表，原因是这个列表中每个元素都不同。如图 8 所示，这个列表中序号为 4 的元素 (从左到右第 5 个元素) 还是个列表，相当于嵌套。

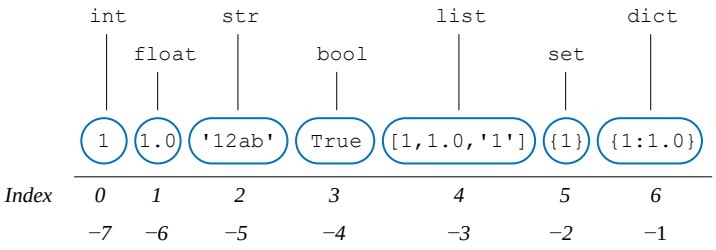


图 8. 混合列表

图 10 还给出 list 常用的索引方法，请大家在 JupyterLab 中练习。列表的索引、切片方式和字符串类似，我们不再展开。其中大家需要注意的是如果列表中的某个元素也是列表，我们可以通过二次索引来进一步索引、切片，如图 9 所示。

请大家在 JupyterLab 中练习图 11 给出的 list 常见方法、操作。

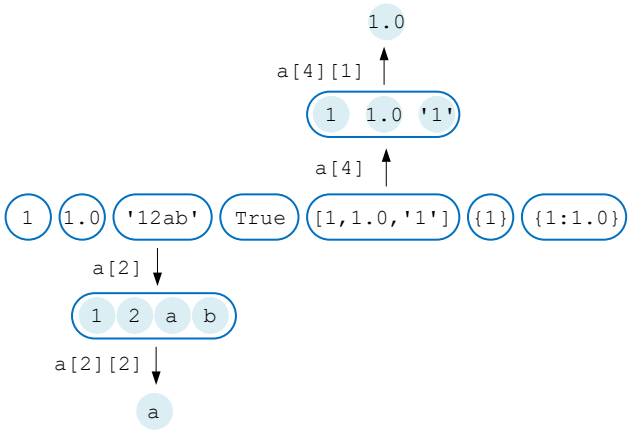


图 9. 混合列表的索引

列表索引、切片

```python
# 创建一个混合列表

my_list = [1, 1.0, '1', True,
           [1, 1.0, '1'], {1}, {1:1.0}]
print('列表长度为')
print(len(my_list))

# 打印每个元素和对应的序号
for index, item in enumerate(my_list):
    type_i = type(item)
    print(f"元素：{item}，序号：{index}，类型：{type_i}")

# 列表索引
print(my_list[0])
print(my_list[1])

print(my_list[-1])
print(my_list[-2])

# 列表切片
# 取出前3个元素，序号为0、1、2
print(my_list[:3])

# 取出序号1、2、3，不含0，不含4
print(my_list[1:4])

# 指定步长2，取出第0、2、4、6
print(my_list[::2])

# 指定步长-1，倒序
print(my_list[::-1])

# 提取列表中的列表某个元素
print(my_list[4][1])
```

图 10. 列表索引和切片

列表常见方法和操作

```
# 创建一个混合列表

my_list = [1, 1.0, '12ab', True,
           [1, 1.0, '1'], {1}, {1:1.0}]
print(my_list)

# 修改某个元素
my_list[2] = '123'
print(my_list)

# 在列表指定位置插入元素
my_list.insert(2, 'inserted')
print(my_list)

# 在列表尾部插入元素
my_list.append('tail')
print(my_list)

# 通过索引删除
del my_list[-1]
print(my_list)

# 删除某个元素
my_list.remove('123')
print(my_list)

# 判断一个元素是否在列表中

if '123' in my_list:
    print("Yes")
else:
    print("No")

# 列表翻转
my_list.reverse()
print(my_list)

# 将列表用所有字符连接，连接符为下划线 _
letters = ['J', 'a', 'm', 'e', 's']
word = '_'.join(letters)
print(word)
```

图 11. 列表常用方法、操作

视图 vs 浅复制 vs 深复制

如果用 = 直接赋值，是非拷贝方法，结果是产生一个视图 (view)。这两个列表是等价的，修改其中任何 (原始列表、视图) 一个列表都会影响到另一个列表。

如图 12 所示，用等号 = 赋值得到的 list_2 和 list_1 共享同一地址，这就是我们为什么称 list_2 为视图。视图这个概念是借用自 NumPy。

我们在本书后续还要聊到 NumPy array 的视图和副本这两个概念。

而通过 copy() 获得的 list_3 和 list_1 地址不同。请大家自行在 JupyterLab 中练习图 13。

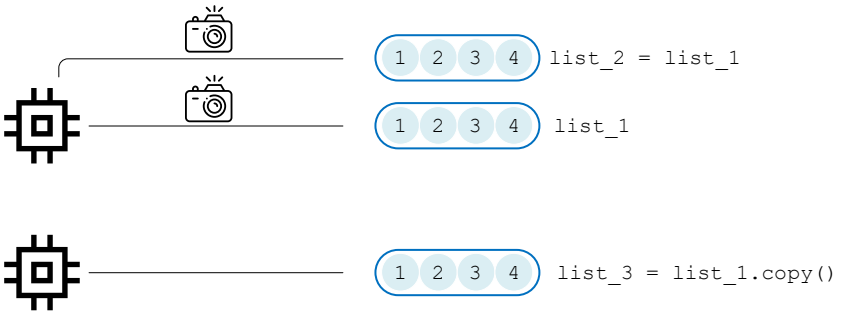


图 12. 视图，还是副本？

视图 vs 副本

```python
list1 = [1, 2, 3, 4]

# 赋值，视图
list2 = list1
# 拷贝，副本 （浅拷贝）
list3 = list1.copy()

list2[0] = 'a'
list2[1] = 'b'
list3[2] = 'c'
list3[3] = 'd'

print(list1)
print(list2)
print(list3)
```

图 13. 视图 vs 副本

可惜事情并没有这么简单。在 Python 中，列表是可变对象，因此在复制列表时会涉及到深复制和浅复制的概念。

浅复制 (shallow copy) 只对 list 的第一层元素完成拷贝，深层元素还是和原 list 共用。

深复制 (deep copy) 是创建一个完全独立的列表对象，该对象中的元素与原始列表中的元素是不同的对象。

注意，特别是对于嵌套列表，建议大家采用 copy.deepcopy() 深复制。图 14 代码比较不同复制，请大家自行学习。

深复制

```python
import copy

list1 = [1, 2, 3, [4, 5]]
print('原始list')
print(list1)

# 深复制，适用于嵌套列表
list_deep = copy.deepcopy(list1)

# 只深复制一层
list2 = list1.copy()
list3 = list1[:]
list4 = list(list1)
list5 = [*list1]

# 修改元素
list_deep[3][0] = 'deep'
list_deep[2] = 'worked_0'

list2[3][0] = 'abc'
list2[2] = 'worked_1'

list3[3][0] = 'X1'
list3[2] = 'worked_2'

list4[3][0] = 'X2'
list4[2] = 'worked_3'

list5[3][0] = 'X3'
list5[2] = 'worked_4'

print('新list')
print(list1)
print(list_deep)

print(list2)
print(list3)
print(list4)
print(list5)
```

图 14. 浅复制、深复制

5.5 其他数据类型

元组

在 Python 中，元组 (tuple) 是一种不可变的序列类型，用圆括号 () 来表示。元组一旦创建就不能被修改，这意味着你不能添加或删除其中的元素。

tuple 和 list 都是序列类型，可以存储多个元素，它们都可以通过索引访问和修改元素，支持切片操作。但是，两者有明显区别，元组使用圆括号 ( ) 表示，而列表使用方括号 [ ] 表示。元组是不可变的，而列表是可变的。这意味着元组的元素不能被修改、添加或删除，而列表可以进行这些操作。

元组的优势在于它们比列表更轻量级，这意味着在某些情况下，它们可以提供更好的性能和内存占用。本书不展开介绍元组。

集合

在 Python 中，集合 (set) 是一种无序的、可变的数据类型，可以用来存储多个不同的元素。使用花括号 {} 或者 set() 函数创建集合，或者使用一组元素来初始化一个集合。

```
number_set = {1, 2, 3, 4, 5}
word_set = set(["apple", "banana", "orange"])
```

可以使用 add() 方法向集合中添加单个元素，使用 update() 方法向集合中添加多个元素。

```
fruit_set = set(["apple", "banana"])
fruit_set.add("orange")
fruit_set.update(["grape", "kiwi"])
```

删除元素：使用 remove() 或者 discard() 方法删除集合中的元素，如果元素不存在，remove() 方法会引发 KeyError 异常，而 discard() 方法则不会。

```
fruit_set.remove("banana")
fruit_set.discard("orange")
```

集合的好处是可以用交集、并集、差集等集合操作来操作集合，如图 15 所示。

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
set3 = set1 & set2  # 交集
set4 = set1 | set2  # 并集
set5 = set1 - set2  # 差集
```

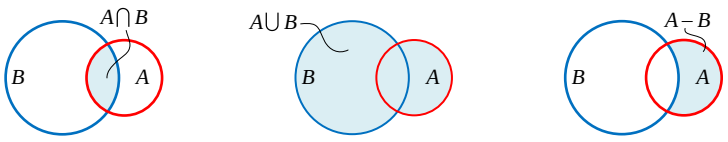


图 15. 交集、并集、差集

字典

在 Python 中，字典是一种无序的键值对 (key-value pair) 集合。

可以使用大括号 {} 或者 dict() 函数创建字典，键 (key) 值 (value) 对之间用冒号 : 分隔。有关字典这种数据类型本书不做展开，请大家自行学习图 16。

再次强调，数据分析、机器学习实践中，我们更关注的数据类型是 NumPy 数组、Pandas 数据帧，这是本书后续要着重讲解的内容。

字典常见操作

```
# 使用大括号创建字典
person = {'name': 'James', 'age': 18, 'gender':
'male'}

# 使用 dict() 函数创建字典
fruits = dict(apple=3, banana=2, cherry=5)

# 访问字典中的值
print(person['name'])
print(fruits['cherry'])

# 修改字典中的值
person['age'] = 30
print(person)

# 添加键值对
person['city'] = 'Toronto'
print(person)

# 删除键值对
del person['gender']
print(person)

# 获取键、值、键值对列表
print(person.keys())
print(person.values())
print(person.items())
```

图 16. 有关字典的常见操作

5.6 矩阵、向量：线性代数概念

抛开本章前文这些数据类型，数学上我们最关心的类型是——矩阵、向量。

简单来说，矩阵是一个由数值排列成的矩形阵列，其中每个数值都称为该矩阵的元素。矩阵通常使用大写、斜体、粗体字母来表示，比如 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{V}$、$\boldsymbol{X}$。

向量是一个有方向和大小的量，通常表示为一个由数值排列成的一维数组。向量通常使用小写字母加粗体来表示，例如 $\boldsymbol{x}$、$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{v}$、$\boldsymbol{u}$。

如图 17 所示，一个 $n \times D$ (n by capital D) 矩阵 $\boldsymbol{X}$，n 是**矩阵行数** (number of rows in the matrix)，D 是**矩阵列数** (number of columns in the matrix)。矩阵 $\boldsymbol{X}$ 的行索引就是 1、2、3、...、n。矩阵 $\boldsymbol{X}$ 的列索引就是 1、2、3、...、D。

$x_{1,1}$ 代表矩阵第 1 行、第 1 列元素，$x_{i,j}$ 代表矩阵第 i 行、第 j 列元素。

$$\boldsymbol{X}_{n\times D} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{bmatrix}$$

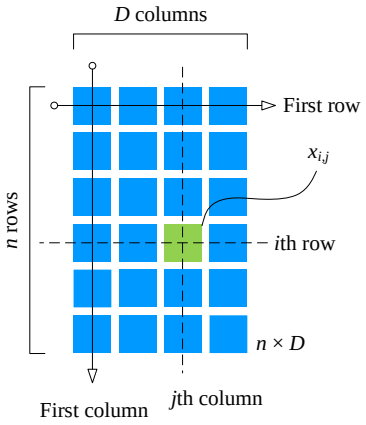
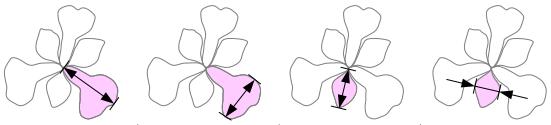


图 17. $n \times D$ 矩阵 $\boldsymbol{X}$

从统计数据角度，n 是样本个数，D 是样本数据特征数。如图 18 所示，鸢尾花数据集，不考虑标签 (即鸢尾花三大类 setosa、versicolor、virginica)，数据集本身 $n = 150$，$D = 4$。



Index	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Species C
1	5.1	3.5	1.4	0.2	
2	4.9	3	1.4	0.2	
3	4.7	3.2	1.3	0.2	Setosa C_1
...	...	...	...	...	
49	5.3	3.7	1.5	0.2	
50	5	3.3	1.4	0.2	
51	7	3.2	4.7	1.4	
52	6.4	3.2	4.5	1.5	
53	6.9	3.1	4.9	1.5	Versicolor C_2
...	...	...	...	...	
99	5.1	2.5	3	1.1	
100	5.7	2.8	4.1	1.3	
101	6.3	3.3	6	2.5	
102	5.8	2.7	5.1	1.9	
103	7.1	3	5.9	2.1	Virginica C_3
...	...	...	...	...	
149	6.2	3.4	5.4	2.3	
150	5.9	3	5.1	1.8	

图 18. 鸢尾花数据，数值数据单位为厘米 (cm)

从数据、统计、线性代数、几何角度解释，什么是矩阵？

矩阵是一个由数字或符号排列成的矩形阵列。简单来说，矩阵就是个表格。矩阵在数据、统计、线性代数和几何学中扮演着重要的角色。

从数据的角度来看，矩阵可以表示为一个包含行和列的数据表。每个单元格中的数值可以代表某种测量结果、观察值或特征。数据科学家和分析师使用矩阵来存储和处理数据，从中提取有用的信息。比如，一张黑白照片中的数据就可以看做是个矩阵。

从统计学的角度来看，矩阵可以用于描述多个变量之间的关系。例如，协方差矩阵用于衡量变量之间的相关性，而相关矩阵则提供了变量之间的线性相关性度量。统计学家使用这些矩阵来推断模式、关联和依赖性，以及进行数据分析和建模。

从线性代数的角度来看，矩阵可以用于表示线性方程组的系数矩阵。通过矩阵运算，例如矩阵乘法、求逆和特征值分解，可以解决线性方程组、求解特征向量和特征值等问题。线性代数中的矩阵理论提供了处理线性关系的强大工具。

从几何学的角度来看，矩阵可以用于表示几何变换。通过将向量表示为矩阵的列或行，可以应用平移、旋转、缩放等几何变换。矩阵乘法用于组合多个变换，从而实现更复杂的几何操作。在计算机图形学和计算机视觉中，矩阵在处理和表示二维或三维对象的位置、方向和形状方面起着重要作用。

总而言之，矩阵是一个在数据、统计、线性代数和几何学中广泛应用的数学工具，它能够表示和处理多个变量之间的关系、解决线性方程组、进行几何变换等。

什么是鸢尾花数据集？

鸢尾花数据集是一种经典的用于机器学习和模式识别的数据集。数据集的全称为安德森鸢尾花卉数据集 (Anderson's Iris data set)，是植物学家埃德加·安德森 (Edgar Anderson) 在加拿大魁北克加斯帕半岛上的采集的鸢尾花样本数据。它包含了 150 个样本，分为三个不同品种的鸢尾花 (山鸢尾、变色鸢尾和维吉尼亚鸢尾)，每个品种 50 个样本。每个样本包含了四个特征：花萼长度、花萼宽度、花瓣长度和花瓣宽度。

鸢尾花数据集由统计学家罗纳德·费舍尔 (Ronald Fisher) 在 1936 年引入，并被广泛用于模式识别和机器学习的教学和研究。这个数据集是机器学习领域的一个基准测试数据集，被用来评估分类算法的性能。

鸢尾花数据集在机器学习应用中有很多用途。它经常被用来进行分类任务，即根据花的特征将其分为不同的品种。许多分类算法和模型，如 K 近邻、决策树、支持向量机和神经网络等，都可以使用鸢尾花数据集进行训练和测试。

由于鸢尾花数据集是一个相对简单的数据集，它也常用于机器学习的入门教学和实践。通过对这个数据集的分析和建模，学习者可以了解特征工程、模型选择和评估等机器学习的基本概念和技术。矩阵是一个由数字或符号排列成的矩形阵列。简单来说，矩阵就是个表格。矩阵在数据、统计、线性代数和几何学中扮演着重要的角色。

行向量、列向量

行向量 (row vector) 是由一系列数字或符号排列成的一行序列。列向量 (column vector) 是由一系列数字或符号排列成的一列序列。

矩阵可以视作由一系列行向量、列向量构造而成。

如图 19 所示，$\boldsymbol{X}$ 任一行向量代表一朵特定鸢尾花样本花萼长度、花萼宽度、花瓣长度和花瓣宽度测量结果。而 $\boldsymbol{X}$ 某一列向量为鸢尾花某个特征 (花萼长度、花萼宽度、花瓣长度、花瓣宽度) 的样本数据。

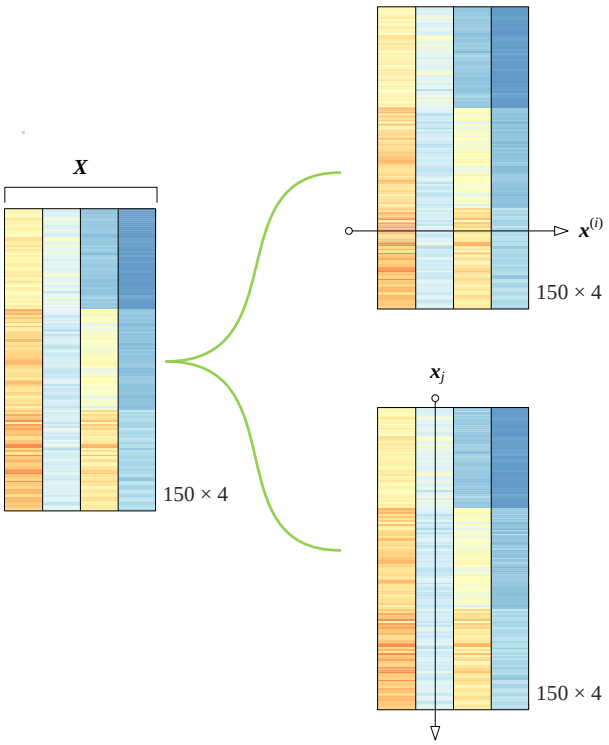


图 19. 矩阵可以分割成一系列行向量或列向量

Basic Calculations in Python

6 Python 常见运算

从加减乘除开始学运算符

有时人们不想听到真相，因为他们不想打碎自己的幻象。

Sometimes people don't want to hear the truth because they don't want their illusions destroyed.

—— 弗里德里希·尼采 (Friedrich Nietzsche) | 德国哲学家 | 1844 ~ 1900

6.1 几类运算符

Python 中的运算符可以分为以下几类：

- 算术运算符：用于数学运算，例如加法 (+)、减法 (-)、乘法 (*)、除法 (/)、取余数 (%)、乘幂 (**) 等。
- 比较运算符：用于比较两个值之间的关系，例如等于 (==)、不等于 (!=)、大于 (>)、小于 (<)、大于等于 (>=)、小于等于 (<=) 等。
- 逻辑运算符：用于处理布尔型数据，例如与 (and)、或 (or)、非 (not) 等。
- 赋值运算符：用于给变量赋值，例如等号 (=)、自加运算 (+=)、自减运算 (-=)、自乘运算 (*=)、自除运算 (/=)。
- 成员运算符：用于检查一个值是否为另一个值的成员，例如 in、not in 等。
- 身份运算符：用于检查两个变量是否引用同一个对象，例如 is、is not 等。

以上是 Python 中常见的运算符，可以根据不同的场景选择合适的运算符进行操作。

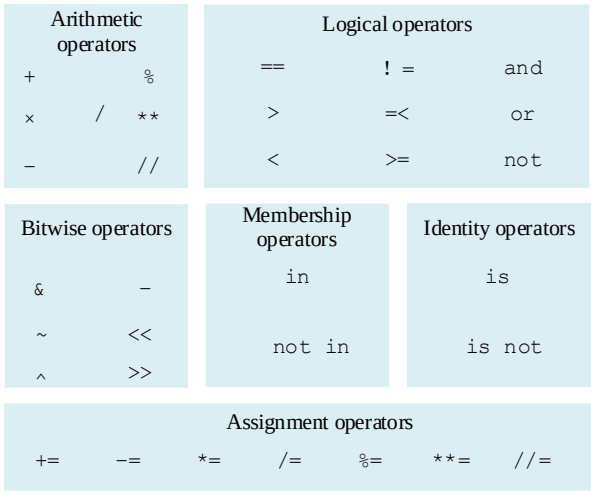


图 1. 常用运算符

6.2 算术运算符

Python 算术运算符用于数学运算，包括加法、减法、乘法、除法、取模和幂运算等。下面分别介绍这些算术运算符及其使用方法。

加减法

加法运算符 (+) 用于将两个数值相加或将两个字符串拼接起来。

请大家在 JupyterLab 中自行练习图 2。

注意，当进行加法运算时，如果操作数的类型不一致，Python 会自动进行类型转换。如果一个数是整数，而另一个是浮点数，则整数会被转换为浮点数，然后进行加法运算。运算结果为浮点数。加法时，如果一个数是整数，而另一个是复数，则整数会被转换为复数，然后进行加法运算。结果为复数。如果一个操作数是浮点数，而另一个是复数，则浮点数会被转换为复数，然后进行加法运算。运算结果为复数。

减法运算符 - 用于将两个数值相减，不支持字符串运算，错误信息为 `TypeError: unsupported operand type(s) for -: 'str' and 'str'`。

+运算符

```python
# 数值加法
a = 10    # 整数
b = 20.0  # 浮点数
c = a + b
print(c)

# 字符串拼接
str_a = "10"    # str(a)
str_b = "20.0"  # str(b)
str_c = str_a + str_b
print(str_c)
```

图 2. 加法

乘除法

乘法运算符 (*) 用于将两个数值相乘或将一个字符串重复多次。

注意，NumPy 数组完成矩阵乘法 (matrix multiplication) 时用的运算符为 @。

```python
# 数值乘法
a = 10    # 整数
b = 20.0  # 浮点数
c = a * b
print(c)

# 字符串复制
str_a = "10"    # str(a)
str_b = "20.0"  # str(b)
str_c = str_a * 5
str_d = str_b * 5
print(str_c)
print(str_d)
```

*运算符

图 3. 乘法

除法运算符 / 用于将两个数值相除，结果为浮点数。

在 Python 中，正斜杠 / (forward slash) 和反斜杠 \ (backward slash) 具有不同的用途和含义。在路径表示中，正斜杠 / 用作目录分隔符，用于表示文件系统路径。在除法运算中，正斜杠用作除法操作符。

在 Windows 文件路径表示中，反斜杠用作目录分隔符。在字符串中，反斜杠 \ 用作转义字符，用于表示特殊字符或字符序列，比如：

- \n 换行符，将光标位置移到下一行开头。
- \r 回车符，将光标位置移到本行开头。
- \t 水平制表符，也即 Tab 键，一般相当于四个空格。
- \\ 反斜线；在使用反斜杠作为转义字符时，为了表示反斜杠本身，需要使用两个连续的反斜杠\\。
- \' 单引号
- \" 双引号
- \ 在字符串行尾的续行符，即一行未完，转到下一行继续写。

取模运算符 % 用于获取两个数值相除的余数，比如 10 % 3 的结果为 1。幂运算符 ** 用于将一个数值的幂次方，比如 2**3 的结果为 8。

什么是转义字符？

转义字符是一种在字符串中使用的特殊字符序列，以反斜杠 \ 开头。在 Python 中，转义字符用于表示一些特殊字符、控制字符或无法直接输入的字符。通过使用转义字符，我们可以在字符串中插入换行符、制表符、引号等特殊字符。

括号

在 Python 中，运算符有不同的优先级。有时我们需要改变运算符的优先级顺序，可以使用圆括号 (parentheses) 来改变它们的顺序。圆括号可以用于明确指定某些运算的执行顺序，确保它们在其他运算之前或之后进行。

请大家自行比较下两例：

```
result = 2 + 3 * 4
result = (2 + 3) * 4
```

根据运算符的优先级规则，乘法运算 * 具有更高的优先级，因此先执行乘法，然后再进行加法。所以结果是 14。如果我们想先执行加法运算，然后再进行乘法运算，可以使用圆括号来改变优先级。

6.3 比较运算符

Python 比较运算符用于比较两个值，结果为 True 或 False。

相等、不等

相等运算符 == 比较两个值是否相等，返回 True 或 False。不等运算符 != 比较两个值是否不相等，返回 True 或 False。

```
x = 5
y = 3
print(x == y)     # False
print(x == 5)     # True
print(x != y)     # True
print(x != 5)     # False
print(x != 5.0)   # False
```

==, !=

图 4. 相等、不等

大于、大于等于

大于运算符 > 比较左边的值是否大于右边的值，返回 True 或 False。大于等于运算符 >= 比较左边的值是否大于等于右边的值，返回 True 或 False。

```python
x = 5
y = 3

print(x > y)   # True
print(x > 10)  # False
print(x >= y)  # True
print(x >= 5)  # True
```

>, >=

图 5. 大于、大于等于

小于、小于等于

小于运算符 < 比较左边的值是否小于右边的值，返回 True 或 False。小于等于运算符 <= 比较左边的值是否小于等于右边的值，返回 True 或 False。

```python
x = 5
y = 3

print(x < y)   # False
print(x < 10)  # True
print(x <= y)  # False
print(x <= 5)  # True
```

<, <=

图 6. 小于、小于等于

6.4 逻辑运算符

Python 中有三种逻辑运算符，分别为 and、or 和 not，这些逻辑运算符可用于布尔类型的操作数上。这三种逻辑运算符实际上体现的是真值表 (truth table) 的逻辑。

如图 7 所示，真值表是一个逻辑表格，用于列出逻辑表达式的所有可能的输入组合和对应的输出结果。它展示了在不同的输入情况下，逻辑表达式的真值 True 或假值 False。下面对每种逻辑运算符进行详细的讲解。

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

A	not A
True	False
False	True

图 7. 真值表

和运算符 and 当左右两边的操作数都为 True 时，返回 True，否则返回 False。或运算符 or 当左右两边的操作数至少有一个为 True 时，返回 True，否则返回 False。取非运算符 not 对一个布尔类型的操作数取反，如果操作数为 True，返回 False，否则返回 True。请大家在 JupyterLab 自行练习图 8。

逻辑运算符常用于条件判断、循环控制等语句中。通过组合不同的逻辑运算符，可以实现复杂的逻辑表达式。

逻辑运算符

```
# 和 and
print(True and True)
print(True and False)
print(False and True)
print(False and False)

# 或 or
print(True or True)
print(True or False)
print(False or True)
print(False or False)

# 非 not
print(not True)
print(not False)
```

图 8. 逻辑运算符

6.5 赋值运算符

Python 中的赋值运算符用于将值分配给变量，下面逐一讲解。

等号 = 将右侧的值赋给左侧的变量。加等于 += 将右侧的值加到左侧的变量上，并将结果赋给左侧的变量。

减等于 -= 将右侧的值从左侧的变量中减去，并将结果赋给左侧的变量。

乘等于 *= 将右侧的值乘以左侧的变量，并将结果赋给左侧的变量。

除等于 /= 将左侧的变量除以右侧的值，并将结果赋给左侧的变量。

取模等于 %= 将左侧的变量对右侧的值取模，并将结果赋给左侧的变量。

幂等于 **= 将左侧的变量的值提高到右侧的值的幂，并将结果赋给左侧的变量。

赋值运算符

```
a = 5
print(a)
a += 3  # 等同于 a = a + 3，此时 a 的值为 8
print(a)
a -= 3  # 等同于 a = a - 3，此时 a 的值为 5
print(a)
a *= 2  # 等同于 a = a * 2，此时 a 的值为 10
print(a)
a /= 5  # 等同于 a = a / 5，此时 a 的值为 2.0
print(a)
a %= 3  # 等同于 a = a % 3，此时 a 的值为 2.0
print(a)
a **= 3 # 等同于 a = a ** 3，此时 a 的值为 8.0
print(a)
```

图 9. 赋值运算

6.6 成员运算符

Python 中成员运算符用于测试是否存在于序列中。共有两个成员运算符：a) in：如果在序列中找到值，返回 True，否则返回 False。b) not in：如果在序列中没有找到值，返回 True，否则返回 False。

图 10 是成员运算符的示例代码，请大家在 JupyterLab 中自行练习。

成员运算符

```
# 定义一个列表
my_list = [1, 2, 3, 4, 5]

# 判断元素是否在列表中
print(3 in my_list)   # True
print(6 in my_list)   # False

# 判断元素是否不在列表中
print(3 not in my_list)   # False
print(6 not in my_list)   # True
```

图 10. 成员运算

6.7 身份运算符

Python 身份运算符包括 is 和 is not，用于判断两个对象是否引用同一个内存地址。请大家回顾上一章介绍的视图、浅复制、深复制这三个概念。简单来说，浅复制只复制对象的一层内容，不涉及到嵌套的可变对象。深复制创建一个全新的对象，并递归地复制原始对象及其嵌套的可变对象。每个对象的副本都是独立的，修改原始对象或其嵌套对象不会影响深复制的对象。深复制涉及到多层嵌套的可变对象，确保每个对象都被复制。

请大家自行练习图 11 给出代码。

身份运算符

```python
import copy
a = [1, 2, 3]
b = a
# 视图 b 引用 a 的内存地址
c = [1, 2, 3]
d = a.copy()

print(a is b)
# 输出 True，因为 a 和 b 引用同一个内存地址
print(a is not c)
# 输出 True，因为 a 和 c 引用不同的内存地址
print(a == c)
# 输出 True，因为 a 和 c 的值相等
print(a is not d)
# 输出 True，因为 a 和 d 引用不同的内存地址
print(a == d)
# 输出 True，因为 a 和 d 的值相等

a_2_layers = [1, 2, [3, 4]]
d_2_layers = a_2_layers.copy()
e_2_layers = copy.deepcopy(a_2_layers)

print(a_2_layers is d_2_layers)
print(a_2_layers[2] is d_2_layers[2]) # 请特别关注

print(a_2_layers is e_2_layers)
print(a_2_layers[2] is e_2_layers[2])
```

图 11. 身份运算

6.8 优先级

在 Python 中，不同类型的运算符优先级是不同的，当一个表达式中有多个运算符时，会按照优先级的顺序依次计算，可以使用括号改变运算顺序。下面是 Python 中常见的运算符优先级列表，从高到低排列：

- 括号运算符：`()`，用于改变运算顺序。
- 正负号运算符：`+x`，`-x`，用于对数字取正负。
- 算术运算符：`**`，`*`，`/`，`//`，`%`，用于数字的算术运算。
- 位运算符：`~`，`&`，`|`，`^`，`<<`，`>>`，用于二进制位的运算。
- 比较运算符：`<`，`<=`，`>`，`>=`，`==`，`!=`，用于比较大小关系。
- 身份运算符：`is`，`is not`，用于判断两个对象是否相同。
- 成员运算符：`in`，`not in`，用于判断一个元素是否属于一个集合。
- 逻辑运算符：`not`，`and`，`or`，用于逻辑运算。

这部分我们不再展开介绍，如果后续用到的话，请大家自行学习。

什么是位运算符？

Python 提供了一组位运算符 (bitwise operator)，用于在二进制级别对整数进行操作。这些位运算符将整数的二进制表示作为操作数，并对每个位进行逻辑运算。

请大家完成下面 1 道题目。

Q1. 本章的唯一的题目就是请大家在 JupyterLab 中练习本章正文给出的示例代码。

* 不提供答案。

Control Flow Statements in Python

7 Python 控制结构

日后尽量避免 for 循环，学会向量化绕行

幸存下来的不是最强壮的物种，也不是最聪明的物种，而是对变化最敏感的物种。

It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change.

—— 查尔斯·达尔文 (Charles Darwin) | 进化论之父 | 1809 ~ 1882

7.1 什么是控制结构?

在 Python 中，控制结构是一种用于控制程序流程的结构，包括条件语句、循环语句和异常处理语句。这些结构可以根据不同的条件决定程序运行的路径，并根据需要重复执行代码块或捕获和处理异常情况。

这一节我们用实例全景展示这几种常见的控制结构。

条件语句

条件语句在程序中用于根据不同的条件来控制执行不同的代码块。Python 中最常用的条件语句是 if 语句，if 语句后面跟一个布尔表达式，如果布尔表达式为真，就执行 if 语句块中的代码，否则执行 else 语句块中的代码。还有 elif 语句可以用来处理多种情况。

图 1 是一个简单例子，如果成绩大于等于 60 分，输出 "及格"，否则输出 "不及格"。图 1 中代码对应的流程图 (flowchart) 如图 2 所示。

注意，代码中用到了本书第 4 章讲的“四空格”缩进，还用到了上一章讲过的 >= 判断运算，忘记的话请回顾。此外，大家在 JupyterLab 练习图 1 给出代码时，注意字符串要用半角引号。

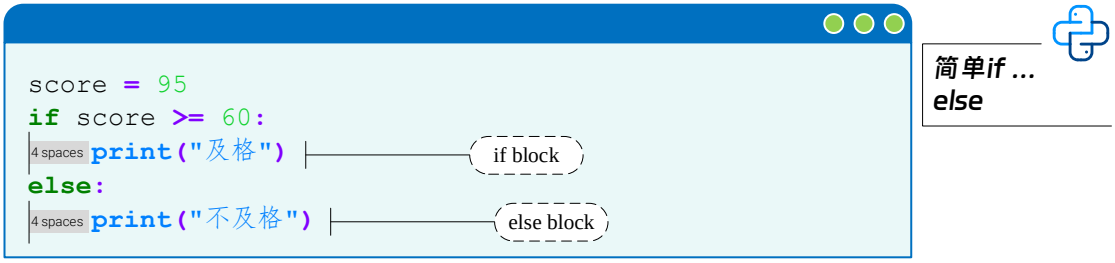


图 1. 用 if 判断是否成绩及格

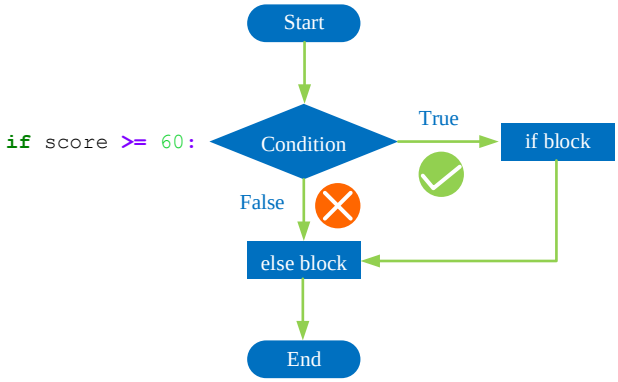


图 2. 用 if 判断是否成绩及格，流程图

循环语句

循环语句用于在程序中重复执行相同的代码块，直到某个条件满足为止。Python 中有两种循环语句：for 循环和 while 循环。

本书前文几次使用过 for 循环，相信大家已经不再陌生。简单来说，for 循环通常用于遍历序列，例如列表或字符串。在 for 循环中，代码块会在每个元素上执行一次，直到循环结束。

请大家在 JupyterLab 中自行练习图 3 代码。

for循环

```
# 循环字符串内字符

str_for_loop = 'Matplotlib'

for str_idx in str_for_loop:
    print(str_idx)

# 循环list中元素
list_for_loop = ['Matplotlib', 'NumPy', 'Seaborn',
                 'Pandas', 'Plotly', 'SK-learn']
for item_idx in list_for_loop:
    print(item_idx)

# 循环中嵌入 if 判断
packages_visual = ['Matplotlib', 'Seaborn', 'Plotly']
for item_idx in list_for_loop:
    print('================')
    print(item_idx)
    if item_idx in packages_visual:
        print('A visualization tool')

# 嵌套 for 循环
for item_idx in list_for_loop:
    print('===============')
    print(item_idx)

    for item_idx in item_idx:
        print(item_idx)
```

图 3. 四个 for 循环例子

while 循环会重复执行代码块，直到循环条件不再满足为止。循环条件在每次循环开始前都会被检查。

图 4 给出的例子为使用 while 循环输出 0 到 4。本书不展开介绍 while 循环。

```
i = 0
while i < 5:
    print(i)
    i += 1
```

while循环

图 4. while 循环例子

异常处理语句

异常处理语句用于捕获和处理程序中出现的异常情况。Python 中的异常处理语句使用 try 和 except 关键字，try 语句块包含可能引发异常的代码，而 except 语句块用于处理异常情况。

图 5 是一个例子，使用 try 和 except 捕获除数为零的异常。本章不展开讲解 try … except，大家日后用到时再深入探究。

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("除数不能为零")
```

try ...
except ...

图 5. 用 try … except 捕捉异常

7.2 条件语句

嵌套 if 判断

大家可能好奇，如果图 1 中赋值能否为用户输入？此外，如果用户输入错误是否有提示信息？我们当然可以在图 2 基础上用多层判断完成这些需求。图 6 给出的代码可以完成上述要求，对应的流程图如图 7 所示。

在这个代码中：首先，使用 input() 函数获取用户输入的数值，并将其存储在 value 变量中。使用 isdigit()方法检查输入是否是一个数值。如果是数值，则执行 if 语句块内的代码。将数值转换为整数类型，并存储在 number 变量中。

使用嵌套的 if 语句来检查 number 是否在 0~100 之间。如果在该范围内，则继续执行内部的 if 语句块。

在内部的 if 语句块中，判断 number 是否小于 60。如果小于 60，则打印"不及格"；否则打印"及格"。

如果输入的数值不在 0 ~ 100 之间，将打印"数值不在 0 ~ 100 之间"。

如果输入不是一个数值，将打印"输入不是一个数值"。

请大家判断图 6 中代码第一层 if 对应的代码块是什么？

注意，上述代码假设用户输入的数值为整数。如果需要支持浮点数，请相应地调整代码。

三层if ... else

```
value = input("请输入一个数值： ")

# 第一层
if value.isdigit():
    number = int(value)

    # 第二层
    if 0 <= number <= 100:

        # 第三层
        if number < 60:
            print("不及格")
        else:
            print("及格")
        # 第三层结束

    else:
        print("数值不在0~100之间")
    # 第二层结束

else:
    print("输入不是一个数值")
# 第一层结束
```

2nd level: if block
3rd level: if block
3rd level: else block
2nd level: else block
1st level: else block

图 6. 用 if 判断是否成绩及格，三层判断

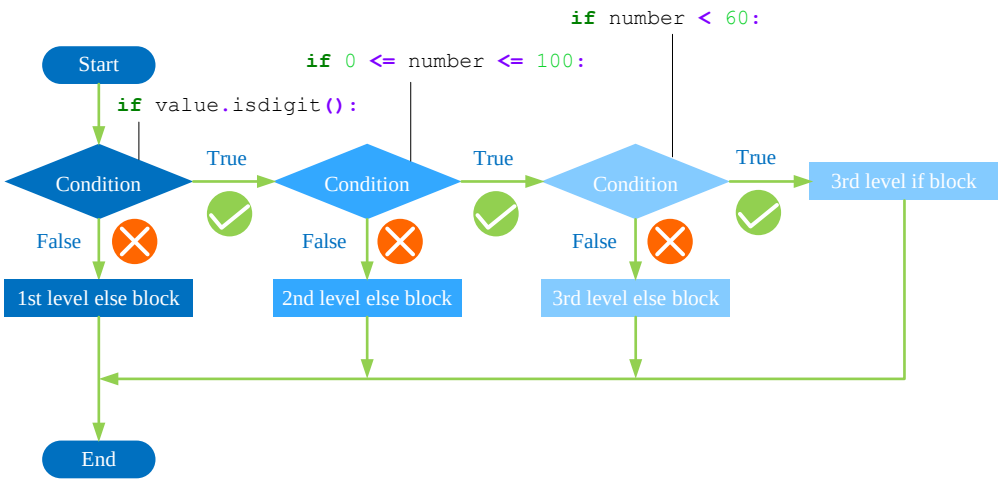


图 7. 用 if 判断是否成绩及格，三层判断，流程图

if...elif...else 语句

if...elif...else 语句用于判断多个条件，如果第一个条件成立，则执行 if 语句中的代码块；如果第一个条件不成立，但第二个条件成立，则执行 elif 语句中的代码块；如果前面的条件都不成立，则执行 else 语句中的代码块。

注意，elif 的语句数量没有上限。但是，如果代码中 elif 数量过多，需要考虑简化代码结构。

图 8 代码判断一个数是正数、负数还是 0。请大家根据图 7 修改图 8 中代码。

```
num = input('输入一个整数')
num = int(num)

if num > 0:
    print("num is positive")
elif num < 0:
    print("num is negative")
else:
    print("num is zero")
```

if ... elif ... else ...

图 8. if...elif...else 语句

break、continue、pass 语句

在 Python 的 if 条件语句、for 循环语句中，可以使用 break、continue 和 pass 来控制循环的行为。

break 语句可以用来跳出当前循环。当循环执行到 break 语句时，程序将立即跳出循环体，继续执行循环外的语句。下面是一个使用 break 的例子，该循环会在 i 等于 3 时跳出。

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

break

图 9. 使用 break

continue 语句可以用来跳过当前循环中的某些语句。当循环执行到 continue 语句时，程序将立即跳过本次循环，继续执行下一次循环。下面是一个使用 continue 的例子，该循环会在 i 等于 3 时跳过本次循环。

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

图 10. 使用 continue

pass 语句什么也不做，它只是一个空语句占位符。在需要有语句的地方，但是暂时不想编写任何语句时，可以使用 pass 语句。下面是一个使用 pass 的例子，该循环中的所有元素都会被输出。

```
for i in range(1, 6):
    pass
    print(i)
```

图 11. 使用 pass

7.3 for 循环语句

本节介绍 for 循环一些常见用法。

使用 enumerate()

在 Python 中，enumerate() 是一个用于在迭代时跟踪索引的内置函数。enumerate() 函数可以将一个可迭代对象转换为一个由索引和元素组成的枚举对象。

下面是一个简单的例子，展示了如何在 for 循环中使用 enumerate() 函数。

```
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(index, fruit)
```

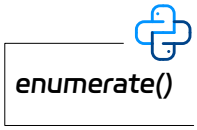


图 12. 使用 enumerate()，从 0 开始

在这个例子中，fruits 列表中的每个元素都会被遍历一遍，每次遍历都会获得该元素的值和其在列表中的索引。这些值分别被赋给 index 和 fruit 变量，并打印输出。

需要注意的是，enumerate 函数的默认起始索引为 0，但是也可以通过传递第二个参数来指定起始索引。例如，如果想要从 1 开始索引，可以使用以下代码。

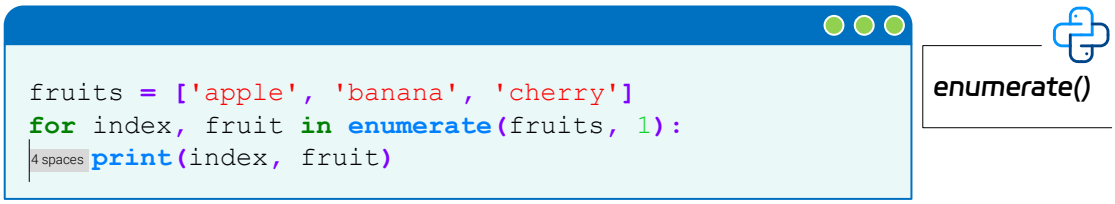


图 13. 使用 enumerate()，从 1 开始

使用 zip()

在 Python 中，zip() 函数可以将多个可迭代对象的元素组合成元组，然后返回这些元组组成的迭代器。在 for 循环中使用 zip() 函数可以方便地同时遍历多个可迭代对象，且当这些可迭代对象的长度不同时，zip() 函数会以最短长度的可迭代对象为准进行迭代。

如果想要打印出每个学生的姓名和对应的成绩，可以使用 zip() 函数和 for 循环，代码如所示。

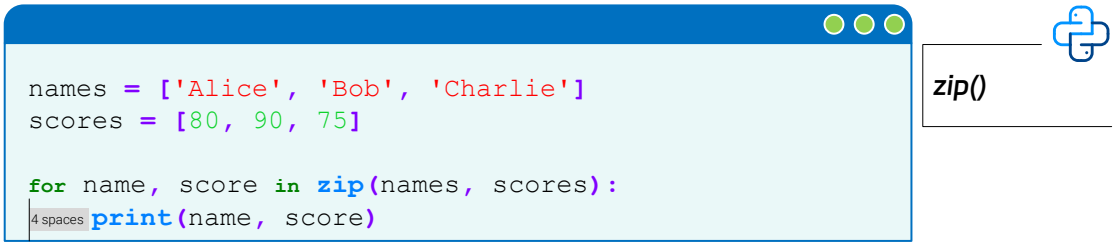


图 14. 使用 zip() 同时遍历多个对象

在这个例子中，zip() 函数将 names 和 scores 两个列表按照位置进行组合，然后返回一个迭代器，其中的每个元素都是一个元组，元组的第一个元素为 names 列表中对应位置的元素，第二个元素为 scores 列表中对应位置的元素。在 for 循环中使用了两个变量 name 和 score，分别用来接收每个元组中的两个元素，然后打印出来即可。

需要注意的是，如果可迭代对象的长度不相等，zip()函数会以最短长度的可迭代对象为准进行迭代。

计算向量内积：使用 zip()

可以使用 Python 的内置函数 zip() 和运算符 *，对两个向量中的对应元素逐一相乘并相加，实现向量内积运算。是一个示例代码。

zip() 向量内积

```python
# 计算向量内积

# 定义向量a和b
a = [1, 2, 3, 4, 5]
b = [6, 7, 8, 9, 0]

# 初始化内积为0
dot_product = 0

# 使用for循环计算内积
for i in range(len(a)):
    dot_product += a[i] * b[i]

# 打印内积
print("向量内积为：", dot_product)
```

图 15. 使用 zip() 计算向量内积

在此示例中，通过 zip() 函数将两个 list，a 和 b，中对应位置的元素组合成了元组，然后使用 for 循环逐个遍历并相乘求和，最终得到了向量内积的结果。

什么是向量内积？

向量内积 (inner product)，也称为点积 (dot product)、标量积 (scalar product)，是在线性代数中常见的一种运算，它是两个向量之间的一种数学运算。

给定两个 n 维向量 $\boldsymbol{a} = [a_1, a_2, ..., a_n]$ 和 $\boldsymbol{b} = [b_1, b_2, ..., b_n]$，它们的内积定义为 $\boldsymbol{a} \cdot \boldsymbol{b} = a_1b_1 + a_2b_2 + ... + a_nb_n$。这个公式的意义是将两个向量的对应分量相乘，然后将乘积相加，从而得到它们的内积。

例如，如果有两个二维向量分别为 $\boldsymbol{a} = [1, 2]$ 和 $\boldsymbol{b} = [3, 4]$，则它们的内积为：$\boldsymbol{a} \cdot \boldsymbol{b} = 1 \times 3 + 2 \times 4 = 11$。向量内积的结果是一个标量，也就是一个值，而不是向量。它可以用来计算向量之间的夹角，衡量它们的相似性，以及用于向量空间的正交分解等。

在实际应用中，向量内积被广泛用于机器学习、计算机视觉、信号处理、物理学等领域。在机器学习中，向量内积常用于计算特征之间的相似度，从而进行分类、聚类等任务。在计算机视觉中，向量内积可以用于计算两个图像之间的相似度。

range(start [, stop, step])

range() 是 Python 内置的函数，用于生成一个整数序列，常用于 for 循环中的计数器。参数为：

- start 是序列起始值；
- stop 是序列结束值 （不包含）；
- step 是序列中相邻两个数之间的步长 （默认为 1）。

range() 函数生成的是一个可迭代对象，而不是一个列表。这样做的好处是，可以节省内存空间，尤其在需要生成很长的序列时。

下面是一些使用 range() 函数的示例：

a) 生成从 0 到 4 的整数序列

```
for i in range(4 + 1):
    print(i)
```

b) 生成从 10 到 20 的整数序列

```
for i in range(10, 20 + 1):
    print(i)
```

c) 生成从 1 到 10 的奇数序列

```
for i in range(1, 10 + 1, 2):
    print(i)
```

d) 生成从 10 到 1 的倒序整数序列

```
for i in range(10, 1 - 1, -1):
    print(i)
```

d) 将 range() 生成的可迭代对象变成 list：

```
list(range(10, 1 - 1, -1))
```

请大家在 JupyterLab 中自行运行如上几段段代码。

矩阵乘法：嵌套 for 循环

下面介绍如何使用嵌套 for 循环完成矩阵乘法。

图 16 所示为两个 2 × 2 矩阵相乘如何得到矩阵 $\boldsymbol{C}$ 的每一个元素。

矩阵 $\boldsymbol{A}$ 的第一行元素和矩阵 $\boldsymbol{B}$ 第一列对应元素分别相乘，再相加，结果为矩阵 $\boldsymbol{C}$ 的第一行、第一列元素 $c_{1,1}$。

矩阵 $\boldsymbol{A}$ 的第一行元素和矩阵 $\boldsymbol{B}$ 第二列对应元素分别相乘，再相加，得到 $c_{1,2}$。

同理，依次获得矩阵 $\boldsymbol{C}$ 的 $c_{2,1}$ 和 $c_{2,2}$ 两个元素。

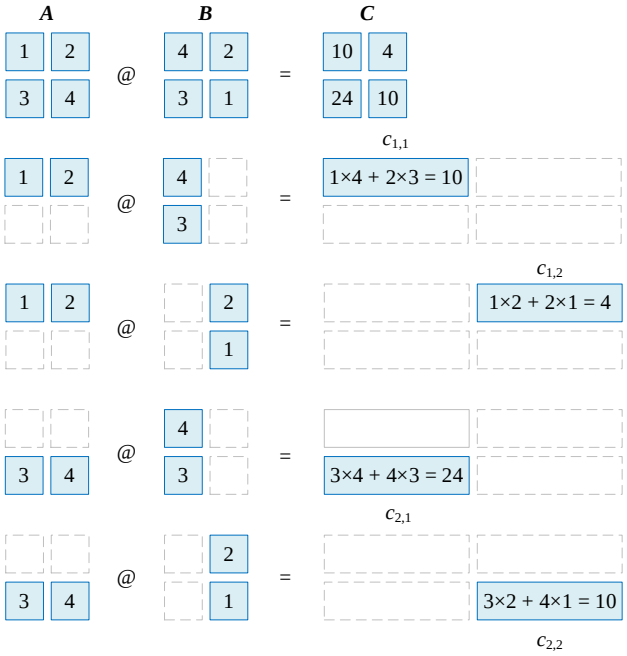


图 16. 矩阵乘法规则，两个 2 × 2 矩阵相乘为例

什么是矩阵乘法？

矩阵乘法 (matrix multiplication) 是一种线性代数运算，用于将两个矩阵相乘。对于两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，它们的乘积 $\boldsymbol{AB}$ 的元素是通过将 $\boldsymbol{A}$ 的每一行与 $\boldsymbol{B}$ 的每一列进行内积运算得到的。

具体而言，假设 $\boldsymbol{A}$ 是一个 $m \times n$ 的矩阵，$\boldsymbol{B}$ 是一个 $n \times p$ 的矩阵，则它们的乘积 $\boldsymbol{C} = \boldsymbol{AB}$ 是一个 $m \times p$ 的矩阵，其中第 i 行第 j 列的元素 $c_{i,j}$ 为 $\boldsymbol{A}$ 的第 i 行与 $\boldsymbol{B}$ 的第 j 列的内积。如果 $\boldsymbol{A}$ 的第 i 行为 $a_{i,1}, a_{i,2}, ..., a_{i,n}$，$\boldsymbol{B}$ 的第 j 列为 $b_{1,j}, b_{2,j}, ..., b_{n,j}$，则 $\boldsymbol{C} = \boldsymbol{AB}$ 的第 i 行第 j 列的元素为 $a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + ... + a_{i,n}b_{n,j}$。

矩阵乘法在许多领域都有广泛的应用，例如线性代数、信号处理、图形学和机器学习等。在机器学习中，矩阵乘法通常用于计算神经网络的前向传播过程，其中输入矩阵与权重矩阵相乘，得到隐藏层的输出矩阵。

嵌套for循环

```python
# 定义矩阵 A 和 B
A = [[1, 2, 10, 20],
     [3, 4, 30, 40]]

B = [[4, 2],
     [3, 1],
     [40, 20],
     [30, 10]]

# 定义全 0 矩阵 C 用来存放结果
C = [[0, 0],
     [0, 0]]

# 矩阵乘法

# 遍历 A 的行
for i in range(len(A)): # len(A) 给出 A 的行数

    # 遍历 B 的列
    for j in range(len(B[0])):
    # len(B[0]) 给出 B 的列数

        # 这一层相当于消去 p 所在的维度，即压缩
        for k in range(len(B)):
            C[i][j] += A[i][k] * B[k][j]
            # 完成对应元素相乘，再求和

# 输出结果
for row in C:
    print(row)
```

图 17. 使用嵌套 for 循环计算矩阵乘法

列表生成式

在 Python 中，列表生成式 (list comprehension) 是一种简洁的语法形式，用于快速生成新的列表。

它的语法形式为 [expression for item in iterable if condition]，其中 expression 表示要生成的元素，item 表示迭代的变量，iterable 表示迭代的对象，if condition 表示可选的过滤条件。

举个例子，假设我们想要生成一个包含 1 到 10 之间所有偶数的列表，我们可以使用如下列表生成式。

```
even_numbers = [num for num in range(1, 11)
                if num % 2 == 0] # 一行放不下
print(even_numbers) # Output: [2, 4, 6, 8, 10]
```

列表生成式

图 18. 使用列表生成式，获得 1 ~ 10 之间所有偶数列表

在上述代码中，我们使用列表生成式创建了一个包含 1 到 10 之间所有偶数的列表。具体来说，我们使用 range(1, 11)迭代 1 到 10 的数字，对每个数字进行取模操作，只保留余数为 0 的数字，即偶数，最终将这些数字存储到一个新的列表中。

使用列表生成式还可以嵌套，比如下例。

```
matrix = [[i * j for j in range(1, 4)]
                 for i in range(1, 4)]
print(matrix)
# Output: [[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

列表生成式

图 19. 使用列表生成式，获得 1 ~ 10 之间所有偶数列表

在上述代码中，我们使用嵌套的列表生成式创建了一个 3 × 3 的矩阵。具体来说，我们使用外部的列表生成式迭代 1 到 3 的数字，对每个数字使用内部的列表生成式迭代 1 到 3 的数字，计算它们的乘积并将结果存储到一个新的二维列表中。请大家用上述代码生成图 17 中全 0 矩阵 $\boldsymbol{C}$。

使用列表生成式可以大大简化代码，提高代码的可读性和可维护性。

向量化

向量化运算是使用 NumPy 等库的一种高效运算处理方式，可以避免使用 for 循环。图 20、图 21 所示为利用 NumPy 完成向量内积、矩阵乘法运算。但是这不意味着前文自己写代码计算向量内积、矩阵乘法是无用功！在前文的代码练习中，一方面我们掌握了 for 循环的使用，此外理解了向量内积、矩阵乘法两种数学工具。

```python
import numpy as np

# 定义向量a和b
a = np.array([1, 2, 3, 4, 5])
b = np.array([6, 7, 8, 9, 0])

# 计算向量内积
dot_product = np.dot(a,b)

# 打印内积
print("向量内积为: ", dot_product)
```

numpy.dot()

图 20. 使用 numpy.dot() 计算向量内积

```python
import numpy as np

# 定义矩阵 A 和 B
A = np.array([[1, 2, 10, 20],
              [3, 4, 30, 40]])

B = np.array([[4, 2],
              [3, 1],
              [40, 20],
              [30, 10]])

C = A @ B
print(C)
```

@ NumPy矩阵乘法运算符

图 21. 使用 NumPy 计算矩阵乘法

7.4 迭代器

itertools 是 Python 标准库中的一个模块，提供了用于创建和操作迭代器的函数。迭代器是一种用于遍历数据集合的对象，它能够逐个返回数据元素，而无需提前将整个数据集加载到内存中。

itertools 模块包含了一系列用于高效处理迭代器的工具函数，这些函数可以帮助我们在处理数据集时节省内存和提高效率。它提供了诸如组合、排列、重复元素等功能，以及其他有关迭代器操作的函数。

不放回排列

itertools.permutations 是 Python 标准库中的一个函数，用于返回指定长度的所有可能排列方式。下面举例如何使用 itertools.permutations 函数。

假设有一个字符串 string = 'abc'，我们想要获取它的所有字符排列方式，可以按照以下步骤操作。

```
import itertools
string = 'abc'
perms_all = itertools.permutations(string)
# 返回一个可迭代对象perms，其中包含了string的所有排列方式

# 全排列
for perm_idx in perms_all:
    print(''.join(perm_idx))
```

itertools.permutations

图 22. 3 个字符全排列

这就好比，一个袋子里有三个球，它们分别印有 a、b、c，先后将所有球取出排成一排共有 6 种排列，具体如图 23 所示。

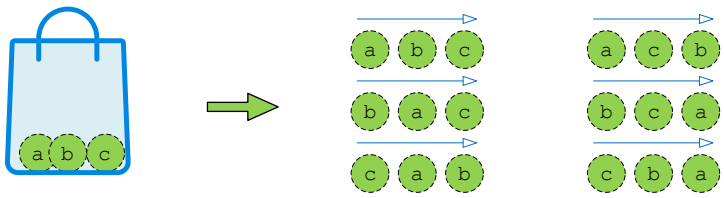


图 23. 3 个元素无放回抽取 3 个，结果有 6 个排列

itertools.permutations 函数还有一个可选参数 r，用于指定返回的排列长度。如果不指定 r，则默认返回与输入序列长度相同的排列。例如，我们可以通过以下方式获取 string 的所有长度为 2 的排列。

```
import itertools
string = 'abc'

# 3个不放回取2个的排列
perms_2 = itertools.permutations(string, 2)
# 返回一个包含所有长度为2的排列的可迭代对象perms

for perm_idx in perms_2:
    print(''.join(perm_idx))
```

itertools.permutations

图 24. 3 个字符无放回取 2 个排列

还是以前文小球为例，如图 25 所示，3 个元素无放回抽取 2 个，结果有 6 个排列。大家可能已经发现这个结果和一致。这也不难理解，袋子里一共有 3 个球，无放回拿出两个之后，第三个球是什么字母已经确定，没有任何悬念。

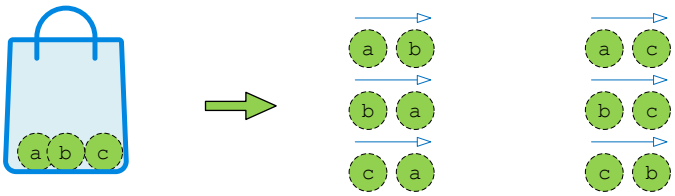


图 25. 3 个元素无放回抽取 2 个，结果有 6 个排列

不放回组合

itertools.combinations 是 Python 中的一个模块，它提供了一种用于生成组合的函数。

使用 itertools.combinations 函数，需要导入 itertools 模块，然后调用 combinations 函数，传入两个参数：一个可迭代对象和一个整数，表示要选择的元素个数。该函数会返回一个迭代器，通过迭代器你可以获得所有可能的组合。

```
import itertools
string = 'abc'

# 3个取2个的组合
combs_2 = itertools.combinations(string, 2)
# 返回一个包含所有长度为2的组合的可迭代对象combs_2

for combo_idx in combs_2:
    print(''.join(combo_idx))
```

itertools.combinations

图 26. 3 个字符无放回取 2 个组合

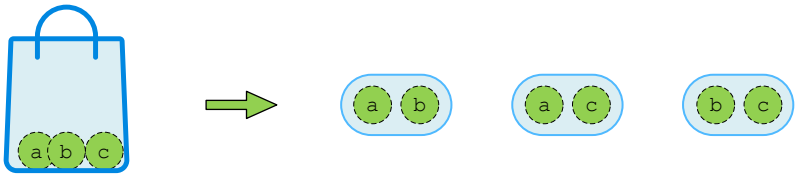


图 27. 3 个元素无放回抽取 2 个，结果有 3 个组合

什么是排列？什么是组合？

排列是指从一组元素中按照一定顺序选择若干个元素形成的不同序列，每个元素只能选取一次。

组合是指从一组元素中无序地选择若干个元素形成的不同集合，每个元素只能选取一次。

有放回排列

前文介绍的排列、组合都是无放回抽样，下面聊聊有放回抽样。还是以小球为例，如图 28 所示，有放回抽样就是从口袋中摸出一个球之后，记录字母，然后将小球再放回口袋。下一次抽取时，这个球还有被抽到的机会。

什么是有放回？什么是无放回？

有放回抽取是指在进行抽样时，每次抽取后将被选中的元素放回原始集合中，使得下一次抽取时仍然有可能选中同一个元素。无放回抽取是指在进行抽样时，每次抽取后将被选中的元素从原始集合中移除，使得下一次抽取时不会再选中相同的元素。简而言之，有放回抽取可以多次选中相同元素，而无放回抽取每次选中后都会从集合中移除，确保不会重复选中同一元素。

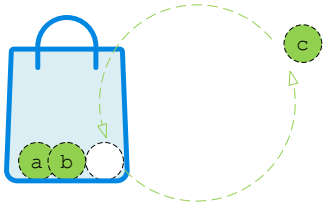


图 28. 有放回抽样

itertools 模块中的 itertools.product 函数可以用于生成有放回排列。它接受一个可迭代对象和一个重复参数，用于指定每个元素可以重复出现的次数。

```python
import itertools

string = 'abc'
# 定义元素列表
elements = list(string)
# 指定重复次数
repeat = 2

# 生成有放回排列
permutations = itertools.product(elements,
repeat=repeat)

# 遍历并打印所有排列
for permutation_idx in permutations:
    print(''.join(permutation_idx))
```

itertools.product

图 29. 3 个字符有放回取 2 个排列

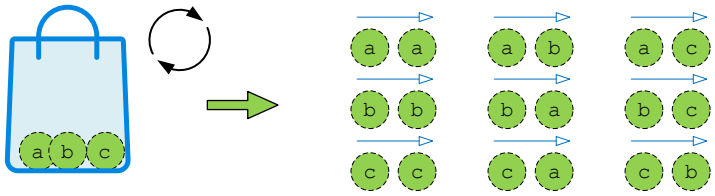


图 30. 3 个元素有放回抽取 2 个，结果有 9 个排列

有放回组合

itertools 模块中的 itertools.combinations_with_replacement 函数可以用于生成有放回组合。该函数接受一个可迭代对象和一个整数参数，用于指定从可迭代对象中选择元素的个数。

```python
import itertools

string = 'abc'
# 定义元素列表
elements = list(string)

# 指定组合长度
length = 2

# 生成有放回组合
combos = itertools.combinations_with_replacement(
         elements, length)

# 遍历并打印所有组合
for combination_idx in combos:
    print(''.join(combination_idx))
```

itertools.combinations_with_replacement

图 31. 3 个字符有放回取 2 个的组合

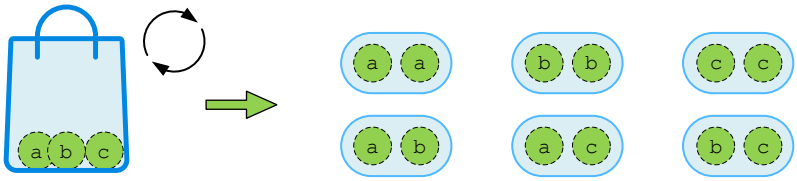


图 32. 3 个元素有放回抽取 2 个，结果有 6 个组合

除了练习本章给出的代码示例之外，请大家完成下面 5 道题目。

Q1. 给定一个整数列表 [3, 5, 2, 7, 1]，找到其中的最大值和最小值，并打印两者之和。

Q2. 使用 while 循环输出 1 到 10 的所有奇数。

Q3. 输入一个数字并将其转换为整数，如果输入的不是数字，则提示用户重新输入直到输入数字为止。

Q4. 求 100 以内的素数。

Q5. 请用至少两种不同办法计算 1-100 中奇数之和。

* 题目答案在 Bk1_Ch07_01.ipynb。

Functions in Python

8 Python 函数

内置函数、自定义函数、Lambda 函数 …

很多人在二十五岁便垂垂老矣，直到七十五岁才入土为安。

Many people die at twenty five and aren't buried until they are seventy five.

—— 本杰明·富兰克林 (Benjamin Franklin) | 美国政治家 | 1706 ~ 1790

8.1 什么是 Python 函数?

这本书学到这里，相信大家对函数这个概念已经不陌生。简单来说，在 Python 中，函数是一段可重复使用的代码块，用于执行特定任务或完成特定操作。函数可以接受输入参数，并且可以返回具体值、或者不返回任何值作为结果。

比如，大家已经非常熟悉的 print()，这个函数的输入参数是要打印的字符串，在完成打印之后，这个函数并没有任何的输出值。

再举个几例子，很多函数都返回具体值，比如 len() 返回 list 元素个数，range() 生成一个可以用在 for 循环的整数序列，list() 可以将。

再者，很多数值操作、科学计算的函数都打包在 NumPy、SciPy 这样的库中，比如大家已经见过的 numpy.array() 等等。

通过使用函数，可以将代码分解成小块，每个块都完成一个特定的任务。这使得代码更易于理解、测试和维护。同时，函数也可以在不同的上下文中重复使用，提高代码的重用性和可维护性。

代数角度，什么是函数?

从代数角度来看，函数是一种数学概念，描述了输入和输出之间的关系。它将一个集合中的每个元素映射到另一个集合中的唯一元素。函数用公式、图表或描述性语言定义，具有定义域和值域的概念。函数在数学中被用于解决问题、建模现实世界，并具有单值性、唯一性等特性。代数中的函数描述了数学方程、曲线和变换，并帮助我们理解数学关系及其应用。

几种函数类型

在 Python 中，有以下几种函数类型：

- 内置函数：Python 解释器提供的函数，例如 print()、len()、range()等。
- 自定义函数：由用户定义的函数。
- Lambda 函数：也称为匿名函数，是一种简单的函数形式，可以通过 lambda 关键字定义。
- 生成器函数：是一种特殊的函数，用于生成一个迭代器，可以使用 yield 关键字定义。本章不展开介绍生成器函数。
- 方法：是与对象相关联的函数，可以使用"."符号调用。例如字符串类型的方法，可以使用字符串变量名.方法名()的形式调用。大家会在 Pandas 中经常看到这种用法。

为什么需要自定义函数?

既然 NumPy、SciPy、SymPy 等等库中提供大量可重复利用的函数，为什么还要兴师动众“自定义函数”?

这个答案其实很简单。现成的函数不能满足“私人订制”需求。

此外，自定义函数在 Python 中的作用是提高代码复用性、模块化和组织性，抽象和封装复杂问题，使代码结构和逻辑更清晰，增加可扩展性和灵活性。通过封装可重复使用的代码块为函

数，避免重复编写相同的代码，并将大型任务分解为小型函数，使程序更易理解和维护。自定义函数提高代码的可读性、可维护性，并支持程序扩展和修改，使代码更结构化和可管理。

包、模块、函数

在 Python 中，一个包 (package) 是一组相关模块 (module) 的集合，一个模块是包含 Python 定义和语句的文件。而一个函数则是在模块或者在包中定义的可重用代码块，用于执行特定任务或计算特定值。

通常情况下，一个模块通常是一个.py 文件，包含了多个函数和类等定义。一个包则是一个包含了多个模块的目录，通常还包括一个特殊的__init__.py 文件，用于初始化该包。在使用时，需要使用 import 关键字导入模块或者包，从而可以使用其中定义的函数和类等。而函数则是模块或包中定义的一段可重用的代码块，用于完成特定的功能。

因此，包中可以包含多个模块，模块中可以包含多个函数，而函数是模块和包中的可重用代码块。

以 NumPy 为例，NumPy 是 Python 中用于科学计算的一个库，其包含了很多有用的数值计算函数和数据结构。下面是 NumPy 库中常见的模块和函数的介绍：

numpy.linalg 这个模块提供了一些线性代数相关的函数，包括矩阵分解、行列式计算、特征值和特征向量计算等。常见的函数有：

- numpy.linalg.det：计算一个方阵的行列式。
- numpy.linalg.inv：计算一个方阵的逆矩阵。
- numpy.linalg.eig：计算一个方阵的特征值和特征向量。
- numpy.linalg.svd：计算一个矩阵的奇异值分解。

numpy.random 这个模块提供了随机数生成的函数，包括生成服从不同分布的随机数。常见的函数有：

- numpy.random.rand：生成 0~1 之间均匀分布的随机数。
- numpy.random.randn：生成符合标准正态分布的随机数。
- numpy.random.randint：生成指定范围内的整数随机数。

数学函数 vs 编程函数

从代数角度来看，函数是一种数学对象，用于描述两个集合之间的关系。一个函数将一个集合中的每个元素 (称为输入) 映射到另一个集合中的唯一元素 (称为输出)。

一元函数通常表示为 $f(x)$，其中 x 是输入变量，$f(x)$ 是输出变量。比如一元一次函数，$f(x) = 2x + 1$，一元二次函数 $f(x) = x^2 + 2x$，正弦函数 $f(x) = \sin(x)$，指数函数 $f(x) = \exp(x)$。函数可以用各种方式定义，包括通过公式、算法、图表或描述性语言。它可以是连续的、离散的或混合的，具体取决于输入和输出的集合的性质。

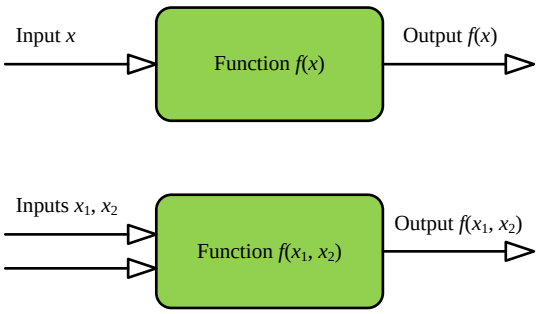


图 1. 一元函数、二元函数的映射

数学上的函数也可以有不止一个输入，比如二元函数 $f(x_1, x_2)$ 便有 2 个输入，三元函数 $f(x_1, x_2, x_3)$ 有 3 个输入，多元函数 $f(x_1, x_2, \ldots, x_D)$ 有 D 个输入。

数学上，函数的定义包括以下要素：

- 定义域 (domain)：定义域是输入变量可能的取值范围。它是函数的输入集合。
- 值域 (range)：值域是函数的输出可能的取值范围。它是函数的输出集合。
- 规则 (rule)：规则定义了输入和输出之间的映射关系。它描述了如何根据给定的输入计算输出。

代数角度的函数概念与计算机编程中的函数概念有些相似，但也有一些不同之处。在代数中，函数是描述输入和输出之间关系的抽象概念，而在编程中，函数是可执行的代码块，用于执行特定的任务。然而，两者之间的基本思想都是处理输入并生成输出。

数学上的函数和编程上的函数在概念和应用上存在一些异同之处。

无论是数学上的函数还是编程上的函数，它们都涉及输入和输出。数学函数接受输入值并产生相应的输出值，而编程函数接受参数但是未必返回结果。

数学上的函数还是编程上的函数都有一个定义域和一个规则，描述了如何将输入转换为输出。无论是通过公式、算法还是逻辑操作，函数都定义了输入和输出之间的关系。

数学上的函数还是编程上的函数的概念都具有可重用性。无论是在数学中还是在编程中，函数可以在多个场景中被多次调用和使用，避免了重复编写相同的代码。

数学上的函数和编程上的函数显然也有很大区别。数学函数通常用符号、公式或描述性语言来表示，如 $f(x) = x^2$。而编程函数则以编程语言的语法和结构来定义和表示，如 `def square(x): return x**2`。编程函数可以包含额外的程序控制结构，如条件语句、循环等，以实现更复杂的逻辑和操作。

总体而言，数学上的函数更关注描述数学关系，而编程上的函数更侧重于实现特定的计算或操作。虽然两者有相似的概念，但具体的表示方式、范围和应用场景可能会有所不同。

8.2 自定义函数

无输入、无返回

在 Python 中，我们可以自定义函数来完成一些特定的任务。函数通常接受输入参数并返回输出结果。但有时我们需要定义一个函数，它既没有输入参数，也不返回任何结果。这种函数被称为没有输入、没有返回值的函数。

定义这种函数的方法和定义其他函数类似，只是在定义函数时省略了输入参数和 return 语句。比如下例，这个函数名为 say_hello，它不接受任何输入参数，执行函数体中的代码时会输出字符串 "Hello!"。

```
def say_hello():
    # 自定义函数：打印问候
    # 输入：无
    # 输出：无
    print("Hello!")

# 调用自定义函数
say_hello()
```

Function block

无输入、无输出自定义函数

图 2. 无输入、无输出函数

下面，我们再看一个复杂的例子。这个例子，我们也定义了一个无输入、无输出函数用来美化线图。图 3 所示为利用 Matplotlib 绘制的一元一次函数、一元二次函数线图美化之后的结果。

本书第 10 章将专门介绍如何绘制线图，此外鸢尾花书《可视之美》将专门介绍 Python 可视化专题。

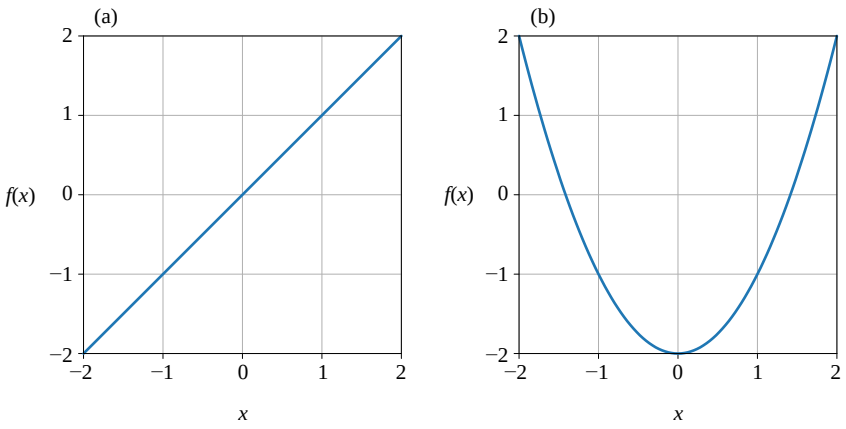


图 3. 绘制线图并美化

无输入、无输出自定义函数

```
# 导入包
import matplotlib.pyplot as plt
# 导入 Matplotlib 库中的 pyplot 模块，并将其命名为 plt
import numpy as np
# 导入 NumPy 库，并将其命名为 np

# 自定义函数
def beautify_line_chart():
    # 添加标签
    plt.xlabel("x")
    plt.ylabel("f(x)")
    # 设置坐标轴范围
    plt.xlim(-2, 2)
    plt.ylim(-2, 2)
    # 设置横纵轴刻度
    plt.xticks([-2,-1,0,1,2])
    plt.yticks([-2,-1,0,1,2])
    # 添加网格线
    plt.grid(True)
    # 横纵轴统一标尺
    plt.gca().set_aspect('equal', adjustable='box')
    # 显示图形
    plt.show()

x_array = np.linspace(-2,2,101)
# 使用NumPy的linspace函数创建一个包含101个元素的数组
# 这些元素均匀地分布在区间[-2, 2]上，左闭右闭

# 绘制直线
fig, ax = plt.subplots(figsize = (4,4))
# plt.subplots()返回值解包为两个变量：fig 和 ax
# fig图形窗口对象，可以用于设置图形窗口的属性
# ax 是坐标轴对象，用于绘制具体的图形和设置坐标轴的属性
# figsize=(4, 4) 表示图形窗口的宽度为4英寸，高度为4英寸

y_array = x_array # 一次函数 y = x
plt.plot(x_array, y_array)
beautify_line_chart() # 调用自定义函数绘制美化的线图

# 绘制抛物线
fig, ax = plt.subplots(figsize = (4,4))
y_array = x_array**2 - 2 # 二次函数
plt.plot(x_array, y_array)
beautify_line_chart() # 调用自定义函数绘制美化的线图
```

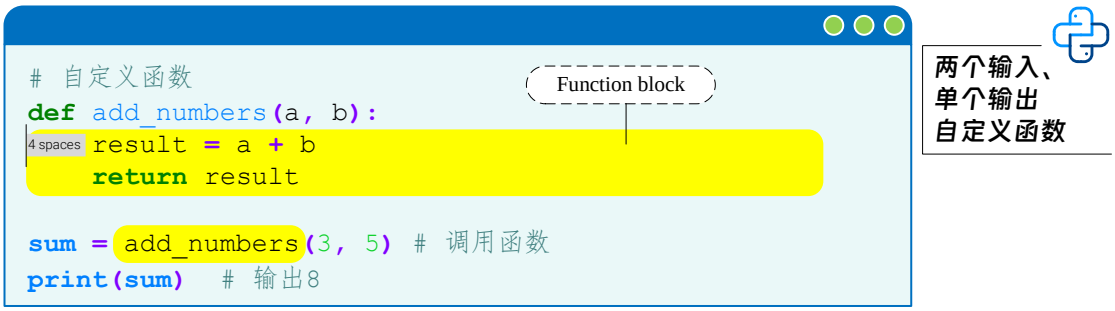
Function block

4 spaces

图 4. 无输入、无输出函数，装饰线图

多个输入、单一返回

一个函数可以有多个输入参数，一个或多个返回值。下面是一个示例函数，它有两个输入参数 a 和 b，返回它们的和。

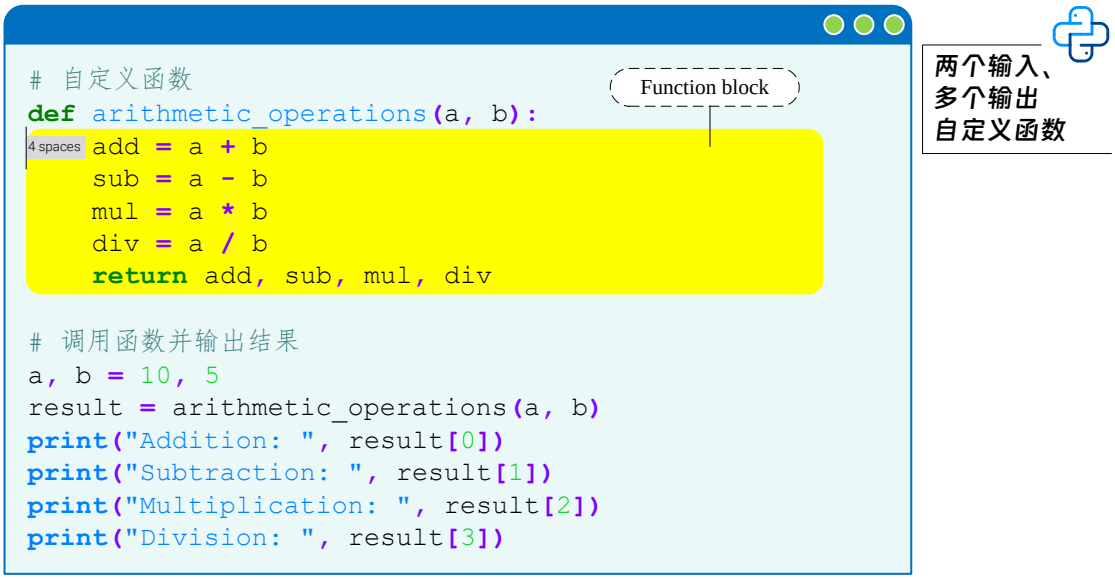



```python
# 自定义函数
def add_numbers(a, b):
    result = a + b
    return result

sum = add_numbers(3, 5) # 调用函数
print(sum)  # 输出8
```

图 5. 两个输入、一个输出函数

下面这个例子中，我们定义了一个名为 arithmetic_operations()的函数，它有两个参数 a 和 b。在函数体内，我们进行了四个基本的算术运算，并将其结果存储在四个变量中。最后，我们使用 return 语句返回这四个变量。当我们调用这个函数时，我们将 a 和 b 的值作为参数传递给函数，函数将返回四个值。我们将这四个返回值存储在一个元组 result 中，并使用索引访问和打印这四个值。




```python
# 自定义函数
def arithmetic_operations(a, b):
    add = a + b
    sub = a - b
    mul = a * b
    div = a / b
    return add, sub, mul, div

# 调用函数并输出结果
a, b = 10, 5
result = arithmetic_operations(a, b)
print("Addition: ", result[0])
print("Subtraction: ", result[1])
print("Multiplication: ", result[2])
print("Division: ", result[3])
```

图 6. 两个输入、多个输出函数

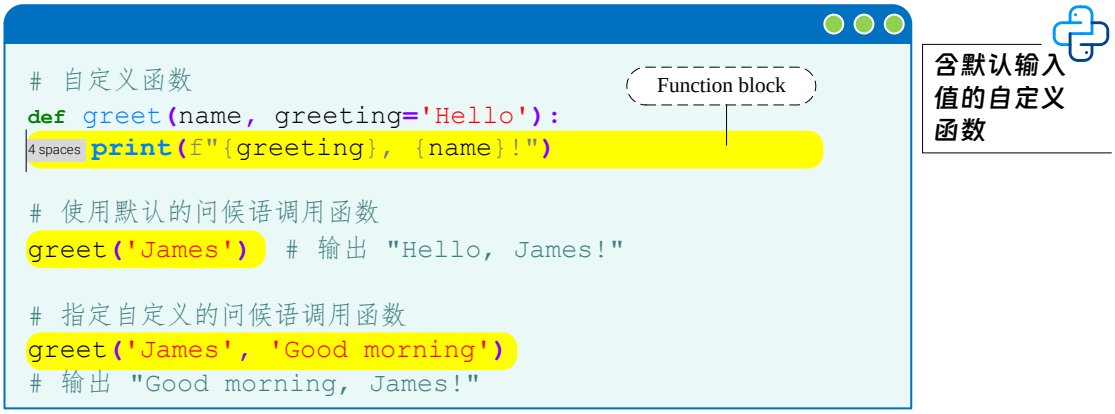
部分输入有默认值

在 Python 中，我们可以为自定义函数中的某些参数设置默认值，这样在调用函数时，如果不指定这些参数的值，就会使用默认值。这种设置默认值的参数称为默认参数。

下面是一个例子，展示如何在自定义函数中设置默认参数。greet()函数有两个参数：name 和 greeting。name 是必需的参数，没有默认值。而 greeting 是可选的，默认值为'Hello'。

当我们调用 greet()函数时，如果只传入了 name 参数，那么 greeting 就会使用默认值'Hello'。如果需要自定义问候语，可以在调用时传入自定义的值，如上面的第二个调用例子所示。

需要注意的是，默认参数必须放在非默认参数的后面。在函数定义中，先定义的参数必须先被传入，后定义的参数后被传入。如果违反了这个顺序，Python 解释器就会抛出 SyntaxError 异常。




```
# 自定义函数
def greet(name, greeting='Hello'):
    print(f"{greeting}, {name}!")

# 使用默认的问候语调用函数
greet('James')  # 输出 "Hello, James!"

# 指定自定义的问候语调用函数
greet('James', 'Good morning')
# 输出 "Good morning, James!"
```

图 7. 函数输入有默认值

将矩阵乘法打包成一个函数

上一章中，我们自定义了计算矩阵乘法代码。为了方便“多次调取”，下面我们将这段代码写成一个自定义函数。改良版的自定义函数，根据输入函数的形状，自行判断矩阵乘法结果矩阵的形状。

矩阵乘法

```
# 自定义函数
def matrix_multiplication(A,B):
    # 定义全 0 矩阵 C 用来存放结果
    C = [[0] * len(B[0]) for i in range(len(A))]

    # 遍历 A 的行
    for i in range(len(A)): # len(A) 给出 A 的行数

        # 遍历 B 的列
        for j in range(len(B[0])):
        # len(B[0]) 给出 B 的列数

            # 这一层相当于消去 p 所在的维度，即压缩
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
                # 完成对应元素相乘，再求和

    return C

# 定义矩阵 A 和 B
A = [[1, 2, 10, 20],
     [3, 4, 30, 40]]
B = [[4, 2],
     [3, 1],
     [40, 20],
     [30, 10]]

C = matrix_multiplication(A,B) # 调用自定义函数

print('A @ B = ')
# 输出结果
for row in C:
    print(row)

# 定义矩阵 X 和 Y
X = [[1], [2], [3]]
Y = [[1, 2, 3]]

print('X @ Y = ')
Z = matrix_multiplication(X,Y) # 调用自定义函数
# 输出结果
for row in Z:
    print(row)
```

Function block

4 spaces

图 8. 将矩阵乘法打包成一个函数

大家可能会问怎么在自定义函数内添加一个判断语句来检查两个矩阵的尺寸是否匹配。如果不匹配，就抛出一个异常并提示错误信息。以下是修改后的代码示例。在函数中，我们使用 len(A[0]) 和 len(B) 来检查第一个矩阵的列数是否等于第二个矩阵的行数。如果不相等，我们就使用 raise 语句抛出一个 ValueError 异常，并输出错误信息。这样，在调用函数时，如果输入的两个矩阵无法相乘，就会得到一个错误提示。

矩阵乘法，检测矩阵形状

```python
# 自定义函数
def matrix_multiplication(A,B):

    # 检查两个矩阵形状是否匹配
    if len(A[0]) != len(B):
        raise ValueError("Error: check matrix sizes")

    else:
        # 定义全 0 矩阵 C 用来存放结果
        C = [[0] * len(B[0]) for i in range(len(A))]

        # 遍历 A 的行
        for i in range(len(A)): # len(A) 给出 A 的行数

            # 遍历 B 的列
            for j in range(len(B[0])):
            # len(B[0]) 给出 B 的列数

                # 这一层相当于消去 p 所在的维度，即压缩
                for k in range(len(B)):
                    C[i][j] += A[i][k] * B[k][j]
                    # 完成对应元素相乘，再求和

        return C

# 定义矩阵 A 和 B
A = [[1, 2, 10, 20],
     [3, 4, 30, 40]]

B = [[4, 2],
     [3, 1]]

# 调用自定义函数
C = matrix_multiplication(A,B) # 报错
```

图 9. 将矩阵乘法打包成一个函数，增加矩阵形状不匹配的报错信息

帮助文档

在 Python 中，可以使用 docstring 来编写函数的帮助文档，即在函数定义的第一行或第二行写入字符串来描述函数的作用、参数、返回值等信息。通常使用三个单引号（'''）或三个双引号（"""）来表示 docstring，如下所示。如果要查询这个文档，可以使用 Python 内置的 help() 函数或者 __doc__ 属性来查看。

帮助文档

```
# 计算向量内积
def inner_prod(a,b):

    '''
    自定义函数计算两个向量内积
    输入:
    a: 向量，类型为数据列表
    b: 向量，类型为数据列表
    输出:
    c: 标量
    参考:
    https://mathworld.wolfram.com/InnerProduct.html
    '''

    # 检查两个向量元素数量是否相同
    if len(a) != len(b):
        raise ValueError("Error: check a/b lengths")

    # 初始化内积为0
    dot_product = 0
    # 使用for循环计算内积
    for i in range(len(a)):
        dot_product += a[i] * b[i]

    return dot_product

# 查询自定义函数文档，两种办法
help(inner_prod)
print(inner_prod.__doc__)

# 定义向量a和b
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
b = a[::-1]

# 调用函数
c = inner_prod(a,b)

# 打印内积
print("向量内积为: ", c)
```

图 10. 帮助文档

8.3 匿名函数

在 Python 中，匿名函数也被称为 lambda 函数，是一种快速定义单行函数的方式。使用 lambda 函数可以避免为简单的操作编写大量的代码，而且可以作为其他函数的参数来使用。

匿名函数的语法格式为：lambda arguments: expression。其中，arguments 是参数列表，expression 是一个表达式。当匿名函数被调用时，它将返回 expression 的结果。

下面是一些使用匿名函数的例子。

```
my_list = [1, 2, 3, 4, 5]

# 将列表中的所有元素加 1
list_plus_1 = list(map(lambda x: x+1, my_list))
print(list_plus_1)
# [2, 3, 4, 5, 6]

# 将列表中的所有元素分别求平方
list_squared = list(map(lambda x: x**2, my_list))
print(list_squared)
# [1, 4, 9, 16, 25]
```

lambda函数

图 11. lambda 函数

在这个例子中，我们定义了一个匿名函数 lambda x: x + 1，该函数接受一个参数 x 并返回 x 加 1。然后我们使用 map()将这个函数应用于列表 my_list 中的每个元素，并将结果存储在 list_plus_1 列表中。类似地，我们还计算了 my_list 中的每个元素的平方。

在 Python 中，map()是一种内置的高阶函数，它接受一个函数和一个可迭代对象作为输入，将函数应用于可迭代对象的每个元素并返回一个可迭代对象，其中每个元素都是应用于原始可迭代对象的函数的结果。

8.4 构造模块、库

简单来说，若干函数可以打包成一个模块，几个模块可以打包成一个库。本节简单聊一聊如何创建模块、创建库，对于大部分读者来说这一节可以跳过不读。

自定义模块

在 Python 中，我们可以将几个相关的函数放在一个文件中，这个文件就成为一个模块。下面是一个例子。假设我们有两个函数，一个是计算圆的面积，一个是计算圆的周长，我们可以将这两个函数放在一个文件中，例如我们可以创建一个名为 "circle.py" 的文件，并将以下代码添加到该文件中。我们首先导入了 math 模块，然后定义了两个函数 area()和 circumference()，分别用于计算圆的面积和周长。

```python
import math

def area(radius):
    return math.pi * radius**2

def circumference(radius):
    return 2 * math.pi * radius

# 将其存为文件circle.py
```

构造模块

图 12. 构造模块 circle.py

在本章配套的代码中，我们调用了 circle.py。使用 import 语句导入了 circle 模块，并命名为 cc，然后通过 cc.area()、cc.circumference() 调用函数。

自定义库

在 Python 中，可以使用 setuptools 库中的 setup()函数将多个模块打包成一个库。本章配套代码中给出的例子对应的具体步骤如下：

创建一个文件夹，用于存放库的代码文件，例如命名为 mylibrary。

在 mylibrary 文件夹中创建一个名为 setup.py 的文件，引入 setuptools 库，并使用 setup()函数来描述库的信息，包括名称、版本、作者、依赖、模块文件等信息。

在 mylibrary 文件夹中创建一个名为__init__.py 的空文件 (内容空白)，用于声明这个文件夹是一个 Python 包。

在 mylibrary 文件夹中创建多个模块文件，这些模块文件包含需要打包的函数或类。比如，mylibrary 中含有 linear_alg.py 和 circle.py 两个模块。linear_alg.py 有矩阵乘法、向量内积两个函数。circle.py 有计算圆面积、周长两个函数。

本章配套的代码中给出如何调用自定义库。

请大家完成下面 1 道题目。

Q1. 本章的唯一的题目就是请大家在 JupyterLab 中练习本章正文给出的示例代码。

* 不提供答案。

10

Fundamentals of Visualization

聊聊可视化

主要了解 Matplotlib、Plotly 两个工具

你能想象的所有东西都是真的。

Everything you can imagine is real.

—— 毕加索 (Pablo Picasso) | 西班牙艺术家 | 1881 ~ 1973

10.1 解剖一幅图

本章和接下来两章介绍如何实现鸢尾花书中最常见的可视化方案。这三章内容本着“够《编程不难》用就好”为原则，不会特别深究某个具体可视化方案中的呈现细节，也不会探究其他高阶的可视化方案。

鸢尾花书《可视之美》专注提供可视化的“家常菜菜谱”。

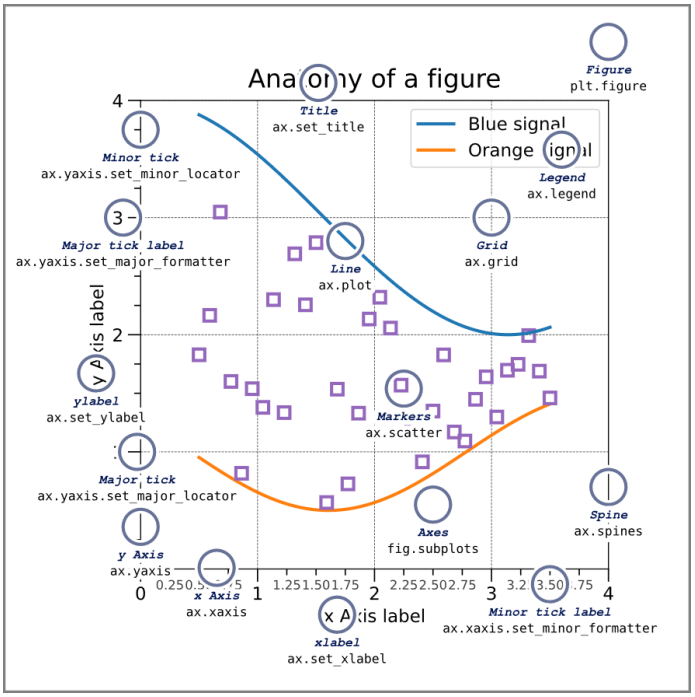


图 1. 解剖一幅图，来源 https://matplotlib.org/stable/gallery/showcase/anatomy.html

如图 1 所示，一幅图的基本构成部分包括以下几个部分：

- **图像区域** (Figure)：整个绘图区域的边界框，可以包含一个或多个子图。
- **子图区域** (Axes)：实际绘图区域，包含坐标轴、绘制的图像和文本标签等。
- **坐标轴** (Axis)：显示子图数据范围并提供刻度标记和标签的对象。
- **脊柱** (Spine)：连接坐标轴和图像区域的线条，通常包括上下左右四条。
- **标题** (Title)：描述整个图像内容的文本标签，通常位于图像的中心位置或上方，用于简要概括图像的主题或内容。
- **刻度** (Tick)：刻度标记，表示坐标轴上的数据值。
- **标签** (Label)：用于描述坐标轴或图像的文本标签。
- **图例** (Legend)：标识不同数据系列的图例，通常用于区分不同数据系列或数据类型。

▶ **艺术家** (Artist)：在 Matplotlib 中，所有绘图元素都被视为艺术家对象，包括图像区域、子图区域、坐标轴、刻度、标签、图例等等。

可视化工具

图 1 这幅图是用 Matplotlib 库绘制。Matplotlib 是 Python 中最基础的绘图工具。鸢尾花书中最常用的绘图库包括：Matplotlib、Seaborn、Plotly。

Matplotlib 可能是 Python 中最常用的绘图库，Matplotlib 具有丰富的绘图功能和灵活的使用方式。Matplotlib 可以绘制多种类型的图形，包括折线图、散点图、柱状图、饼图、等高线图等各种二维、三维图像，还可以进行图像处理和动画制作等。图 17、图 18、图 19 给出 Matplotlib 中常见的可视化方案。

Seaborn 是基于 Matplotlib 的高级绘图库，专注于统计数据可视化。它提供了多种高级数据可视化技术，包括分类散点图、热图 (热力图)、箱线图、分布图等，可以快速生成高质量的统计图表。Seaborn 适用于数据分析、数据挖掘和机器学习等领域。

⚠ 注意，Matplotlib 和 Seaborn 生成的都是静态图，即图片。

Plotly 是一个交互式可视化库，可以生成高质量的静态和动态图表。它提供了丰富的图形类型和交互式控件，可以通过滑块、下拉列表、按钮等方式动态控制图形的显示内容和样式。Plotly 适用于 Web 应用、数据仪表盘和数据科学教育等领域。类似 Plotly 的 Python 库还有 Bokeh、Altair、Pygal 等。

鸢尾花书中，大家会发现 PDF 书稿、纸质书图片一般会使用 Matplotlib、Seaborn 生成的矢量图，配套的 JupyterLab Notebook、Streamlit 则倾向于采用 Plotly。

本书第六大板块“数据”会介绍 Pandas 本身、Seaborn 的统计描述可视化方案。

10.2 使用 Matplotlib 绘制线图

下面我们聊一下如何用 Matplotlib 可视化正弦、余弦函数，图 2 所示代码生成图 3。下面我们逐块讲解这段代码；此外，请大家在 JupyterLab 中复刻这段代码，并绘制图 3。

大家会在鸢尾花书中发现，我们用 Python 代码生成的图像和书中的图像很多细节上并不一致。产生这种偏差的原因有很多。

首先，为了保证矢量图像质量及可编辑性，每幅 Python 代码生成的图形都会经过多道后期处理。后期处理的工具包括 (但不限于) Inkscape、MS Visio、Adobe Illustrator。使用怎样的工具要根据图片类型、图片大小等因素考虑。

也就是说哪怕图 2 这种简单的线图中的所有“艺术家 (artist)”，即所有元素，都被加工过。比如，图中的数字、英文、希腊字母都是手动添加上去的 (为了保证文本可编辑)。此外，从时间角度来看，一些标注、艺术效果用 Python 写代码方生成并不“划算”。

但是，加工过程仅仅是为了美化图像，并没有篡改数据本身。不篡改数据是一条铁律，希望大家谨记。

matplotlib 绘制线图

```
# 导入包
import numpy as np
import matplotlib.pyplot as plt

# 生成横轴数据
x_array = np.linspace(0, 2*np.pi, 100)
# 正弦函数数据
sin_y = np.sin(x_array)
# 余弦函数数据
cos_y = np.cos(x_array)

# 设置图片大小
fig, ax = plt.subplots(figsize=(8, 6))

# 绘制正弦和余弦曲线
ax.plot(x_array, sin_y,
        label='sin', color='blue', linewidth=2)
ax.plot(x_array, cos_y,
        label='cos', color='red', linewidth=2)

# 设置标题、横轴和纵轴标签
ax.set_title('Sine and cosine functions')
ax.set_xlabel('x')
ax.set_ylabel('f(x)')

# 添加图例
ax.legend()

# 设置横轴和纵轴范围
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1.5, 1.5)

# 设置横轴标签和刻度标签
x_ticks = np.arange(0, 2*np.pi+np.pi/2, np.pi/2)
x_ticklabels = [r'$0$', r'$\frac{\pi}{2}$',
                r'$\pi$', r'$\frac{3\pi}{2}$',
                r'$2\pi$']
ax.set_xticks(x_ticks)
ax.set_xticklabels(x_ticklabels)

# 横纵轴采用相同的scale
ax.set_aspect('equal')

# 将图片存成SVG格式
plt.savefig('正弦_余弦函数曲线.svg', format='svg')

# 显示图形
plt.show()
```

图 2. 用 Matplotlib 绘制正弦、余弦线图

Inkscape 是开源免费的矢量图形编辑软件，支持多种矢量图形格式，适用于绘制矢量图形、图标、插图等。MS Visio 特别适合做示意图、流程图等矢量图像。Adobe Illustrator 是 Adobe 公司

开发的专业矢量图形编辑软件，功能强大，广泛用于图形设计、插图、标志设计等。比如鸢尾花书的封面都是用 Adobe Illustrator 设计，鸢尾花书中复杂的图像也都是在这个软件设计生成。此外，也推荐大家使用 CorelDRAW。CorelDRAW 是 Corel 公司开发的矢量图形编辑软件，具有类似于 Adobe Illustrator 的功能，是一种流行的矢量图形处理工具。

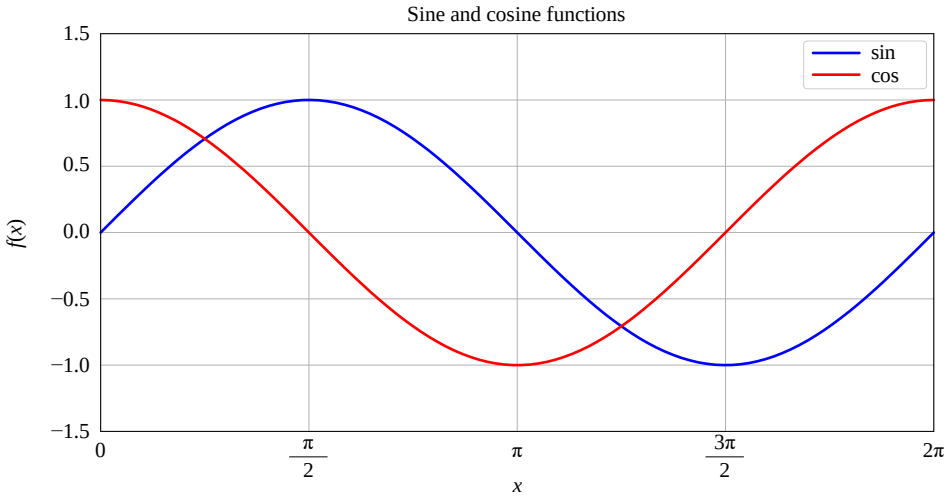


图 3. 正弦、余弦函数线图

产生等差数列

`import numpy as np` 这句代码的意思是将 NumPy (Python 代码中叫 `numpy`) 库导入到当前的 Python 程序中，并为其取一个简短的别名 np。

这意味着我们可以使用 `np` 来代替 `numpy` 来调用 NumPy 库中的函数和方法，例如 `np.linspace()`，`np.sin()`，`np.cos()` 等。这样做的好处是可以简化代码，减少打字量，并且提高代码的可读性。通常，人们将 `numpy` 取别名为 `np`，这是因为它的缩写简短且容易记忆。

`numpy.linspace()` 是 NumPy 库中的一个函数，用于生成在给定范围内等差数列。由于在导入 `numpy` 时，我们将其命名为 `np`，因此代码中大家看到的是 `np.linspace()`。

图 4. 用 numpy.linspace() 生成等差数列

上面的代码中，`0` 是数值序列的起始值，`2*np.pi` 是数值序列的结束值，`100` 是数值序列的数量。因此，`x_array = np.linspace(0, 2*np.pi, 100)` 在 [0, 2π] 闭区间内生成一个 100 个数值等差数列。

numpy.linspace(start, stop, num=50, endpoint=True)

这个函数的重要输入参数：

- start：起始点的值。
- stop：结束点的值。
- num：要生成的数据点数量，默认为 50。
- endpoint：布尔值，指定是否包含结束点。如果为 True，则生成的数据点包括结束点；如果为 False，则生成的数据点不包括结束点。默认为 True。

请大家在 JupyterLab 中自行学习下例。

```
import numpy as np

arr = np.linspace(0, 1, num=11)
print(arr)

arr_no_endpoint = np.linspace(0, 1, num=10, endpoint=False)
print(arr_no_endpoint)
```

什么是 NumPy 数组 array?

NumPy 中最重要的数据结构是 ndarray (n-dimensional array)，即多维数组。一维数组是最简单的数组形式，类似于 Python 中的列表。它是一个有序的元素集合，可以通过索引访问其中的元素。一维数组只有一个轴。二维数组是最常见的数组形式，可以看作是由一维数组组成的表格或矩阵。它有两个轴，通常称为行和列。我们可以使用两个索引来访问二维数组中的元素。多维数组是指具有三个或更多维度的数组。

正弦、余弦

如图 5 所示， numpy.sin() 和 numpy.cos() 是 NumPy 库中的数学函数，用于计算给定角度的正弦和余弦值。这两个函数的输入既可以是弧度值 (比如 numpy.pi/2)，也可以是数组 (一维、二维、多维)。

⚠ 注意，NumPy 中 numpy.deg2rad()将角度转换为弧度，numpy.rad2deg()将弧度转换为角度。

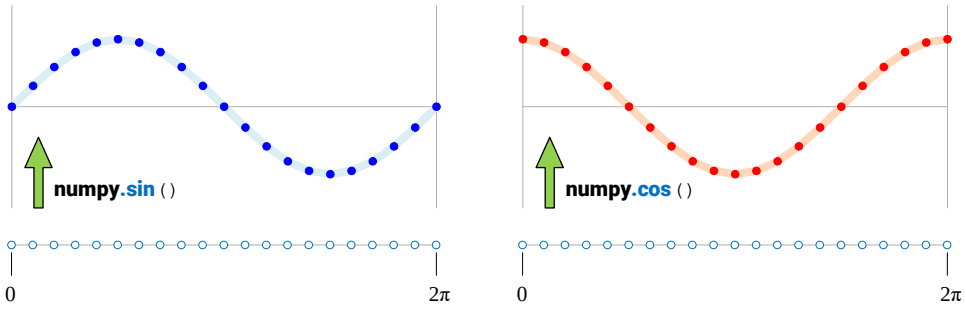


图 5. 生成正弦、余弦数据

创建图形、轴对象

`fig, ax = plt.subplots(figsize=(8, 6))` 用于创建一个新的 Matplotlib 图形 `fig` 和一个轴 `ax` 对象，并设置图形的大小为 (8, 6)，单位为英寸。

通过创建图形和轴对象，我们可以在轴上绘制图表、设置轴的标签和标题、调整轴的范围等。`fig, ax = plt.subplots()` 这一句代码常常是开始绘图的第一步，它创建了一个具有指定大小的图形和轴对象，为后续绘图操作提供了一个可用的基础。

需要注意的是，`plt` 是 Matplotlib 的一个常用的别名，通常通过 `import matplotlib.pyplot as plt` 引入。所以在使用 `plt.subplots()` 函数之前，需要确保已经正确导入了 Matplotlib 库。

添加子图

此外，我们还可以使用使用 `add_subplot()` 方法创建一个新的子图对象，并指定其所在的行、列、编号等属性。

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

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
plt.show()
```

add_subplot() 创建新的子图对象

图 6. add_subplot() 方法创建一个新的子图对象

在这个例子中，我们使用 add_subplot() 方法创建了一个新的子图对象，并将其添加到 Figure 对象中。其中，1, 1, 1 参数表示子图在 1 行 1 列的第 1 个位置，即占据整个 Figure 对象的空间。然后，我们在子图中绘制了一个正弦曲线。最后，使用 plt.show()函数显示 Figure 对象，即可在屏幕上显示绘制的图像。

绘制曲线

`ax.plot(x_array, sin_y, label='sin', color='blue', linewidth=2)` 用于在轴对象 ax 上绘制正弦曲线。x_array 为 x 轴数据，sin_y 为 y 轴数据。

`label='sin'` 设置了曲线的标签为 `'sin'`，`color='blue'` 设置曲线的颜色为蓝色，`linewidth=2` 设置曲线的线宽为 2。在 Matplotlib 中，linewidth 参数表示线条的宽度。它的单位是点 (point, pt)，通常用于测量线条、字体等绘图元素的大小。在 Matplotlib 中，默认情况下，一个点等于 1/72 inch。

颜色

在 Matplotlib 中，可以使用多种方式指定线图的颜色，包括 RGB 值、预定义颜色名称、十六进制颜色码和灰度值。

可以使用 RGB (R 是 red，G 是 green，B 是 blue) 来指定颜色，其中每个元素的值介于 0 到 1 之间。例如，(1, 0, 0) 表示纯红色，(0, 1, 0) 表示纯绿色。使用 RGBA 值指定“颜色 + 透明度 (A)”。

如图 8 所示，RGB 三原色模型实际上构成了一个色彩“立方体”——一个色彩空间。

鸢尾花书《矩阵力量》将会用 RGB 三原色模型讲解线性代数中向量空间 (vector space) 这个重要概念。

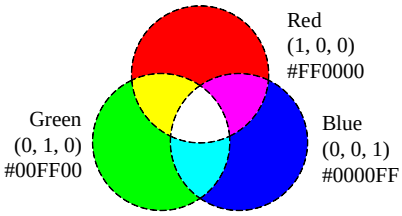


图 7. RGB 三原色模型

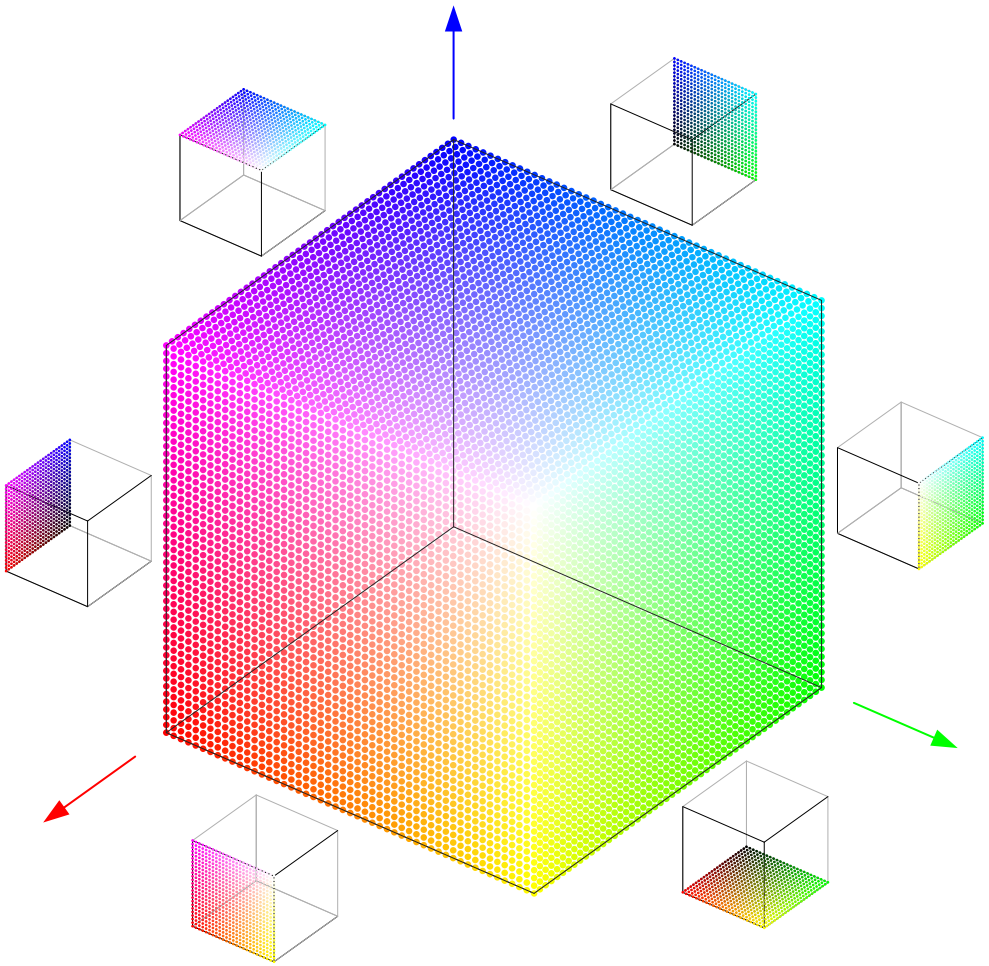


图 8. RGB 三原色模型“立方体”

什么是 RGB 颜色模式？

RGB (红绿蓝) 颜色模式是一种使用红、绿、蓝三个基本颜色通道来表示颜色的方法。在 RGB 模式中，通过调整每个通道的强度 (从 0 到 255 的值，Matplotlib 中 0 到 1 的值) 来创建各种颜色。通过组合不同强度的红、绿和蓝，可以形成几乎所有可见光颜色。RGB 颜色模式被广泛应用于计算机图形、数字图像处理和网页设计等领域，它提供了一种直观、灵活且广泛支持的方式来表示和操作颜色。

Matplotlib 提供了一些常见颜色的预定义名称，例如 'red'、'green'、'blue' 等。图 16 所示为在 Matplotlib 中已经预定义名称的颜色。

大家还可以使用十六进制颜色码来指定颜色。它以 '#' 开头，后面跟着六位十六进制数。例如，'#FF0000' 表示纯红色，'#00FF00' 表示纯绿色。

我们还可以使用灰度值来指定颜色，取值介于 0 到 1 之间，表示不同的灰度级别。'0' 表示黑色，'1' 表示白色。比如，color='0.5' 代表灰度值为 0.5 的灰色。

使用色谱

Matplotlib 中还有一种渐变配色方案——colormap。在 Matplotlib 中，colormap 用于表示从一个端到另一个端的颜色变化。这个变化可以是连续的，也可以是离散的。

在 Matplotlib 中，colormap 主要用于绘制二维图形，如热图、散点图、等高线图等。它用于将数据值映射到不同的颜色，以显示数据的变化和模式。Colormap 可以直译为“色彩映射”，鸢尾花书一般称之为“色谱”。图 9 所示为几种常见的色谱。鸢尾花书中最常用的色谱为 RdYlBu。

《可视之美》将专门讲解色谱。

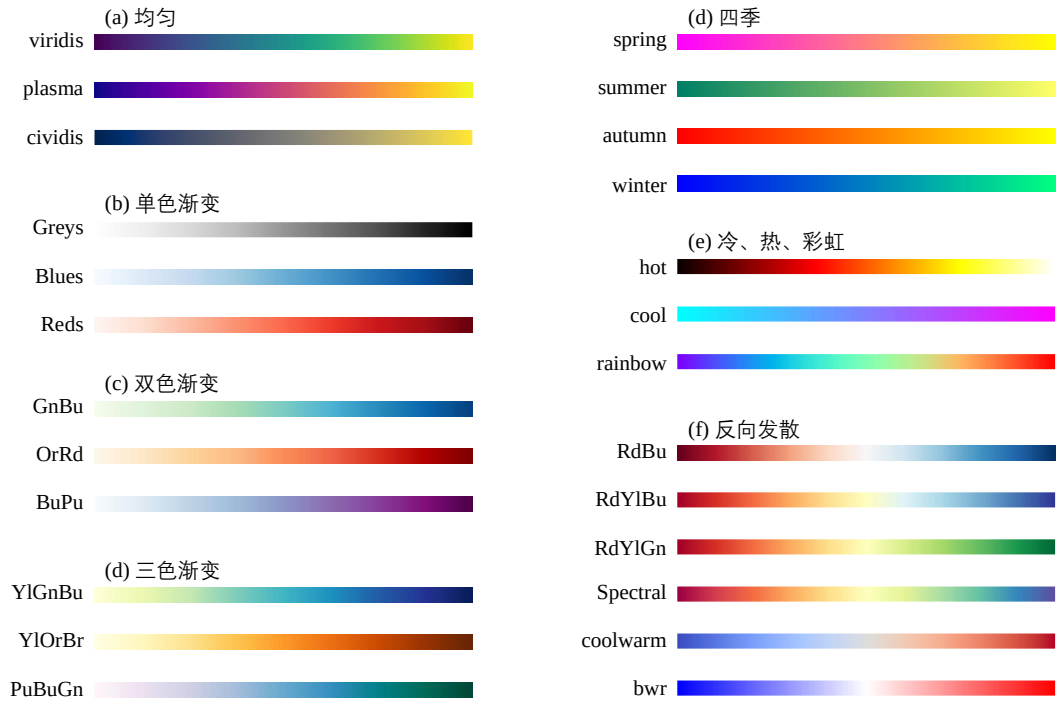


图 9. 几种常用色谱

图 10 所示为利用色谱渲染一组曲线。图 10 左图所示为一元高斯概率密度分布曲线随均值 μ 变化，图 10 右图所示为曲线随标准差 σ 变化。大家可以在本章配套 Jupyter Notebook 找到对应代码。

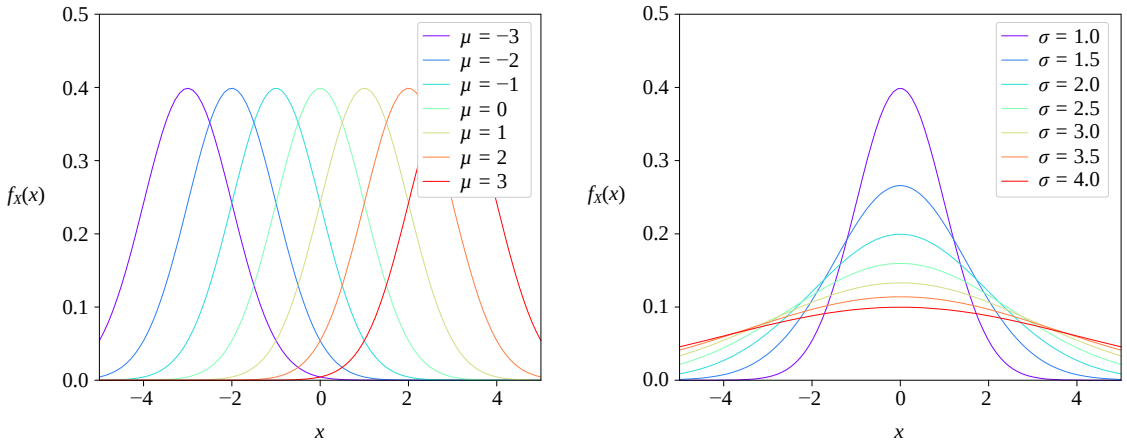


图 10. 用色谱渲染一组曲线

什么是高斯分布？

高斯分布 (Gaussian distribution)，也称为正态分布 (Normal distribution)，是统计学中常用的概率分布模型之一。它具有钟形曲线的形状，呈对称分布。高斯分布的概率密度函数可以由两个参数完全描述：均值 (mean) 和标准差 (standard deviation)。均值决定了分布的中心位置，标准差决定了分布的展开程度。

高斯分布在自然界和社会现象中广泛存在，例如身高、体重、温度等连续型随机变量常常服从高斯分布。中心极限定理也说明了许多独立同分布的随机变量的总和趋向于高斯分布。

高斯分布在统计学和数据分析中有着重要的应用，可用于描述数据集的分布特征、进行假设检验、构建回归模型等。在机器学习和人工智能领域，高斯分布在概率密度估计、聚类分析、异常检测等算法中被广泛使用。

什么是概率密度函数?

概率密度函数 (Probability Density Function，简称 PDF) 是概率论和统计学中用于描述连续型随机变量的概率分布的函数。它表示了变量落在某个特定取值范围内的概率密度，而不是具体的概率值。

一元连续随机变量的概率密度函数是非负函数，并且在整个定义域上的积分等于 1。对于给定的连续型随机变量，通过 PDF 可以计算出在不同取值范围内的概率密度值，从而了解变量的分布特征和概率分布形状。

以正态分布为例，其概率密度函数即高斯函数，可以描述变量取值的概率密度。在某个特定取值处，概率密度函数的值越高，表示该取值的概率越大。概率密度函数在统计分析、数据建模、概率推断等领域广泛应用，可用于计算概率、推断参数、生成模拟数据等。

其他细节美化

图 2 中还提供图片美化命令，下面逐一说明。

- `ax.set_title('Sine and cosine functions')` 设置图表的标题为 "Sine and cosine functions"，即正弦和余弦函数。
- `ax.set_xlabel('x')` 设置横轴标签为 "x"。`ax.set_ylabel('f(x)')` 设置纵轴标签为 "f(x)"。
- `ax.legend()` 添加图例 legend，用于标识不同曲线或数据系列。
- `ax.set_xlim(0, 2*np.pi)` 设置横轴范围从 0 到 2π。`ax.set_ylim(-1.5, 1.5)` 设置纵轴范围从 −1.5 到 1.5。
- `x_ticks = np.arange(0, 2*np.pi+np.pi/2, np.pi/2)` 生成横轴刻度的位置，从 0 到 2π，间隔为 π/2。
- `x_ticklabels = [r'$0$', r'$\frac{\pi}{2}$', r'$\pi$', r'$\frac{3\pi}{2}$', r'$2\pi$']` 设置横轴刻度的标签，分别为 0, π/2, π, 3π/2, 2π。在代码中，`r'$\frac{\pi}{2}$'` 是一个特殊的字符串，用于表示数学公式中的文本。在这个字符串前面的 `r` 前缀表示该字符串是一个“原始字符串”，即不对字符串中的特殊字符进行转义。
- 在这个特殊字符串中，使用了 LaTeX 符号来表示一个分数。具体来说，`\frac{\pi}{2}` 表示一个分数，分子是π，分母是 2。当这个字符串被用作横轴刻度的标签时，它会在图表中显示为 "π/2" 的形式。这种表示方法可以用于在图表中显示复杂的数学公式或符号。
- `ax.set_xticks(x_ticks)` 设置横轴刻度的位置。
- `ax.set_xticklabels(x_ticklabels)` 设置横轴刻度的标签。
- `ax.set_aspect('equal')` 设置横纵轴采用相同的比例，保持图形在绘制时不会因为坐标轴的比例问题而产生形变。

图片输出格式

Matplotlib 可以输出多种格式的图片，其中一些是矢量图。以下是一些常见的输出格式及其特点：

- PNG (Portable Network Graphics)：PNG 是一种常见的位图格式，支持透明度和压缩。PNG 格式输出的图片不是矢量图，因此在放大时会失去清晰度，但是可以保持较高的分辨率和细节。
- JPG/JPEG (Joint Photographic Experts Group)：JPG 是一种常见的有损压缩位图格式，用于存储照片和复杂的图像。与 PNG 不同，JPG 格式输出的图片是有损的，压缩率高时会失去一些细节，但是文件大小通常较小。
- EPS (Encapsulated PostScript)：EPS 是一种矢量图格式，可以在很多绘图软件中使用。EPS 格式输出的图片可以无限放大而不失真，适合于需要高品质图像的打印和出版工作。
- PDF (Portable Document Format)：PDF 是一种常见的文档格式，可以包含矢量图和位图。与 EPS 类似，PDF 格式输出的图片也是矢量图，可以无限放大而不失真，同时具有可编辑性和高度压缩的优势。存成 PDF 很方便插入 Latex 文档。
- SVG (Scalable Vector Graphics)：SVG 是一种基于 XML 的矢量图格式，可以用于网页和打印等多种用途。SVG 格式输出的图片可以无限放大而不失真，且文件大小通常较小。鸢尾花书的图片首选 SVG 格式。

注意，EPS、PDF 和 SVG 是矢量图格式，可以无限放大而不失真 (比如图 11 (b))，适合于需要高品质图像的打印和出版工作。在需要高品质图像的场合，最好使用这些矢量图格式。

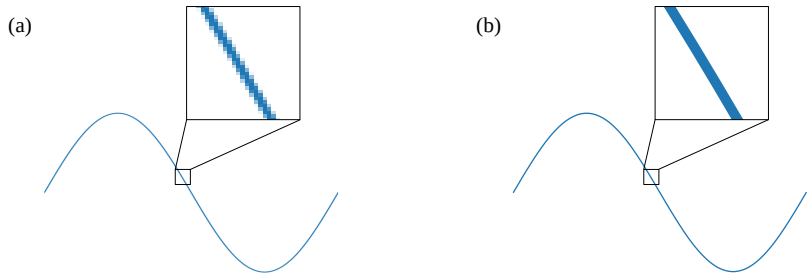


图 11. 比较非矢量、矢量图

子图

图 12 所示一行两列子图。请大家在 JupyterLab 中给图 13 代码逐行添加注释，并复刻图 12。

《可视之美》将介绍更多子图可视化方案。

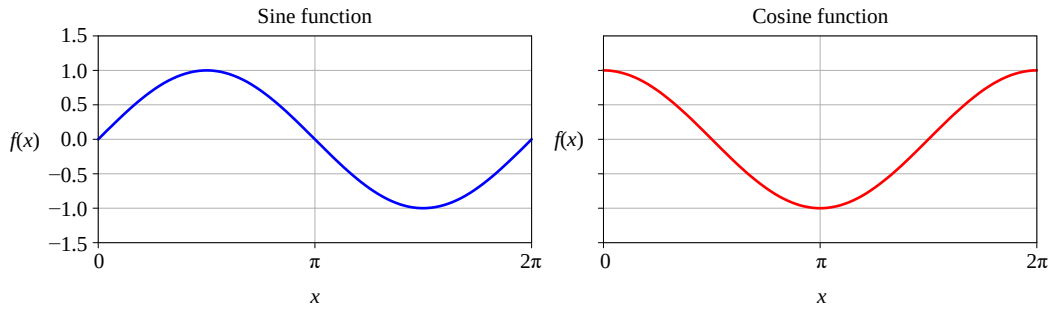


图 12. 一行、两列子图

subplots() 一行两列子图

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

x = np.linspace(0, 2 * np.pi, 100)
y_sin = np.sin(x)
y_cos = np.cos(x)

# 创建图形对象和子图布局
fig, (ax1, ax2) = plt.subplots(1, 2,
                  figsize=(10, 4),
                  sharey=True)

# 在左子图中绘制正弦函数曲线，设置为蓝色
ax1.plot(x, y_sin, color='blue')
ax1.set_title('Sine function')
ax1.set_xlabel('x')
ax1.set_ylabel('f(x)',
               rotation='horizontal',
               ha='right')
ax1.set_xlim(0, 2*np.pi)
ax1.set_ylim(-1.5, 1.5)
x_ticks = np.arange(0, 2*np.pi+np.pi/2, np.pi)
x_ticklabels = [r'$0$', r'$\pi$', r'$2\pi$']
ax1.set_xticks(x_ticks)
ax1.set_xticklabels(x_ticklabels)
ax1.grid(True)
ax1.set_aspect('equal')

# 在右子图中绘制余弦函数曲线，设置为红色
ax2.plot(x, y_cos, color='red')
ax2.set_title('Cosine function')
ax2.set_xlabel('x')
ax2.set_ylabel('f(x)',
               rotation='horizontal',
               ha='right')
ax2.set_xlim(0, 2*np.pi)
ax2.set_ylim(-1.5, 1.5)
ax2.set_xticks(x_ticks)
ax2.set_xticklabels(x_ticklabels)
ax2.grid(True)
ax2.set_aspect('equal')

# 调整子图之间的间距
plt.tight_layout()

# 显示图形
plt.show()
```

图 13. 绘制一行两列子图

10.3 使用 Plotly 绘制线图

此外，我们还可以用 Plotly 绘制具有交互属性的图形，比如图 14，对应的代码如图 15。

`plotly.graph_objects` 是 Plotly 库中的一个模块，它提供了创建和操作图形对象的类和方法。通过 `go` 的别名，我们可以方便地使用 `plotly.graph_objects` 模块中的各种类和函数。在 `plotly.graph_objects` 模块中，有许多类可用于创建各种类型的图形，如 `scatter`、`bar`、`surface` 等。

通过 `go` 模块，我们可以创建一个 `Figure` 对象，用于容纳和管理我们的图形。Figure 对象是一个图形容器，可以添加多个轨迹 (trace)，设置整体布局和样式，并最终显示或保存图形。

`fig.add_trace(go.Scatter(x=x, y=y_sin, mode='lines', name='Sine'))` 的作用是向图形对象 fig 中添加一个轨迹，其中包含了一条正弦曲线的数据和样式。`go.Scatter` 创建了一个散点图 (scatter plot) 的轨迹对象。`x=x` 指定了横轴的数据，即之前生成的 x 值数组。`y=y_sin` 指定了纵轴的数据，即正弦函数的 `y` 值数组。`mode='lines'` 设置了散点图的显示模式为连线模式，表示将数据点用线连接起来。`name='Sine'` 设置了轨迹的名称为 `"Sine"`，在图例中显示。通过 `add_trace` 方法，我们将该轨迹添加到 fig 图形对象中，使得该正弦曲线在图形中显示出来。

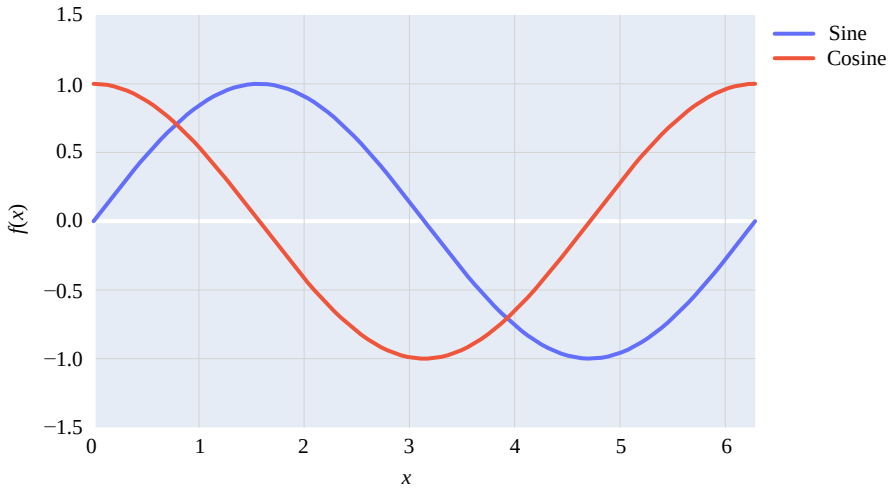


图 14. 用 Plotly 绘制具有交互性质的曲线

用Plotly绘制线图

```
import numpy as np
import plotly.graph_objects as go
import plotly.io as pio
pio.kaleido.scope.default_format = "svg"

# 生成 x 值
x = np.linspace(0, 2 * np.pi, 100)

# 生成正弦和余弦函数的 y 值
y_sin = np.sin(x)
y_cos = np.cos(x)

# 创建图形对象
fig = go.Figure()

# 添加正弦曲线
fig.add_trace(go.Scatter(x=x, y=y_sin, mode='lines',
name='Sine'))

# 添加余弦曲线
fig.add_trace(go.Scatter(x=x, y=y_cos, mode='lines',
name='Cosine'))

# 设置横轴和纵轴范围
fig.update_layout(xaxis_range=[0, 2 * np.pi],
                  yaxis_range=[-1.5, 1.5])

# 设置横轴和纵轴标签
fig.update_xaxes(title_text='x')
fig.update_yaxes(title_text='f(x)')

# 添加网格
fig.update_xaxes(showgrid=True, gridwidth=0.25,
gridcolor='lightgray')
fig.update_yaxes(showgrid=True, gridwidth=0.25,
gridcolor='lightgray')

# 显示图形
fig.show()

# 将fig保存为SVG格式
fig.write_image("fig.svg")
```

图 15. 用 Plotly 绘制线图

请大家完成下面 3 道题目。

Q1. 大家可以在本章配套代码中找到图 1 对应的 Matplotlib 官方提供的代码文件。本书将 Python 代码文件命名为 Q1_Assignment_Anatomy_of_a_figure.py。请大家给这个代码文件中的代码逐行中文注释，并在 JupyterLab 中进行探究式学习。

Q2. Matplotlib 提供丰富的可视化方案实例，图 17、图 18、图 19 大部分子图对应的代码都在如下链接中，请大家在 JupyterLab 复刻每幅子图，并补充必要注释。

https://matplotlib.org/stable/plot_types/index.html

* 本章习题不提供答案。

图 16. Matplotlib 已定义名称的颜色

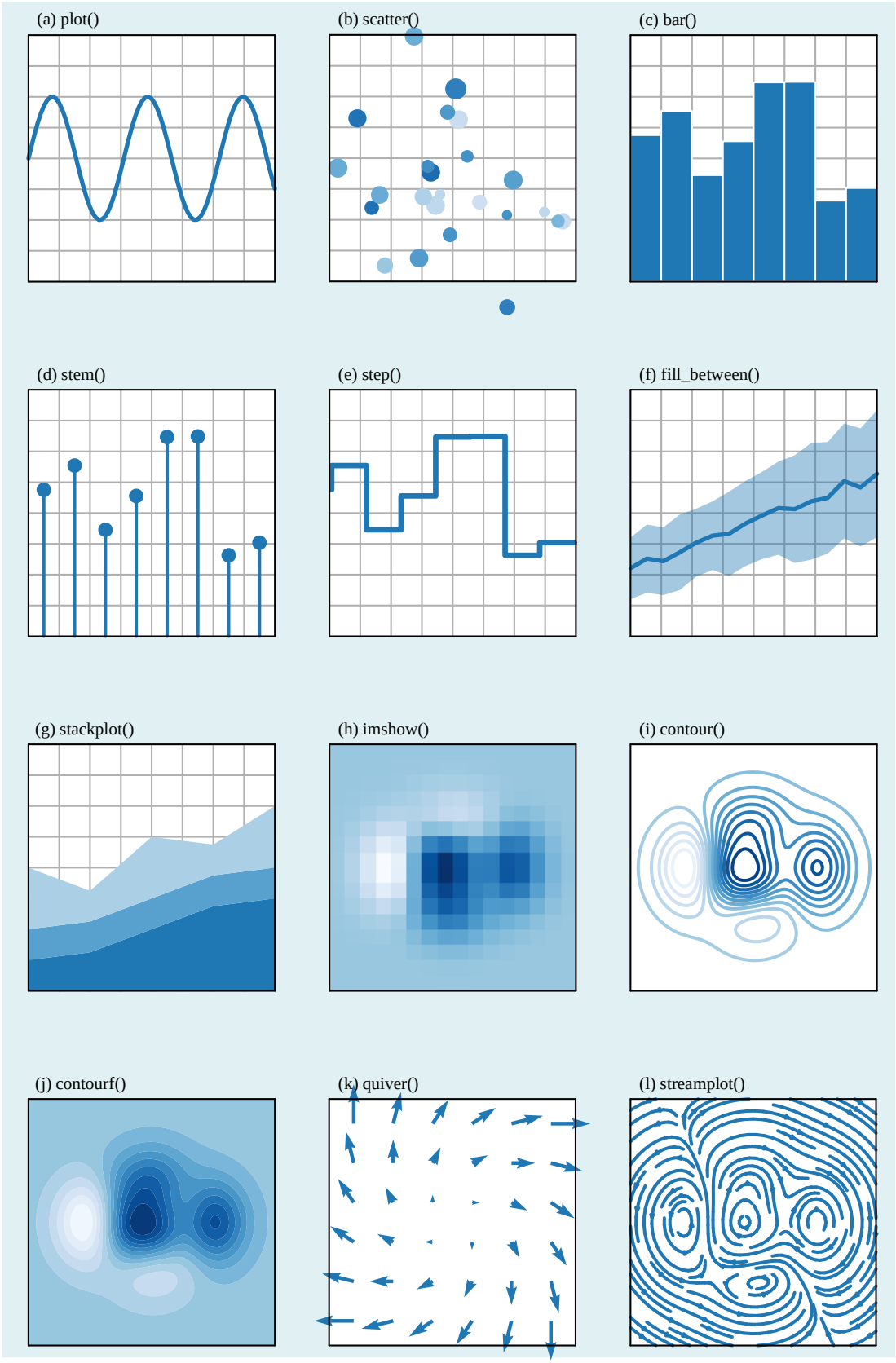


图 17. Matplotlib 常见可视化方案，第一组

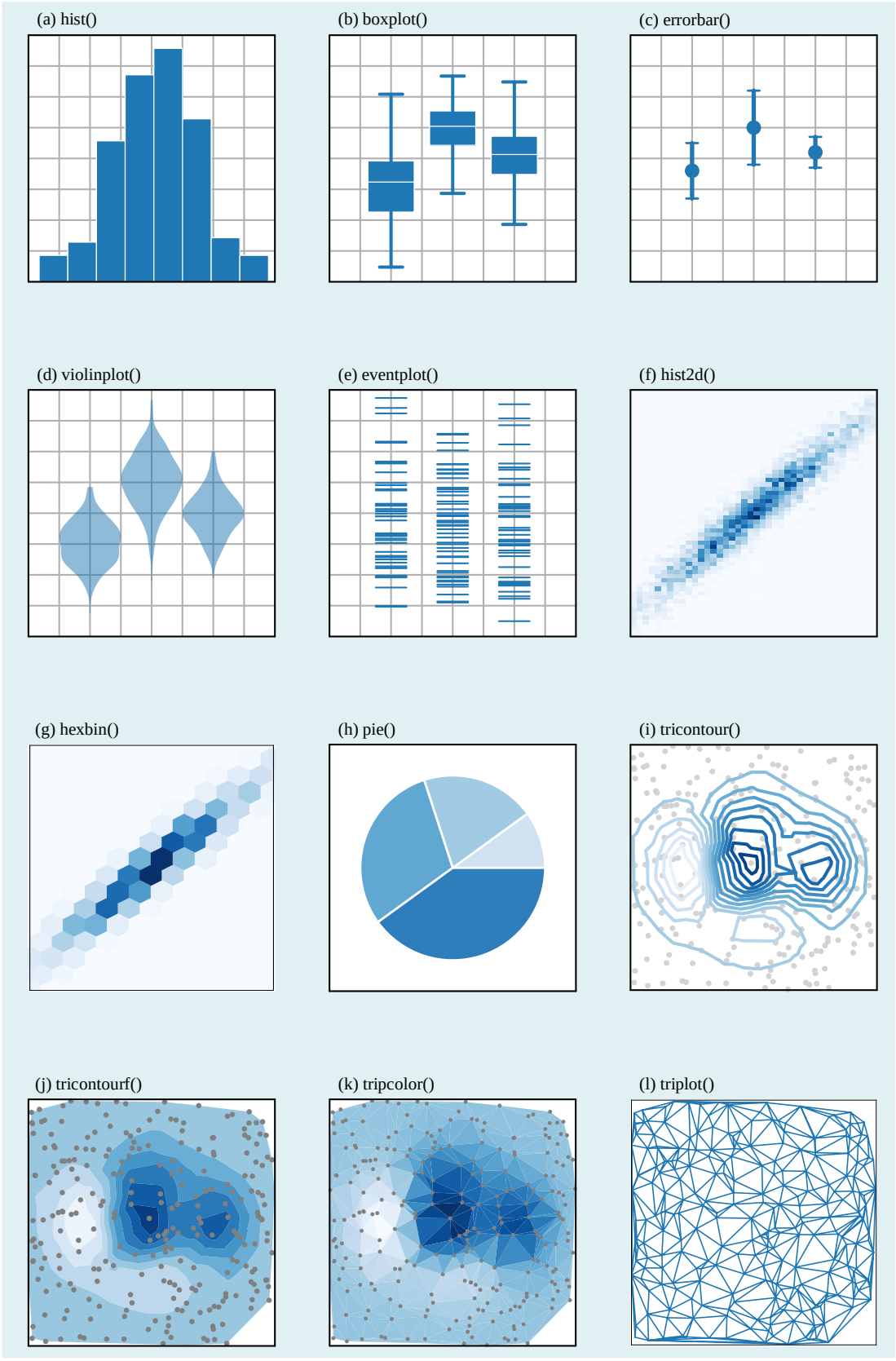


图 18. Matplotlib 常见可视化方案，第二组

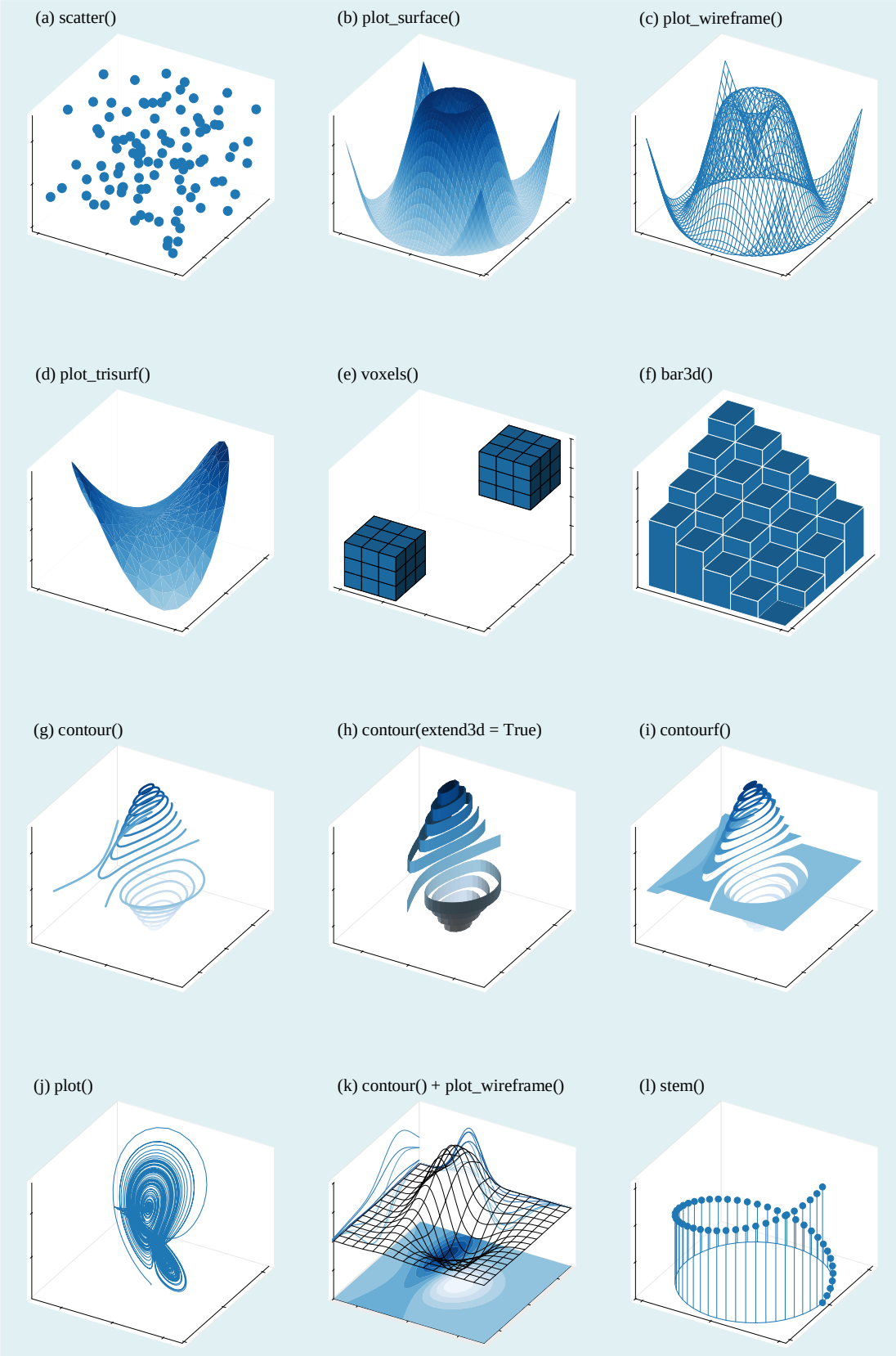


图 19. Matplotlib 常见可视化方案，第三组

11 2D Visualizations
二维可视化

平面上的散点图、等高线、热图

文明的传播像是星星之火可以燎原；首先是星星之火，然后是闪烁的炬火，最后是燎原烈焰，排山倒海、势不可挡。

The spread of civilization may be likened to a fire; first, a feeble spark, next a flickering flame, then a mighty blaze, ever increasing in speed and power.

—— 尼古拉·特斯拉 (Nikola Tesla) | 发明家、物理学家 | 1856 ~ 1943

11.1 二维可视化方案

本章将介绍鸢尾花书最常见的四类平面可视化方案：散点、线图、等高线、热图。

- 散点图 (scatter plot)：散点图用于展示两个变量之间的关系，其中每个点的位置表示两个变量的取值。可以通过设置点的颜色、大小、形状等属性来表示其他信息。
- 线图 (line plot)：线图用于展示数据随时间或其他变量而变化的趋势。线图由多个数据点连接而成，通常用于展示连续数据。
- 等高线图 (contour plot)：等高线图用于展示二维数据随着两个变量的变化而变化的趋势。每个数据点的值表示为等高线的高度，从而形成连续的轮廓线。
- 热图 (heatmap)：热图用于展示二维数据的值，其中每个值用颜色表示。热图常用于数据分析中，用于显示数据的热度、趋势等信息。建议使用 Seaborn 库绘制热图。

上一章，我们介绍了如何用 Matplotlib 和 Plotly 绘制线图，本章将主要介绍散点图、平面等高线、热图这三大类可视化方案。

11.2 散点

二维散点图是平面直角坐标系 (也叫笛卡尔坐标系) 中一种用于可视化二维数据分布的图形表示方法。它由一系列离散的数据点组成，其中每个数据点都由两个坐标值。

Matplotlib 中的 scatter() 函数可以用于创建散点图。可以使用 matplotlib.pyplot.scatter() 函数来指定数据点的坐标和其他绘图参数，例如颜色、大小等。请大家自行在 JupyterLab 中实践图 2 给出的代码。

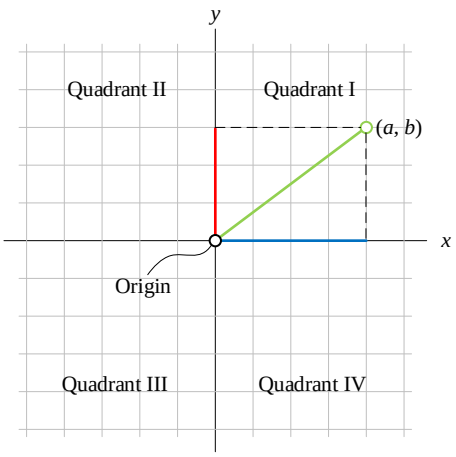


图 1. 笛卡儿坐标系

什么是平面直角坐标系？

平面直角坐标系，也称笛卡儿坐标系，是一种二维空间中的坐标系统，由两条相互垂直的直线组成。其中一条直线称为 x 轴，另一条直线称为 y 轴。它们的交点称为原点，通常用 O 表示。平面直角坐标系可以用来描述二维空间中点的位置，其中每个点都可以由一对有序实数 (a, b) 表示，分别表示点在 a 轴和 b 轴上的距离。x 轴和 y 轴的正方向可以是任意方向，通常 x 轴向右，y 轴向上。平面直角坐标系是解析几何中重要的工具，用于研究点、直线、曲线以及它们之间的关系和性质。

Matplotlib绘制散点图

```
# 导入包
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import numpy as np

# 加载鸢尾花数据集
iris = load_iris()

# 提取花萼长度和花萼宽度作为变量
sepal_length = iris.data[:, 0]
sepal_width = iris.data[:, 1]
target = iris.target

fig, ax = plt.subplots()

# 创建散点图
plt.scatter(sepal_length, sepal_width, c=target,
cmap='rainbow')

# 添加标题和轴标签
plt.title('Iris sepal length vs width')
plt.xlabel('Sepal length (cm)')
plt.ylabel('Sepal width (cm)')

# 设置横纵轴刻度
ax.set_xticks(np.arange(4, 8 + 1, step=1))
ax.set_yticks(np.arange(1, 5 + 1, step=1))

# 设定横纵轴尺度1:1
ax.axis('scaled')

# 增加刻度网格，颜色为浅灰
ax.grid(linestyle='--', linewidth=0.25,
color=[0.7,0.7,0.7])

# 设置横纵轴范围
ax.set_xbound(lower = 4, upper = 8)
ax.set_ybound(lower = 1, upper = 5)

# 显示图形
plt.show()
```

图 2. 用 Matplotlib 绘制散点图

特别推荐大家使用 seaborn.scatterplot() 函数来创建二维散点图，并传递数据点的坐标和其他可选参数。还可以使用 plotly.graph_objects.Scatter() 函数创建可交互的散点图，并指定数据点的坐标、样式等参数。本节下面利用 Seaborn 和 Plotly 这两个库中函数绘制散点图。

Seaborn

图 3 所示为利用 seaborn.scatterplot() 绘制鸢尾花数据集的散点图。这两幅散点图的横轴都是花萼长度，纵轴为花萼宽度。图 3 (b) 用颜色标识鸢尾花类别。

使用 seaborn.scatterplot() 函数的基本语法如下：

```
import seaborn as sns
sns.scatterplot(data=data_frame, x="x_variable", y="y_variable")
```

其中，`x_variable` 是数据集中表示 x 轴的变量列名，`y_variable` 是表示 y 轴的变量列名，`data_frame` 是包含要绘制的数据的 Pandas DataFrame 对象。

我们还可以指定 `hue` 参数，用于对数据点进行分组并在图中用不同颜色表示的列名，`size` 参数指定了数据点的大小根据 `value` 列的值进行缩放。除了 `hue` 和 `size`，还可以使用其他参数如 `style`、`palette`、`alpha` 等来进一步定制散点图的外观和风格。

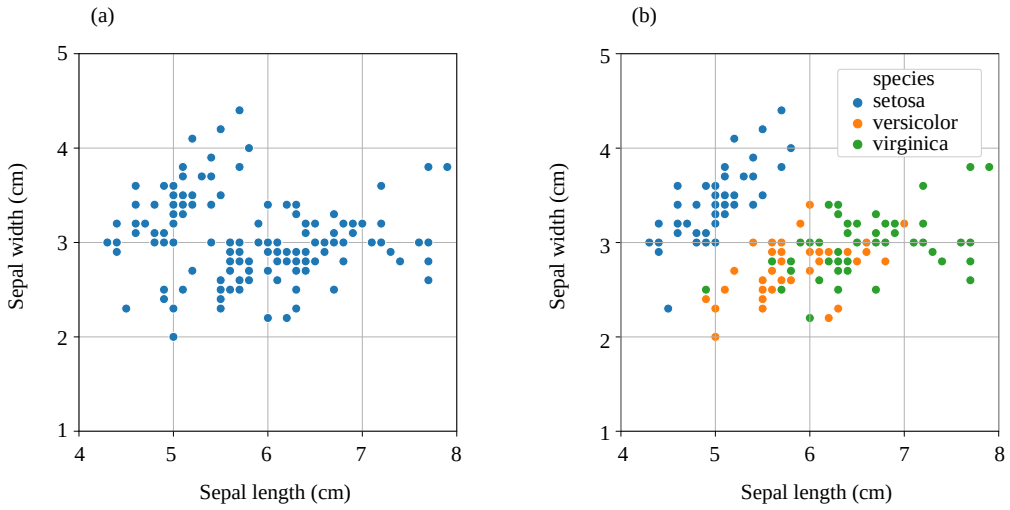


图 3. 使用 seaborn.scatterplot() 绘制鸢尾花数据集散点图

Plotly

图 4 所示为使用 `plotly.express.scatter()` 绘制鸢尾花数据集散点图。在本章配套的 Jupyter Notebook 中大家可以看到这两幅子图为可交互图像。

`plotly.express.scatter()` 用来可视化两个数值变量之间的关系，或者展示数据集中的模式和趋势。这个函数的基本语法如下：

```
import plotly.express as px
```

```
fig = px.scatter(data_frame, x="x_variable", y="y_variable")
fig.show()
```

其中，data_frame 是包含要绘制的数据的 Pandas DataFrame 对象，x_variable 是数据集中表示 *x* 轴的变量列名，y_variable 是表示 *y* 轴的变量列名。可以根据需要添加其他参数，例如 color、size、symbol 等，以进一步定制散点图的外观。

最后，通过 fig.show() 方法显示绘制好的散点图。

《可视之美》将专门讲解散点图。

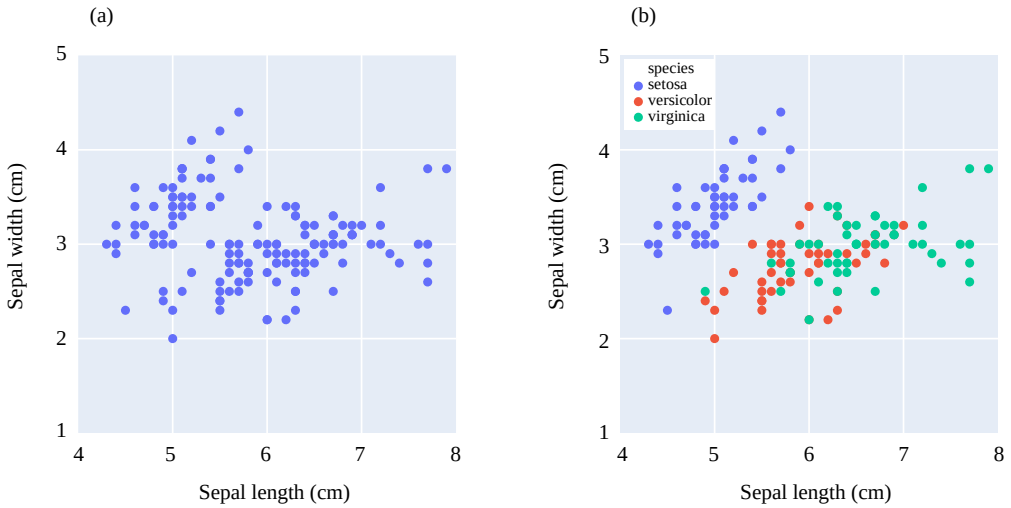


图 4. 使用 plotly.express.scatter() 绘制鸢尾花数据集散点图

11.3 等高线

等高线原理

等高线图是一种展示三维数据的方式，其中相同数值的数据点被连接成曲线，形成轮廓线。

形象地说，如图 5 所示，二元函数相当于一座山峰。在平行于 x_1x_2 平面在特定高度切一刀，得到的轮廓线就是一条等高线。这是一条三维空间等高线。然后，将等高线投影到 x_1x_2 平面，我们便得到一条平面等高线。

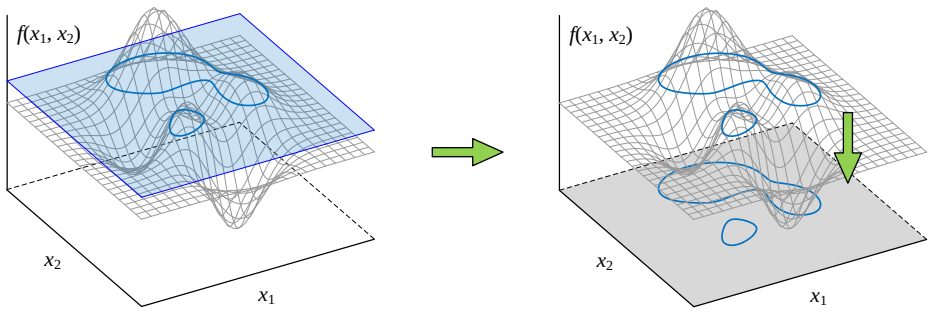


图 5. 平行 x_1x_2 平面切 $f(x_1, x_2)$ 获得等高线，然后等高线投影到 x_1x_2 平面

什么是二元函数？

二元函数是指具有两个自变量和一个因变量的函数。它接受两个输入，并返回一个输出。一般表示为 $y = f(x_1, x_2)$，其中 x_1 和 x_2 是自变量，y 是因变量。二元函数常用于描述和分析具有两个相关变量之间关系的数学模型。它可以用于表示二维空间中的曲面、表达物理或经济关系、进行数据建模和预测等。在可视化二元函数时，常使用三维图形或等高线图。三维图形以 x_1 和 x_2 作为坐标轴，将因变量 y 的值映射为曲面的高度。等高线图则使用等高线来表示 y 值的等值线，轮廓线的密集程度反映了函数值的变化。

一系列轮廓线的高度一般用不同的颜色或线型表示，使得我们可以通过视觉化方式看到数据的分布情况。如图 6 所示，将一组不同高度的等高线投影到平面便得到右侧平面等高线。右侧子图还增加了色谱条，用来展示不同等高线对应的具体高度。这一系列高度可以是一组用户输入的数值。大家可能已经发现，等高线图和海拔高度图原理完全相同。类似的图还有，等温线、等降水线、等距线等等。

Matplotlib 的填充等高线是在普通等高线的基础上添加填充颜色来表示不同区域的数据密度。可以使用 contourf() 函数来绘制填充等高线。

图 6 左图则是三维等高线，这是下一章要介绍的内容。

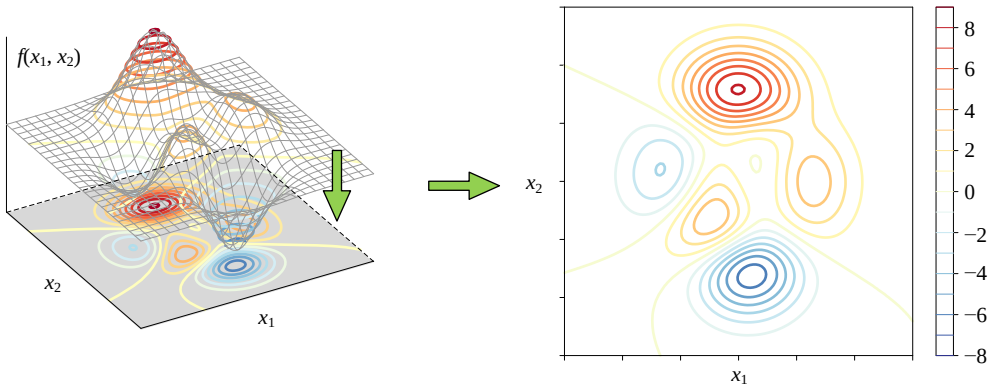


图 6. 将不同高度 (值) 对应的一组等高线投影到 x_1x_2 平面

网格数据

为了绘制平面等高线，我们需要利用 `numpy.meshgrid()` 产生网格数据。`numpy.meshgrid()` 接受一维数组作为输入，并生成二维、三维乃至多维数组来表示网格坐标。

原理上，如图 7 所示，`numpy.meshgrid()` 函数会将输入的一维数 (`x1_array` 和 `x2_array`) 组扩展为二维数组 (`xx1` 和 `xx2`)，其中一个数组的每一行都是输入数组的复制，而另一个数组的每一列都是输入数组的复制。这样，通过组合这两个二维数组的元素，就形成了一个二维网格。

```
xx1, xx2 = numpy.meshgrid(x1, x2)
```

提供两个一维数组 x1 和 x2 作为输入。函数将生成两个二维数组 xx1 和 xx2，用于表示一个二维网格。

请大家在 JupyterLab 中自行学习下例。

```
import numpy as np

x1 = np.arange(10)
# 第一个一维数组
x2 = np.arange(5)
# 第二个一维数组

xx1, xx2 = np.meshgrid(x1, x2)
```

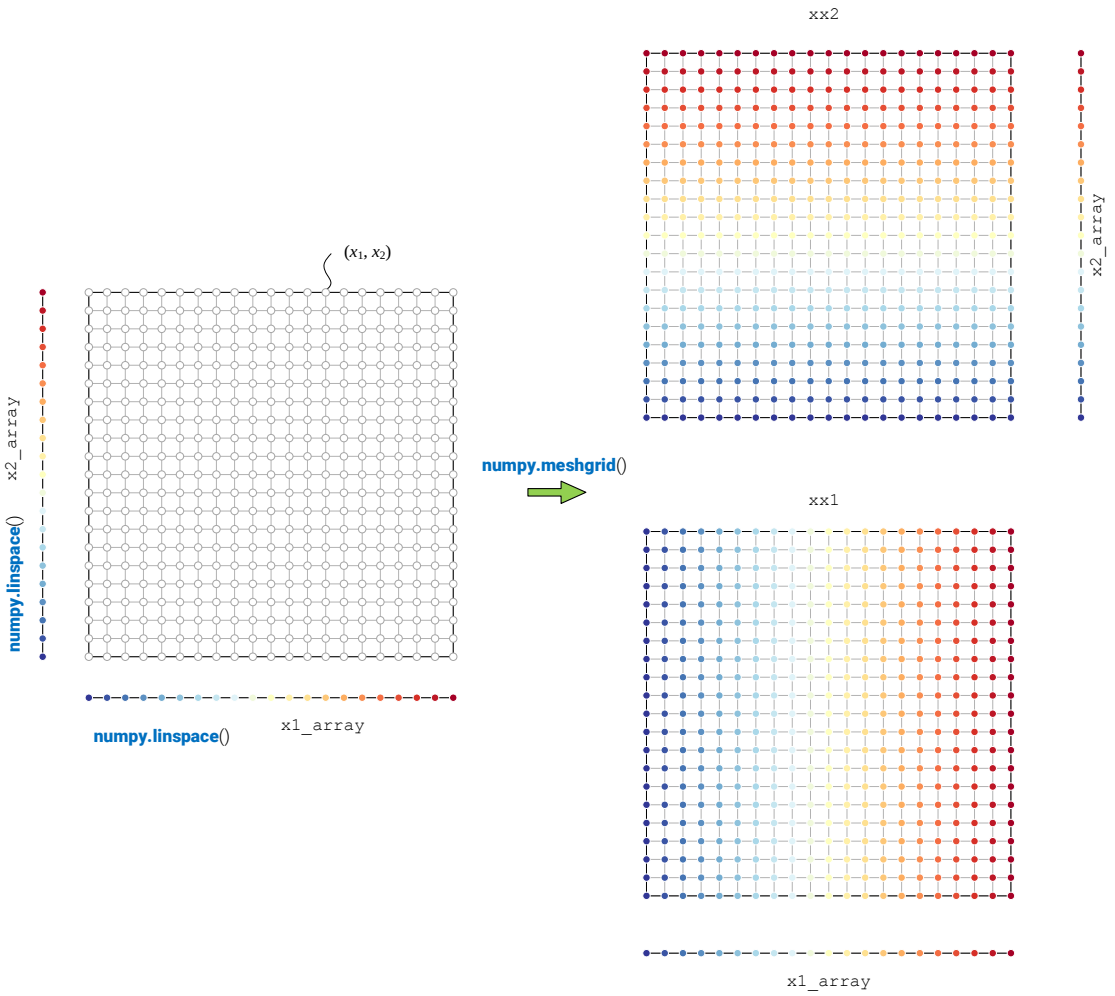


图 7. 用 numpy.meshgrid() 生成二维网络数据

Matplotlib

图 8 所示为利用 Matplotlib 中等高线可视化二元函数 $f(x_1, x_2) = x_1 \exp\left(-x_1^2 - x_2^2\right)$。填充等高线的原理是通过在等高线之间创建颜色渐变来表示不同区域的数值范围。这样可以增强等高线图的可视化效果，更直观地展示数据的分布和变化。

在 Matplotlib 中，填充等高线可以通过使用 contourf() 函数实现。该函数与 contour() 函数类似，但会填充等高线之间的区域。

```
matplotlib.pyplot.contour(X,Y,Z,levels,cmap)
```

下面是 contour() 函数的常用输入参数:

- X：二维数组，表示数据点的横坐标。
- Y：二维数组，表示数据点的纵坐标。
- Z：二维数组，表示数据点对应的函数值或高度。
- levels：用于指定绘制的等高线层级或数值列表。
- colors：用于指定等高线的颜色，可以是单个颜色字符串、颜色序列或 colormap 对象。
- cmap：颜色映射，用于将数值映射为颜色。可以是预定义的 colormap 名称或 colormap 对象。
- linestyles：用于指定等高线的线型，可以是单个线型字符串或线型序列。
- linewidths：用于指定等高线的线宽，可以是单个线宽值或线宽序列。
- alpha：用于指定等高线的透明度。

请大家在 JupyterLab 中自行学习下例。

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

# 创建二维数据
x = np.linspace(-2, 2, 100)
y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)
Z = X**2 + Y**2  # 示例函数，可以根据需要自定义

# 绘制等高线图
plt.contour(X, Y, Z, levels = np.linspace(0,8,16 + 1), cmap = 'RdYlBu_r')

# 添加颜色图例
plt.colorbar()

# 显示图形
plt.show()
```

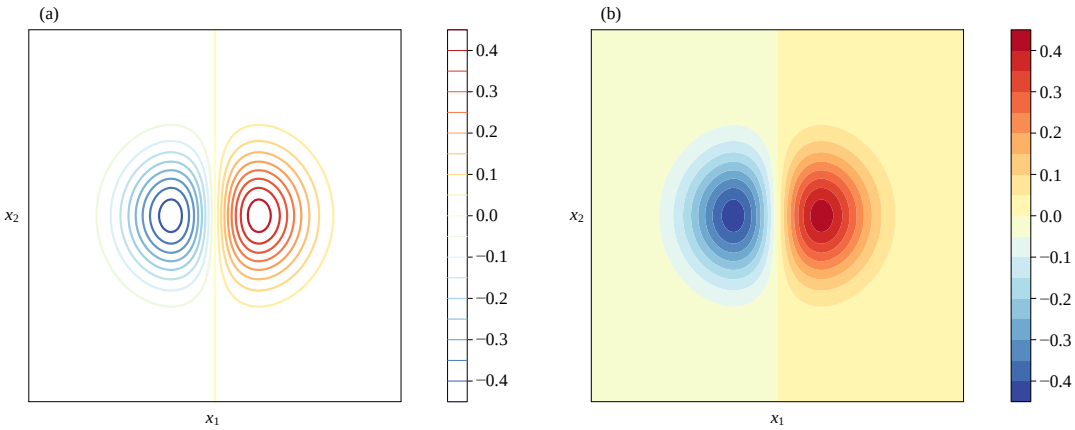


图 8. 用 Matplotlib 生成的平面 (填充) 等高线

Plotly

图 9 所示为利用 plotly.graph_objects.Contour() 绘制的 (填充) 等高线。

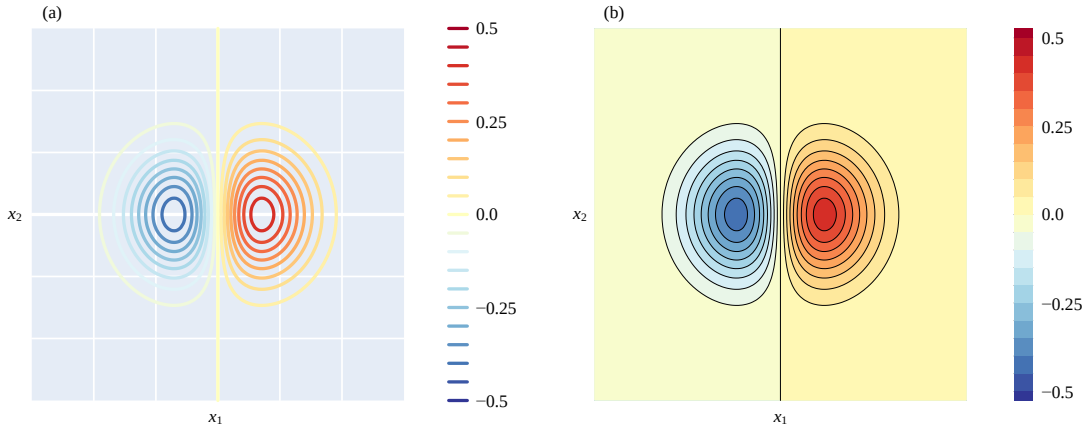


图 9. 用 Plotly 生成的平面 (填充) 等高线

11.4 热图

在 Matplotlib 中，可以使用 matplotlib.pyplot.imshow() 函数来绘制热图 (heatmap)，也叫热力图。imshow() 函数可以将二维数据矩阵的值映射为不同的颜色，从而可视化数据的密度、分布或模式。

鸢尾花书中一般会用 Seaborn 绘制静态热图，特别是在可视化矩阵运算。

```
seaborn.heatmap(data, vmin, vmax, cmap, annot)
```

下面是函数的常用输入参数：

- data：二维数据数组，要绘制的热图数据。
- vmin：可选参数，指定热图颜色映射的最小值。
- vmax：可选参数，指定热图颜色映射的最大值。
- cmap：可选参数，指定热图的颜色映射。可以是预定义的颜色映射名称或 colormap 对象。
- annot：可选参数，控制是否在热图上显示数据值。默认为 False，不显示数据值；设为 True 则显示数据值。
- xticklabels：可选参数，控制是否显示 X 轴的刻度标签。可以是布尔值或标签列表。
- yticklabels：可选参数，控制是否显示 Y 轴的刻度标签。可以是布尔值或标签列表。

请大家在 JupyterLab 中自行学习下例。

```
import seaborn as sns
import numpy as np

# 创建二维数据
data = np.random.rand(10,10)

# 绘制热图
sns.heatmap(data, vmin=0, vmax=1,
            cmap='viridis',
            annot=True,
            xticklabels=True,
            yticklabels=True)
```

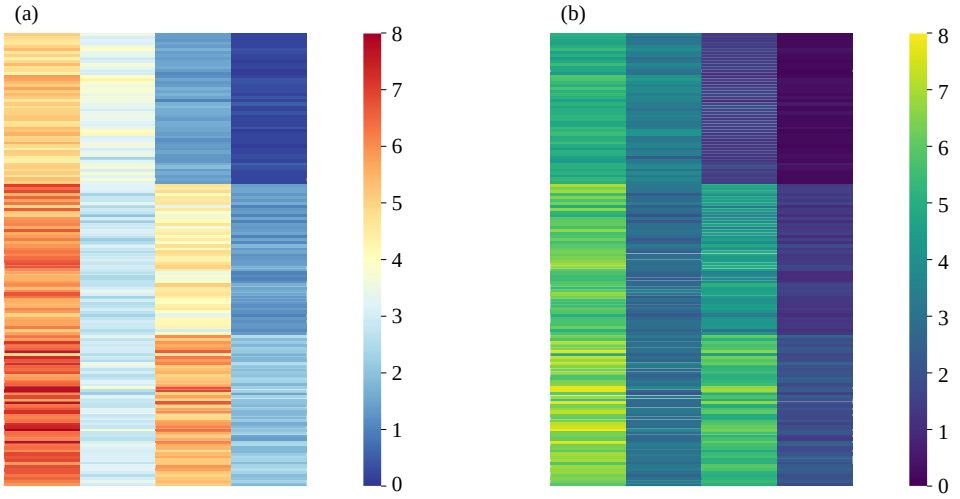


图 10. 使用 Seaborn、Plotly 热图可视化鸢尾花数据集

请大家完成下面 3 道题目。

Q1. 分别用 Matplotlib、Seaborn、Plotly 绘制鸢尾花数据集，花瓣长度、宽度散点图，并适当美化图像。

Q2. 分别用 Matplotlib 和 Plotly 绘制如下二元函数等高线图，并用语言描述图像特点 (等高线形状、疏密分布、增减、最大值、最小值等等)。

$$f(x_1,x_2)=x_1$$
$$f(x_1,x_2)=x_2$$
$$f(x_1,x_2)=x_1+x_2$$
$$f(x_1,x_2)=x_1-x_2$$
$$f(x_1,x_2)=x_1^2+x_2^2$$
$$f(x_1,x_2)=-x_1^2-x_2^2$$
$$f(x_1,x_2)=x_1^2+x_2^2+x_1x_2$$
$$f(x_1,x_2)=x_1^2-x_2^2$$
$$f(x_1,x_2)=x_1^2$$
$$f(x_1,x_2)=x_1x_2$$
$$f(x_1,x_2)=x_1^2+x_2^2+2x_1x_2$$

* 本章不提供答案。

12

3D Visualizations

三维可视化

三维直角坐标系中的散点、线图、网格面、等高线

> 善良一点，因为你遇到的每个人都在打一场更艰苦的战斗。
>
> ***Be kind, for everyone you meet is fighting a harder battle.***
>
> —— 柏拉图 (Plato) | 古希腊哲学家 | 424/423 ~ 348/347 BC

12.1 三维可视化方案

本章介绍常见四种三维空间可视化方案。图 1 所示为三维直角坐标系和三个平面。

散点图 (scatter plot) 用于展示三维数据的离散点分布情况。每个数据点在三维空间中的位置由其对应的三个数值确定。通过散点图，可以观察数据点的分布、聚集程度和可能的趋势。

线图 (line plot) 可用于表示在三维空间中的曲线或路径。通过将连续的点用线段连接，可以呈现数据的演变过程或路径的形态。线图在表示运动轨迹、时间序列数据等方面很有用。

网格面图 (mesh surface plot) 展示了三维空间中表面或曲面的形状。通过将空间划分为网格，然后根据每个网格点的数值给予相应的高度或颜色，可以可视化复杂的三维数据，例如地形地貌、物理场、函数表面等。

三维等高线图 (3D contour plot) 在三维空间中绘制了等高线的曲线。这种图形通过将等高线与垂直于平面的轮廓线相结合，可以同时显示三个维度的信息。它适用于表示等值线密度、梯度分布等。

鸢尾花书《数学要素》第 6 章专门介绍三维直角坐标系。

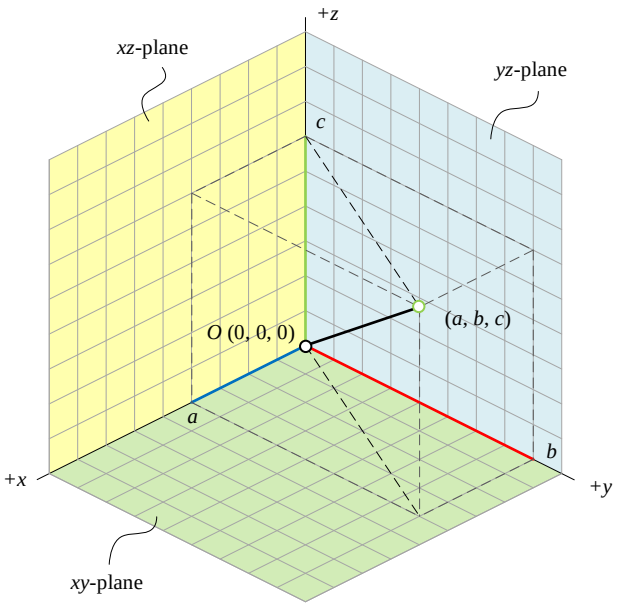


图 1. 三维直角坐标系和三个平面

三维视图视角

学过机械工程制图的同学知道，在三维空间中，我们可以将立体物体的投影投射到不同的平面上，以便更好地理解其形状和结构。图 2、图 3 展示咖啡杯在六个不同方向的投影。

以下是常见的三维立体在不同面的投影方式：

- 俯视投影 (top view) 把立体物体在垂直于其底面的平面上投影的方式。这种投影显示了物体的顶部视图，可以揭示物体在水平方向上的外形和布局。
- 侧视投影 (side view) 将立体物体在垂直于其侧面的平面上投影的方式。这种投影显示了物体的侧面视图，可以展示物体在垂直方向上的外形和结构。
- 正视投影 (front view) 把立体物体在垂直于其正面的平面上投影的方式。这种投影显示了物体的正面视图，可以展示物体在前后方向上的外形和特征。
- 斜视投影 (isometric view) 将立体物体在等角度投射到平面上的方式。它显示了物体的斜面视图，保留了物体在三个维度上的比例关系，使观察者能够同时感知物体的长度、宽度和高度。

这些不同面的投影方式可以提供不同的视角，帮助我们从多个方面理解和分析立体物体。选择合适的投影方式取决于我们关注的特定方面和目的。特别是用 Matplotlib、Plotly 绘制三维图像时，选择合适的投影方式至关重要。

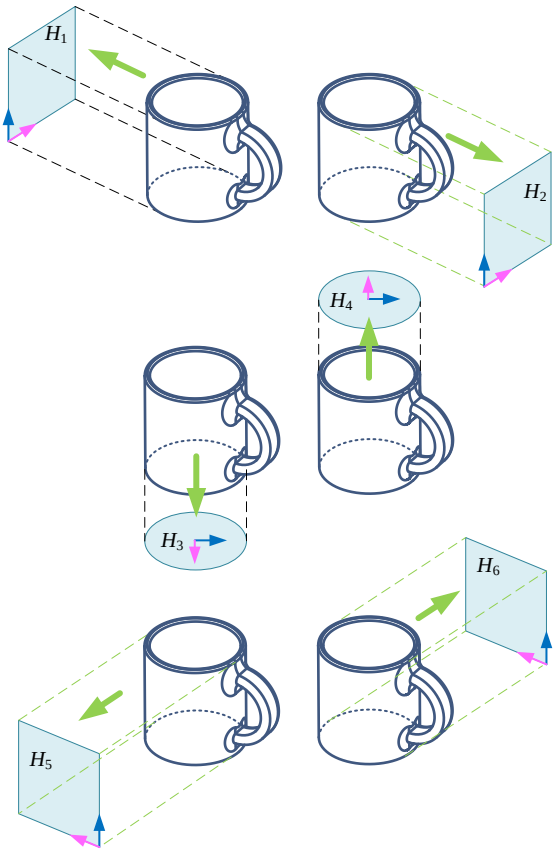


图 2. 咖啡杯六个投影方向

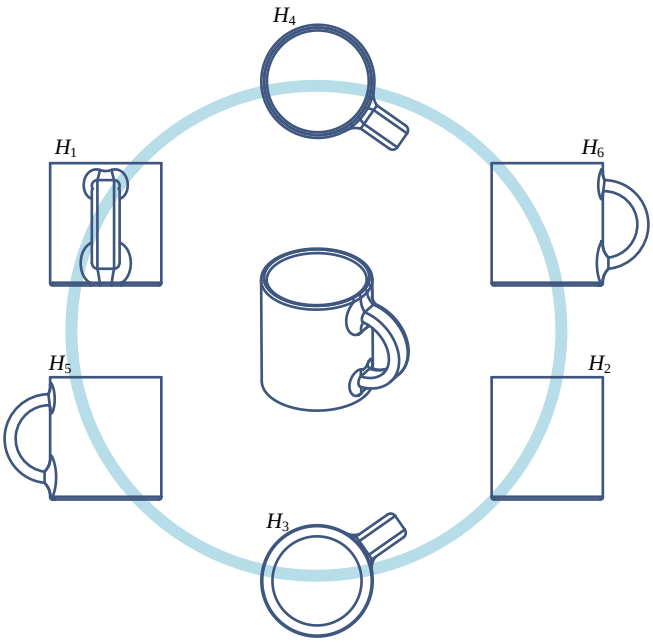


图 3. 咖啡杯在六个方向投影图像

在 Matplotlib 中，`ax.view_init(elev, azim, roll)` 方法用于设置三维坐标轴的视角，也叫相机照相位置。这个方法接受三个参数：`elev`、`azim` 和 `roll`，它们分别表示仰角、方位角和滚动角。

- 仰角 (elevation)：elev 参数定义了观察者与 *xy* 平面之间的夹角，也就是观察者与 *xy* 平面之间的旋转角度。当 elev 为正值时，观察者向上倾斜，负值则表示向下倾斜。
- 方位角 (azimuth)：azim 参数定义了观察者绕 *z* 轴旋转的角度。它决定了观察者在 *xy* 平面上的位置。azim 的角度范围是 −180 到 180 度，其中正值表示逆时针旋转，负值表示顺时针旋转。
- 滚动角 (roll)：roll 参数定义了绕观察者视线方向旋转的角度。它决定了观察者的头部倾斜程度。正值表示向右侧倾斜，负值表示向左侧倾斜。

通过调整这三个参数的值，可以改变三维图形的视角，从而获得不同的观察效果。例如，增加仰角可以改变观察者的俯视角度，增加方位角可以改变观察者在 XY 平面上的位置，增加滚动角可以改变观察者的头部倾斜程度。

类比的话，这三个角度和图 4 所示飞机的三个姿态角度类似。

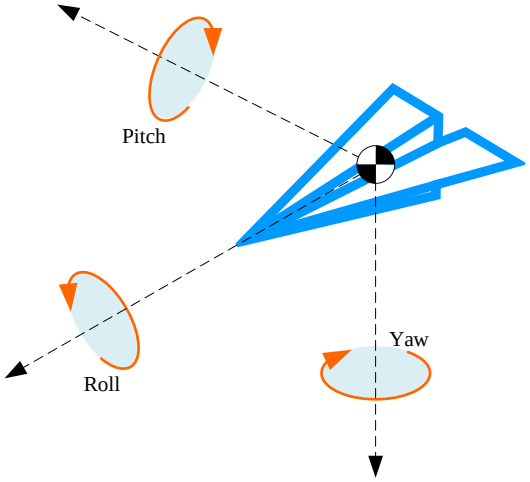


图 4. 飞机姿态的的三个角度

如图 5 所示，鸢尾花书中调整三维视图视角一般只会用 `elev`、`azim`，几乎不用使用 `roll`。

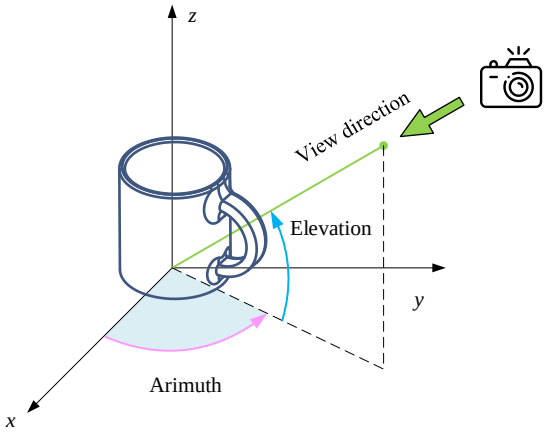


图 5. 仰角和方位角示意图

请大家在 JupyterLab 中练习如下代码，并调整仰角、方位角大小观察图像变化。

注意，ax = fig.gca(projection='3d') 已经被最新版本 Matplotlib 弃用，正确的语法为 ax = fig.add_subplot(projection='3d')。

Matplotlib设置观察视角

```
import matplotlib.pyplot as plt
# 导入Matplotlib的绘图模块

fig = plt.figure()
# 创建一个新的图形窗口

ax = fig.add_subplot(projection='3d')
# 在图形窗口中添加一个3D坐标轴子图

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
# 设置坐标轴的标签

ax.set_proj_type('ortho')
# 设置投影类型为正交投影 (orthographic projection)

ax.view_init(elev=30, azim=30)
# 设置观察者的仰角为30度，方位角为30度，即改变三维图形的视角

ax.set_box_aspect([1,1,1])
# 设置三个坐标轴的比例一致，使得图形在三个方向上等比例显示

plt.show()
# 显示图形
```

图 6. 设置三维图像观察视角

有关 Matplotlib 三维视图视角，请参考：

https://matplotlib.org/stable/api/toolkits/mplot3d/view_angles.html

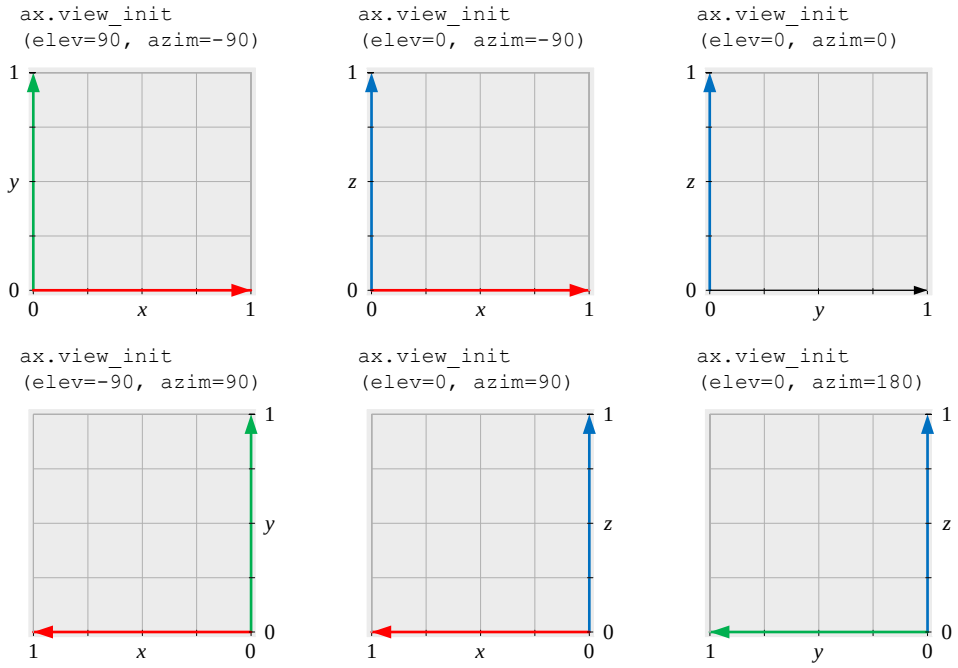


图 7. 几个特殊视角

两种投影方法

此外，大家还需要注意投影方法。上述代码采用的是正交投影。

在 Matplotlib 中，ax.set_proj_type() 方法用于设置三维坐标轴的投影类型。Matplotlib 提供了两种主要的投影类型：

- 透视投影 (perspective projection) 是默认的投影类型，如图 8 (a) 所示。简单来说就是近大远小，它模拟了人眼在观察远处物体时的视觉效果，使得远离观察者的物体显得较小。透视投影通过在观察者和图形之间创建一个虚拟的透视点，从而产生远近比例和景深感。设置方式为：ax.set_proj_type('persp')。
- 正交投影 (orthographic projection) 是另一种投影类型，如图 8 (b) 所示。它在观察者和图形之间维持固定的距离和角度，不考虑远近关系，保持了物体的形状和大小。正交投影在某些情况下可能更适合于一些几何图形的呈现，尤其是在需要准确测量物体尺寸或进行定量分析时。设置方式为：ax.set_proj_type('ortho')。

Plotly 的三维图像也是默认透视投影，想要改成正交投影对应的语法为：

```
fig.layout.scene.camera.projection.type = "orthographic"
```

图 9 展示了 3D 绘图时改变焦距对透视投影的影响。需要注意的是，Matplotlib 会校正焦距变化所带来的“缩放”效果。

透视投影中，默认焦距为 1，对应 90 度的视场角 (Field of View, FOV)。增加焦距 (1 至无穷大) 会使图像变得扁平，而减小焦距 (1 至 0 之间) 则会夸张透视效果，增加图像的视觉深度。当焦距趋近无穷大时，经过缩放校正后，会得到正交投影效果。

注意，鸢尾花书中三维图像绝大部分都是正交投影。

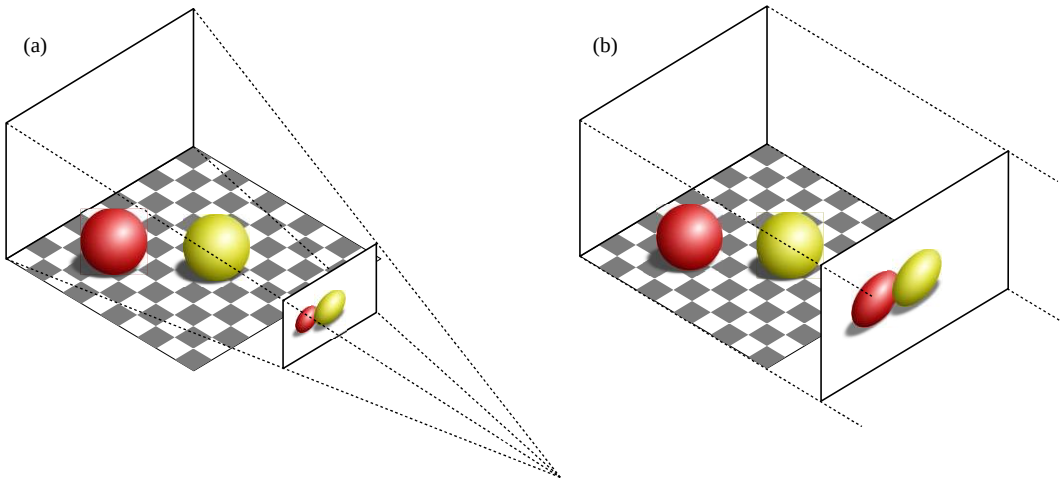


图 8. 两种投影方式，来源：https://github.com/rougier/scientific-visualization-book

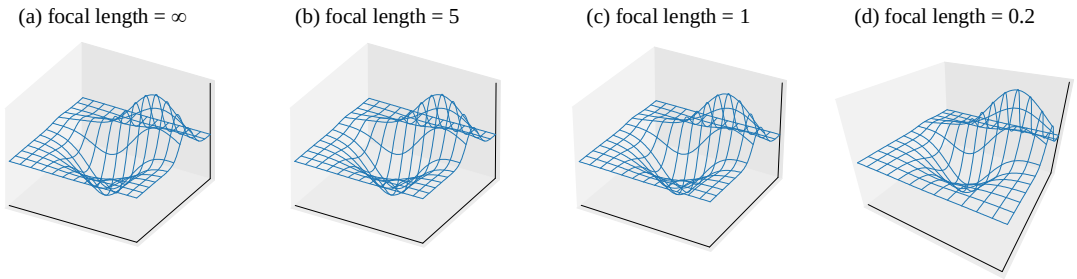


图 9. 投影焦距对结果影响；参考：https://matplotlib.org/stable/gallery/mplot3d/projections.html

12.2 散点

上一章我们利用平面散点可视化鸢尾花数据集，这一节将用三维散点图可视化这个数据集。图 10 所示为利用 Matplotlib 绘制的三维散点图，这幅图用不同颜色表征鸢尾花分类。类似图 6，请大家将图 10 投影到不同平面上。

本章配套的 Jupyter Notebook 还用 Plotly 绘制了散点图，请大家自行学习。

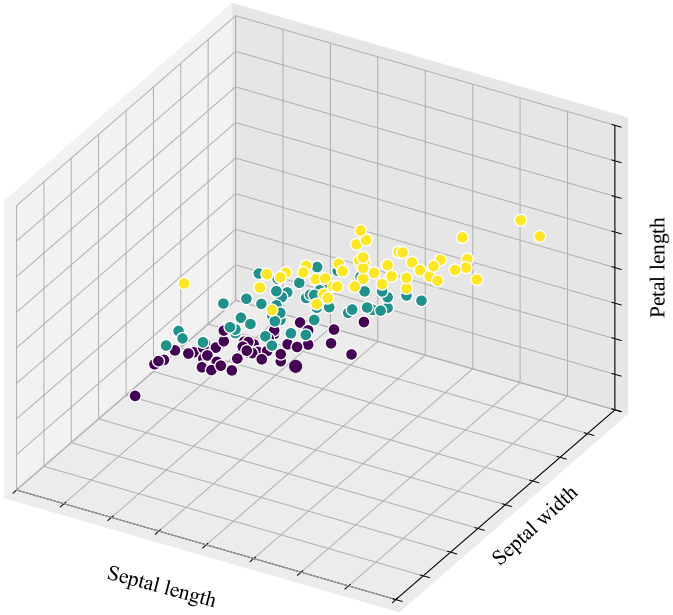


图 10. 用 Matplotlib 和绘制散点图

12.3 线图

图 11 所示为利用 Matplotlib 绘制“线图 + 散点图”可视化微粒的随机漫步。并且用散点的颜色渐进变化展示时间维度。本章配套的 Jupyter Notebook 也用 Plotly 绘制相同图像，请大家自行学习。

《数据有道》将专门介绍随机漫步。

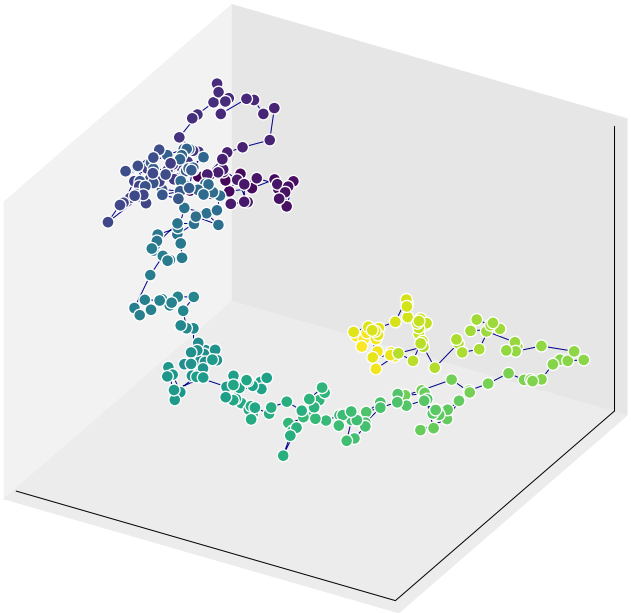


图 11. 用 Matplotlib 绘制微粒随机漫步线图

什么是随机漫步？

随机漫步是指一个粒子或者一个系统在一系列离散的时间步骤中，按照随机的方向和大小移动的过程。每个时间步骤，粒子以随机的概率向前或向后移动一个固定的步长，而且每个时间步骤之间的移动是相互独立的。随机漫步模型常用于模拟不确定性和随机性的系统，例如金融市场、扩散过程、分子运动等。通过模拟大量的随机漫步路径，可以研究粒子或系统的统计特性和概率分布。

12.4 网格面

图 12 所示为利用 Axes3D.plot_surface() 绘制的三维网格曲面。请大家思考如何在图片中加入 colorbar。本章配套的 Jupyter Notebook 也用 Plotly 绘制的三维曲面，请大家自行学习。

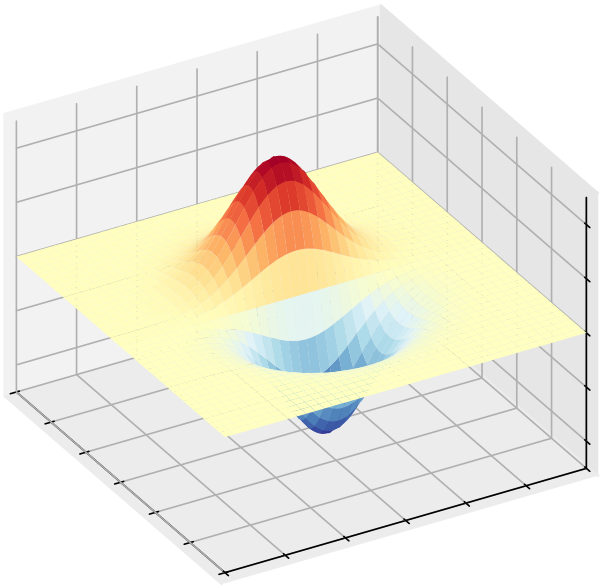


图 12. 用 Matplotlib 绘制网格曲面

12.5 三维等高线

图 13 所示为用 Matplotlib 绘制的三维等高线，这些等高线投影到水平面便得到上一章介绍的平面等高线。本章配套的 Jupyter Notebook 也用 Plotly 绘制的三维“曲面 + 等高线”，请大家自行学习。

鸢尾花书《可视之美》将介绍更多三维等高线的用法。

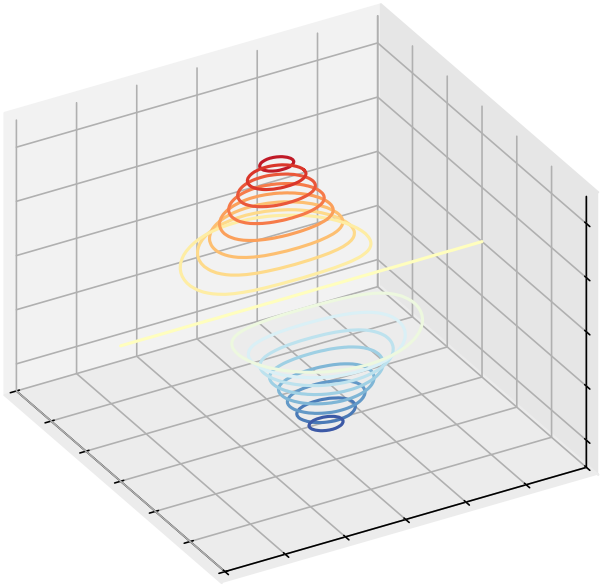


图 13. 用 Matplotlib 绘制三维等高线

请大家完成下面这道题目。

Q1. 请用分别用 Matplotlib 和 Plotly 中网格面、三维等高线可视化上一章 Q2 给出的几个二元函数。

* 本章不提供答案。

13

Fundamentals of NumPy

聊聊 NumPy

本节的核心是用 NumPy 产生不同类型数组

重要的不是生命的长度，而是深度。

It is not the length of life, but the depth.

—— 拉尔夫·沃尔多·爱默生 (Ralph Waldo Emerson) | 美国思想家、文学家 | 1942 ~ 2018

13.1 什么是 NumPy?

NumPy 是 Python 科学计算中非常重要的一个库，它提供了快速、高效的多维数组对象及其操作方法，是众多其他科学计算库的基础。

NumPy 最重要的功能之一是提供了高效的多维数组对象 ndarray，可以用来表示向量、矩阵和更高维的数组。它是 Python 中最重要的科学计算数据结构，支持广泛的数值运算和数学函数操作。

此外，如果大家需要处理有标签、多维数组数据的话，推荐使用 xarray。xarray 可以看作是在 ndarray 的基础上，增加了标签和元数据的功能。xarray 可以对多个数组进行向量化计算，避免了循环操作，提高了计算效率。xarray 提供了多种统计分析函数，可以方便地对多维数组数据进行统计分析。本书将不会展开讲解 xarray。

NumPy 提供了多种数组操作方法，包括数组索引、切片、迭代、转置、变形、合并等，以及广播 (broadcasting) 机制，使得数组操作更加方便、高效。这些话题是本书后续要展开讲解的内容。本书后文会专门讲解广播。

NumPy 提供了丰富的数学函数库，包括三角函数、指数函数、对数函数、逻辑函数、统计函数、随机函数等，能够满足大多数科学计算需要。

“鸢尾花书”中《数学要素》一册将大量使用这些函数库来可视化常见函数。

NumPy 支持多种文件格式的读写操作，包括文本文件、二进制文件、CSV 文件等。NumPy 基于 C 语言实现，因此可以利用底层硬件优化计算速度，同时还支持多线程、并行计算和向量化操作，使得计算更加高效。

NumPy 提供了丰富的线性代数操作方法，包括矩阵乘法、求逆矩阵、特征值分解、奇异值分解等，可以方便地解决线性代数问题。

本书中会简要介绍这些常见线性代数操作，详细讲解请大家参考“鸢尾花书”中的《矩阵力量》一册。

NumPy 可以于 Matplotlib 库集成使用，方便地生成各种图表，如线图、散点图、柱状图等。相信大家在本书前文已经看到基于 NumPy 数据绘制的平面、三维图像。

NumPy 提供了一些常用的数据处理方法，如排序、去重、聚合、统计等，方便对数据进行预处理。即便如此，“鸢尾花书”中我们更常用 Pandas 处理数据，本书后续将专门介绍 Pandas。

Python 中许多数据分析和机器学习的库都是基于 NumPy 创建。Scikit-learn 是一个流行的机器学习库，它基于 NumPy、SciPy 和 Matplotlib 创建，提供了各种机器学习算法和工具，如分类、回归、聚类、降维等。PyTorch 是一个开源的机器学习框架，它基于 NumPy 创建，提供了张量计算和动态计算图等功能，可以用于构建神经网络和其他机器学习算法。TensorFlow 是一个深度学习框架，它基于 NumPy 创建，提供了各种神经网络算法和工具，包括卷积神经网络、循环神经网络等。

“鸢尾花书”中的《数据有道》专门讲解回归、降维这两类机器学习算法，而《机器学习》一册则侧重分类、聚类。

13.2 手动构造数组

从 numpy.array() 说起

我们可以利用 **numpy.array**() 手动生成一维、二维、三维等数组。下面首先介绍如何使用 **numpy.array**() 这个函数。

numpy.array(object, dtype)

这个函数的重要输入参数：

- object 转换为数组的输入数据，可以是列表、元组、其他数组或类似序列的对象。
- dtype 参数用于指定数组的数据类型。如果不指定 dtype 参数，则 NumPy 会自动推断数组的数据类型。

请大家在 JupyterLab 中自行学习下例。

```
import numpy as np

# 从列表中创建一维数组
arr1 = np.array([1, 2, 3, 4])

# 指定数组的数据类型
arr2 = np.array([1, 2, 3, 4], dtype=float)

# 从元组中创建二维数组
arr3 = np.array([(1, 2, 3), (4, 5, 6)])

# 指定最小维度
arr4 = np.array([1, 2, 3, 4], ndmin=2)
```

NumPy 中的 array 是什么？

在 NumPy 中，array 是一种多维数组对象，它可以用于表示和操作向量、矩阵和张量等数据结构。array 是 NumPy 中最重要的数据结构之一，它支持高效的数值计算和广播操作，可以用于处理大规模数据集和科学计算。与 Python 中的列表不同，array 是一个固定类型、固定大小的数据结构，它可以支持多维数组操作和高性能数值计算。array 的每个元素都是相同类型的，通常是浮点数、整数或布尔值等基本数据类型。在创建 array 时，用户需要指定数组的维度和类型。例如，可以使用 numpy.array() 函数创建一个一维数组或二维数组，也可以使用 numpy.zeros() 函数或 numpy.ones() 函数创建指定大小的全 0 或全 1 数组，还可以使用 numpy.random 模块生成随机数组等。除了基本操作之外，NumPy 还提供了许多高级的数组操作，例如数组切片、数组索引、数组重塑、数组转置、数组拼接和分裂等。

本节配套的 Jupyter Notebook 文件 BK_2_Topic_4.01_1.ipynb，请大家边读正文边在 Notebook 中探究学习。

手动生成一维数组

在 NumPy 中，一维数组是最基本的数组类型，也被称为一维 ndarray。它只有一个维度，并且可以包含多个元素，其中每个元素都是相同的数据类型。

图 1 所示为利用 **numpy.array**() 生成的一维数组。这个数组的形状为 (7,)，长度为 7，维度为 1。和本书前文介绍的 list 一样，NumPy 数组的索引也是从 0 开始。下一话题专门讲解 NumPy 数组索引和切片。再次强调，如图 1 所示，本书可视化一维数组时用圆形。

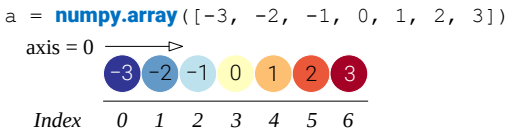


图 1. 手动生成一维数组

下面区分一下形状、长度、维度、大小这四个特征：

- 形状：可以使用 **shape** 属性来获取数组的形状，即每个维度上的大小，例如，如果数组 arr 是一个二维数组，则可以使用 arr.**shape** 来获取其形状。
- 长度：可以使用 **len**()函数来获取数组的长度，例如，如果数组 arr 是一个一维数组，则可以使用 **len**(arr)来获取其长度。
- 维数：可以使用 **ndim** 属性来获取数组的维数，例如，如果数组 arr 是一个二维数组，则可以使用 arr.**ndim** 来获取其维数。
- 大小：可以使用 **size** 属性来获取数组的大小，即所有元素的个数，例如，如果数组 arr 是一个二维数组，则可以使用 arr.**size** 来获取其大小。

手动生成二维数组

图 2 所示为利用 **numpy.array**() 生成的二维数组。利用 V 方法，大家可以发现图 2 中数组的维度都是 2。此外，**numpy.matrix**() 专门用来生成二维矩阵，请大家自行学习。

⚠ 请大家注意图 2 中中括号 [] 的数量。特别强调，本书中，行向量、列向量都被视作特殊的二维数组。也就是说，行向量是一行多列矩阵，而列向量是多行一列矩阵。

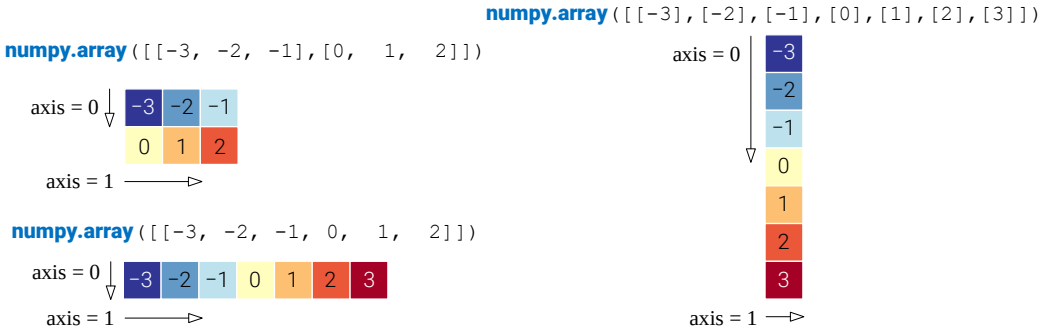


图 2. 手动生成二维数组

手动生成三维数组

图 3 所示为利用 **numpy.array**() 生成的三维数组，这个数组的形状为 (2, 3, 4)，也就是 2 页、3 行、4 列。Jupyter Notebook 文件展示如何获取三维数组的第 0 页和第 1 页。

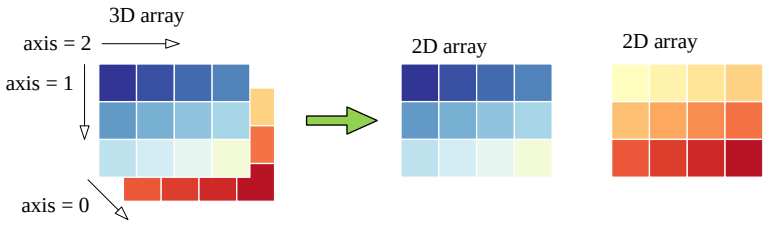


图 3. 手动生成三维数组

13.3 生成数列

在 NumPy 中我们常用以下三个函数生成数列：

- **numpy.arange**(start, stop, step)。生成等差数列，从起始值 start 开始，以步长 step 递增，直到结束值 stop (不包含 stop)。例如，**numpy.arange**(1, 11, 2) 将生成一个等差数列 [1, 3, 5, 7, 9]。实际上，**numpy.arange**() 和前文介绍的 **range**() 函数颇为相似。
- **numpy.linspace**(start, stop, num, endpoint)。生成等间距数列，从起始值 start 开始，到结束值 stop 结束，num 指定数列的长度 (元素的个数)，默认为 50。endpoint 参数指定是否包含结束值。例如，numpy.linspace(1, 10, 6) 生成一个等间距数列 [1, 3.25, 5.5, 7.75, 10]。
- **numpy.logspace**(start, stop, num, endpoint, base)：生成等比数列，从 base 的 start 次幂开始，到 base 的 stop 次幂结束，num 指定数列的长度，默认为 50。endpoint 和 dtype 参数与 **numpy.linspace**() 函数相同。例如，**numpy.logspace**(0, 4, 5, base=2) 将生成一个等比数列 [1, 2, 4, 8, 16]。

相信大家对 **numpy.linspace**() 函数已经不陌生，本书前文在讲可视化时已经介绍过这个函数。我们经常会在二维可视化中用到 **numpy.linspace**()。

什么是数列？

数列是指一列按照一定规律排列的数，它通常用一个公式来表示，也可以用递推关系式来定义。数列中的每个数称为数列的项，用 a_n 来表示第 n 项。数列在数学中具有广泛的应用，它是许多数学分支的基础，如数学分析、概率论、统计学、离散数学和计算机科学等。在数学中，数列是一种有序的集合，通常用于研究数学对象的性质和行为，例如函数、级数、微积分和代数等。数列可以分为等差数列、等比数列和通项公式不规则数列等几种类型。等差数列的项之间的差是固定的，比如 1、2、3、4 … 100。等比数列的相邻项之间的比是固定的，比如 2、4、8、16 … 2048。

13.4 生成网格数据

本书前文提过 **numpy.meshgrid**() 函数。**numpy.meshgrid**() 可以生成多维网格数据，它可以将多个一维数组组合成一个 N 维数组，并且可以方便地对这个 N 维数组进行计算和可视化。

在科学计算中，常常需要对多维数据进行可视化，比如绘制 3D 曲面图、等高线图等。**numpy.meshgrid**() 可以方便地生成网格数据，使得我们可以对多维数据进行可视化。

例如，如图 4 所示，对于二元函数 $f(x_1, x_2)$，我们可以使用 **numpy.meshgrid**() 生成横坐标和纵坐标的网格点，然后计算每个网格点的函数值，最后将网格点和对应的函数值作为输入，绘制出如图 5 所示的 3D 曲面图。

《可视之美》将介绍如何生成图 5 这幅图。

如图 6 所示，**numpy.meshgrid**() 还可以用来生成三维网格数据。在《可视之美》一册中，大家可以看到大量利用三维网格数据完成的可视化方案。

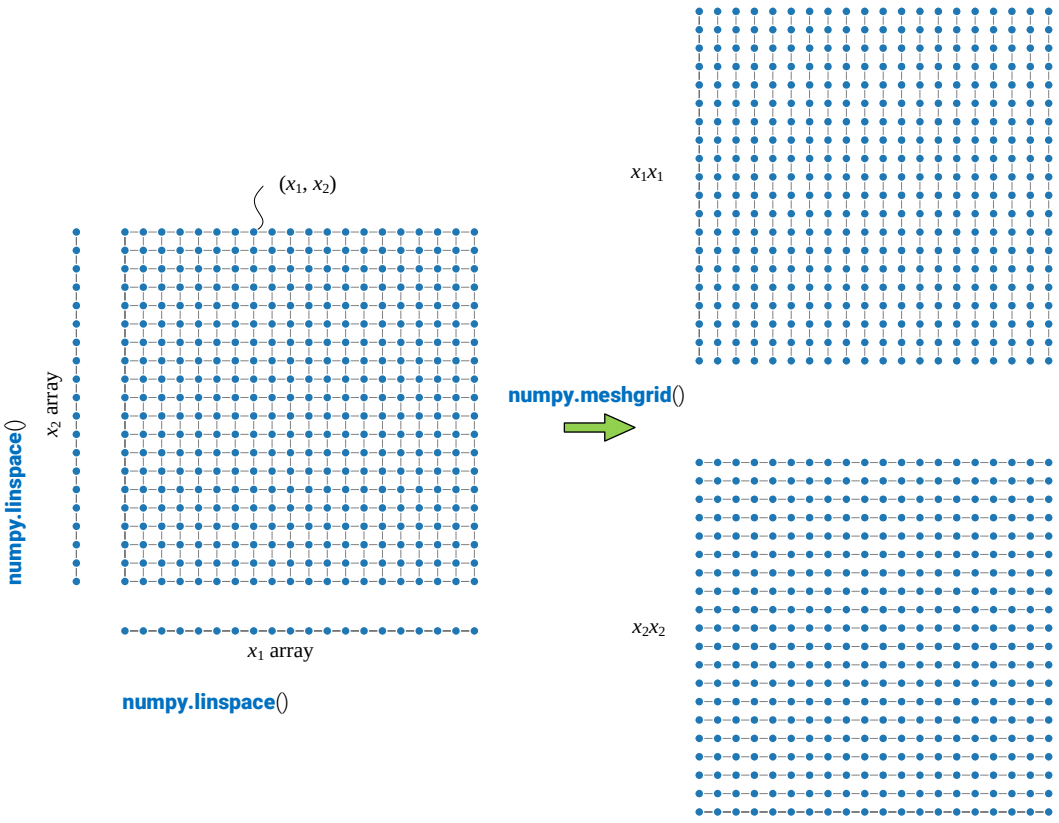


图 4. 用 numpy.meshgrid() 生成二维网络状坐标，x_1x_1 代表散点的横轴坐标，x_2x_2 代表散点的纵轴坐标

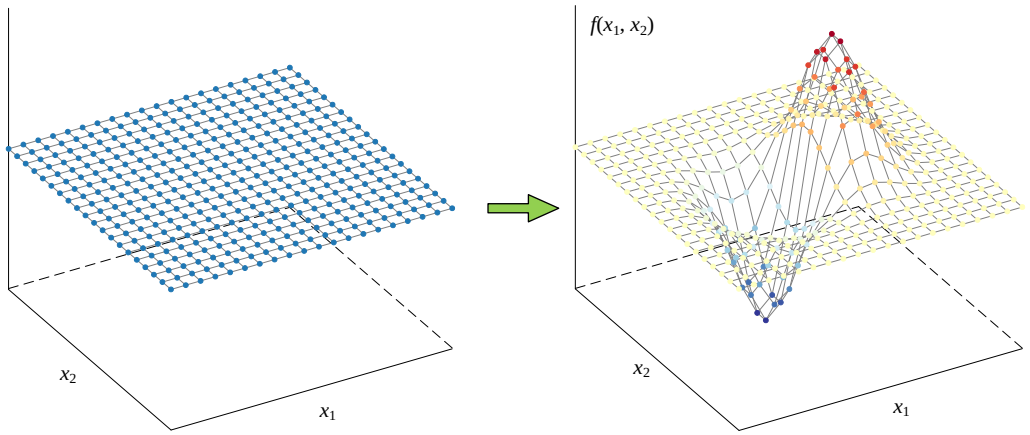


图 5. 三维空间看二维网络状坐标

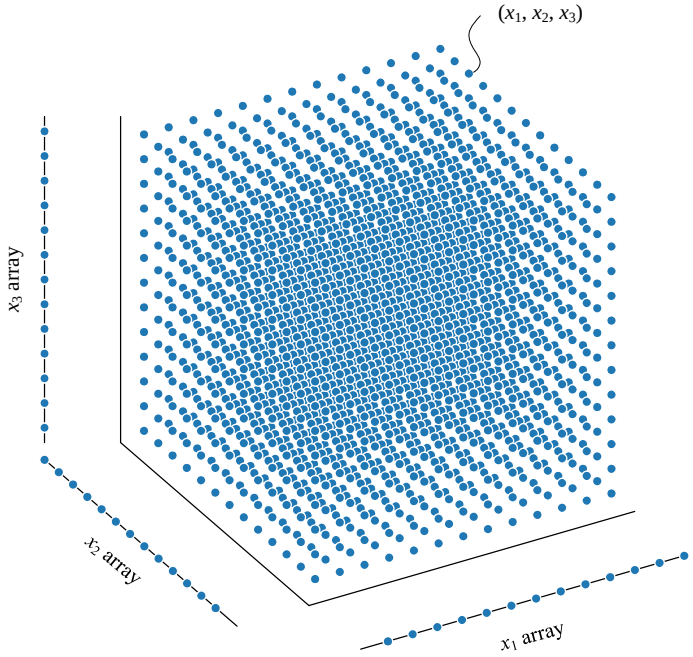


图 6. 三维网格

13.5 特殊数组

表 1 总结 NumPy 中常用来生成特殊数组的函数、用途、示例。表 1 第二列都是由 ChatGPT 生成的答案。请大家在 JupyterLab 中练习使用这些函数。

表 1. 用 NumPy 函数生成特殊数组

函数	用途	代码示例
numpy.empty()	numpy.empty()是一个用于创建一个指定大小的、未初始化的数组的函数。它返回一个数组对象，其元素的值是随机的，取决于数组在内存中的位置。因此，使用 numpy.empty()创建的数组的值是不确定的。	`import numpy as np` `np.empty([4,4])`

numpy.empty_like()	numpy.empty_like()是一个用于创建与给定数组具有相同形状和数据类型的未初始化数组的函数。它返回一个新的数组对象，其元素的值是随机的，取决于数组在内存中的位置。因此，使用 numpy.empty_like()创建的数组的值是不确定的。	`import numpy as np`
		`A = np.array([[1, 2, 3],`
		`[4, 5, 6]])`
		`np.empty_like(A)`

numpy.eye()	numpy.eye()是一个用于创建一个二维数组，表示单位矩阵的函数。它返回一个 $N \times N$ 的矩阵，其中对角线上的元素为 1，其他元素为 0。可以通过指定参数 N，来指定矩阵的大小。	`import numpy as np` `np.eye(5)`
numpy.full()	numpy.full()是一个用于创建一个指定大小和给定值的数组的函数。它返回一个数组对象，其所有元素都初始化为指定的值。可以通过指定	`import numpy as np` `np.full((3,3), np.inf)`

	参数来指定数组的大小和数据类型，以及所填充的值。	

numpy.full_like()	numpy.full_like()是一个用于创建与给定数组具有相同形状和数据类型，且所有元素都是指定值的数组的函数。它返回一个新的数组对象，其所有元素都初始化为指定的值。可以通过指定参数来指定所填充的值。	`import numpy as np`
		`A = np.array([[1, 2, 3],`
		`[4, 5, 6]])`
		`np.full_like(A, 100)`

numpy.ones()	numpy.ones()是一个用于创建一个指定大小的全 1 数组的函数。它返回一个数组对象，其所有元素都是 1。可以通过指定参数来指定数组的大小和数据类型。	`import numpy as np` `np.ones((5,5))`

numpy.ones_like()	numpy.ones_like()是一个用于创建与给定数组具有相同形状和数据类型，且所有元素都是 1 的数组的函数。它返回一个新的数组对象，其所有元素都是 1。可以通过指定参数来指定所创建数组的数据类型。	`import numpy as np`
		`A = np.array([[1, 2, 3],`
		`[4, 5, 6]])`
		`np.ones_like(A)`

numpy.zeros()	numpy.zeros()是一个用于创建一个指定大小的全 0 数组的函数。它返回一个数组对象，其所有元素都是 0。可以通过指定参数来指定数组的大小和数据类型。	`import numpy as np` `np.zeros((5,5))`

numpy.zeros_like()	numpy.zeros_like()是一个用于创建与给定数组具有相同形状和数据类型，且所有元素都是 0 的数组的函数。它返回一个新的数组对象，其所有元素都是 0。可以通过指定参数来指定所创建数组的数据类型。	`import numpy as np`
		`A = np.array([[1, 2, 3],`
		`[4, 5, 6]])`
		`np.zeros_like(A)`

什么是单位矩阵？

单位矩阵是一个非常特殊的方阵，它的对角线上的元素全都是 1，而其余元素全都是 0。常用符号表示单位矩阵的是 $\boldsymbol{I}$ 或者 $\boldsymbol{E}$，它的大小由下标表示，例如，$\boldsymbol{I}_2$ 表示 2 × 2 的单位矩阵。类似地，$\boldsymbol{I}_3$ 表示 3 × 3 的单位矩阵，以此类推。单位矩阵是在矩阵运算中非常重要的一个概念，它可以被看作是矩阵乘法中的“1”，即任何矩阵与单位矩阵相乘，其结果都是该矩阵本身。单位矩阵在许多应用中都有广泛的应用，例如，单位矩阵可以用来表示标准正交基等。在计算矩阵的逆时，单位矩阵也起到了关键作用，因为一个矩阵 $\boldsymbol{A}$ 的逆矩阵可以通过 $\boldsymbol{A}$ 和单位矩阵的运算来计算，即 $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$。

13.6 随机数

NumPy 中还有大量产生随机数的函数。图 7 所示为满足二元连续均匀分布、二元高斯分布的随机数。请大家翻阅帮助文档了解这些函数的用法，并在 JupyterLab 中动手实践。**Error! Reference source not found.**总结 NumPy 中和随机数有关的常用函数

“鸢尾花书”《统计至简》一册将专门讲解各种常用概率分布。

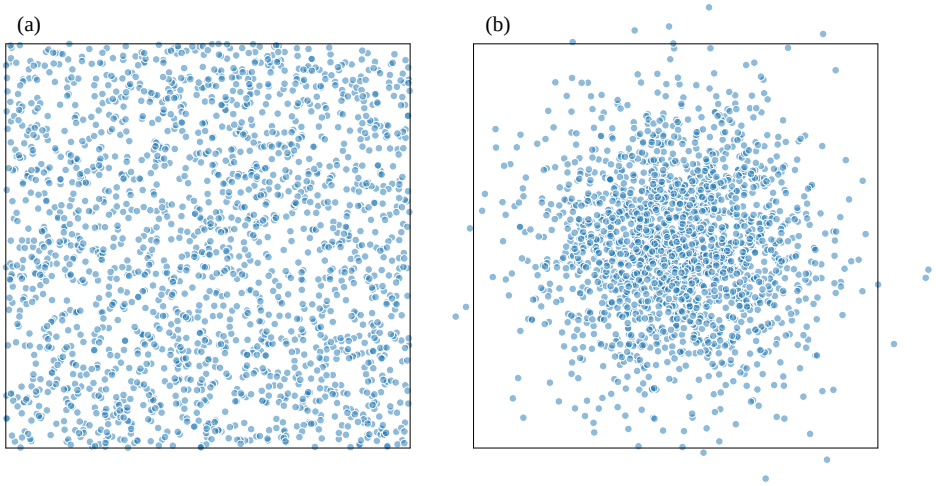
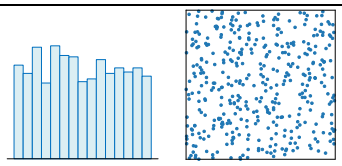
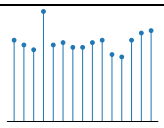
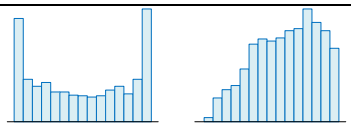
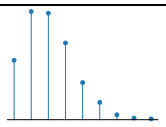
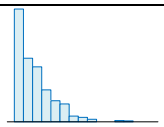
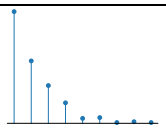
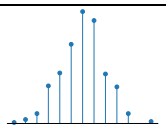
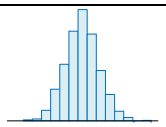
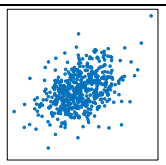
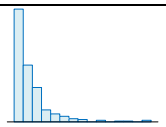
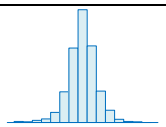
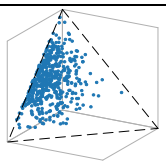


图 7. 分别满足二元连续均匀分布、二元高斯分布的随机数

表 1 总结常用随机数发生器函数和随机数分布图像。

表 2. 常用随机数发生器

随机数服从的分布	函数	随机数分布图像
连续均匀分布	numpy.random.uniform()	
均匀整数	numpy.random.randint()	
Beat 分布	numpy.random.beta()	
泊松分布	numpy.random.poisson()	
指数分布	numpy.random.exponential()	

几何分布	numpy.random.geometric()	
二项分布	numpy.random.binomial()	
正态分布	numpy.random.normal()	
多元正态分布	numpy.random.multivariate_normal()	
对数正态分布	numpy.random.lognormal()	
学生 *t*-分布	numpy.random.standard_t()	
Dirichlet 分布	numpy.random.dirichlet()	

概率统计中，随机是什么意思？

在概率统计中，随机指的是一个事件的结果是不确定的，而且每种可能的结果出现的概率是可以计算的。随机事件是由各种随机变量所描述的，随机变量是一个具有不确定结果的数学变量，其值取决于随机事件的结果。概率统计学家使用随机变量和概率分布来描述随机事件的结果和出现的概率。随机事件的结果可能是离散的，例如掷骰子的结果是 1、2、3、4、5 或 6，也可能是连续的，例如衡量人的身高或重量。概率统计学家使用各种数学方法和技术，例如概率、期望值和方差等，来分析和理解随机事件和随机变量的性质和行为。概率统计的研究在现代科学和工程中有着广泛的应用，例如金融、生物学、医学、物理学等领域。

什么是随机数发生器？

随机数生成器是一种用于生成随机数的计算机程序或硬件设备。随机数生成器可分为真随机数生成器和伪随机数生成器两种。真随机数生成器的输出完全基于物理过程，如大气噪声、放射性衰变或者热噪声等，其生成的随机数序列是完全随机且不可预测的。真随机数生成器通常需要专门的硬件设备支持。伪随机数生成器则使用计算机算法生成伪随机数，其看似随机，但是实际上是可预测的，因为它们是由固定的算法和种子值生成的。伪随机数生成器通常使用伪随机数序列和随机

种子，以便在需要时生成随机数。随机数生成器在计算机科学、加密学、模拟实验、游戏设计、统计分析等领域中被广泛使用。在加密学中，随机数生成器通常用于生成安全密钥和初始化向量等关键数据，以保证加密算法的强度和安全性。在模拟实验和游戏设计中，随机数生成器用于模拟不可预测的因素，如掷骰子、扑克牌等。

13.7 数组导入、导出

numpy.savetxt() 可以把 numpy array 写成 txt、CSV 文件。numpy.genfromtxt() 可以用来读入 txt、CSV 文件。图 8 所示为鸢尾花表格和热图。大家在本书后文，特别是在《矩阵力量》一册中会看到，我们大量使用热图可视化矩阵运算。

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
4.8	3.0	1.4	0.1
4.3	3.0	1.1	0.1
...	...	...	...
5.9	3.0	5.1	1.8

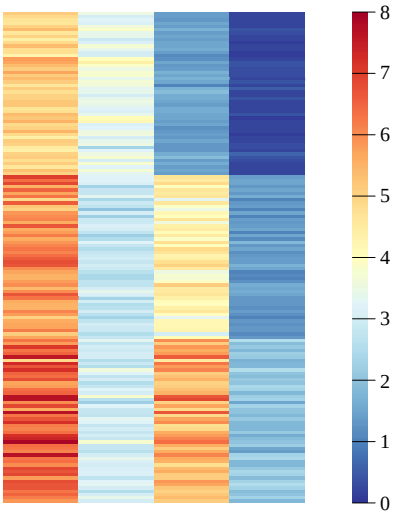


图 8. 鸢尾花数据表格和热图

请大家完成下面 3 道题目。

Q1. 用至少两种办法生成一个 3 × 4 二维 NumPy 数组，数组的每个值都是 10。

Q2. 利用 numpy.meshgrid() 和 matplotlib.pyplot.contour() 绘制二元函数 $f(x_1, x_2) = x_2 \exp\left(-x_1^2 - x_2^2\right)$ 的平面等高线。

Q3. 在 [0, 1] 范围内生成 1000 个满足连续均匀随机数，并用 matplotlib.pyplot.hist()绘制频率直方图。

* 题目答案在 Bk1_Ch13_02.ipynb。

14 NumPy 索引和切片

Indexing and Slicing NumPy Arrays

获取数组的部分成分

做数学的艺术在于找到包含所有普遍性萌芽的特殊情况。

The art of doing mathematics consists in finding that special case which contains all the germs of generality.

—— 大卫·希尔伯特 (David Hilbert) | 德国数学家 | 1862 ~ 1943

14.1 什么是索引、切片?

这个话题聊一聊 NumPy 数组的索引 (indexing) 和切片 (slicing)。简单来说，数组中的某个元素可以通过索引来访问。切片指的是从数组中提取“子数组”的操作。

需要注意的是，NumPy 的索引和切片都是基于 0 开始的。此外，NumPy 的切片操作返回的是原数组的视图 (view) 而不是副本 (copy)，因此对切片操作所得到的数组进行修改会直接影响到原数组。本话题后续将专门讲解视图和副本之间的区别。

本节配套的 Jupyter Notebook 文件是 Bk1_Ch14_01.ipynb。

14.2 一维数组索引、切片

索引

一维数组可以使用索引来访问和操作数组中的某个元素。如图 1 所示，索引是一个整数值，它指定了要访问的元素在数组中的位置。一维数组的索引从 0 开始，到数组长度 (len(a)) 减 1 结束。如图 1 所示，想要取出数组 a 的第一个元素，可以用 a[0] 或 a[-11]。a[-1] 或 a[10] 则取出数组 a 的最后一个元素。请大家在 BK_2_Topic_4.02_1.ipynb 尝试取出数组不同位置元素。

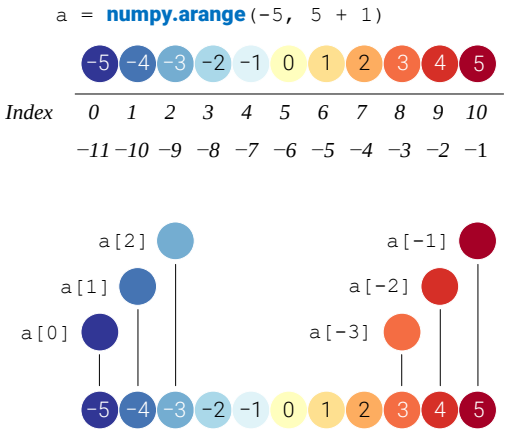


图 1. 一维数组的索引

行向量、列向量

上一个话题特别强调过，本书中行向量、列向量都被视作特殊的二维数组。也就是说，行向量是一行多列矩阵，而列向量是多行一列矩阵。

在 NumPy 中，**numpy.newaxis** 是一个特殊的索引，用于增加数组的维度。它的作用是在数组的某个位置添加一个新的轴，从而改变数组的维度。

具体来说，使用 **numpy.newaxis** 将会在数组的一个指定位置添加一个新的维度。如图 2 所示，对于一个一维数组 a，我们可以使用 a[:, **numpy.newaxis**] 将其转换为一个二维数组，其中新的维度被添加在列的方向上。这个操作将会把数组变成一个列向量。

a[**numpy.newaxis**, :] 则把一维数组变成行向量。本书后文还会介绍利用 **numpy.reshape**() 函数完成“升维”及其他变形。Bk1_Ch14_01.ipynb 还给出其他“升维”方法，请大家自行学习。

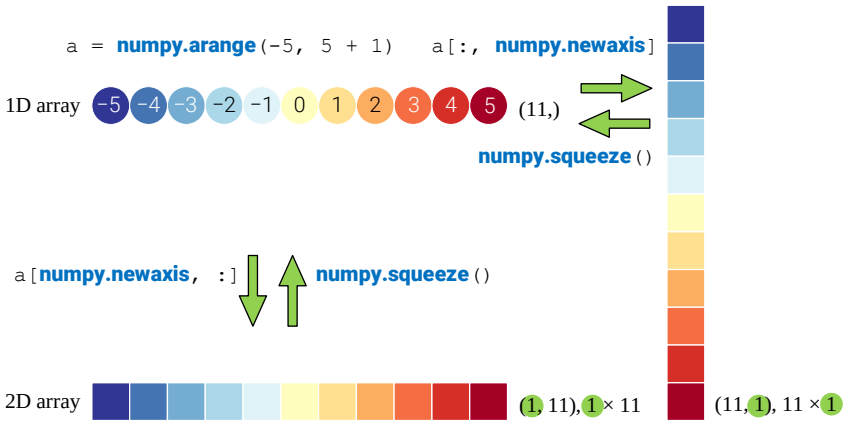


图 2. 一维数组“升维”

相反地，在 NumPy 中，**numpy.squeeze**() 函数用于从数组的形状中删除单维度的条目。这意味着它可以去掉数组中的长度为 1 的维度，并返回一个新的数组，其维度数目更少。

例如，对于一个形状为 (1, 3, 1, 5) 的四维数组，可以使用 **numpy.squeeze**(a) 函数将其转换为形状为 (3, 5) 的二维数组，其中长度为 1 的第 1 和第 3 维被删除。如果在调用 **numpy.squeeze**() 时指定了参数 axis，则只有该轴上长度为 1 的维度会被删除。

numpy.squeeze() 函数可以帮助我们简化数组的形状，使其更符合我们的需求。在某些情况下，例如在机器学习模型的输入中，我们需要使用具有特定形状的数组，而 **numpy.squeeze**() 可以帮助我们将数据变形为所需的形状。

切片

切片访问一维数组中的“子数组”，即多个元素。切片是一个包含开始索引和结束索引的范围，用冒号分隔。开始索引指定要获取的第一个元素的位置，结束索引指定要获取的最后一个元素的位置+1。

图 3 所示为一维数组连续切片。图 4 中，将步长设为 2 分别提取数组中的奇数、偶数。图 5 中，将步长设为-1 将数组倒序排序。Bk1_Ch14_01.ipynb 中还给出其他步长设置，请大家自行学习。

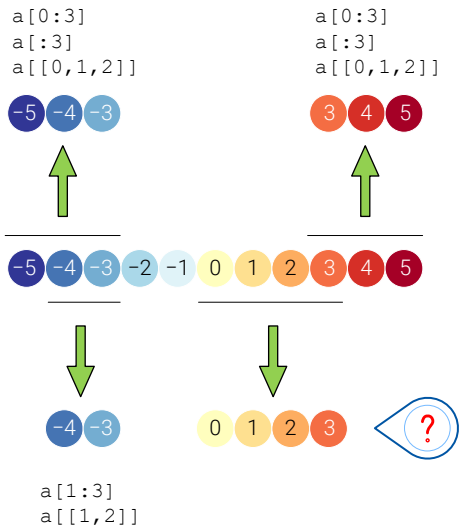


图 3. 一维数组连续切片

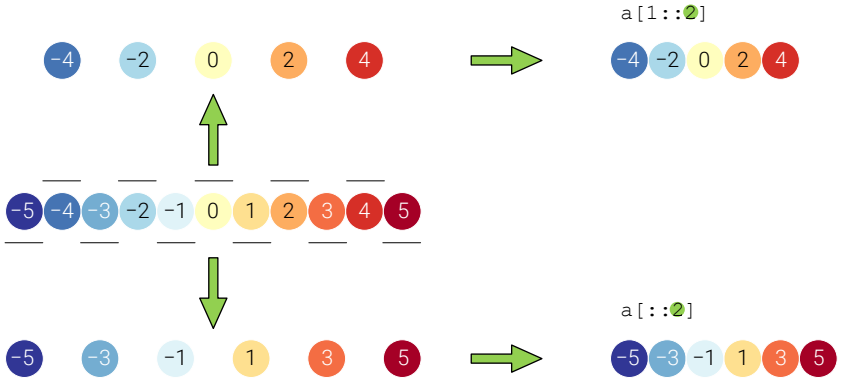


图 4. 一维数组以固定步长切片，步长为 2

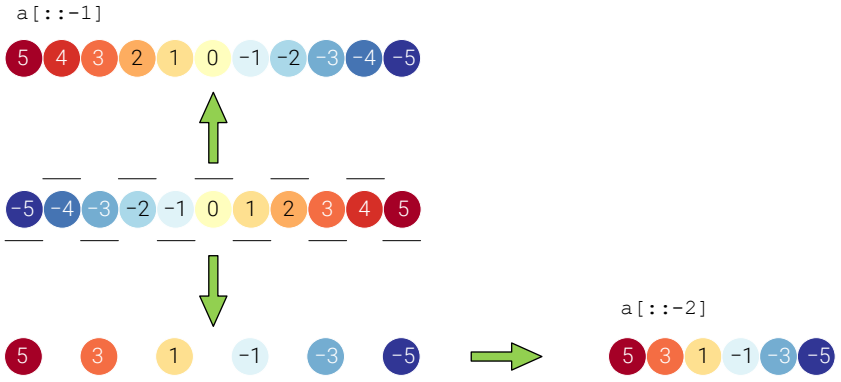


图 5. 一维数组倒序

整数索引、切片

在 NumPy 中，可以使用整数索引来访问和修改数组中的元素。整数索引是一种非常基本的索引方法，它允许使用一个整数或整数数组来访问数组的元素。

使用整数索引时，大家可以传递一个整数来访问数组的单个元素，或者传递一个整数数组来访问数组的多个元素。大家已经在图 1 看到这一点。

如果传递一个整数数组，则该数组的每个元素将被视为索引，从而返回一个新的数组，该数组包含原始数组中相应索引处的元素。如图 6 所示，整数索引为数组 [0, 1, 2, -1]，我们提取一维数组的第 1、2、3 和最后一个 (-1) 元素，结果还是一维数组。

同时，我们可以用 **np.r_**[0:3, -1]构造一个数组，也能提取相同的元素组合。**numpy.r_**() 是一个用于将切片对象转换为一个沿着第一个轴堆叠的 NumPy 数组的函数。它可以在数组创建和索引时使用。它的作用类似于 **numpy.concatenate**() 和 **numpy.vstack**()，但是使用切片对象作为索引来方便快捷地创建数组。

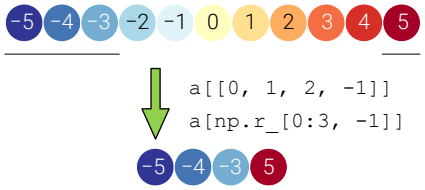


图 6. 一维数组整数索引，输入为数组

布尔索引、切片

布尔索引 (Boolean indexing) 是一种使用布尔值来选择数组中的元素的技术。在使用布尔索引时，可以通过一些条件来生成一个布尔数组，该布尔数组与要索引的数组具有相同的形状，然后使用该布尔数组来选择要访问的数组元素。图 7 所示为利用布尔值切片我们分别提取数组中大于 1、小于 0 的元素。

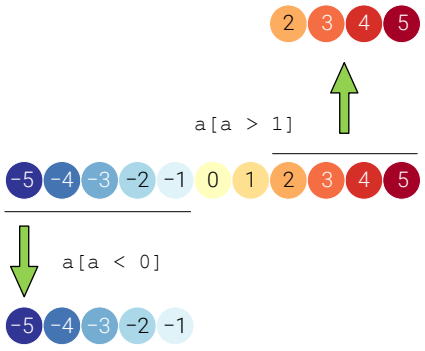


图 7. 一维数组布尔值切片

14.3 视图 vs 副本

在 NumPy 中，有两种不同的方式来创建新的数组对象：视图 (view) 和副本 (copy)。

视图是原始数组的一个新视图，而副本是原始数组的一个新副本。它们的区别在于它们如何处理原始数据的内存和共享。

视图是原始数组的一个新视图，一种重新排列、重新解释。视图是原始数组共享相同的数据，不会创建新的内存。换句话说，视图是原始数组的一个不同的“窗口”，它可以访问原始数组的相同数据块。当对视图进行更改时，原始数组也会发生相应的更改。

副本则是原始数组的一份完整的拷贝，修改副本不会影响原始数组。当对数组进行切片或使用 **numpy.copy**() 方法时，将生成一个副本。副本的创建可以使用 **numpy.copy**() 方法或者 **numpy.array**() 函数的参数 copy = True 来实现。

如图 8 所示，本节之前的各种索引、切片方法实际上创建的都只是原数组的视图，改变这些视图就会原数组，并“牵一发动全身”地改变所有视图。而 **numpy.copy**() 则创建了全新的内存，即副本。

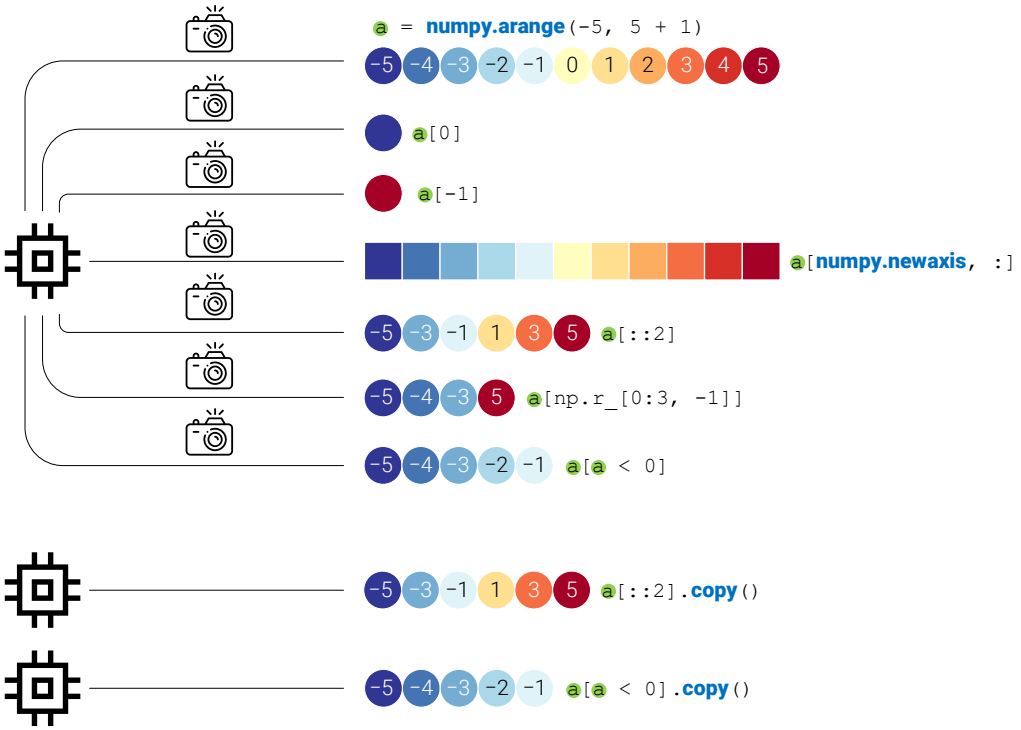


图 8. 视图，还是副本?

在 Bk1_Ch14_01.ipynb 这个示例中，首先创建了一个一维数组 a，然后创建了一个切片视图 s，该视图选择了数组 a 中的第二个和第三个元素。接下来，将视图中的第一个元素设置为 1000，这也会修改原始数组 a 中的元素。最后，用 a.**copy**() 创建了一个整数数组索引副本 c，该副本选择了数组 a 中的第二个和第四个元素。然后，将副本中的第一个元素设置为 888，但这不会修改原始数组 a 中的元素。本章后文的二维、三维数组在视图、副本方面的性质和一维数组完全一致。

可以使用 **numpy.may_share_memory**() 函数来判断两个数组是否共享内存。

在 NumPy 中，还有一些函数需要注意视图和副本的问题，比如 **numpy.reshape**()、**numpy.transpose**()、**numpy.ravel**()、**numpy.flatten**() 等等。这个话题非常重要，本书后文还会涉及。

14.4 二维数组索引、切片

取出单一元素

要取出二维 NumPy 数组中特定索引的元素，可以使用索引操作符 [] 来访问。可以将需要访问的元素的行索引和列索引作为参数传递给这个操作符。图 9 所示为从二维数组 a 中取出单一元素，a[0, 0] 代表第 0 行、第 0 列。请大家特别注意 a[[1], [2]] 的结果为一维数组。

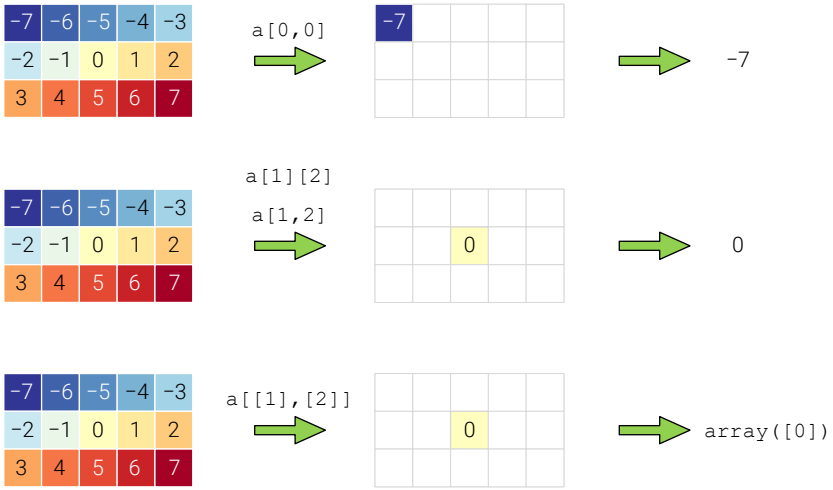


图 9. 取出单一元素

取出行

要取出二维 NumPy 数组中特定行的元素，也是使用索引操作符 [] 来访问。你可以将需要访问的行的索引作为第一个参数传递给这个操作符，用冒号 : 表示需要访问的列范围。图 10 所示，取出第 0 行，只需 a[0]，结果为一维数组。而 a[[0], :] 取出第 0 行，结果为二维数组。

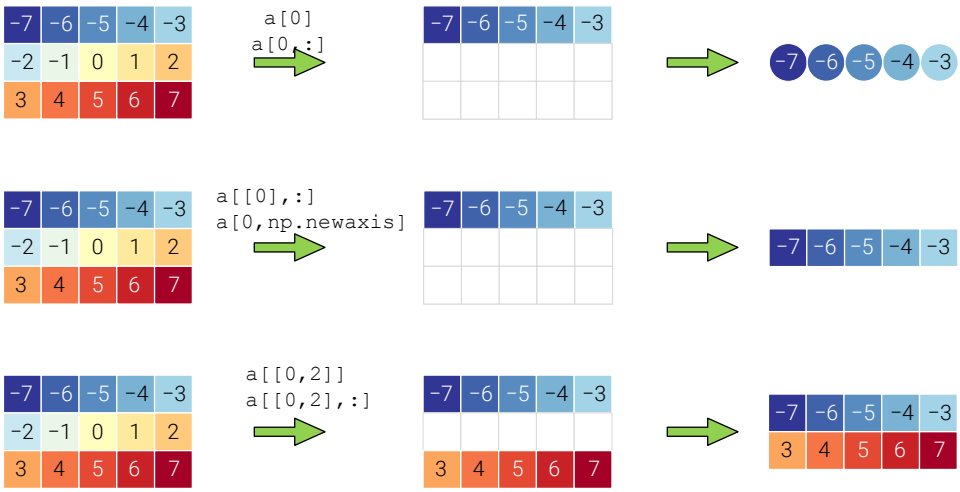


图 10. 取出行

取出列

类似地，如图 11 所示，我们也可以取出特定列。本书前文提过，**numpy.newaxis** 是一个常用的 NumPy 函数，它用于在数组中添加一个新的维度。具体来说，**numpy.newaxis** 用于在现有数组的指定位置插入一个新的维度，从而改变数组的形状。

注意，在 NumPy 多维数组的索引和切片操作中，省略号 ... 可以用于代替多个连续冒号 :，从而简化操作。具体来说，省略号可以用于表示在某个维度上使用完整的切片范围。需要注意的是，省略号只能在索引或切片操作的开头、结尾或中间使用，而不能重复出现。此外，当数组的维度比较大时，省略号可以显著提高代码的可读性和简洁性，因为它避免了写很多个冒号 : 的重复代码。

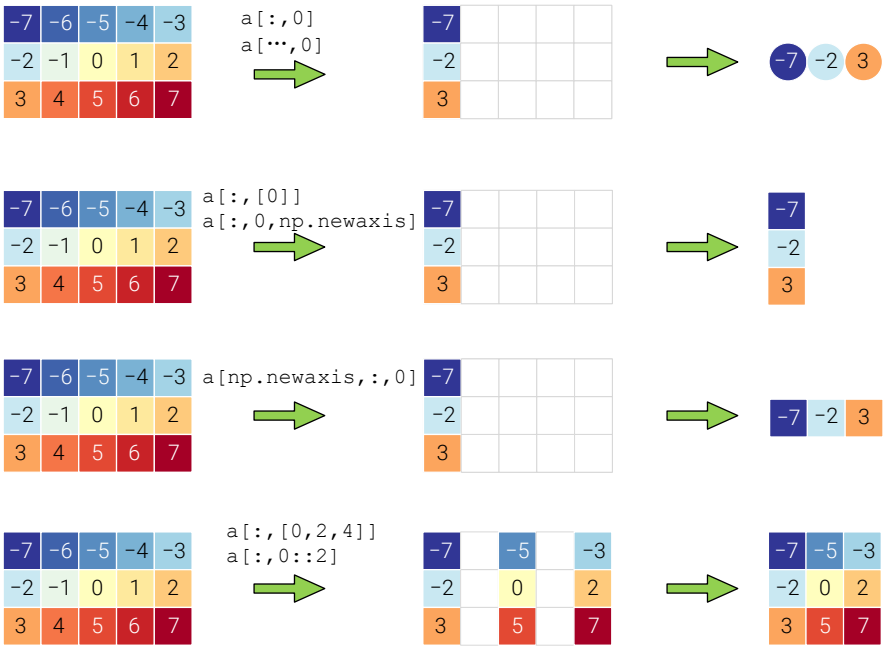


图 11. 取出列

图 12 所示为取出特定行列组合的方法。

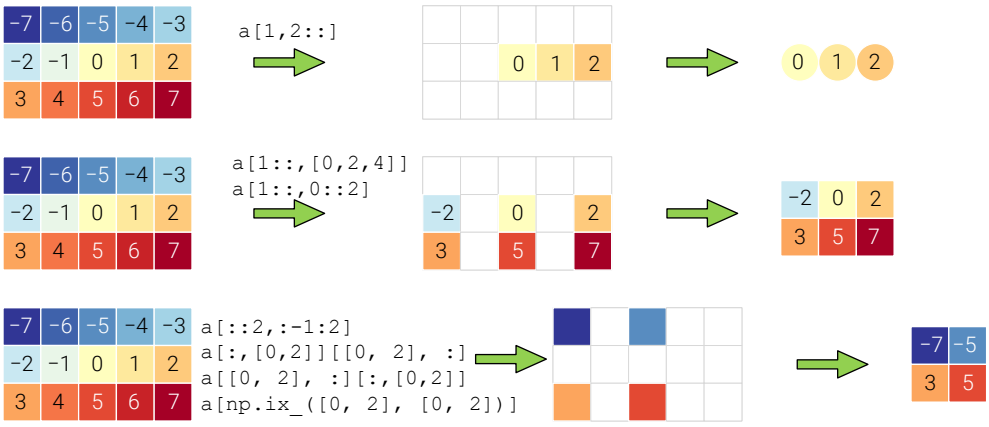


图 12. 取出特定的行列组合

图 12 中，**numpy.ix_**() 是 NumPy 提供的一个函数，用于将多个一维索引数组转换为一个用于多维数组索引的元组。这个元组可以用于同时对多个维度进行索引，从而方便地选择数组中的子集。使用 **numpy.ix_**() 可以让代码更加简洁和易读，避免了使用多个索引数组或切片来对多维数组进行索引的复杂性和难以理解的问题。在科学计算和数据分析中，使用 **numpy.ix_**() 可以方便地进行数据筛选和子集提取，提高代码效率和可读性。

布尔索引、切片

类似本章前文，二维数组也可以采用布尔索引、切片。举个例子，如图 13 所示，取出二维数组大于 0 的元素，结果为一元数组。本章配套代码还提供其他输出形式，请大家自行学习。

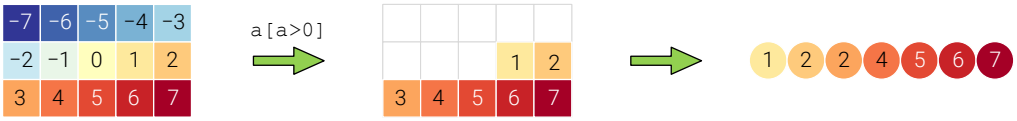


图 13. 取出大于 0 的元素

本章配套代码还介绍如何对三维数组进行索引、切片，也请大家自行学习。

请大家完成下面 3 道题目。

Q1. 创建一个一维数组，形状为 (10,)，用满足在 [−1, 1] 均匀分布随机数填充。切片操作提取前 5 个元素，并将结果倒序输出。

Q2. 创建一个二维数组，形状为 (3, 4)，用满足在 [−1, 1] 均匀分布随机数填充。使用切片操作选取其中的第一行和第三行。同时，使用切片操作取出第二、四列。

Q3. 创建一个三维数组，形状为 (4, 5, 6)，用满足在 [−1, 1] 均匀分布随机数填充。使用切片操作选取其中的 axis = 0、1 维度上的所有元素，以及 axis = 2 维度上的前两个元素。

* 题目不提供答案。

15

Basic Computations in NumPy

NumPy 常见运算

使用 NumPy 完成算数、代数、统计运算

生活只有两件好事：发现数学和教数学。

Life is good for only two things: discovering mathematics and teaching mathematics.

—— 西梅翁·德尼·泊松 (Siméon Denis Poisson) | 法国数学家 | 1781 ~ 1840

15.1 加减乘除乘幂

在 NumPy 中，基本的加减乘除、乘幂运算如下：

- 加法：使用 + 运算符或 **numpy.add()** 函数实现。
- 减法：使用 - 运算符或 **numpy.subtract()** 函数实现。
- 乘法：使用 * 运算符或 **numpy.multiply()** 函数实现。
- 除法：使用 / 运算符或 **numpy.divide()** 函数实现。
- 乘幂：使用 ** 运算符或 **numpy.power()** 函数实现。

下面，我们先聊一聊相同形状的数组之间的加减乘除乘幂运算。

本节配套的 Jupyter Notebook 文件是 Bk1_Ch15_01.ipynb。

一维数组

图 1 所示为两个等长度一维数组之间的加、减、乘、除、乘幂运算。这一组运算都是逐项完成，也就是对应位置完成运算。

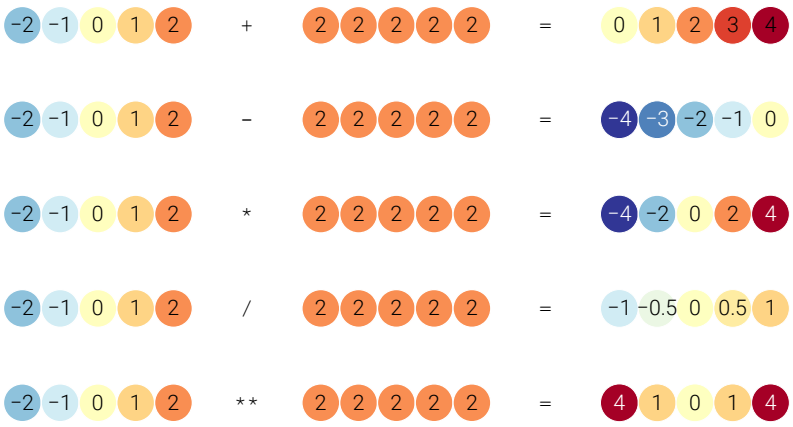


图 1. 一维数组加、减、乘、除、乘幂

二维数组

图 2 所示为二维数组之间的加、减、乘、除、乘幂运算。类似运算也可以用在三维、多维数组上。

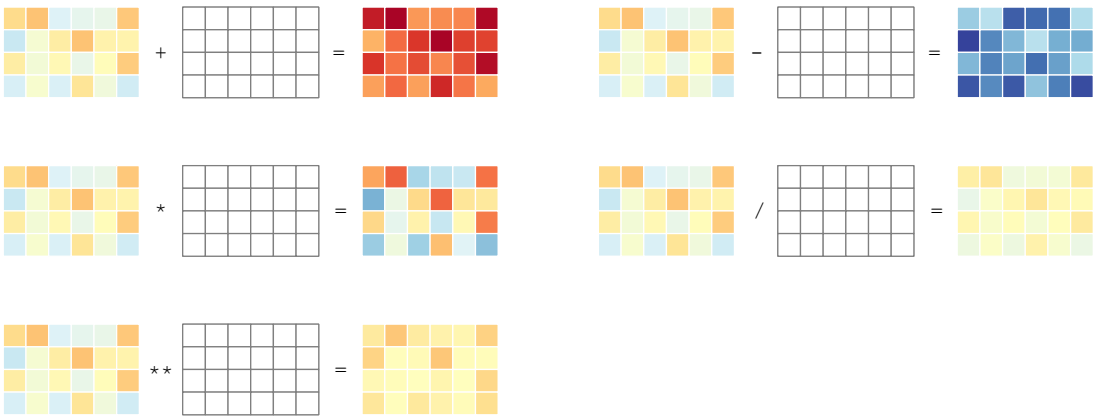


图 2. 二维数组加、减、乘、除、乘幂

15.2 广播原则

简单来说，NumPy 的广播原则 (broadcasting) 指定了不同形状的数组之间的算术运算规则，将形状较小的数组扩展为与形状较大的数组相同，再进行运算，以提高效率。

下面，我们首先以一维数组为例介绍什么是广播原则。

一维数组和标量

图 3 所示一维数组和标量之间完成加、减、乘、除、乘幂运算，大家可以发现图 3 可以替代

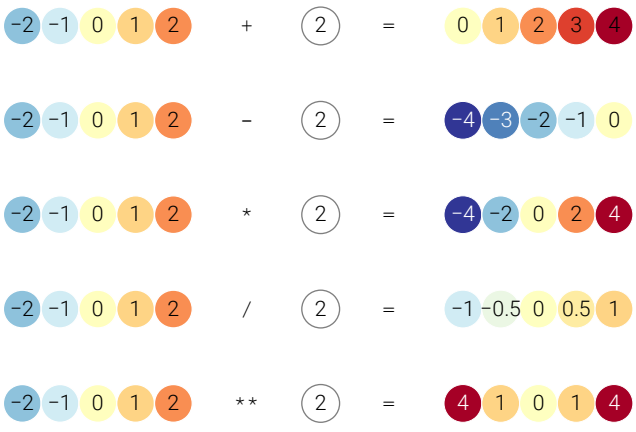


图 3. 一维数组和标量加、减、乘、除、乘幂，广播原则

一维数组和列向量

图 4 和图 5 所示为将广播原则用在一维数组和列向量的加法和乘法上。广播过程相当于把一维数组 (5,) 展成 (3, 5) 二维数组，把列向量 (3, 1) 也展成 (3, 5) 二维数组。运算结果也是二维数组。

这两幅图中，大家还会看到，行向量、列向量之间的运算也可以获得同样的结果，请大家在 JupyterLab 中自己完成。

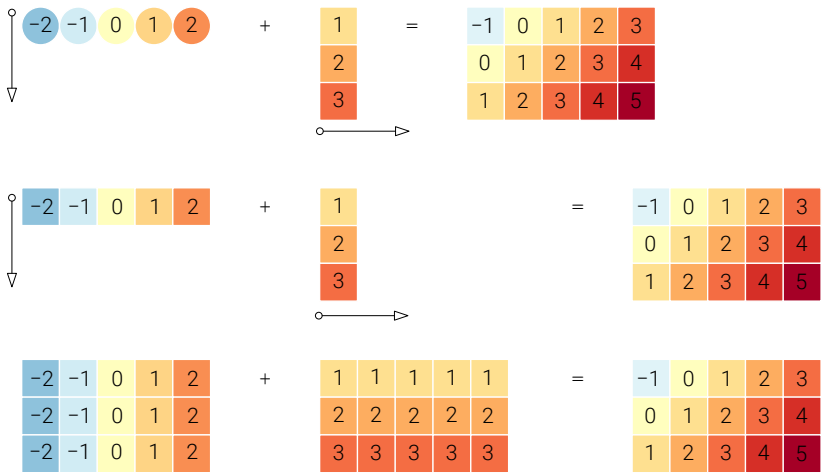


图 4. 一维数组和列向量加法，广播原则

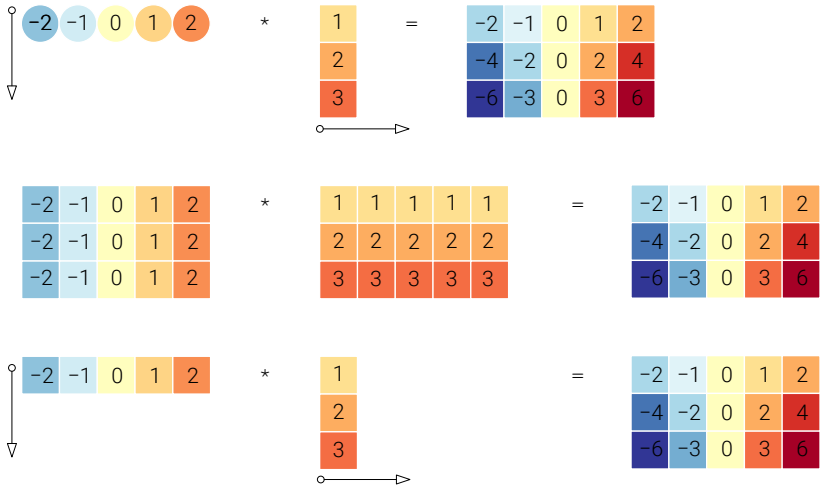


图 5. 一维数组和列向量乘法，广播原则

二维数组和标量

图 6 所示二维数组和标量的运算相当于图 2。

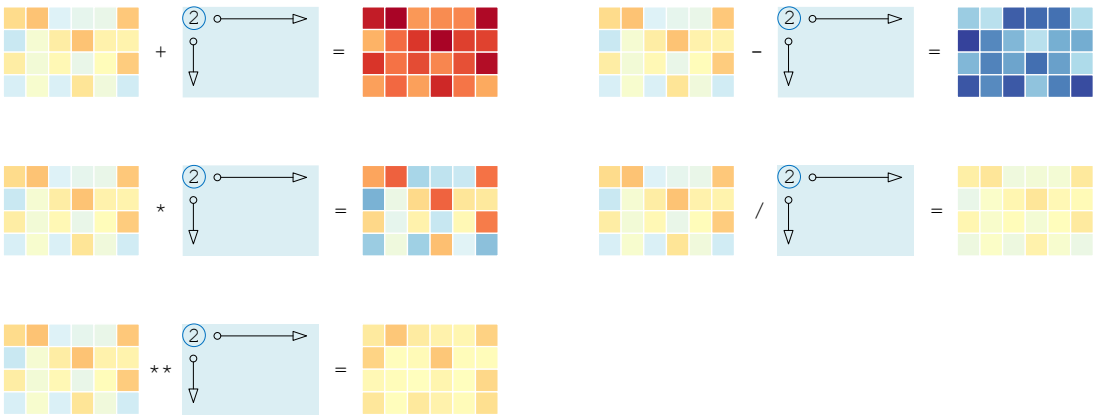


图 6. 二维数组和标量加、减、乘、除、乘幂，广播原则

二维数组和一维数组

图 7 所示为二维数组和一维数组之间的广播原则运算。二维数组的形状为 (4, 6)，一维数组的形状为 (6,)。

图 7 等价于图 8。图 8 中，行向量是二维数组，形状为 (1, 6)。

注意，当前 NumPy 不支持 (4, 6) 和 (4,) 之间的广播运算，会报错。这种情况，要用 (4, 6) 和 (4, 1) 之间的广播原则。

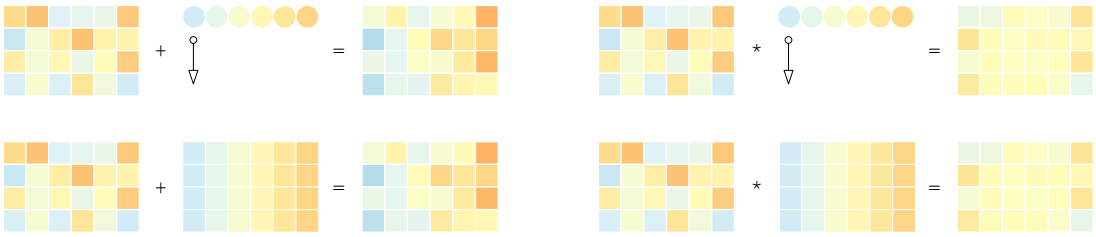


图 7. 二维数组和一维数组加、乘，广播原则

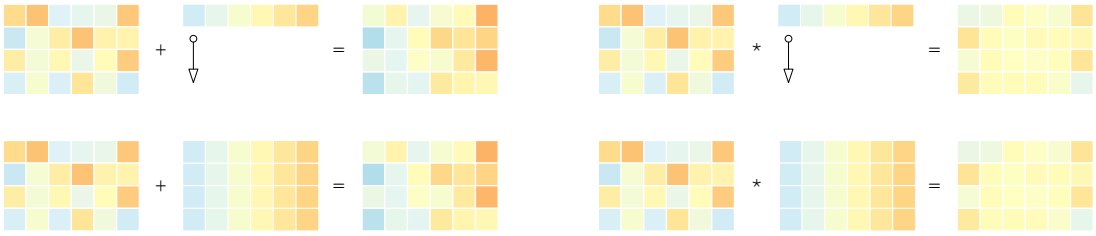


图 8. 二维数组和行向量加、乘，广播原则

二维数组和列向量

图 9 所示为二维数组和列向量之间的广播运算。二维数组的形状为 (4, 6)，列向量形状为 (4, 1)。它们在行数上匹配。

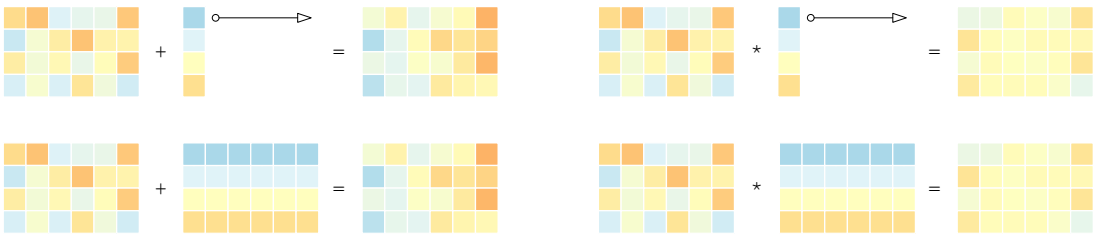


图 9. 二维数组和列向量加、乘，广播原则

15.3 常见函数

NumPy 还提供大量常用函数，如图 10 所示。NumPy 中还给出很多常用常数，比如 **numpy.pi** (圆周率)、**numpy.e** (欧拉数、自然底数)、**numpy.Inf** (正无穷)、**numpy.NAN** (非数) 等等。

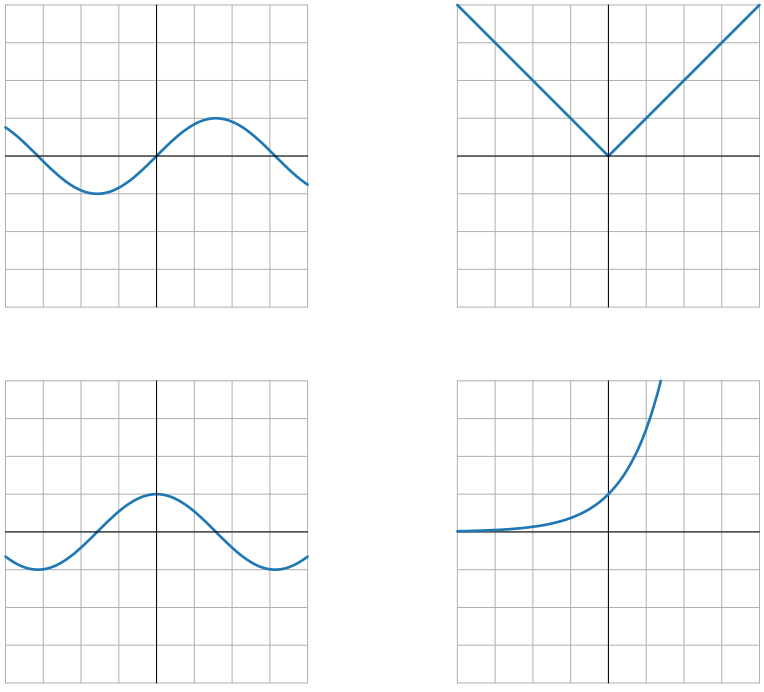


图 10. 四个常用函数

15.4 统计运算

图 11 所示为求最大值的操作。给定二维数组 A，A.max() 计算整个数组中最大值。而 A.max(axis = 0) 沿行最大值，结果为一维数组。A.max(axis = 1) 沿列最大值，结果同样为一维数组。A.max(axis = 1, keepdims = True) 的结果为列向量。

此外，计算最小值、求和、均值、方差、标准差等统计运算遵循相同的规则，请大家参考本章 Jupyter Notebook。

注意，计算方差、标准差时，NumPy 默认分母为 n (样本数量)，而不是 $n-1$；为了计算样本方差或标准差，需要设定 ddof = 1。

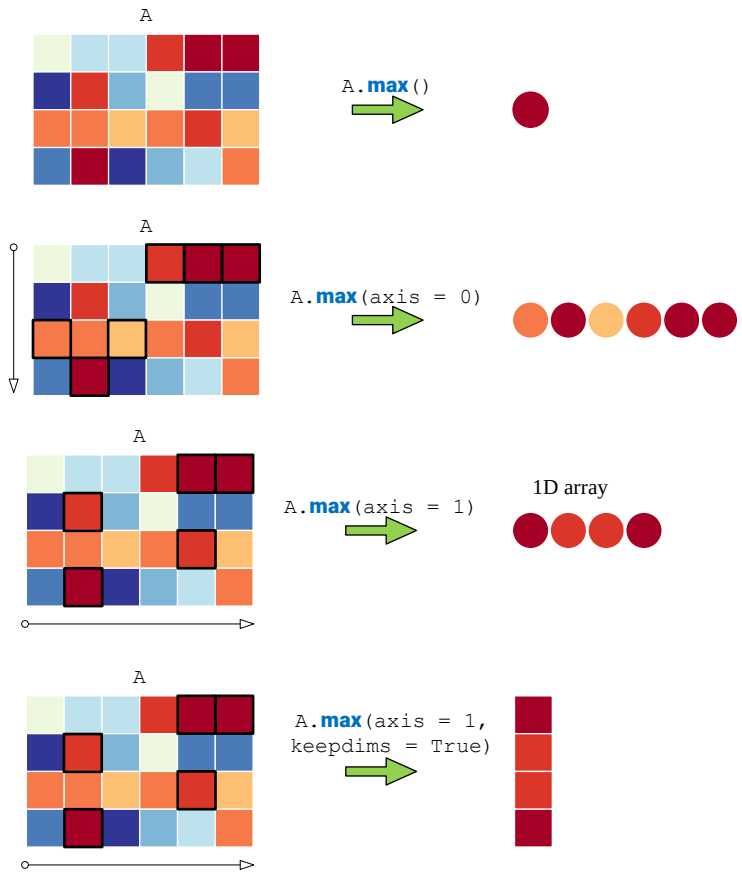


图 11. 沿不同轴求最大值

什么是方差？

方差是统计学中衡量数据分散程度的一种指标，用于衡量一组数据与其平均值之间的偏离程度。方差的计算是将每个数据点与平均值的差的平方求和，并除以数据点的个数 n 减 1，即 $n-1$。方差越大，数据点相对于平均值的离散程度就越高，反之亦然。方差常用于数据分析、建模和实验设计等领域。方差开平方结果为标准差。

NumPy 还提供计算协方差矩阵、相关性系数矩阵的函数。图 12 (a) 所示为鸢尾花数据协方差矩阵，图 12 (b) 为相关性系数矩阵。

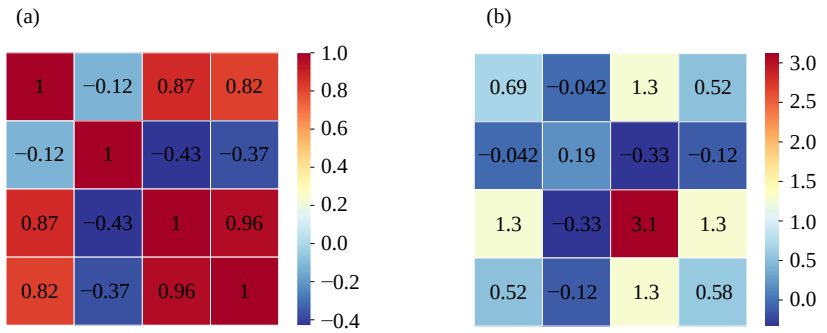


图 12. 鸢尾花数据协方差矩阵、相关性系数矩阵

什么是协方差矩阵？

协方差矩阵是一个方阵，其中的元素代表了数据中各个维度之间的协方差。协方差是用来衡量两个随机变量之间的关系的统计量，它描述的是两个变量的变化趋势是否相似，以及它们之间的相关性强度。协方差矩阵可以用于多变量分析和线性代数中的特征值分解、奇异值分解等计算。在机器学习领域，协方差矩阵常用于数据降维、主成分分析、特征提取等方面。

什么是相关系数矩阵？

相关性系数矩阵是一个方阵，其中的元素代表了数据中各个维度之间的相关性系数。相关性系数是用来衡量两个变量之间线性关系的程度，它取值范围在-1 到 1 之间，数值越接近于 1 或-1，说明两个变量之间的线性关系越强；数值越接近于 0，说明两个变量之间的线性关系越弱或不存在。相关性系数矩阵可以用于多变量分析、线性回归等领域，通常与协方差矩阵一起使用。在机器学习领域，相关性系数矩阵常用于特征选择和数据可视化等方面。

请大家完成下面 3 道题目，它们的目的都是利用 NumPy 计算并可视化公式。

Q1. 给定如下一元高斯函数，参数 $a = 1, b = 2, c = 1$。请用 NumPy 和 Matplotlib 线图可视化函数函数图像。

$$f(x) = a\exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

Q2. 给定如下二元高斯函数。请用 NumPy 和 Matplotlib 三维网格面可视化二元函数图像。

$$f(x_1, x_2) = \exp\left(-x_1^2 - x_2^2\right)$$

Q3. 下式为二元高斯分布的概率密度函数，请用 NumPy 和 Matplotlib 填充等高线可视化这个二元函数图像。参数具体为 $\mu_X = 0$, $\mu_Y = 0$, $\sigma_X = 1$, $\sigma_Y = 1$, $\rho_{X,Y} = 0.6$。

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}}\exp\left(-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]\right)$$

* 题目答案请参考 Bk1_Ch15_02.ipynb。

16

Reshaping NumPy Arrays

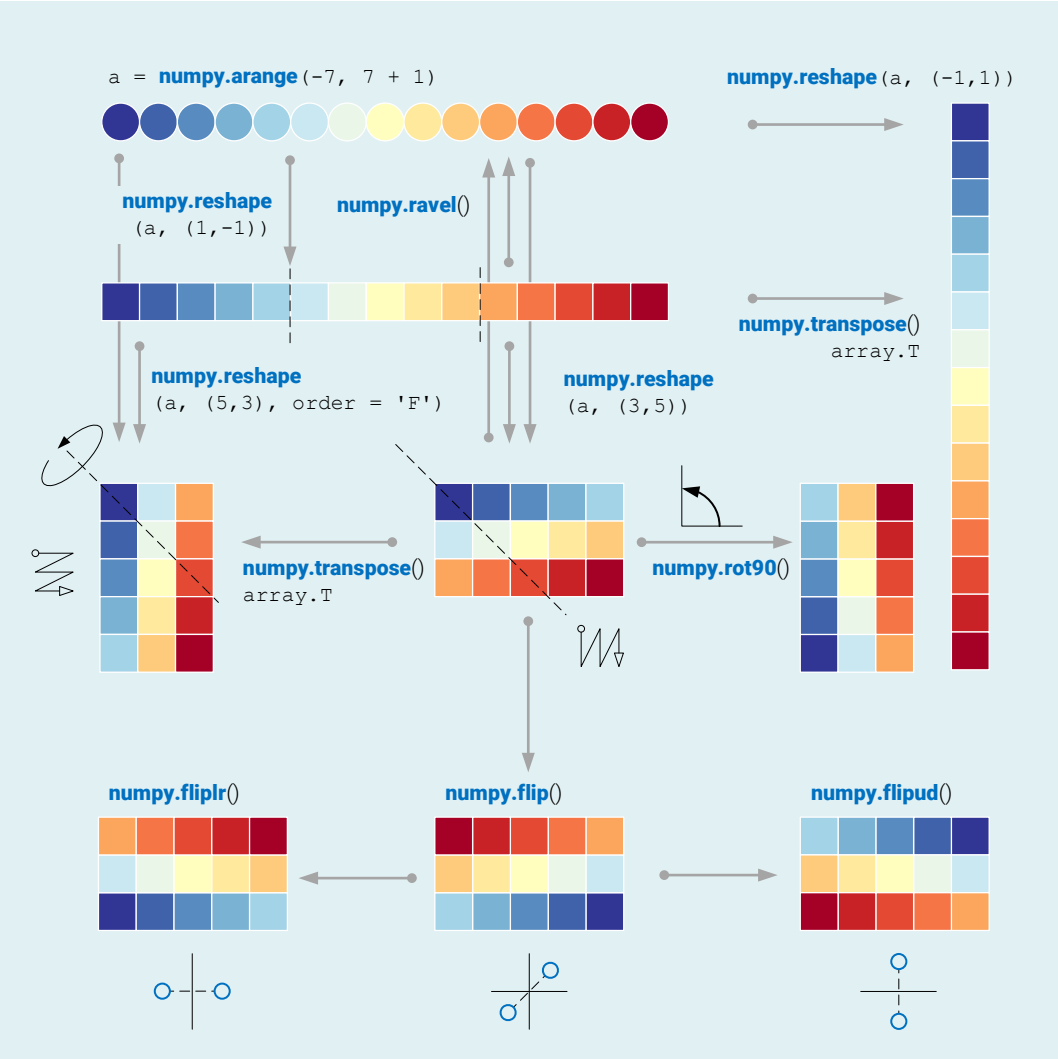
NumPy 数组变形

重塑数组的维数、形状

哪里有物质，哪里就有几何学。

Where there is matter, there is geometry.

—— 约翰内斯·开普勒 (Johannes Kepler) | 德国天文学家、数学家 | 1571 ~ 1630



16.1 从 reshape() 函数说起

在 NumPy 中，要改变数组的形状 (也称重塑数组)，可以使用 **numpy.reshape**() 函数。**reshape**() 函数允许你指定一个新的形状，然后返回一个拥有相同数据但具有新形状的数组。

下面我们先了解一下这个话题的核心函数——**numpy.reshape**()。

numpy.reshape(a, newshape, order='C')

这个函数的重要输入参数：

- a 参数是要被重塑的数组，可以是一个数组对象，也可以是一个 Python 列表、元组等支持迭代的对象。
- newshape 参数是新的形状，可以是一个整数元组或列表，也可以是一个整数序列。
- order 参数表示重塑数组的元素在内存中存储的顺序，可以是 'C' (按行顺序存储) 或 'F' (按列顺序存储)，默认值为 'C'。

下面是 numpy.reshape() 函数一些常见用法：

a) 改变数组的维度：可以将一个数组从一维改为二维、三维等。例如：

```
import numpy as np
a = np.arange(12)             # 创建一个长度为 12 的一维数组
b = np.reshape(a, (3, 4))     # 改变为 3 行 4 列的二维数组
c = np.reshape(a, (2, 3, 2)) # 改变为 2 个 3 行 2 列的三维数组
```

b) 展开数组：可以将一个多维数组展开为一维数组。例如：

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.reshape(a, -1)   # 将二维数组展开为一维数组
```

c) 改变数组的顺序：可以改变数组在内存中的存储顺序。例如：

```
import numpy as np
a = np.arange(6).reshape((2, 3))       # 创建一个 2 行 3 列的二维数组
b = np.reshape(a, (3, 2), order='F')   # 按列顺序存储
```

注意：numpy.reshape() 函数并不会改变数组的数据类型和数据本身，只会改变其形状。如果改变后的形状与原数组的元素数量不一致，将会抛出 ValueError 异常。

请大家在 JupyterLab 中自行运行如上三段代码。

更多有关 numpy.reshape() 函数的用法，请大家参考如下技术文档：

https://numpy.org/doc/stable/reference/generated/numpy.reshape.html

下面结合实例详细讲解如何利用 numpy.reshape() 完成数组变形。

本节配套的 Jupyter Notebook 文件是 BK_2_Topic_4.04_1.ipynb。

16.2 一维数组 → 行向量、列向量

一维数组 → 行向量

本书前文提过，行向量、列向量都是特殊矩阵。因此，行向量、列向量都是二维数组。也就是说，行向量是一行若干列的数组，形状为 $1 \times D$。列向量是若干行一列的数组，形状为 $n \times 1$。

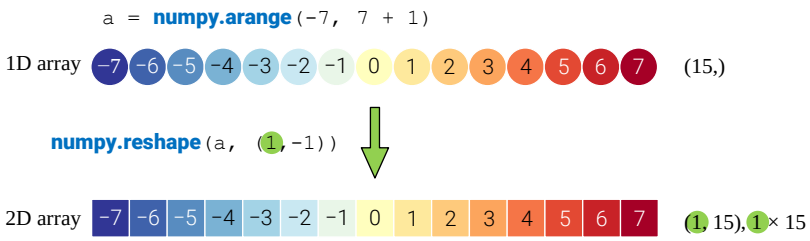


图 1. 将一维数组转换为行向量

如图 1 所示，用 a = **numpy.arange**(-7, 7+1) 生成的是一个一维数组 a，这个数组有 15 个元素。由于数组为一维，所以可视化时采用了“圆圈”，而不是方块。利用 **numpy.reshape**(a, (1, -1))，我们将 a 转化为形状为 (1, 15) 的二维数组，也称行向量，即 1 × 15 矩阵。

⚠ 注意，使用 -1 作为形状参数时，**numpy.reshape**() 会根据数组中的数据数量和其它指定的维数来自动计算该维度的大小。

一维数组 → 列向量

如图 2 所示，利用 **numpy.reshape**(a, (1, -1))，我们可以把一维数组 **numpy.arange**(-7, 7+1) 转化为形状为 (15, 1) 的二维数组，也称列向量，即 15 × 1 矩阵。

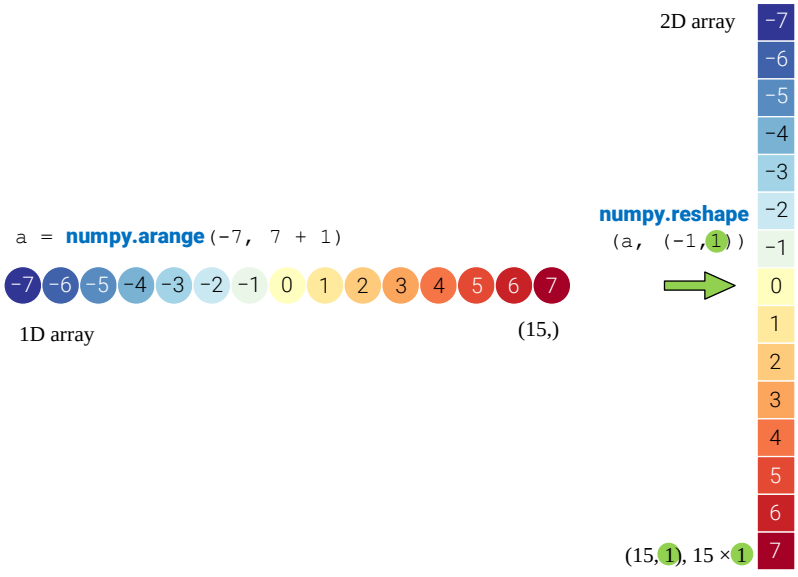


图 2. 将一维数组转换为列向量

16.3 一维数组 → 二维数组

用 a = **numpy.arange**(-7, 7+1) 生成的数组有 15 个元素，可以被 3、5 整除，因此一维数组 a 可以写成 3 × 5 矩阵。如图 3 所示，我们可以分别按先行后列、先列后行两种形式重塑数组。

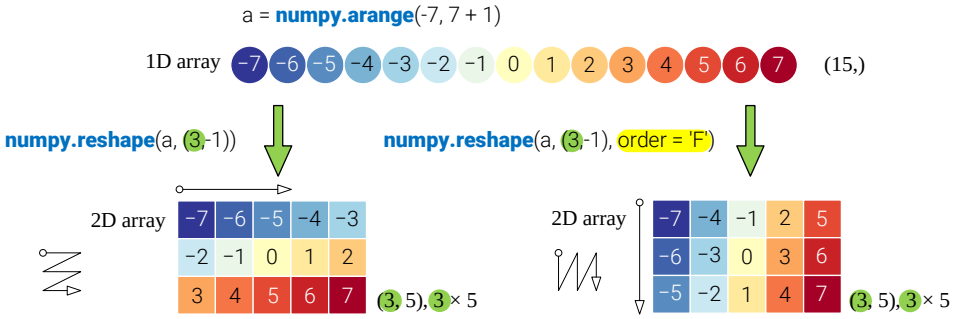


图 3. 将一维数组转换为 3 × 5 矩阵，先行后列，先列后行

图 4 所示为将 **numpy.arange**(-7, 7+1) 一维数组写成 5 × 3 矩阵。图 4 给出了先行后列、先列后行两种顺序。如图 5 所示已经完成转换的 3 × 5 数组，通过 **numpy.reshape**() 可以进一步转化为 5 × 3 数组。

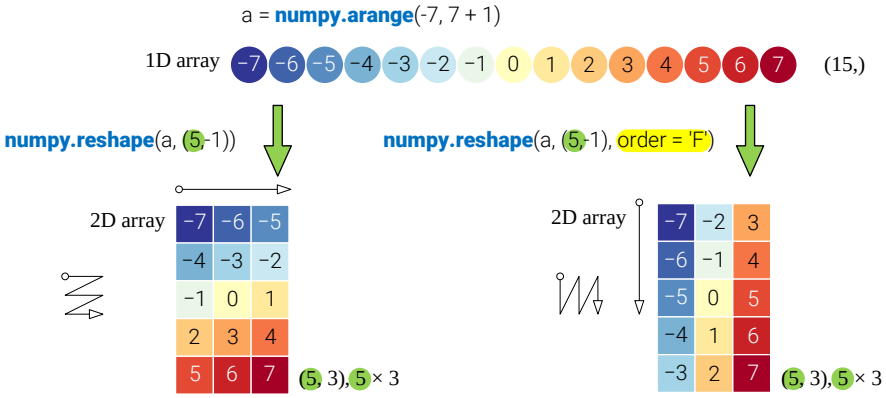


图 4. 将一维数组转换为 5 × 3 矩阵，先行后列，先列后行

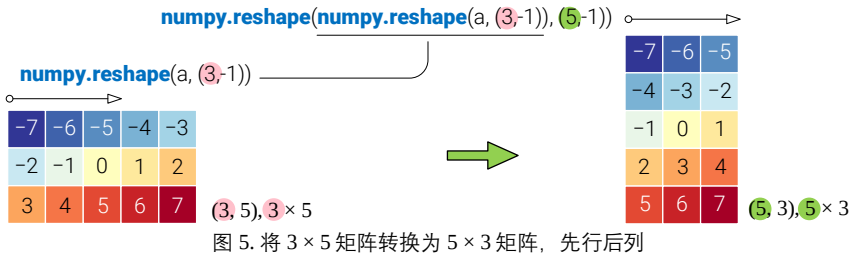


图 5. 将 3 × 5 矩阵转换为 5 × 3 矩阵，先行后列

16.4 一维数组 → 三维数组

图 6 所示为将 **numpy.arange**(-13, 13+1) 一维数组转化成形状为 3 × 3 × 3 的三维数组。

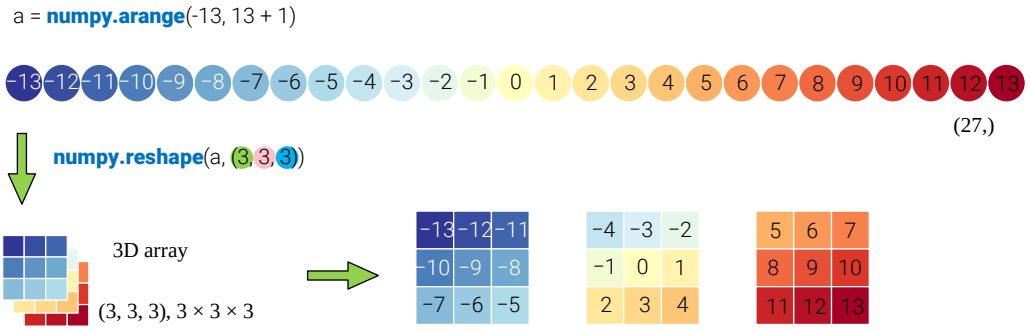


图 6. 将一维数组转换为三维数组

16.5 视图 vs 副本

本书前文特别提过，NumPy 中要特别注意视图 (view)、副本 (copy) 的区别。简单来说，视图和副本是 NumPy 中的两种不同的数组对象。

视图是指一个数组的不同视角或者不同形状的表现方式，视图和原始数组共享数据存储区，因此在对视图进行操作时，会影响原始数组的数据。视图可以通过数组的切片、转置、重塑等操作创建。

副本则是指对一个数组的完全复制，副本和原始数组不共享数据存储区，因此对副本进行操作不会影响原始数组。使用 **numpy.reshape**() 也需要注意视图、副本问题。

本节配套的 Jupyter 笔记中，大家可以看到，我们用 **numpy.shares_memory**() 判断两个数组是否指向同一个内存。

如图 7 所示，**numpy.reshape**() 仅仅改变了观察同一数组的视角，也就是改变了 index。

⚠ 注意，不同函数的历史、未来版本可能存在不一致，需要大家自行判断。

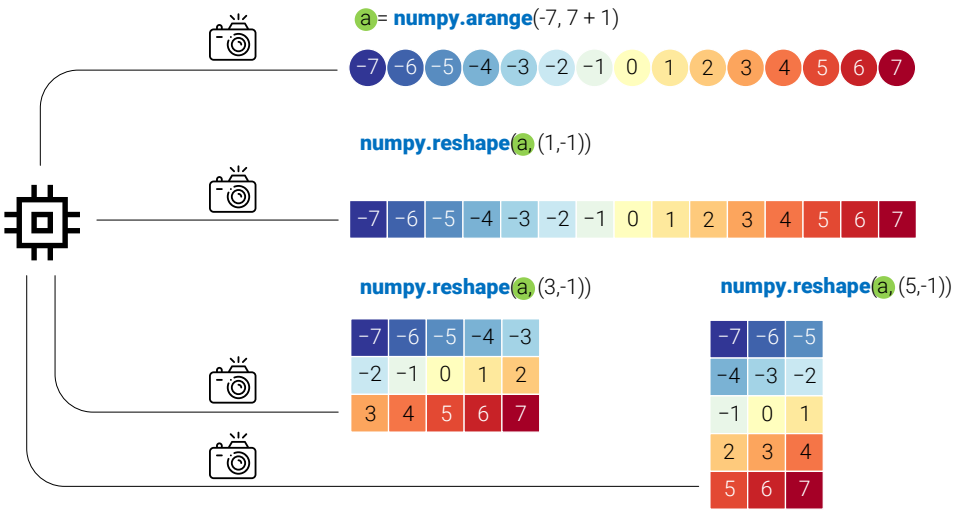


图 7. 视图，还是副本？

16.6 转置

如图 8 所示，一个 $n \times D$ 矩阵 $\boldsymbol{A}$ 转置得到 $D \times n$ 矩阵 $\boldsymbol{B}$，整个过程相当于矩阵 $\boldsymbol{A}$ 绕主对角线镜像。具体来说，矩阵 $\boldsymbol{A}$ 位于 (i, j) 的元素转置后的位置为 (j, i)，即行列序号互换。这就是，为什么位于主对角线上的元素转置前后位置不变。矩阵 $\boldsymbol{A}$ 的转置 (the transpose of a matrix $\boldsymbol{A}$) 记作 $\boldsymbol{A}^{\mathrm{T}}$ 或 $\boldsymbol{A}'$。为了和求导记号区分，本书仅采用 $\boldsymbol{A}^{\mathrm{T}}$ 记法。

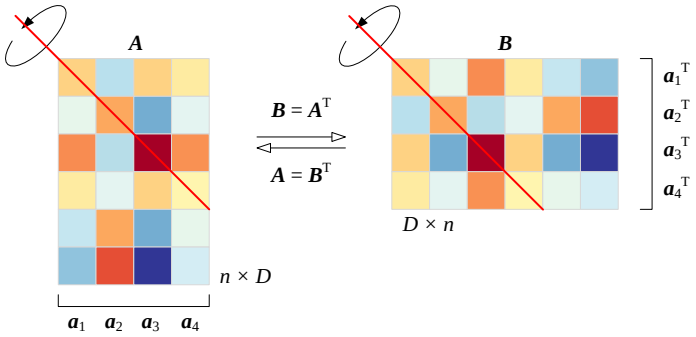


图 8. 矩阵转置，图片来自《矩阵力量》第 4 章

⚠ 需要大家特别注意的是，NumPy 的 **numpy.transpose**() 方法和 .T 属性都返回原始数组的转置，两者都返回原始数组的视图，而不是副本。

➔ “鸢尾花书”中《矩阵力量》第 4 章将专门讲解矩阵的转置运算。

图 9 所示为二维数组的转置。行向量转置得到列向量，反之亦然。3 × 5 矩阵转置得到 5 × 3 矩阵。而一维数组的转置不改变形状。

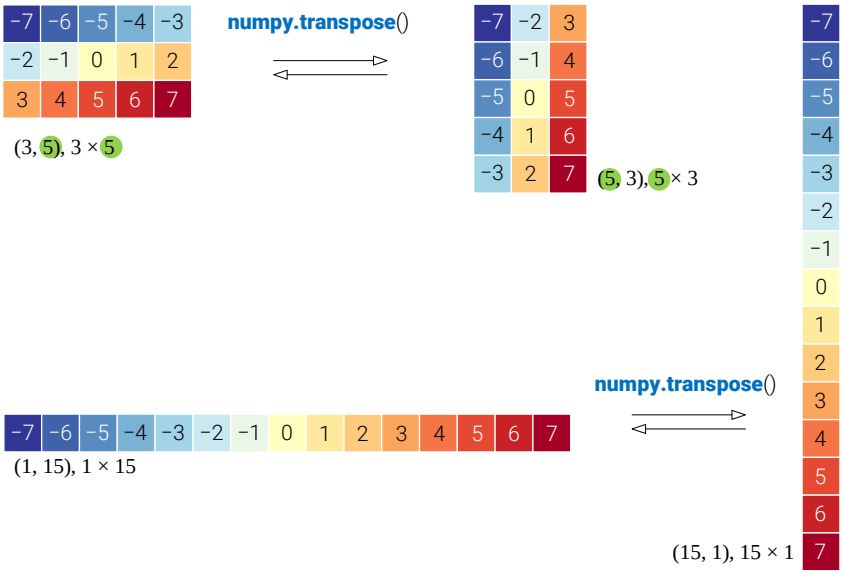


图 9. 二维数组的转置

16.7 扁平化

扁平化可以理解图 1、图 2、图 3 等 **numpy.reshape()** 的“逆操作”。完成扁平化的方法有很多，比如 **array.ravel()**、**array.reshape(-1)**、**array.flatten()**。大家也可以使用 **numpy.ravel()**、**numpy.flatten()** 这两个函数。图 10 所示为将二维转化为一维数组。

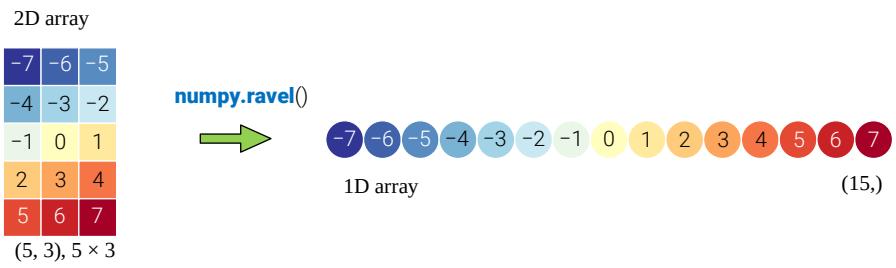


图 10. 二维数组转化为一维数组

请大家格外注意，**ravel()**、**reshape(-1)** 返回的是原始数组的视图，而不是其副本。因此，如果修改新数组中的任何元素，原始数组也会受到影响。如果需要返回一个数组副本，可以使用 **flatten()**函数。本节配套的 Jupyter 笔记中给出一个详细的例子，请大家自行学习。

16.8 其他操作

如图 11 所示，**numpy.rot90**() 的作用是将一个数组逆时针旋转 90 度。默认情况下，这个函数会将数组的前两个维度 axes=(0, 1) 进行旋转。此外，还可以利用参数 k (正整数) 逆时针旋转 $k \times 90$ 度。默认，$k = 1$。

注意，**numpy.rot90**() 的结果也是返回原始数组的视图，而不是副本。

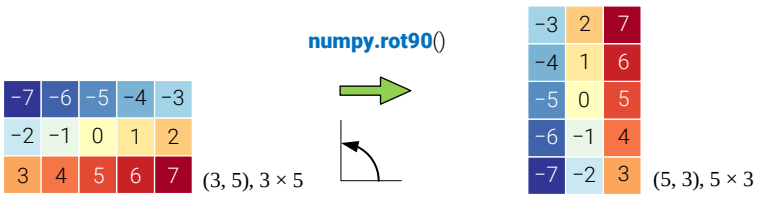


图 11. 3 × 5 矩阵逆时针旋转 90 度

numpy.flip() 函数用于翻转数组中的元素，即将数组沿着一个或多个轴翻转。**numpy.flip**(A, axis=None) 中，A 是要进行翻转的数组，axis 指定要翻转的轴。如图 12 所示，如果不指定 axis，则默认将整个数组沿着所有的轴进行翻转。类似的函数还有 **numpy.fliplr**()、**numpy.flipud**()，请大家自行学习。

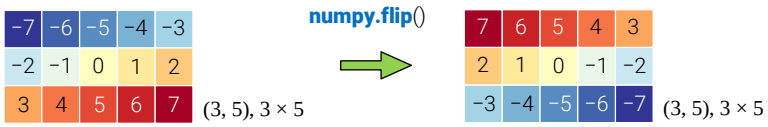


图 12. 3 × 5 矩阵沿着两个轴翻转

下面，是有关使用 **numpy.reshape**() 函数的三道习题，请大家完成。

Q1. 首先生成一个一维数组 [1, 2, 3, 4, 5, 6]，然后将其转换为一个形状为 (2, 3) 的二维数组，并打印结果。注意，元素按先行后列顺序存储。最后，想办法判断转换前后的数组是视图，还是副本。

Q2. 将一个二维数组 [[1, 2], [3, 4], [5, 6]] 转换为一个形状为 (6,) 的一维数组，并打印结果。注意，按先列后行顺序存储。

Q3. 将一个三维数组 [[[1, 2], [3, 4]], [[5, 6], [7, 8]]] 转换为一个形状为 (2, 4) 的二维数组，并按列顺序存储，最后打印结果。

* 这三道题目很基础，本书不给答案。

17

Manipulating NumPy Arrays

NumPy 数组规整

重塑数组的维数、形状

我不能教任何人任何东西。我只能让他们思考。

I cannot teach anybody anything. I can only make them think.

—— 苏格拉底 (Socrates) | 古希腊哲学家 | 470 ~ 399 BC

本书前文介绍的 **numpy.swapaxes**()、**numpy.reshape**()、**numpy.resize**()、**numpy.transpose**()、**numpy.squeeze**()、**numpy.ravel**()等等都算是对 NumPy 数组进行规整的函数。本章将介绍其他几种常用规整函数。

17.1 堆叠

沿行堆叠

用 **numpy.arange**() 产生如图 1 所示的两个一维等长数组。图 2 所示为三种办法将两个等长一维数组沿行 axis = 0 方向堆叠，结果为二维数组。

numpy.stack() 函数将沿着指定轴将多个数组堆叠在一起，返回一个新的数组；默认轴为 axis = 0。**numpy.row_stack**() 函数将多个数组沿着行方向进行堆叠，生成一个新的数组。**numpy.vstack**() 将多个数组沿着垂直方向（行方向）进行堆叠，生成一个新的数组。

图 1. 两个等长一维数组

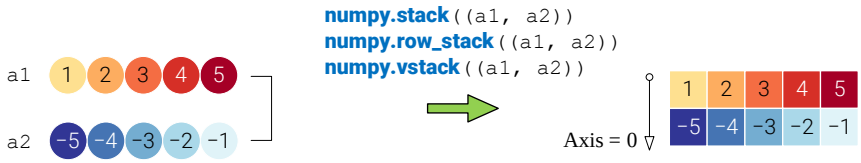


图 2. 沿行 axis = 0 方向堆叠

沿列堆叠

图 3 所示为沿列 axis = 1 方向堆叠两个一维等长数组。图中给出两种办法。

numpy.column_stack() 将多个一维数组沿着列方向进行堆叠，生成一个新的二维数组。

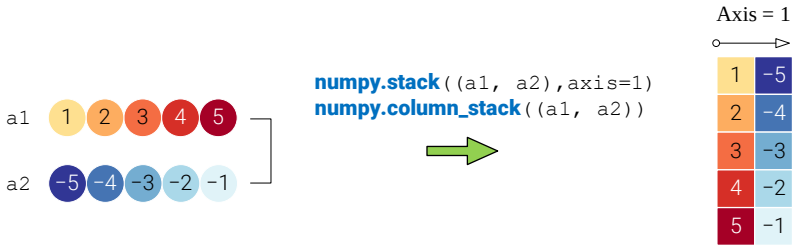


图 3. 沿列 axis = 1 方向堆叠

如图 4 所示，用 **numpy.hstack**() 堆叠一维数组的结果还是一个一维数组。**numpy.hstack**() 将多个数组沿着水平方向 (列方向) 进行堆叠，生成一个新的数组。为了获得图 3 结果，需要先将两个一维数组变形为列向量，然后用 **numpy.hstack**() 函数沿列堆叠，具体如图 5 所示。

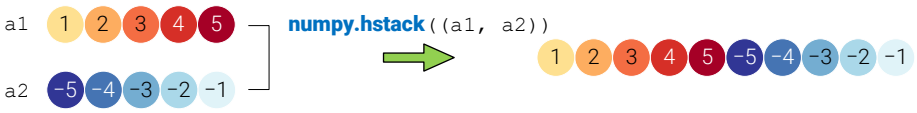


图 4. 沿列 axis = 1 方向堆叠，用 numpy.hstack()

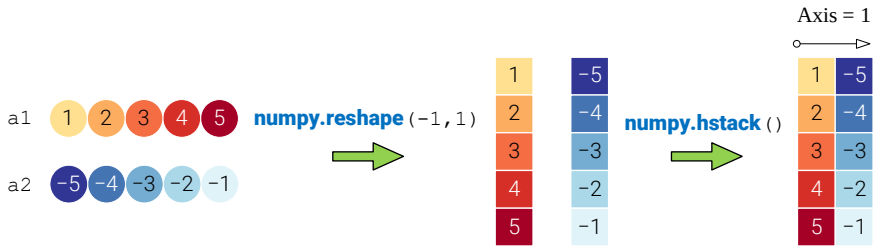


图 5. 沿列 axis = 1 方向堆叠，两个列向量

拼接

我们还可以用 **numpy.concatenate()** 完成数组拼接。如所示，利用 **numpy.concatenate()**，我们可以分别完成沿行、列方向数组拼接。

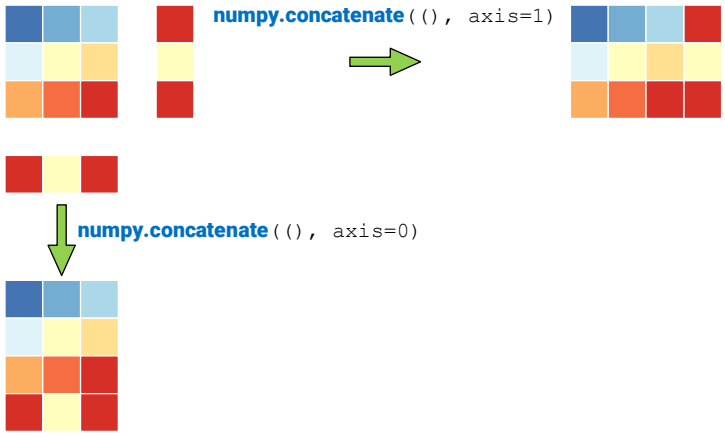


图 6. 用 numpy.concatenate() 拼接

堆叠结果为三维数组

此外，利用 **numpy.stack()**，我们还可以将二维数组堆叠为三维数组。图 7 所示为沿三个不同方向堆叠结果的效果图。

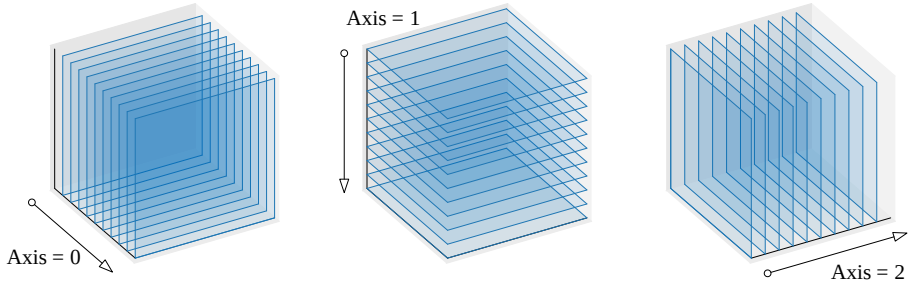


图 7. 沿着三个不同方向堆叠

举个例子，给定图 8 所示两个形状相同的二维数组。它俩按图 7 所示为沿三个不同方向堆叠的结果如图 9 所示。

图 8. 两个形状相同的二维数组

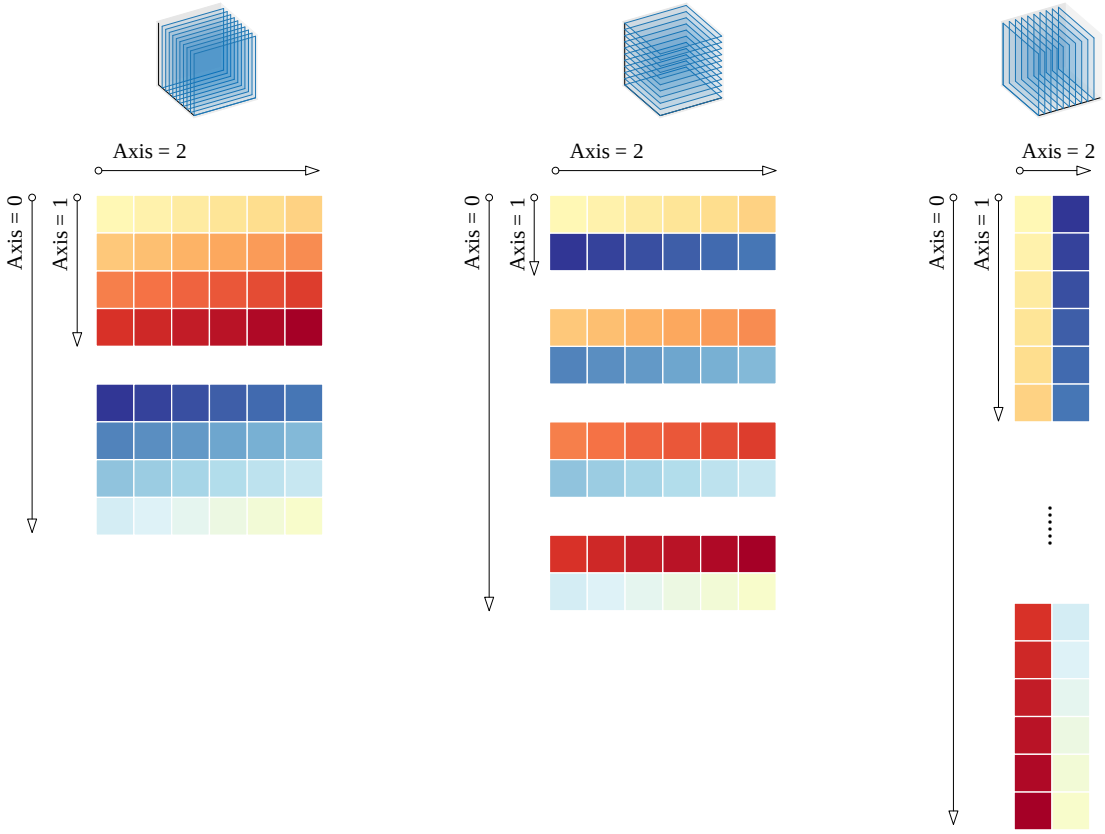


图 9. 得到三个不同的三维数组

17.2 重复

numpy.repeat() 和 **numpy.tile**() 都可以用来重复数据。**numpy.repeat**() 和 **numpy.tile**() 的区别在于重复的对象不同。**numpy.repeat**() 重复的是分别数组中的每个元素。**numpy.repeat**() 还可以指定具体的轴，以及不同元素重复的次数，请大家参考其技术文档。

numpy.tile() 重复的是整个数组，如图 11 所示。本章配套 Jupyter Notebook 还提供其他示例，请大家自行练习。

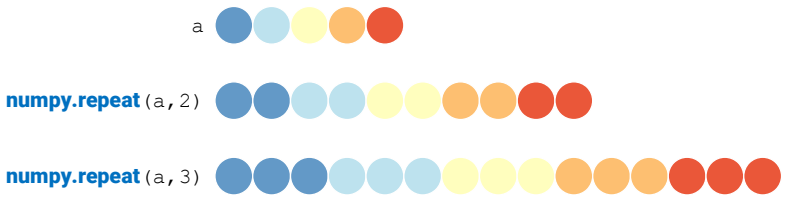


图 10. 利用 numpy.repeat() 重复一维数组

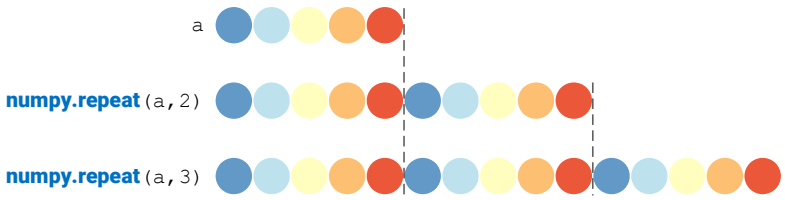


图 11. 利用 numpy.tile() 重复一维数组

17.3 分块矩阵

合成

numpy.block() 函数用于将多个数组沿不同的轴组合成一个分块矩阵。它接受一个嵌套列表作为输入，每个列表代表一个块矩阵，然后根据指定的轴将这些块矩阵组合在一起。

在图 12 给出的例子中，我们创建了四个小的矩阵，并使用 **numpy.block**() 函数将它们组合成一个分块矩阵 $\boldsymbol{M}$。

分块矩阵经常用来简化某些线性代数运算，鸢尾花书《矩阵力量》将专门介绍分块矩阵。

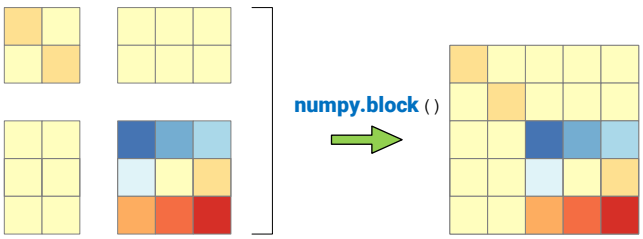


图 12. 四个二维数组合成一个矩阵

什么是分块矩阵？

分块矩阵是由多个小矩阵组合而成的大矩阵。它将一个大的矩阵划分为若干个小的矩阵，这些小矩阵可以是实数矩阵、向量矩阵或者其他的矩阵形式。通常情况下，分块矩阵可以使用一个方括号将小矩阵组合在一起，然后按照一定的规则排列。分块矩阵可以简化一些复杂的矩阵计算，同时也常常用于表示具有特定结构的矩阵，例如对角矩阵或者上下三角矩阵等。

切割

numpy.split() 函数可以将一个数组沿指定轴分割为多个子数组。**numpy.split**() 接受三个参数：要分割的数组、分割的索引位置、沿着哪个轴进行分割。图 13 所示为将一个一维数组三等分得到三个子数组。本章配套的 Jupyter Notebook 中，大家可以看到如何设定分割索引位置，请自行练习。

图 14 所示为利用 **numpy.split**() 将二维数组沿不同轴三等分。大家也可以分别尝试使用 **numpy.hsplit**() 和 **numpy.vsplit**() 完成类似操作。本章配套 Jupyter Notebook 中还介绍如何使用 **numpy.append**()、**numpy.insert**()、**numpy.delete**() 完成附加、插入、删除操作，请大家自行学习。

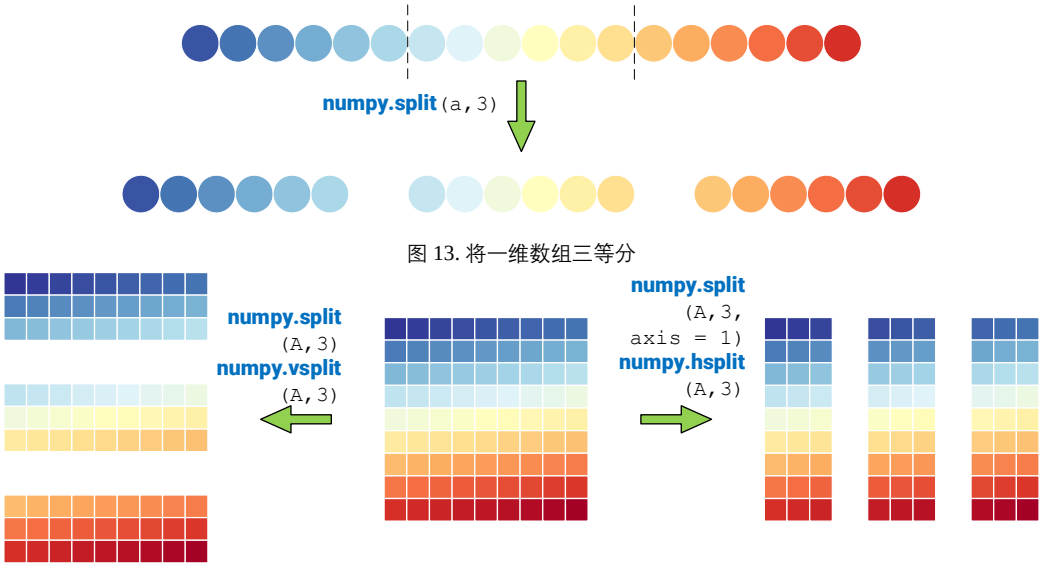


图 13. 将一维数组三等分

图 14. 将二维数组三等分，沿不同轴

下面，是有关 NumPy 数组规整的三道习题，请大家完成。

Q1. 请生成 [0, 1] 区间内的连续均匀两个随机数数组，数组形状为 (10,)。将它俩分别按行、按列堆叠起来形成二维数组。

Q2. 请生成 [0, 1] 区间内的连续均匀一个随机数数组，数组形状为 (12,12)。将它分别按行、按列三等分。

Q3. 请生成 [0, 1] 区间内的连续均匀两个随机数数组，数组形状分别为 (8, 5)、(3, 5)。用几种不同办法将它们拼接成一个数组。

* 这三道题目很基础，本书不给答案。

Linear Algebra in NumPy

18 NumPy 线性代数

NumPy 中的重要线性代数计算

我的大脑只是一个接收器。宇宙中有一个核心，我们从中获得知识、力量和灵感。这个核心的秘密我没有深入了解，但我知道它的存在。

My brain is only a receiver, in the Universe there is a core from which we obtain knowledge, strength and inspiration. I have not penetrated into the secrets of this core, but I know that it exists.

—— 尼古拉·特斯拉 (Nikola Tesla) | 发明家、物理学家 | 1856 ~ 1943

18.1 NumPy 的 linalg 模块

NumPy 库的 **linalg** 模块提供了许多用于线性代数计算的函数，包括矩阵分解和向量计算。

以下是一些常见的 **linalg** 函数：

- **numpy.linalg.inv**()：计算矩阵的逆。
- **numpy.linalg.pinv**()：计算矩阵的 Moore-Penrose 伪逆。
- **numpy.linalg.solve**()：求解线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，其中 $\boldsymbol{A}$ 是一个矩阵，$\boldsymbol{b}$ 是一个向量。
- **numpy.linalg.lstsq**()：最小二乘解。

linalg 模块还提供了许多向量计算函数，包括：

- **numpy.linalg.norm**()：计算向量的范数。
- **numpy.linalg.dot**()：计算向量的点积。

以下是 **linalg** 中常用的矩阵分解函数：

- **numpy.linalg.cholesky**()：计算 Cholesky 分解。
- **numpy.linalg.eig**()：计算矩阵的特征值和特征向量。
- **numpy.linalg.svd**()：计算奇异值分解。

这些函数在许多科学计算中都非常有用，例如，在机器学习中，可以使用矩阵分解函数进行降维和特征提取，而向量计算函数则可用于计算距离和相似性度量等。需要注意的是，这些函数都要求输入参数为 NumPy 数组，并返回 NumPy 数组作为输出。

什么是矩阵分解？

矩阵分解是一种将一个矩阵分解为若干个矩阵的乘积的数学技术。这种分解可以帮助我们更好地理解和处理矩阵数据。常见的矩阵分解包括 Cholesky 分解、特征值分解 (EVD)、奇异值分解 (SVD) 等等。矩阵分解在很多领域都有广泛的应用，比如在机器学习、数据分析、信号处理、图像处理等方面。

本节配套的 Jupyter Notebook 文件是 BK_2_Topic_4.06_1.ipynb。

18.2 拆解矩阵

一组行向量

本书前文提到鸢尾花数据矩阵 $\boldsymbol{X}$ 的形状为 150 × 4。也就是说，如图 1 热图所示，$\boldsymbol{X}$ 可以看成是由 150 个行向量上下堆叠而成。每个行向量的形状为 1 × 4。图 1 特别展示了 $\boldsymbol{x}^{(1)}$ ($\boldsymbol{X}$ 第 1 行，数组第 0 行)、$\boldsymbol{x}^{(2)}$ ($\boldsymbol{X}$ 第 2 行，数组第 1 行)、$\boldsymbol{x}^{(51)}$ ($\boldsymbol{X}$ 第 51 行，数组第 50 行)、$\boldsymbol{x}^{(101)}$ ($\boldsymbol{X}$ 第 101 行，数组第 100 行)。

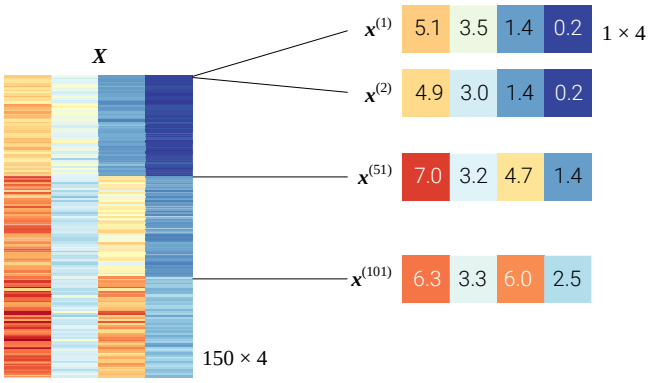


图 1. $\boldsymbol{X}$ 可以看做由一组行向量构成

一组列向量

此外，如图 1 热图所示，$\boldsymbol{X}$ 可以看成是由 4 个列向量左右排列而成，即 $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4]$。每个列向量的形状为 150 × 1。请大家回忆，我们如何设定索引获得 NumPy 数组的行向量、列向量。

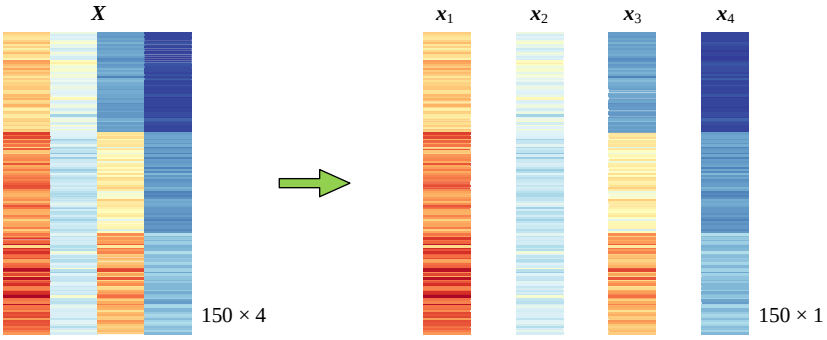


图 2. $\boldsymbol{X}$ 可以看做由一组列向量构成

18.3 向量运算

几何角度看向量

在二维空间中，一个向量 $\boldsymbol{a}$ 可以表示为一个有序的数对 (a_1, a_2)、$[a_1, a_2]$、$[a_1, a_2]^{\mathrm{T}}$。向量也可以用一个有向线段来表示，线段的起点为原点 (0, 0)，终点为 (a_1, a_2)。其中，a_1 表示向量在水平方向上的投影；y 表示向量纵轴方向上的投影。

用勾股定理，我们可求得图 3 中向量 $\boldsymbol{a}$ 的长度，即向量的模，为 $\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2}$。在 NumPy 中计算向量模的函数为 **numpy.linalg.norm**()。

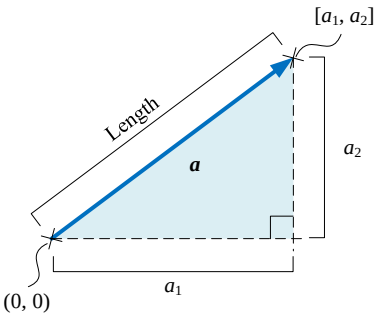


图 3. 向量起点、终点、大小和方向

在 BK_2_Topic_4.06_1.ipynb 中，我们还计算了 $\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$、$\boldsymbol{x}^{(51)}$、$\boldsymbol{x}^{(101)}$这四个向量的单位向量。单位向量是长度为 1 的向量，可以用来表示某个向量方向。比如，的单位向量就是 $\boldsymbol{x}^{(1)}$除以自己的模 $\left\|\boldsymbol{x}^{(1)}\right\|$，即 $\boldsymbol{x}^{(1)} \Big/ \left\|\boldsymbol{x}^{(1)}\right\|$。

什么是向量的模？

向量的模 (也称为向量的长度) 是指一个向量从原点到其终点的距离，它是一个标量，表示向量的大小。向量的模通常用两个竖线 $\|\boldsymbol{a}\|$ 来表示，其中 $\boldsymbol{a}$ 表示向量。对于 n 维向量 $\boldsymbol{a} = [a_1, a_2, ..., a_n]$，它的模定义为 $\|\boldsymbol{a}\| = \sqrt{a_1^2 + a_2^2 + \cdots a_n^2}$。$\|\boldsymbol{a}\|$ 就是向量各个分量的平方和的平方根。这个公式可以用勾股定理推导得出，因为一个向量的模就是从原点到它的终点的距离，而这个距离可以用勾股定理计算。比如，2 维向量 $\boldsymbol{a} = [3, 4]$ 的模 (长度) 为 $\|\boldsymbol{a}\| = \sqrt{3^2 + 4^2} = 5$。

向量内积

本书前文在讲 for 循环时介绍过**向量内积** (inner product)，又叫**标量积** (scalar product)、**点积** (dot product)。给定两个 n 维向量 $\boldsymbol{a} = [a_1, a_2, ..., a_n]$ 和 $\boldsymbol{b} = [b_1, b_2, ..., b_n]$，它们的内积定义为 $\boldsymbol{a} \cdot \boldsymbol{b} = a_1b_1 + a_2b_2 + ... + a_nb_n$。内积结果 $\boldsymbol{a} \cdot \boldsymbol{b}$ 显然为标量。

如图 4 所示，我们分别计算向量内积 $\boldsymbol{x}^{(1)} \cdot \boldsymbol{x}^{(2)}$、$\boldsymbol{x}^{(1)} \cdot \boldsymbol{x}^{(51)}$、$\boldsymbol{x}^{(1)} \cdot \boldsymbol{x}^{(101)}$。建议大家在 JupyterLab 中用手输入算式计算图中三个向量内积。

再次强调，向量内积的运算前提是两个向量维数相同，结果为标量。NumPy 中计算向量内积的函数为 **numpy.dot**()。

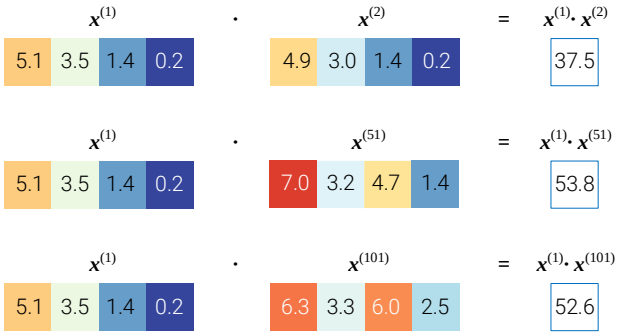


图 4. 向量内积

向量夹角

在 BK_2_Topic_4.06_1.ipynb 中，我们计算得到 $\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$的夹角约为 3°，$\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(51)}$的夹角约为 22°，$\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(101)}$的夹角约为 31°。

这显然不是巧合，$\boldsymbol{x}^{(1)}$、$\boldsymbol{x}^{(2)}$分别代表两朵鸢尾花，它们同属 Setosa，因此最为相似。而 $\boldsymbol{x}^{(51)}$属于 Versicolour，$\boldsymbol{x}^{(101)}$属于 Virginica。这就是向量夹角在机器学习中的一个应用举例。

什么是向量夹角？

向量夹角是指两个向量之间的夹角，它是一个标量，通常用弧度或角度来表示。向量夹角的计算是通过向量内积和向量模的关系得出的。对于两个非零向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$，它们的夹角 θ 定义为 $\cos(\theta) = (\boldsymbol{a} \cdot \boldsymbol{b}) / (\|\boldsymbol{a}\| \|\boldsymbol{b}\|)$。其中 $\boldsymbol{a} \cdot \boldsymbol{b}$ 是向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的内积，$\|\boldsymbol{a}\|$ 和 $\|\boldsymbol{b}\|$ 分别是向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的模。注意，这个公式只适用于非零向量，因为对于零向量，它没有方向，因此无法定义夹角。此外，$\cos(\theta)$ 可以看成是 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的单位向量的向量内积，即 $\cos(\theta) = (\boldsymbol{a}/\|\boldsymbol{a}\|) \cdot (\boldsymbol{b}/\|\boldsymbol{b}\|)$。

通过向量夹角的计算，我们可以判断两个向量之间的相对方向。如果两个向量的夹角为零度，表示它们的方向相同；如果夹角为 90 度，表示它们互相垂直；如果夹角为 180 度，表示它们的方向相反。在机器学习中，可以通过计算向量夹角来度量两个样本之间的相似性。

18.4 矩阵运算

矩阵乘法

本书前文介绍过矩阵乘法，假设 $\boldsymbol{A}$ 是一个 $m \times n$ 的矩阵，$\boldsymbol{B}$ 是一个 $n \times p$ 的矩阵，则它们的乘积 $\boldsymbol{C} = \boldsymbol{AB}$ 是一个 $m \times p$ 的矩阵，相当于“消去”n。在 NumPy 中矩阵乘法的运算符为 @。

在本节配套的 Jupyter Notebook 文件中大家可以看到两个有趣的矩阵乘法。

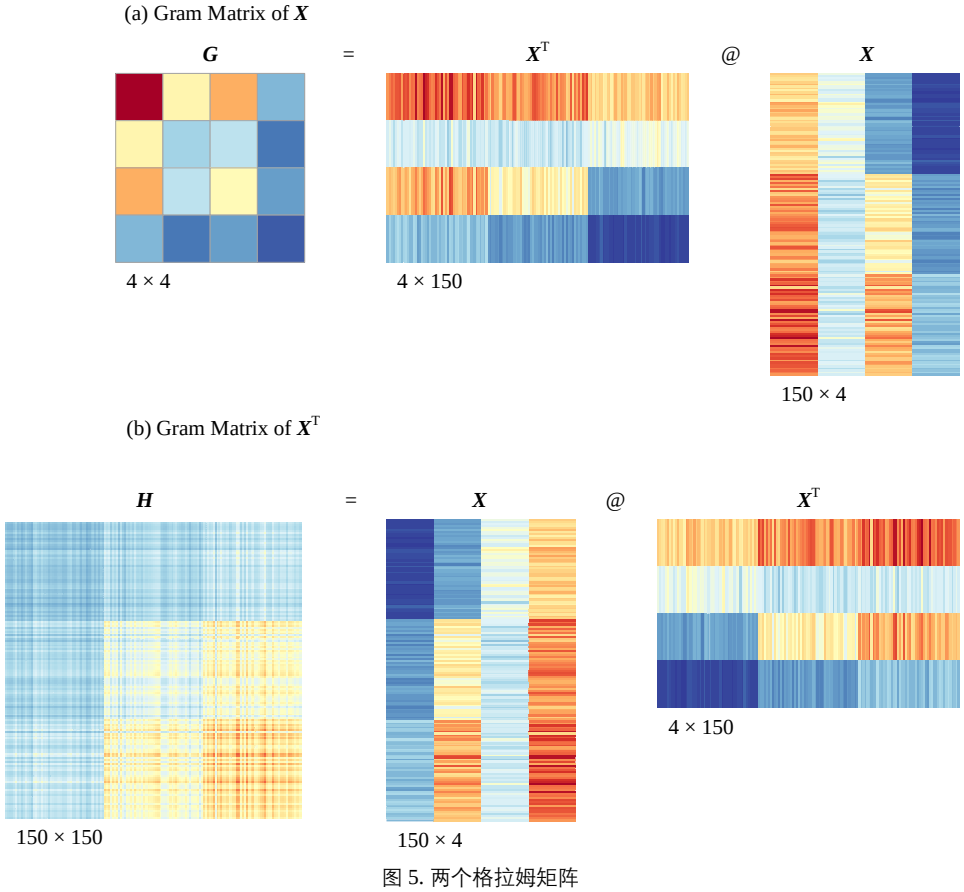


图 5. 两个格拉姆矩阵

如图 5 (a) 所示，鸢尾花数据矩阵的转置 $\boldsymbol{X}^{\mathrm{T}}$ 乘 $\boldsymbol{X}$ 得到 $\boldsymbol{G}$。$\boldsymbol{X}^{\mathrm{T}}$ 的形状为 4 × 150，$\boldsymbol{X}$ 的形状为 150 × 4。$\boldsymbol{G} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 的结果形状为 4 × 4。$\boldsymbol{G}$ 有自己的名字，叫 $\boldsymbol{X}$ 的格拉姆矩阵 (Gram matrix)。图 5 (b) 所示的 $\boldsymbol{H} = \boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ 的结果形状为 150 × 150。$\boldsymbol{H}$ 相当是 $\boldsymbol{X}^{\mathrm{T}}$ 的格拉姆矩阵。

什么是格拉姆矩阵?

格拉姆矩阵 (Gram matrix) 是一个重要的矩阵，它由向量集合的内积组成。给定一个向量集合 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$，则其对应的格拉姆矩阵 $\boldsymbol{G}$ 定义为 $\boldsymbol{G} = [g_{i,j}]$，其中 $g_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$，表示第 i 个向量和第 j 个向量的内积。格拉姆矩阵是对称矩阵。

格拉姆矩阵在许多应用中都有广泛的应用，例如在机器学习中的支持向量机 (Support Vector Machine, SVM) 算法和核方法 (kernel method) 中，格拉姆矩阵可以用来计算向量之间的相似度和距离，从而实现非线性分类和回归。此外，格拉姆矩阵也可以用于矩阵分解、图像处理、信号处理等领域。

格拉姆矩阵有很多有趣的性质，《矩阵力量》一册将详细介绍。这里大家仅仅需要知道格拉姆矩阵为对称矩阵。$\boldsymbol{G}$ 的主对角线上元素是 $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i$，即 $\boldsymbol{x}_i \cdot \boldsymbol{x}_i$。如图 6 上图所示，$\boldsymbol{G}$ 的主对角线第一元素 $g_{1,1} = \boldsymbol{x}_1^{\mathrm{T}}\boldsymbol{x}_1 = \boldsymbol{x}_1 \cdot \boldsymbol{x}_1$。如图 6 下图所示，$\boldsymbol{G}$ 的主对角线第二元素 $g_{2,2} = \boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{x}_2 = \boldsymbol{x}_2 \cdot \boldsymbol{x}_2$。请大家自行计算 $\boldsymbol{G}$ 的主对角线剩余两个元素。

如图 7 所示，显然 $g_{2,1} = g_{1,2}$。也就是说，$\boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{x}_1 = \boldsymbol{x}_1^{\mathrm{T}}\boldsymbol{x}_2 = \boldsymbol{x}_2 \cdot \boldsymbol{x}_1 = \boldsymbol{x}_1 \cdot \boldsymbol{x}_2$。

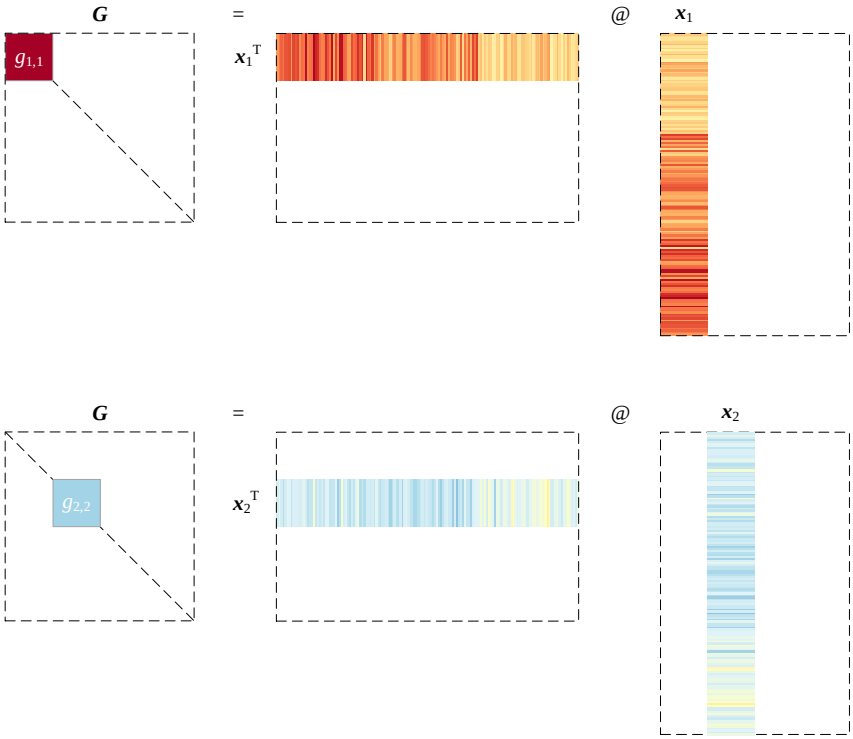


图 6. 格拉姆矩阵 $\boldsymbol{G}$ 主对角线元素

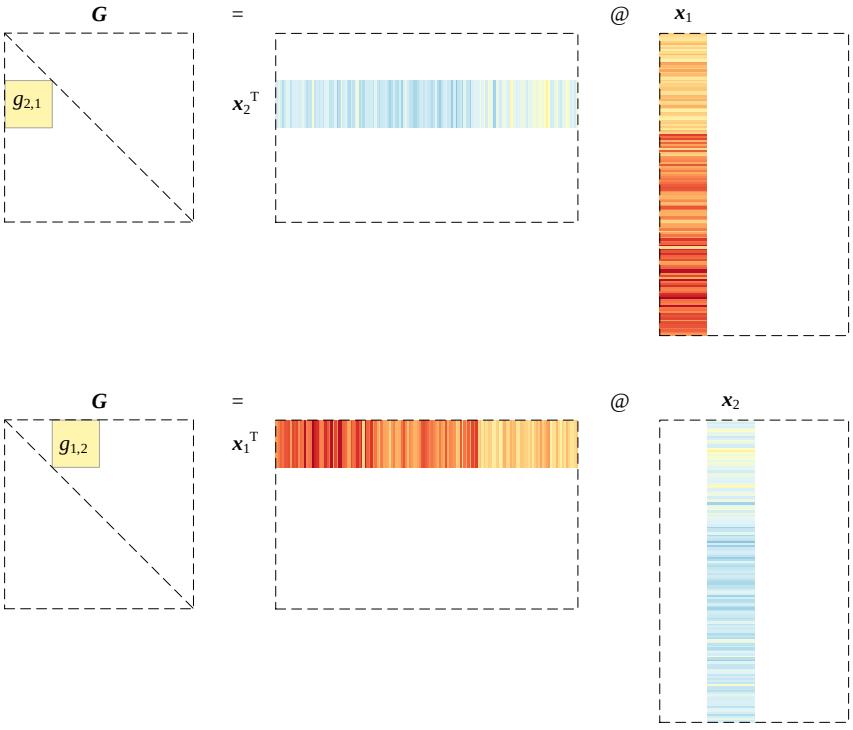


图 7. $\boldsymbol{G}$ 为对称矩阵

矩阵的逆

矩阵的逆可以被看作是一种倒数的概念。并不是所有格拉姆矩阵，恰好前文的格拉姆矩阵 $\boldsymbol{G}$ 存在逆，记做 $\boldsymbol{G}^{-1}$。如图 8 所示，$\boldsymbol{G}$ 乘 $\boldsymbol{G}^{-1}$ 结果为单位阵 $\boldsymbol{I}$。不难看出来，$\boldsymbol{G}^{-1}$ 也是个对称矩阵。

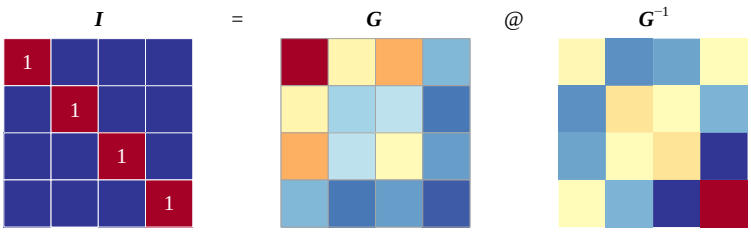


图 8. 格拉姆矩阵 $\boldsymbol{G}$ 的逆

什么是矩阵的逆？

矩阵的逆是一个重要的概念，它是指对于一个可逆的 (即非奇异的) $n \times n$ 矩阵 $\boldsymbol{A}$，存在一个 $n \times n$ 矩阵 $\boldsymbol{B}$，使得 $\boldsymbol{AB} = \boldsymbol{BA} = \boldsymbol{I}$，其中 $\boldsymbol{I}$ 是单位矩阵。$\boldsymbol{B}$ 被称为 $\boldsymbol{A}$ 的逆矩阵，通常用 $\boldsymbol{A}^{-1}$ 表示。矩阵的逆可以被看作是一种倒数的概念，它可以使我们在矩阵运算中除以矩阵，从而解决线性方程组和其他问题。如果我们需要求解一个线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$，其中 $\boldsymbol{A}$ 是一个可逆矩阵，那么可以使用矩阵的逆来计算 $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$，从而得到方程的解。需要注意的是，并非所有矩阵都有逆矩阵，只有可逆矩阵才有逆矩阵。对于一个不可逆矩阵，它可能是奇异的 (即行列式为 0)，也可能是非方阵。在实际应用中，矩阵的逆通常通过 LU 分解、QR 分解、Cholesky 分解等方法来计算，而不是直接求解逆矩阵。

18.5 几个常见矩阵分解

Cholesky 分解

所幸前文的格拉姆矩阵 G 也是个正定矩阵 (positive definite matrix)，我们可以对它进行 Cholesky 分解。如图 9 所示，$\boldsymbol{L}$ 是个下三角矩阵，它的转置 $\boldsymbol{L}^{\mathrm{T}}$ 为上三角矩阵。$\boldsymbol{L}$ 和 $\boldsymbol{L}^{\mathrm{T}}$ 的乘积也相当于“平方”。NumPy 中完成 Cholesky 分解的函数为 **numpy.linalg.cholesky**()。

《矩阵力量》第 12 章专门讲解 Cholesky 分解，这本书第 21 章将介绍正定性。

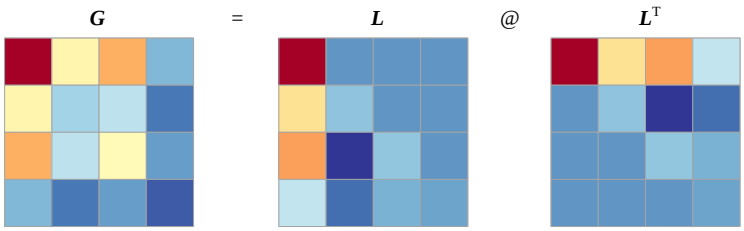


图 9. 对格拉姆矩阵 $\boldsymbol{G}$ 进行 Cholesky 分解

什么是 Cholesky 分解？

Cholesky 分解是一种将对称正定矩阵分解为下三角矩阵和其转置矩阵乘积的数学技术。给定一个对称正定矩阵 $\boldsymbol{A}$，Cholesky 分解可以将其表示为 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$，其中 $\boldsymbol{L}$ 是下三角矩阵，$\boldsymbol{L}^{\mathrm{T}}$ 是其转置矩阵。Cholesky 分解是一种高效的矩阵分解方法，它可以在数值计算中减少误差，同时可以加速线性方程组的求解，特别是对于大型的稠密矩阵。因此，Cholesky 分解在很多领域都有广泛的应用，例如统计学、金融学、物理学、工程学等。Cholesky 分解也是一些高级技术的基础，例如蒙特卡洛模拟、Kalman 滤波等等。

什么是正定矩阵？

Cholesky 分解是一种将对称正定矩阵分解为下三角矩阵和其转置矩阵乘积的数学技术。给定一个对称正定矩阵 $\boldsymbol{A}$，Cholesky 分解可以将其表示为 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$，其中 $\boldsymbol{L}$ 是下三角矩阵，$\boldsymbol{L}^{\mathrm{T}}$ 是其转置矩阵。Cholesky 分解是一种高效的矩阵分解方法，它可以在数值计算中减少误差，同时可以加速线性方程组的求解，特别是对于大型的稠密矩阵。因此，Cholesky 分解在很多领域都有广泛的应用，例如统计学、金融学、物理学、工程学等。Cholesky 分解也是一些高级技术的基础，例如蒙特卡洛模拟、Kalman 滤波等等。

特征值分解 EVD

图 10 所示为对格拉姆矩阵 $\boldsymbol{G}$ 的特征值分解。$\boldsymbol{V}$ 的每一列对应特征向量，$\boldsymbol{\Lambda}$ 的主对角线元素为特征值。

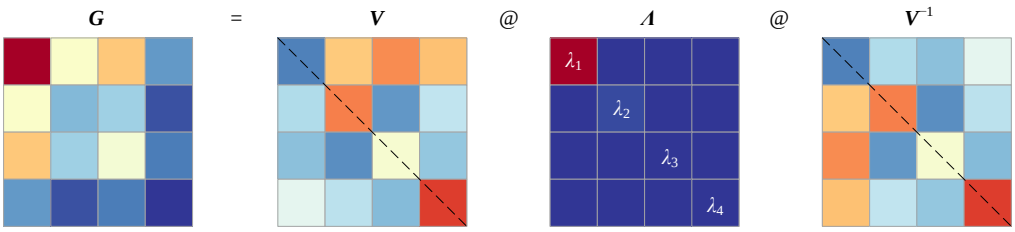


图 10. 对格拉姆矩阵 $\boldsymbol{G}$ 进行 EVD 分解

什么是特征值分解？

特征值分解 (Eigenvalue Decomposition, EVD) 是一种将一个方阵分解为一组特征向量和特征值的数学技术。对于一个 $n\times n$ 的矩阵 $\boldsymbol{A}$，如果存在非零向量 $\boldsymbol{v}$ 和常数 λ，使得 $\boldsymbol{A}\boldsymbol{v}=\lambda\boldsymbol{v}$，那么 $\boldsymbol{v}$ 就是矩阵 $\boldsymbol{A}$ 的特征向量，λ 就是对应的特征值。将所有特征向量排列成一个矩阵 $\boldsymbol{V}$，将所有特征值排列成一个对角方阵 $\boldsymbol{\Lambda}$，那么矩阵 $\boldsymbol{A}$ 就可以表示为 $\boldsymbol{A}=\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$。特征值分解可以帮助我们理解矩阵的性质和结构，以及实现很多数学算法。它在很多领域都有广泛的应用，比如图像处理、机器学习、信号处理、量子力学等。特征值分解也是一些高级技术的基础，例如奇异值分解、QR 分解、LU 分解等。

仔细观察，大家可以已经发现图 10 中 $\boldsymbol{V}$ 和 $\boldsymbol{V}^{-1}$ 关于主对角线对称，即 $\boldsymbol{V}^{\mathrm{T}}=\boldsymbol{V}^{-1}$。这并不是巧合，原因是格拉姆矩阵 $\boldsymbol{G}$ 为对称矩阵。而对称矩阵的特征值分解又叫谱分解 (spectral decomposition)。也就是说，$\boldsymbol{G}$ 的谱分解可以写成 $\boldsymbol{G}=\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathrm{T}}$。

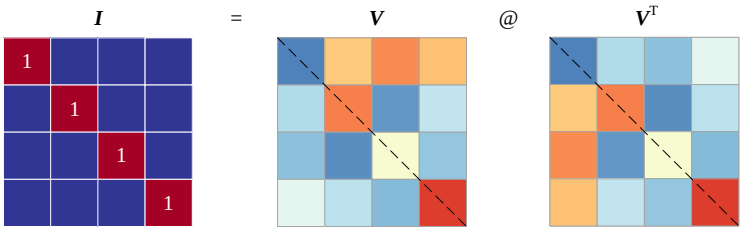


图 11. 谱分解中 $\boldsymbol{V}$ 的特点

《矩阵力量》第 13、14 章专门讲解特征值分解。

什么是谱分解？

谱分解 (Spectral Decomposition) 是将对称矩阵分解为一组特征向量和特征值的数学技术，即对称矩阵的特征值分解。对于一个对称矩阵 $\boldsymbol{A}$，谱分解可以将其分解为 $\boldsymbol{A}=\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\mathrm{T}}$，其中 $\boldsymbol{Q}$ 是由矩阵 $\boldsymbol{A}$ 的特征向量组成的正交矩阵，$\boldsymbol{\Lambda}$ 是由矩阵 $\boldsymbol{A}$ 的特征值组成的对角矩阵。谱分解在很多领域都有广泛的应用，例如图像处理、信号处理、量子力学等。谱分解可以帮助我们理解对称矩阵的性质和结构，从而帮助我们分析和处理各种问题。谱分解也是很多高级技术的基础，例如奇异值分解、主成分分析、矩阵函数等。

奇异值分解 SVD

奇异值分解可谓“最重要的矩阵分解，没有之一”。图 12 所示为对鸢尾花数据矩阵 $\boldsymbol{X}$ 的奇异值分解。图 12 中 $\boldsymbol{S}$ 的主对角线上的元素叫奇异值。大家会在 BK_2_Topic_4.06_1.ipynb 看到，图 10 中特征值开方的结果就是图 12 中的奇异值，这当然不是巧合！

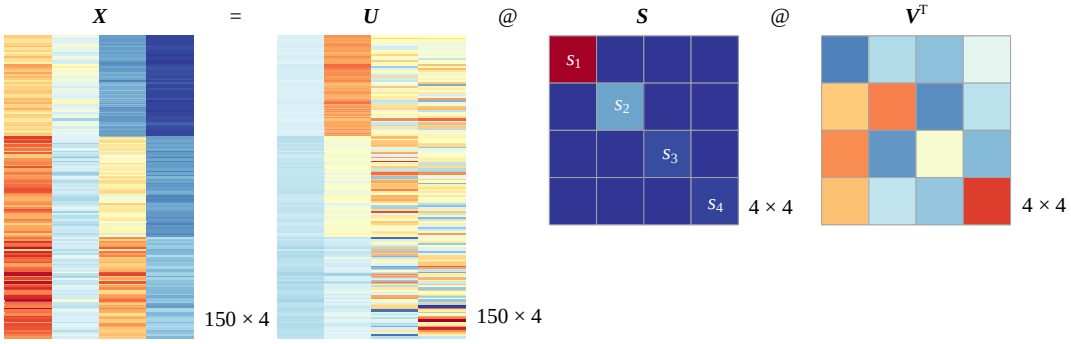


图 12. 对 $\boldsymbol{X}$ 进行 SVD 分解

什么是奇异值分解？

奇异值分解 (Singular Value Decomposition, SVD) 是一种将一个矩阵分解为三个矩阵乘积的数学技术。给定一个矩阵 $\boldsymbol{A}$，它可以表示为 $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\mathrm{T}}$，其中 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 是正交矩阵，$\boldsymbol{S}$ 是对角矩阵，对角线上的元素称为奇异值。SVD 可以将一个矩阵的信息分解为不同奇异值所对应的向量空间，并按照奇异值大小的顺序进行排序，使得我们可以仅使用前面的奇异值和相应的向量空间来近似地表示原始矩阵。这种分解在降维、压缩、数据处理和模型简化等领域中有着广泛的应用，例如推荐系统、图像压缩、语音识别等。

如图 13 所示，$\boldsymbol{U}$ 的转置 $\boldsymbol{U}^{\mathrm{T}}$ 和自己乘积为单位阵。如图 14 所示，$\boldsymbol{V}$ 和自身转置 $\boldsymbol{V}^{\mathrm{T}}$ 乘积为单位阵。大家是否已经发现图 11 和图 14 竟然相同，这当然也不是巧合。

实际上，图 12 是四种奇异值分解中的一种。《矩阵力量》第 15、16 章专门讲解奇异值分解，并揭开各种“巧合”背后的数学原理。

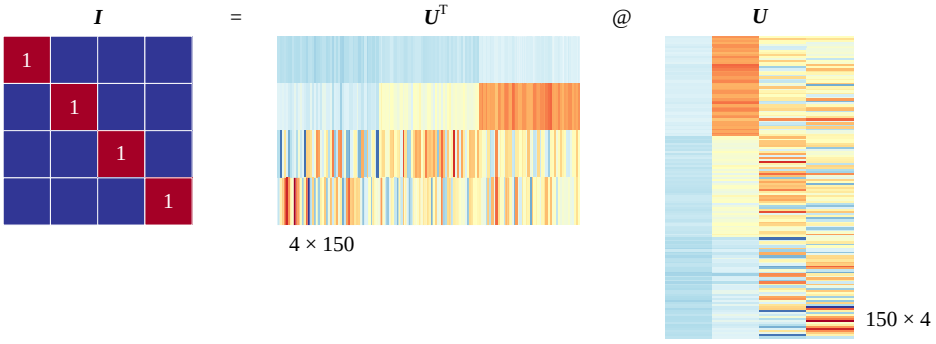


图 13. $\boldsymbol{U}$ 的特点

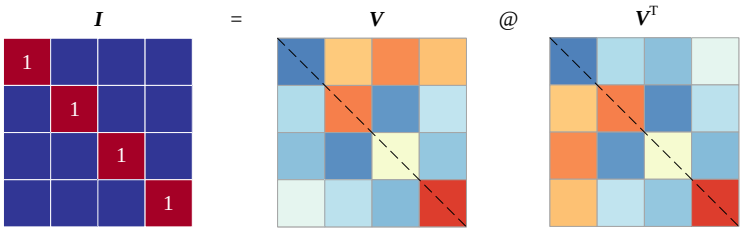


图 14. $\boldsymbol{V}$ 的特点

请大家完成下面 5 道题目。

Q1. 本节配套笔记计算了鸢尾花数据矩阵 $\boldsymbol{X}$ 的若干行向量的模、单位向量、夹角，请大家计算 $\boldsymbol{X}$ 的 4 个列向量的模、单位向量、两两列向量内积、两两夹角。并说明两两列向量内积和图 5 (a) 中格拉姆矩阵的关系。

Q2. 请大家用热图可视化图 5 (a) 中的 $\boldsymbol{G}$ 的第 2 行第 3 列元素如何计算得到。

Q4. 请对图 5 (b) 中的格拉姆矩阵进行 Cholesky 分解，并解释报错的原因。

Q4. 请对图 5 (b) 中的格拉姆矩阵进行特征值分解，并比较其特征值和图 10 中特征值关系。

Q5. 请对 $\boldsymbol{X}^{\mathrm{T}}$ 进行奇异值分解，比较和图 12 中 SVD 分解的关系。

* 本节不提供答案。

Descriptive Statistics Using Seaborn

29 Seaborn 可视化数据

使用 Seaborn 完成样本数据统计描述

理性永恒，其他一切皆有终结之时。

Reason is immortal, all else mortal.

—— 毕达哥拉斯 (Pythagoras) | 古希腊哲学家、数学家 | 570 ~ 495 BC

29.1 Seaborn

本书前文介绍用 Seaborn 绘制热图。实际上，Seaborn 的真正价值体现在统计可视化上。简单来说，Seaborn 是一个用于数据可视化的 Python 库，它基于 Matplotlib，并提供了一组高级的绘图函数和样式设置，可以轻松创建具有吸引力和专业外观的统计图表。

Seaborn 提供了多种可视化方案，包括但不限于：

- 分布图：包括直方图、核密度图、箱线图等，用于展示数据的分布情况。
- 散点图：用于观察两个变量之间的关系，可以通过散点图添加颜色或大小编码第三个变量。
- 线性关系图：通过绘制线性回归模型的置信区间，展示两个变量之间的线性关系。
- 分类图：包括条形图、点图、计数图等，用于比较不同类别之间的数值关系。
- 矩阵图：如热图和聚类图，用于显示数据的相似性和聚类结构。

本章以鸢尾花数据为例介绍如何用 Seaborn 可视化样本数据分布。

样本数据分布是指在统计学中，对于一组收集到的数据，对其进行统计和描述的方式。

一元样本数据分布是指只包含一个随机变量的样本数据分布，例如鸢尾花花萼长度。可视化一元样本分布的方法有：直方图 (histogram)、核密度估计 (Kernel Density Estimation, KDE)、毛毯图 (rug plot)、分散图 (strip plot)、小提琴图 (violin plot)、箱型图 (box plot)、蜂群图 (swarm plot)等等。

二元样本数据分布则涉及两个随机变量，例如鸢尾花花萼长度、花萼关系之间的关系。这种分布一般叫联合分布 (joint distribution)。我们可以通过相关性系数量化联合分布。

边缘分布 (marginal distribution) 是指在多元数据分布中，对某一个或几个变量进行统计，而忽略其他变量的分布。例如，在花萼长度、花萼关系的二元数据分布中，对花萼长度的边缘分布就是仅考虑花萼长度变量的数据分布。

可视化二元样本分布的方法有散点图 (scatter plot)、散点图 + 边缘直方图、散点图 + 毛毯图、散点图 + 回归图、频率热图、二元 KDE 等等图形和图形组合。

多元样本数据分布则涉及两个以上随机变量，例如鸢尾花花萼长度、花萼宽度、花瓣长度、花瓣宽度。多元样本数据的可视化方案有热图、聚类热图 (cluster map)、平行坐标图 (parallel plot)、成对特征散点图、Radviz 等等。特别地，我们还可以用协方差矩阵、相关性系数矩阵来量化随机变量之间的关系。而热图可以用来可视化协方差矩阵、相关性系数矩阵。

除此之外，我们在采用上述可视化方案时，还可以考虑分类，比如鸢尾花种类。

下面我们来逐一展示这些统计可视化方案。

29.2 一元

直方图

直方图是一种常用的数据可视化图表，用于显示数值变量的分布情况。如图 1 所示，将数据划分为不同的区间 (也称"柱子")，一般计算每个区间内的数据频数 (样本数量)；简单来说，这个过程就是“查数”。然后，通过绘制每个区间的柱状条形来表示相应的频数。比如，图 1 中深蓝的“柱子”对应区间的样本数量为 25，因此“柱子”的高度为 25。

直方图的 x 轴表示变量的取值范围，而 y 轴表示频数、概率、概率密度。图 1 中深蓝的“柱子”对应的频数为 25，样本总数为 150，因此这个柱子对应的概率为 25/150。柱子的宽度为 0.2，因此这个深蓝色柱子的概率密度为 25/150/0.2。

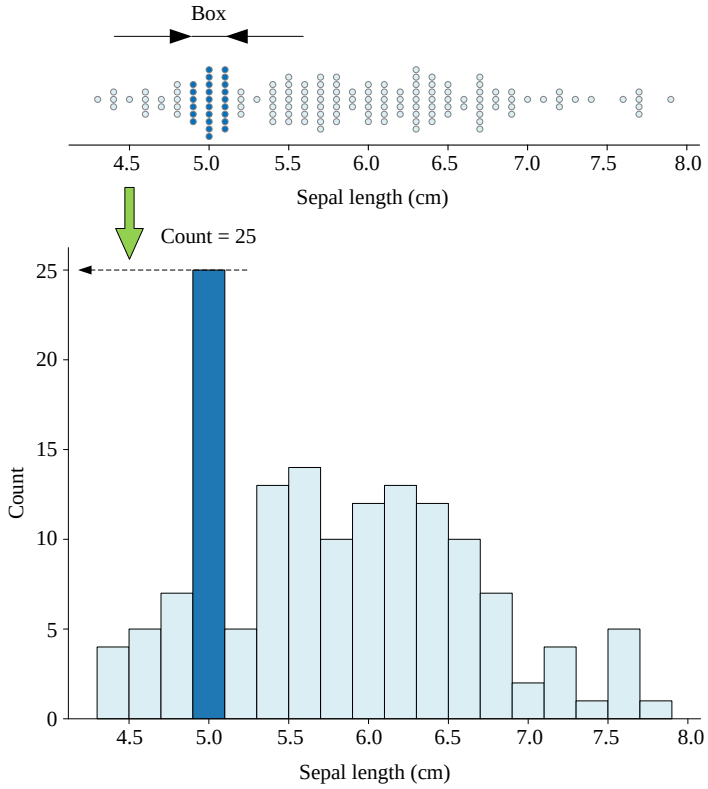


图 1. 直方图原理

图 2 所示为鸢尾花花萼长度样本数据的直方图，纵轴为频数。

如果图 2 的纵轴为概率，图 2 的这些“柱子”的高度之和为 1。如果图 2 的纵轴为概率密度，图 2 的这些“柱子”的面积之和为 1。

图 2 这张图上还用 ax.axvline() 绘制了花萼长度样本均值的位置。请大家修改本章配套 Jupyter Notebook，将“均值 ± 标准差”这两条直线也画上去。

注意，标准差是方差的平方根。样本标准差、样本、均值三者的单位相同。

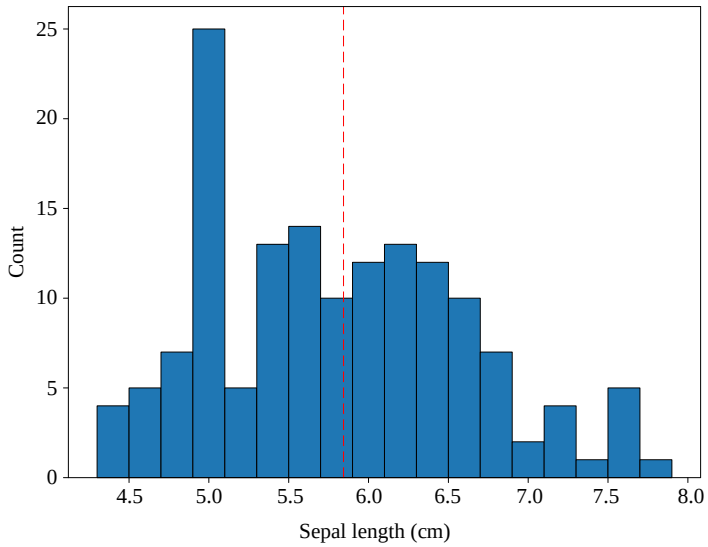


图 2. 鸢尾花花萼长度样本数据直方图，纵轴为频数

`seaborn.histplot()` 是 `Seaborn` 库中用于绘制直方图的函数。这个函数的重要输入有，`data` 一般为 `Pandas` 数据帧，`x` 为横轴标签。此外，`stat` 指定纵轴类型，比如`'count'` 对应频数，`'probability'`对应概率，`'density'` 对应概率密度。可以用 `bins` 指定直方图区间数量，`binwidth` 定义区间宽度。

利用 `seaborn.histplot()`绘制鸢尾花数据直方图时，如果指定 `hue = 'species'`，我们便得到每个类别鸢尾花单独的直方图，具体如图 3 所示。`seaborn.histplot()` 还可以用来绘制二维直方热图，本章后文将介绍。此外，本章配套的 `Jupyter Notebook` 还给出函数其他用法。

注意，图 3 直方图纵轴为概率密度值。

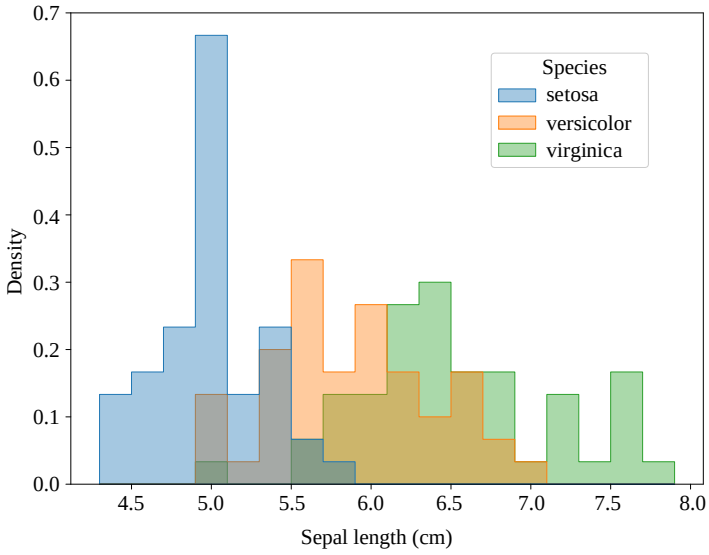


图 3. 鸢尾花花萼长度样本数据直方图，考虑鸢尾花分类，纵轴为概率密度

在直方图中，以下是频数、概率和概率密度的确切定义如下：

频数 (frequency)：直方图中每个区间内的样本数量被称为频数。它表示了数据落入该区间的次数或计数。

概率 (probability)：是指某个事件发生的可能性。在直方图中，可以将频数除以总观测值的数量，得到每个区间的概率。这样计算得到的概率是相对频率，表示该区间中的观测值出现的相对概率。

概率密度 (probability density)：是指在概率分布函数中某一点附近单位自变量取值范围内的概率。在直方图中，概率密度可以通过将每个区间的频数除以该区间的宽度得到。概率密度函数描述了变量的分布形状，而不是具体的概率值。

直方图可以显示数据的分布形状，如对称 (symmetry)、偏态 (skewness)、峰度 (kurtosis) 等，以及数据的中心趋势和离散程度。通过观察直方图，我们可以直观地了解数据的分布特征，如数据的集中程度、范围和异常值等。

《统计至简》第 1 章将专门讲解直方图、偏态、峰度等概念。

核密度估计 KDE

核密度估计 (Kernel Density Estimation, KDE) 是一种非参数方法，用于估计连续变量的概率密度函数 (Probability Density Function, PDF)。它通过将每个数据点视为一个核函数 (通常是高斯核函数)，在整个变量范围内生成一系列核函数，然后将这些核函数进行平滑和叠加，从而得到连续的概率密度估计曲线。具体原理如图 4 所示。

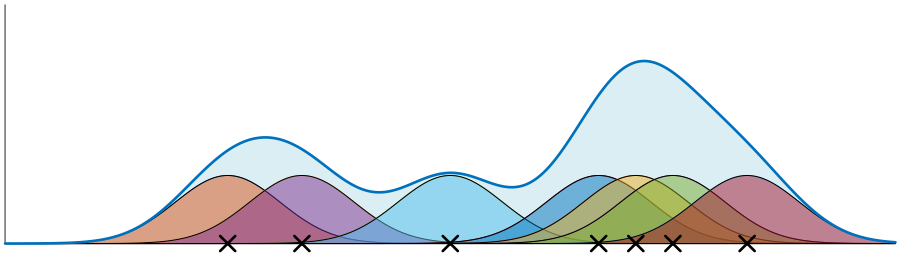


图 4. 高斯核密度估计原理

核密度估计的目标是通过在数据点附近生成高斯分布的核函数，捕捉数据的分布特征和结构。具体地说，每个数据点的核函数会在其附近产生一个小的高斯分布，然后将所有核函数叠加在一起。通过调整核函数的带宽参数，可以控制估计曲线的平滑程度和敏感度。

《统计至简》第 17 章将专门讲解核密度估计原理。

图 5 所示为利用 seaborn.kdeplot() 绘制的鸢尾花花萼长度数据高斯核密度估计 PDF。可以这样理解，图 5 是图 2 直方图的“平滑”处理结果。

图 5 的横轴还有用 seaborn.rugplot() 绘制的毛毯图。毛毯图常用于展示数据在一维空间上的分布。它通过在坐标轴上绘制短线，或称为"毛毯"，表示数据点的位置和密度。这种图形通常用于辅助其他类型的图表，如直方图或密度图，以更清晰地显示数据的分布特征。

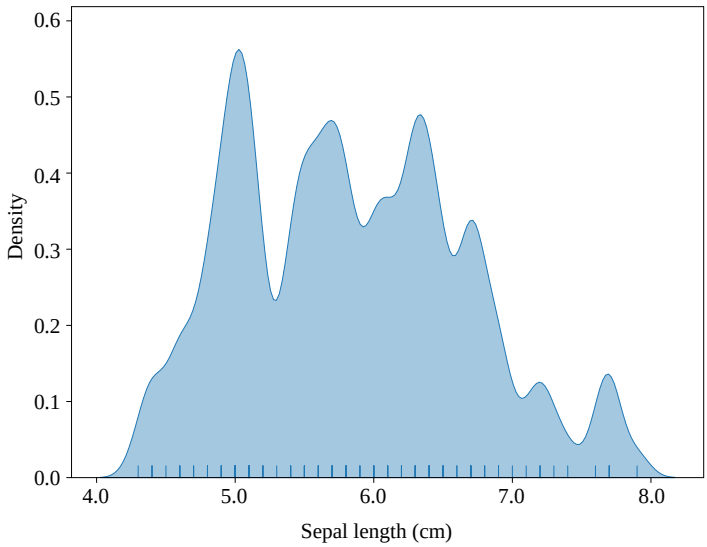


图 5. 鸢尾花花萼长度样本数据核密度估计

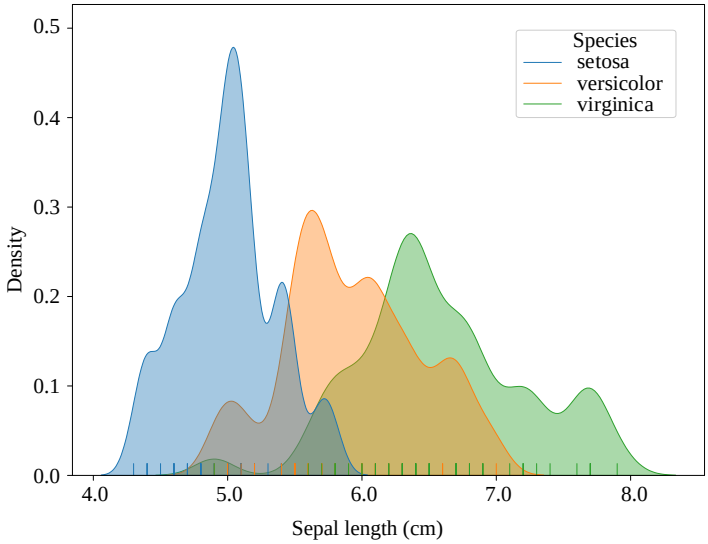


图 6. 鸢尾花花萼长度样本数据核密度估计，考虑鸢尾花分类

在用 seaborn.kdeplot() 绘制花萼长度样本数据核密度估计曲线时，我们还可以用 hue 来绘制三类鸢尾花种类各自的分布，具体如图 6 所示。

换个角度理解图 6，图 6 中三条曲线叠加便得到图 5。图 7 这幅图更好地解释了这一点。用 seaborn.kdeplot() 绘制这幅图时，需要设置 multiple="stack"。

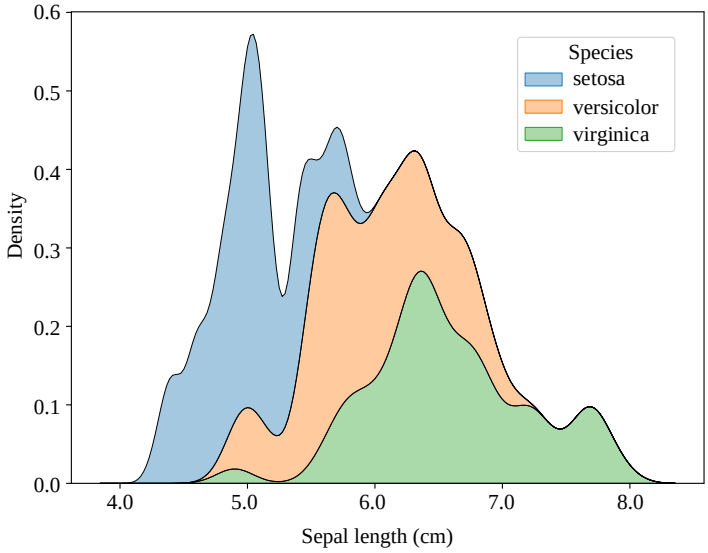


图 7. 三条 KDE 曲线叠加

特别地，在利用绘制核密度估计曲线时，如果设置 multiple = 'fill'，我们便获得图 8。图中每条曲线准确来说，都是“后验概率 (posterior)”。而这个后验概率值可以用来完成分类。也就是说，给定具体花萼长度，比较该点处红蓝绿三条曲线对应的宽度，最宽的的曲线对应的鸢尾花种类可以作为该点的鸢尾花分类预测值。因此，这个后验概率值也叫“成员值 (membership score)”。

想要理解后验概率这个概念，需要大家深入理解贝叶斯定理，《统计至简》第 18、19 章将专门介绍利用贝叶斯定理完成分类。

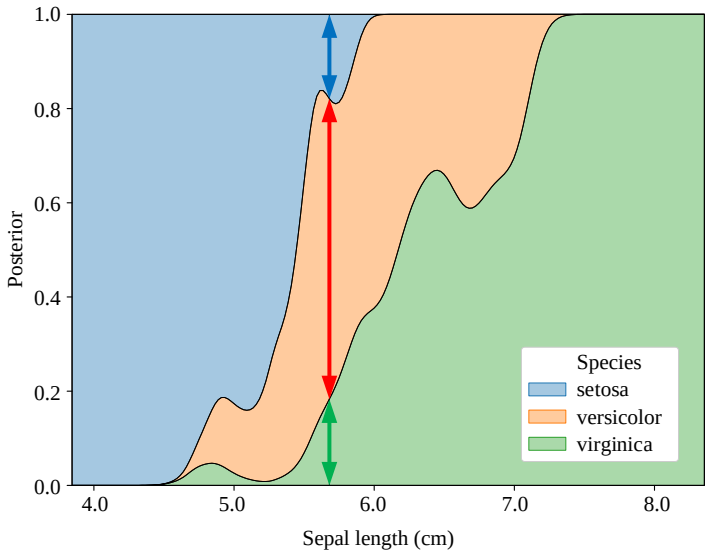


图 8. 后验概率曲线

什么是贝叶斯定理？

贝叶斯定理是一种用于更新概率推断的数学公式。它描述了在获得新信息后如何更新我们对某个事件发生概率的信念。贝叶斯定理基于先验概率（我们对事件发生的初始信念）和条件概率（给定新信息的情况下事件发生的概率），通过计算后

验概率（在获得新信息后事件发生的概率）来实现更新。贝叶斯定理在统计学、机器学习和人工智能等领域具有广泛应用。

分散点图

分散点图 (strip plot) 一般用来可视化一组分类变量与连续变量的关系。在分散图中，每个数据点通过垂直于分类变量的轴上的一个点表示，连续变量的取值则沿着水平轴展示。这种图形通常用于可视化分类变量和数值变量之间的关系，以观察数据的分布、聚集和离散程度，同时也可以用于比较不同分类变量水平下的数值变量。

seaborn.stripplot() 是 Seaborn 库中用于绘制分散点图的函数。需要注意的是，分散点图适用于较小的数据集，当数据点重叠较多时，可考虑使用 seaborn.swarmplot() 函数来避免重叠点问题。

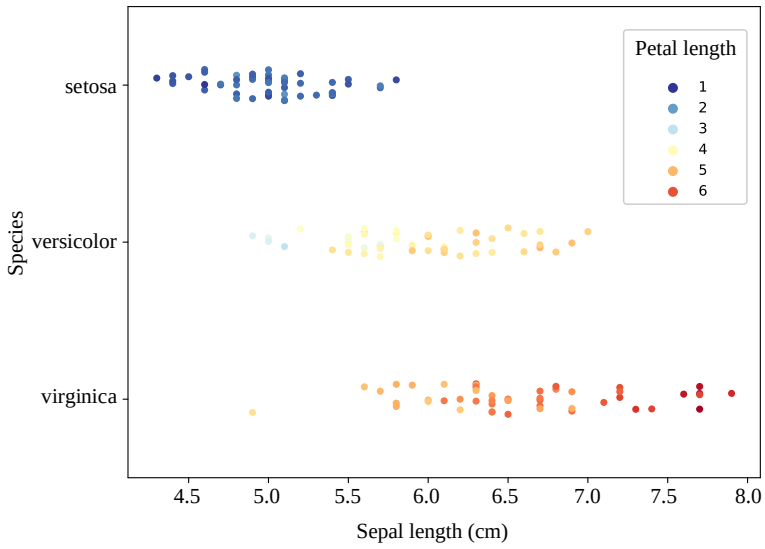


图 9. 分散点图

蜂群图

蜂群图 (swarm plot) 是一种用于可视化分类变量和数值变量关系的图表类型。它通过在分类轴上对数据进行分散排列，避免数据点的重叠，以展示数值变量在不同类别下的分布情况。每个数据点在分类轴上的位置表示其对应的数值大小，从而呈现出数据的密度和分布趋势。

蜂群图可以帮助我们比较不同类别之间的数值差异和趋势，适用于数据探索、特征分析和可视化报告等场景。图 10 所示为利用 seaborn.swarmplot() 绘制蜂群图。图 11 所示为考虑鸢尾花分类的蜂群图。

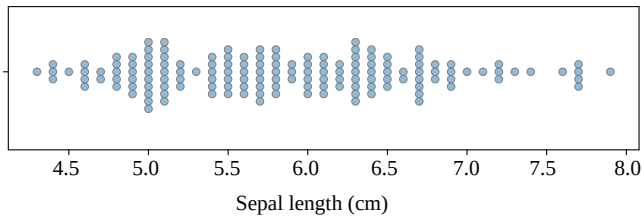


图 10. 蜂群图

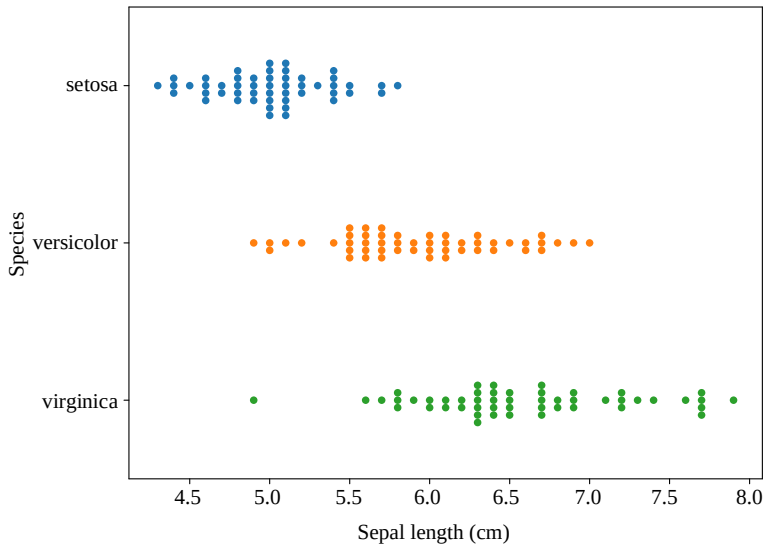


图 11. 蜂群图，考虑鸢尾花分类

箱型图

箱型图 (box plot) 是一种常用的统计图表，用于展示数值变量的分布情况和异常值检测。它通过绘制数据的五个关键统计量 (最小值、第一四分位数 Q_1、中位数 Q_2、第三四分位数 Q_3、最大值) 以及可能存在的异常值来提供对数据的直观概览。

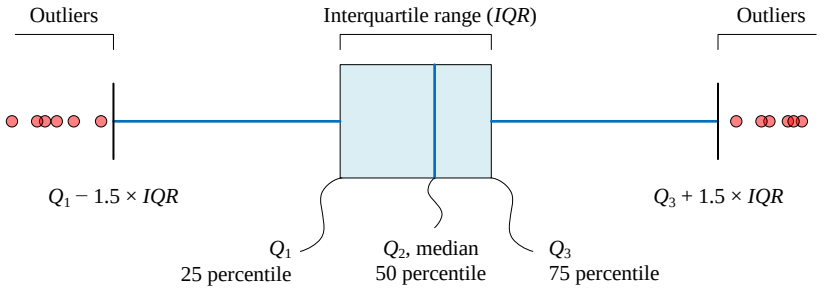


图 12. 箱型图原理

什么是四分位？

四分位是统计学中用于描述数据集分布的概念，将数据按大小顺序分成四等份。第一个四分位数 Q_1 表示 25%的数据小于或等于它，第二个四分位数 Q_2 是中位数，表示 50%的数据小于或等于它，第三个四分位数 Q_3 表示 75%的数据小于或等于它。四分位可以帮助了解数据的中心趋势、分散程度和异常值。四分位与盒须图、离群值检测等统计分析方法密切相关。

箱型图的主要元素包括：

箱体 (box)：由第一四分位数 Q_1 和第三四分位数 Q_3 之间的数据范围组成。箱体的高度表示数据的四分位距 $IQR = Q_3 - Q_1$，箱体的中线表示数据的中位数。

须 (whisker)：延伸自箱体的线段，表示数据的整体分布范围。通常，须的长度为 1.5 倍的四分位距。

异常值 (outliers)：位于须之外的数据点，被认为是异常值，可能表示数据中的极端值或异常观测。

通过观察箱型图，可以快速了解数据的中心趋势、离散程度以及是否存在异常值等关键信息。

图 13 所示为利用 seaborn.boxplot() 绘制的鸢尾花花萼长度样本数据的箱型图。图 14 所示为考虑鸢尾花分类的箱型图。

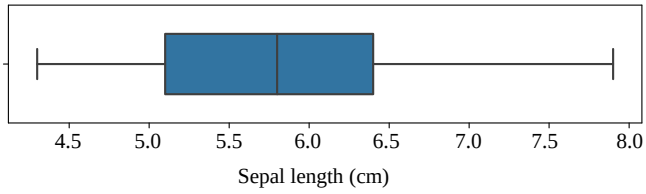


图 13. 箱型图

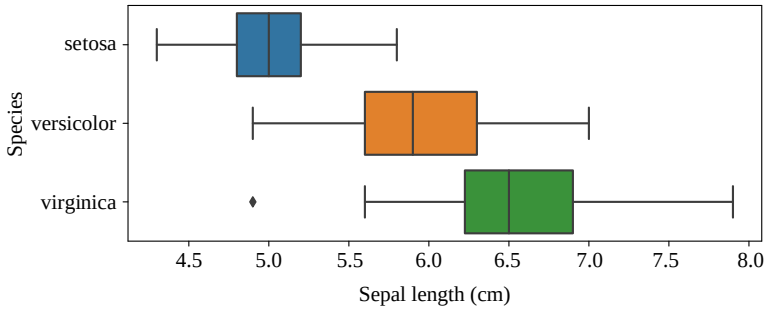


图 14. 箱型图，考虑鸢尾花分类

小提琴图

小提琴图 (violin plot) 是一种用于可视化数值变量分布的图表类型。它结合了核密度估计曲线和箱型图的特点，可以同时展示数据的分布形状、中位数、四分位数和离群值等信息。seaborn.violinplot() 是 Seaborn 库中用于绘制小提琴图的函数。

小提琴图的主要组成部分包括：

背景形状：由核密度估计曲线组成，表示数据在不同值上的概率密度。

中位数线：位于核密度估计曲线的中间位置，表示数据的中位数。

四分位线：分别位于核密度估计曲线的 25% 和 75% 位置，表示数据的四分位范围。

离群值点：位于核密度估计曲线之外的离群值数据点。

图 15 所示为用 seaborn.violinplot() 绘制的鸢尾花花萼长度样本数据的小提琴图。图 16 为考虑鸢尾花分类的小提琴图。图 17 所示为“蜂群图 + 小提琴图”可视化方案。

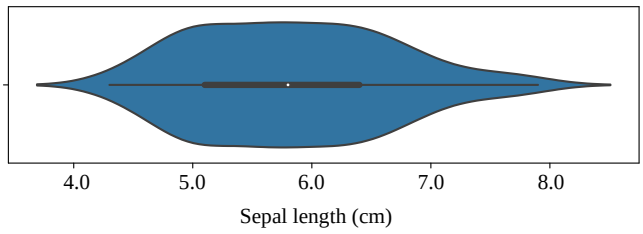


图 15. 小提琴图

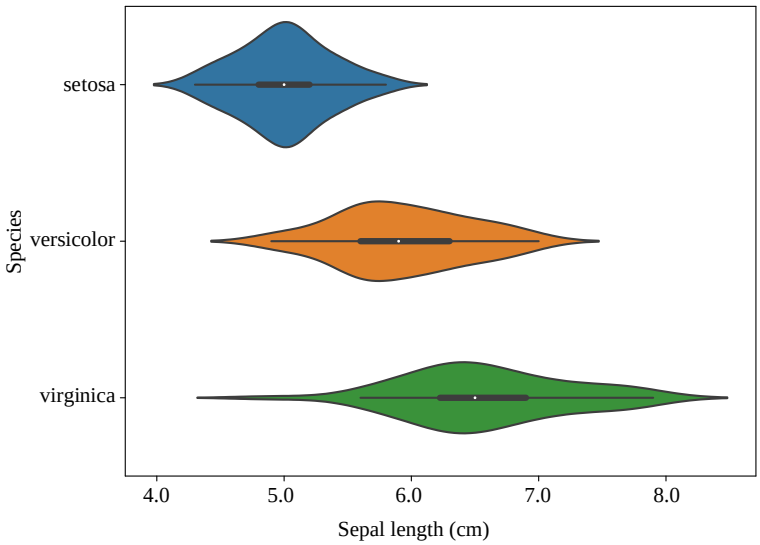


图 16. 小提琴图，考虑鸢尾花分类

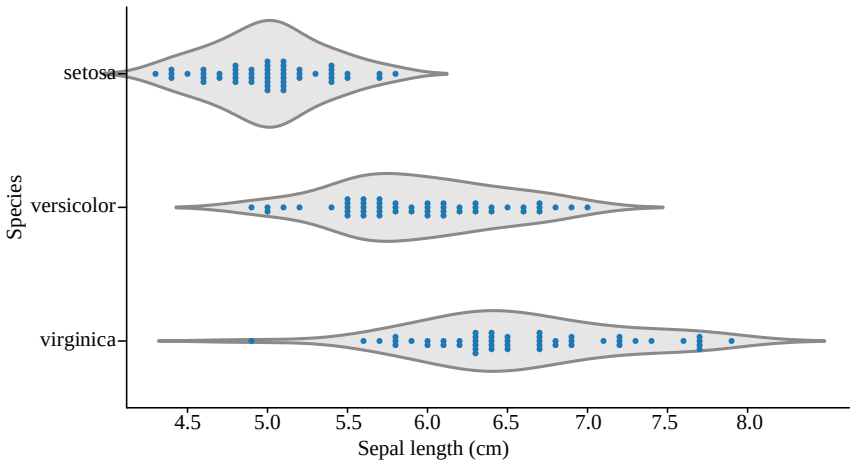


图 17. 蜂群图 + 小提琴图，考虑鸢尾花分类

29.3 二元

散点图

散点图是一种数据可视化图表，用于展示两个变量之间的关系。它通过在坐标系中以点的形式表示每个数据点，横轴代表一个变量，纵轴代表另一个变量。散点图可以帮助我们观察和分析数据点之间的趋势、分布和相关性。通过观察点的聚集程度和分布形状，我们可以推断两个变量之间的关系类型，如线性正相关、线性负相关、线性无关，甚至是非线性关系。

图 18 所示为利用 seaborn.scatterplot() 绘制的散点图，散点图的横轴为花萼长度、纵轴为花萼宽度。通过观察散点趋势，可以发现花萼长度、花萼宽度似乎似乎存在线性正相关。但是实际情况可能并非如此。本章最后将通过线性相关性系数进行量化确认。

图 18 这幅图中，我们还用毛毯图分别可视化花萼长度、花萼宽度的分布情况。

用不同颜色散点代表鸢尾花分类，我们便得到图 19 所示散点图。观察这幅图中蓝色点，即 setosa 类，我们可以发现更强的线性正相关性。

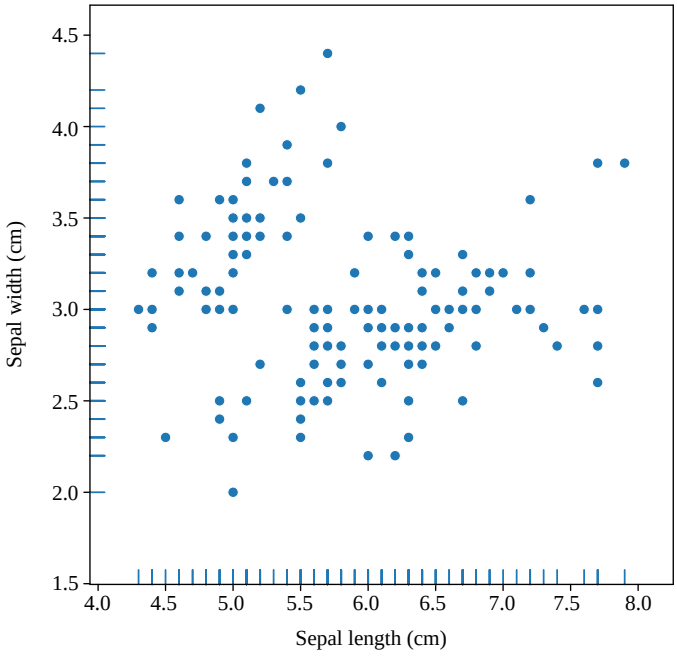


图 18. 散点图 + 毛毯图

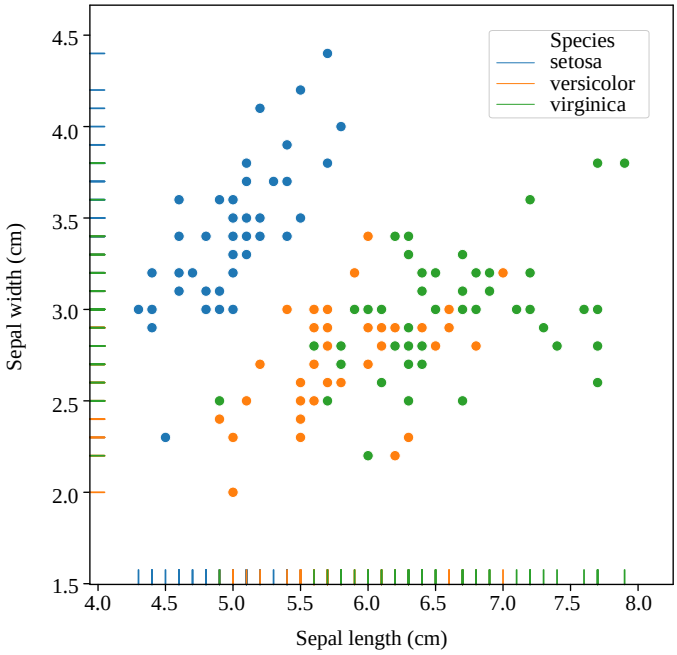


图 19. 散点图 + 毛毯图，考虑鸢尾花分类

二元直方图

本章前文，我们将一元样本数据划分成不同区间便可以绘制一元直方图。类似地，如果我们把图 18 所示平面划分成如图 20 所示一系列格子，计算每个格子中的样本数，我们便可以绘制类似图 21 二元直方图。显然，这种可视化方案并不理想。一方面“柱子”的高度很难确认，而且固定某个特定视角之后，一些较矮的“柱子”必定会被遮挡。因此，在实践中我们常常使用二元直方热图作为可视化方案。

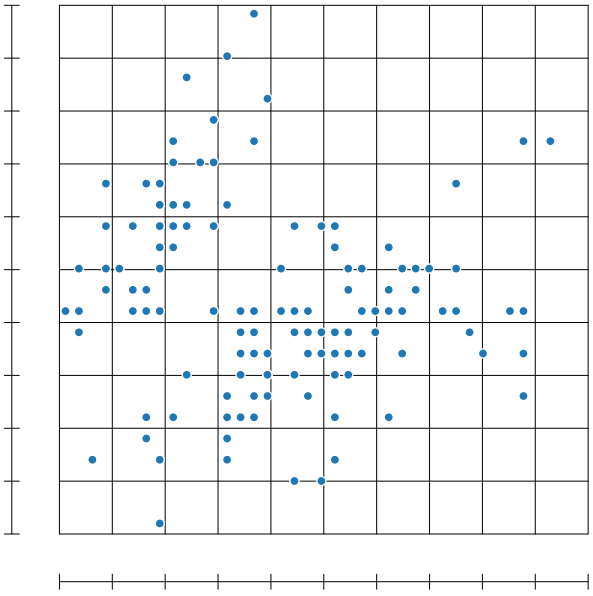


图 20. 二元直方图原理

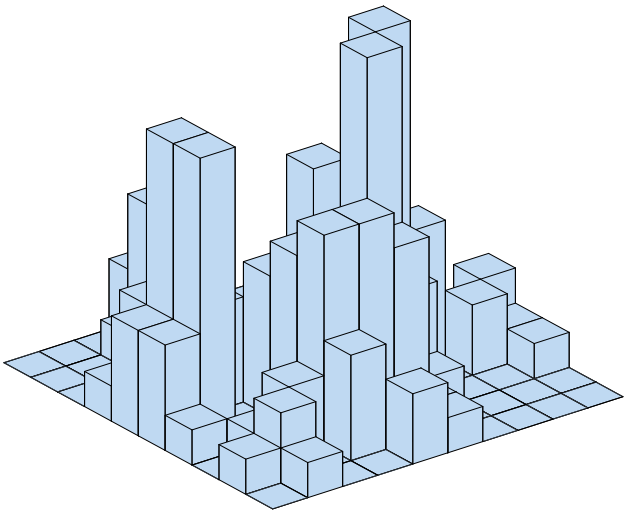


图 21. 二元直方图，柱状图可视化方案

二元直方热图由一个矩形网格组成，其中每个单元格的颜色代表了对应的数据频数、概率、概率密度。通常，行和列代表两个不同的随机变量，而单元格中的颜色强度表示频数、概率、概率密度。

二元直方热图可以帮助我们观察两个变量之间的关系以及它们的分布模式。通过观察颜色的变化和集中区域，我们可以得出关于两个变量之间的相关性、联合分布和潜在模式的初步结论。

所示为利用 seaborn.displot() 绘制的二元直方热图，横轴为鸢尾花花萼长度，纵轴为花萼宽度。如图 23 所示，二元直方热图沿着某个方向压缩便得到一元直方图；反过来看，直方图沿着特定方向展开便得到二元直方热图。

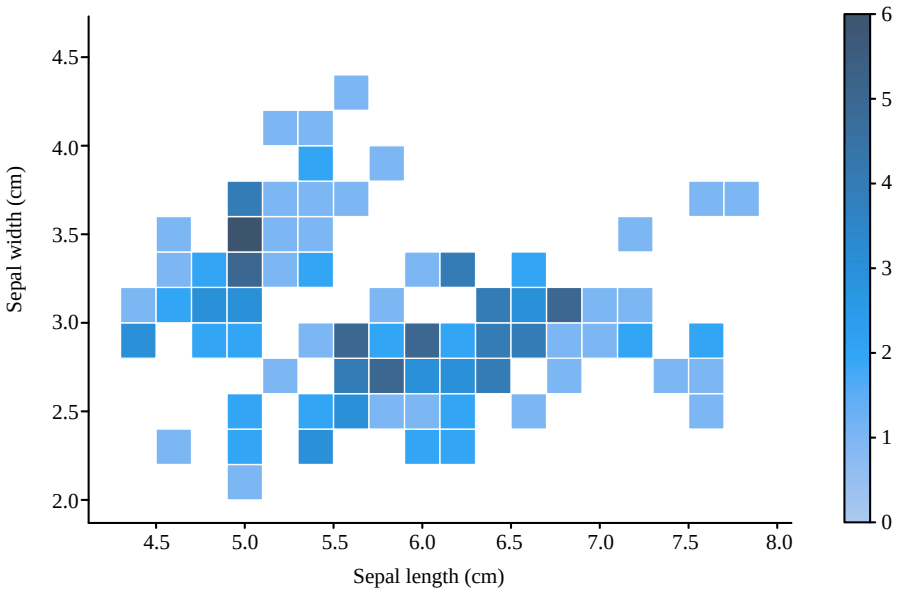


图 22. 鸢尾花花萼长度、花萼宽度的二元直方热图

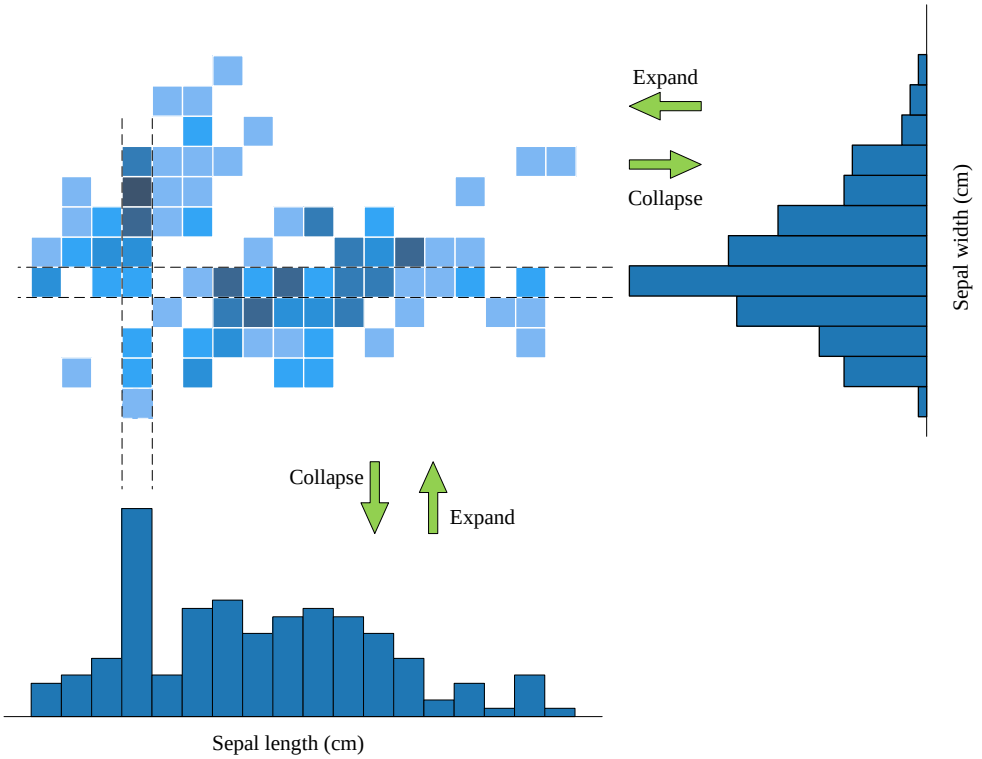


图 23. 一元直方图和二元直方热图之间关系

联合分布

前文的高斯核函数 KDE 也可以用在估算二元联合分布。图 24 所示为 seaborn.kdeplot() 绘制鸢尾花花萼长度、花萼宽度联合分布概率密度估计等高线。图 24 (b) 还考虑了鸢尾花三个不同类别。

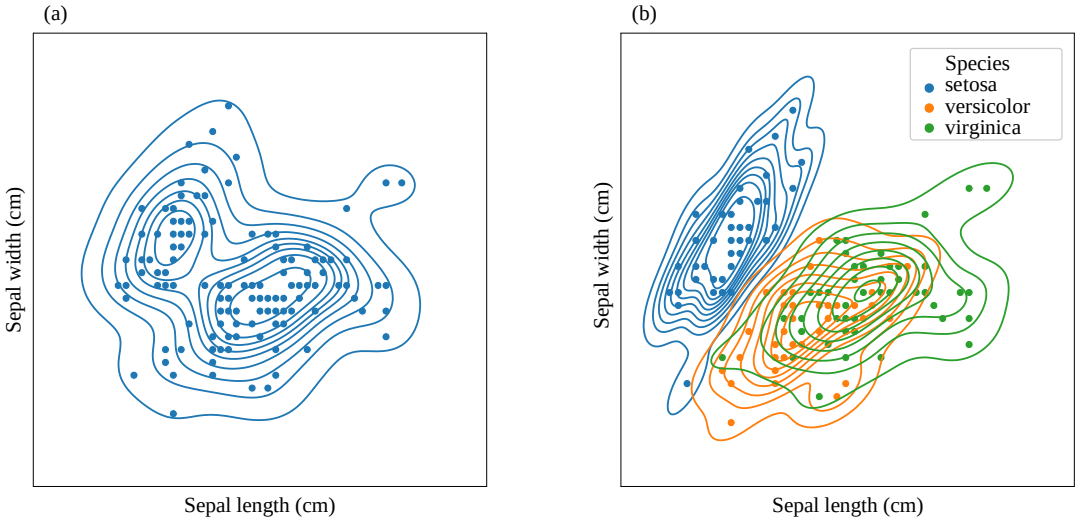


图 24. 鸢尾花花萼长度、花萼宽度的联合分布，高斯核密度估计

什么是联合分布？

联合分布是统计学中用于描述两个或多个随机变量同时取值的概率分布。它提供了关于多个变量之间关系的信息，包括它们的联合概率、相互依赖程度以及共同变化的模式。联合分布可以以多种形式呈现，如概率质量函数（离散变量）或概率密度函数（连续变量）。通过分析联合分布，我们可以洞察变量之间的相关性、条件概率以及预测和推断未来事件的可能性。联合分布在概率论、统计建模、数据分析和机器学习等领域具有广泛应用。

边缘分布

图 25 所示为利用 seaborn.jointplot() 可视化“联合分布 + 边缘分布”。

seaborn.jointplot() 函数用于创建联合图，结合了两个变量的散点图和各自的边缘分布图。它可以帮助我们同时可视化两个变量之间的关系以及它们的边缘分布。seaborn.jointplot() 函数默认情况下会绘制散点图和边缘直方图。散点图展示了两个变量之间的关系，而边缘直方图则分别显示了每个变量的边缘分布情况。

本章配套 Jupyter Notebook 还提供 seaborn.jointplot() 其他几种可视化方案，请大家自行学习。

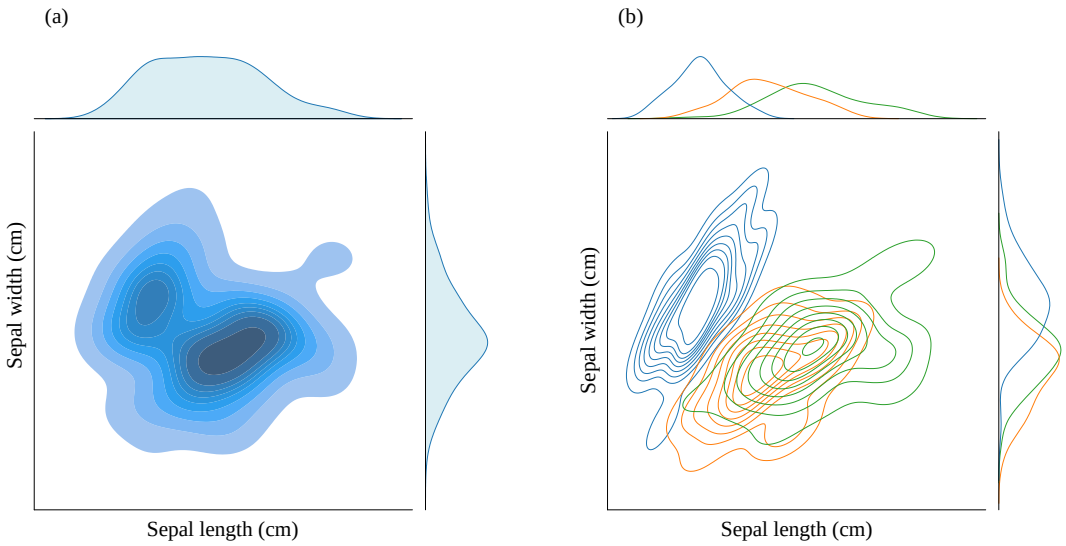


图 25. 鸢尾花花萼长度、花萼宽度的联合分布和边缘分布

什么是边缘分布？

边缘分布是指在多变量数据集中，针对单个变量的分布情况。它表示了某个特定变量在与其他变量无关时的概率分布。边缘分布可以通过将多变量数据集投影到某个特定变量的轴上来获得。通过分析边缘分布，我们可以了解每个变量单独的分布特征，包括均值、方差、偏度、峰度等统计量，以及分布的形状和模式。边缘分布对于探索数据集的特征、进行单变量分析和了解数据的单个方面非常有用。

线性回归

图 26 所示为利用 seaborn.lmplot() 绘制的鸢尾花花萼长度、花萼宽度之间的线性回归关系图。seaborn.lmplot() 函数默认情况下会绘制散点图和拟合的线性回归线。散点图展示了两个变量之间的关系，而线性回归线表示了拟合的线性关系。

除了基本语法外，seaborn.lmplot() 还支持其他参数，例如 hue 参数用于指定一个额外的分类变量，可以通过不同的颜色展示不同类别的数据点和回归线。

《数据有道》第 9、10 章专门介绍线性回归。

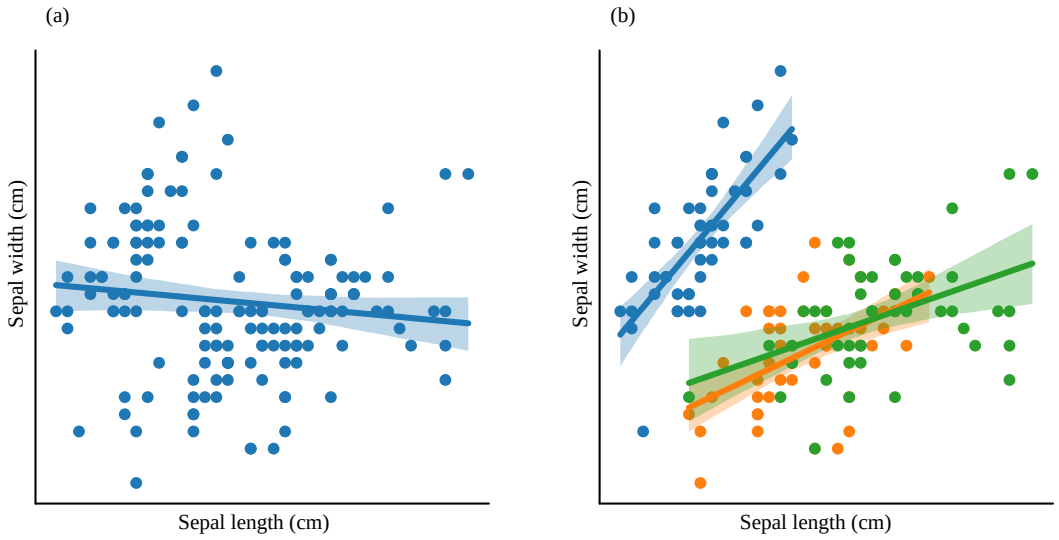


图 26. 鸢尾花花萼长度、花萼宽度的线性回归关系

29.4 多元

分散点图、小提琴图

我们当然可以使用一元可视化方案展示多元数据的特征，如所示。但是这两幅图最致命的缺陷是仅仅展示单个特征分布，并没有展示特征之间的联系。下面我们聊聊其他能够可视化多元特征之间关系的可视化方案。

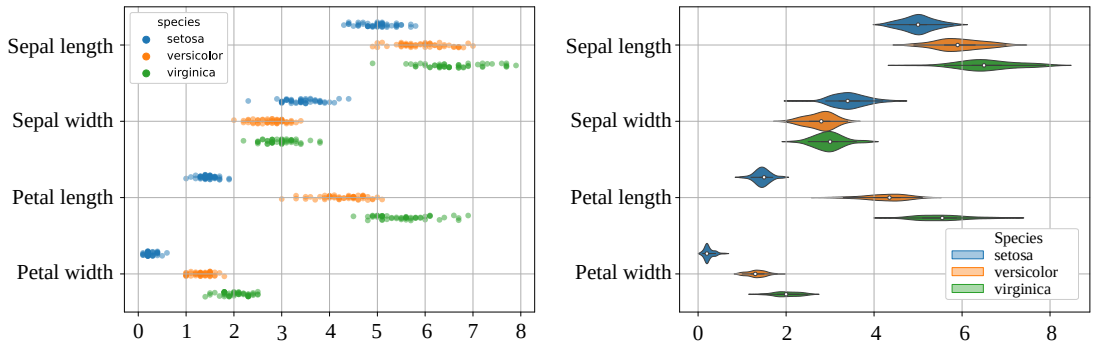


图 27. 分散点图、小提琴图

聚类热图

seaborn.clustermap()函数用于创建聚类热图，它能够可视化数据集中的聚类结构和相似性。聚类热图使用层次聚类算法对数据进行聚类，并以热图的形式展示聚类结果。

聚类热图的原理是通过计算数据点之间的相似性（例如欧几里得距离或相关系数），然后使用层次聚类算法将相似的数据点分组为聚类簇。层次聚类将数据点逐步合并形成聚类树状结构，根据相似性的距离进行聚类的层次化过程。聚类热图将聚类树状结构可视化为热图，同时显示数据点的排序和聚类关系。

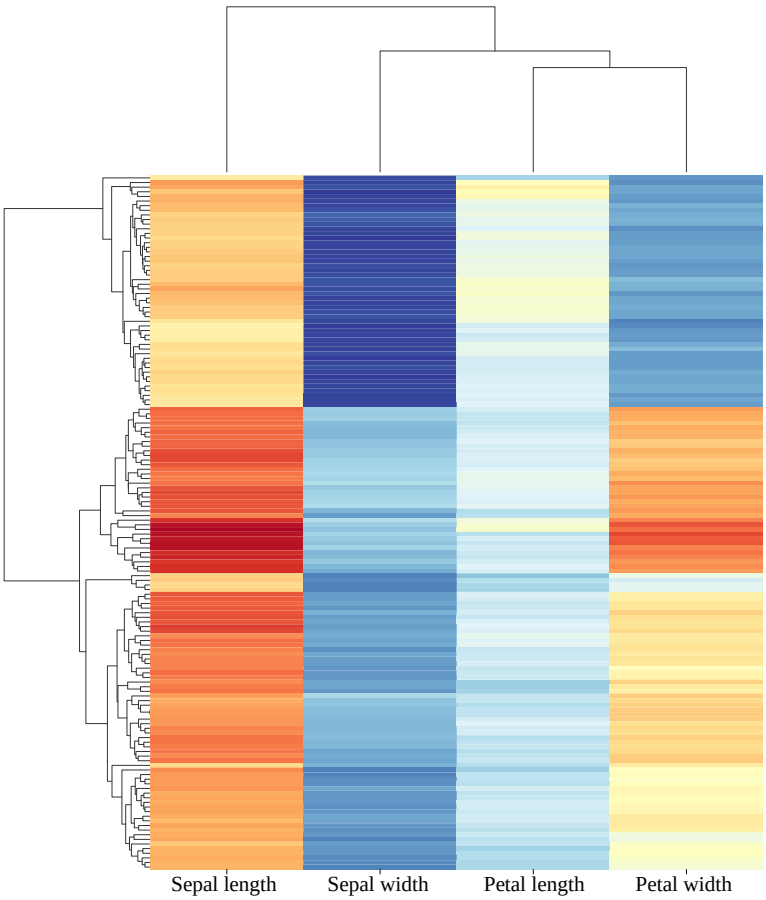


图 28. 鸢尾花数据集，聚类热图

《机器学习》将专门介绍各种聚类算法。

什么是聚类？

机器学习中的聚类是一种无监督学习方法，用于将数据集中的样本按照相似性进行分组或聚集。聚类算法通过自动发现数据的内在结构和模式，将相似的样本归为一类，从而实现数据的分组和分类。聚类的目标是使得同一类别内的样本相似度高，而不同类别之间的样本相似度低。聚类算法通常基于样本之间的距离或相似性度量进行操作，例如欧几里得距离、余弦相似度等。常见的聚类算法包括 K 均值聚类、层次聚类、DBSCAN、高斯混合模型等。

成对特征散点图

seaborn.pairplot() 函数用于创建成对特征散点图矩阵，可视化多个变量之间的关系和分布。它会将数据集中的每对特征绘制为散点图，并展示变量之间的散点关系和单变量的分布。

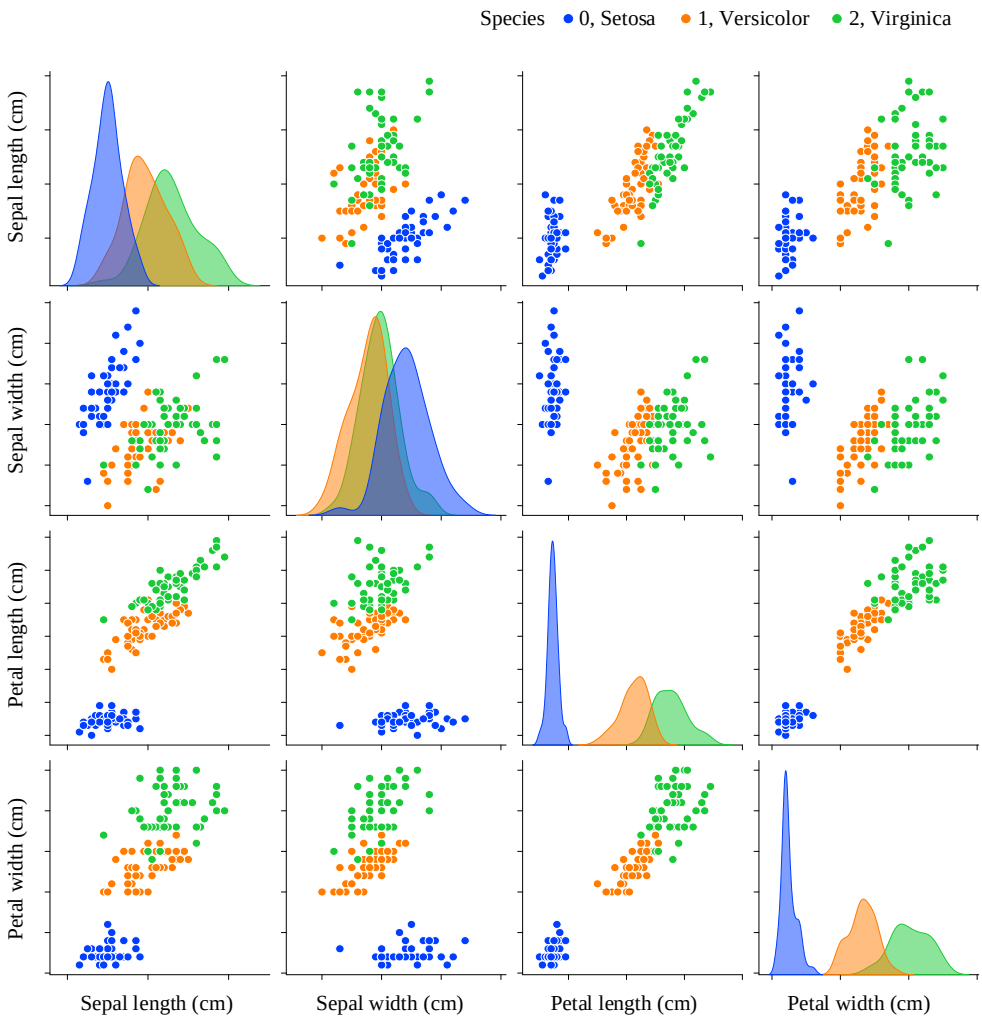


图 29. 鸢尾花数据成对特征散点图，考虑分类标签

seaborn.pairplot() 函数会根据数据集中的每对特征生成散点图，并以网格矩阵的形式展示。对角线上的图形通常是单变量的直方图或核密度估计图，表示每个变量的分布情况。非对角线上的图形是两个变量之间的散点图，展示它们之间的关系。

此外，seaborn.pairplot()函数还支持其他参数，例如 hue 参数用于根据一个分类变量对散点图进行颜色编码，使不同类别的数据点具有不同的颜色。

通过使用 seaborn.pairplot()函数，我们可以轻松地可视化多个变量之间的关系和分布。这对于探索变量之间的相关性、识别数据中的模式和异常值等非常有用。

平行坐标图

平行坐标图是一种可视化多个连续变量之间关系的图形方法。它使用平行的垂直线段来表示每个变量，这些线段相互平行并沿着水平轴排列。每个变量的值通过垂直线段在对应的轴上进行表示。

在平行坐标图中，每个数据样本由一条连接不同垂直线段的折线表示。这条折线的形状和走势反映了数据样本在不同变量之间的关系。通过观察折线的走势，我们可以识别出变量之间的相对关系，例如正相关、负相关或无关系。同时，我们也可以通过折线的位置和形状来比较不同样本之间的差异。

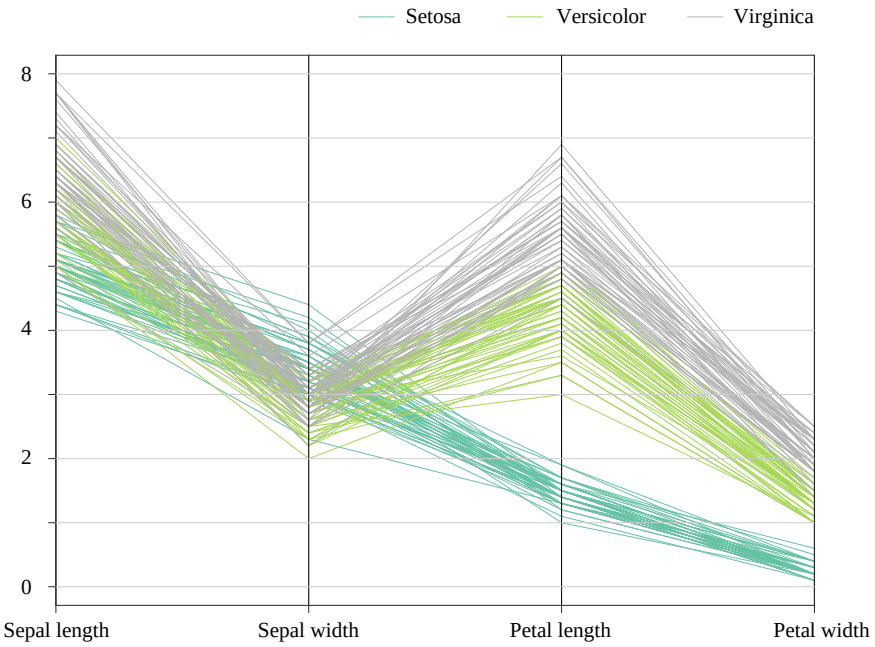


图 30. 鸢尾花数据，平行坐标图

平行坐标图常用于数据探索、特征分析和模式识别等任务。它能够帮助我们发现多个变量之间的关系、观察变量的分布模式，并对数据样本进行可视化比较。此外，通过添加颜色映射或其他可视化元素，还可以在平行坐标图中显示附加信息，例如类别标签或异常值指示。

注意，目前 Seaborn 并没有绘制平行坐标图的工具，本章配套的 Jupyter Notebook 中采用的是 pandas.plotting.parallel_coordinates() 函数。

类似平行坐标图的可视化方案还有安德鲁斯曲线 (Andrews curves)。在安德鲁斯曲线中，每个特征被映射为一个三角函数（通常是正弦函数和余弦函数），并按照给定的顺序排列。本章配套的 Jupyter Notebook 用 pandas.plotting.andrews_curves 绘制了鸢尾花样本数据的安德鲁斯曲线。

量化多特征样本数据任意两个随机变量关系的最方便的工具莫过于协方差矩阵、相关性系数矩阵。这是上一章已经介绍过的内容，本章不再赘叙。

请大家完成下面 2 道题目。

Q1. 请大家分别绘制鸢尾花花萼宽度、花瓣长度、花瓣宽度的直方图、KDE 概率密度估计。

Q2. 请大家绘制鸢尾花花萼长度、花瓣长度的散点图、二元直方热图、联合分布 KDE 等高线。

* 本章题目不提供答案。

Timeseries Data in Pandas

30 Pandas 时间序列数据

重塑数组的维数、形状

很难做出预测，尤其是对未来的预测。

It is difficult to make predictions, especially about the future.

—— 尼尔斯·玻尔 (Niels Bohr) | 丹麦物理学家 | 1885 ~ 1962

30.1 什么是时间序列?

时间序列 (timeseries) 是指按照时间顺序排列的一系列数据点或观测值，通常是等时间间隔下的测量值，如每天、每小时、每分钟等。时间序列数据通常用于研究时间相关的现象和趋势，例如股票价格、气象数据、经济指标等。图 1 (a) 所示为标普 500 (S&P 500) 数据。

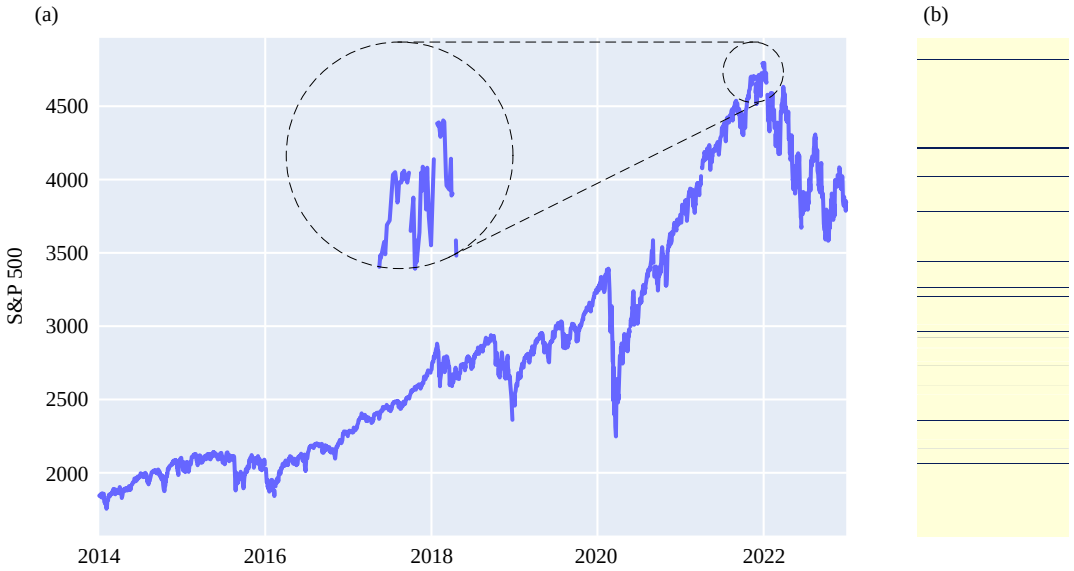


图 1. 标普 500 数据，含有缺失值

时间序列分析是一种重要的数据分析方法，它可以用于预测未来的趋势和变化，评估现有趋势的稳定性和可靠性，并发现异常点和异常趋势。时间序列分析通常包括以下几个步骤：

- 数据预处理：对数据进行清洗、去噪、填补缺失值等操作，以提高数据质量和可靠性。
- 时间序列的可视化：对数据进行绘图，以了解数据的分布、趋势和周期性。
- 时间序列的统计分析：对数据进行时间序列分解、平稳性检验、自相关性检验等统计分析，以评估数据的稳定性和相关性。
- 时间序列的建模和预测：根据统计分析的结果，建立合适的时间序列模型，进行未来趋势的预测和评估。

比如，在图 1 (a) 中被局部放大的曲线上，大家已经看到了缺失值。图 1 (b) 用热图可视化缺失值的位置。在本章配套的代码中，大家会看到经过计算缺失值的占比约为 3.5%。

本章仅仅采用“图解”介绍部分时间序列分析，《数据有道》一册将专门介绍时间序列相关话题。

Pandas 中的时间序列功能

在 Python 中，Pandas 库提供了强大的时间序列处理和分析功能，使得时间序列的处理和分析变得更加简单和高效。在 Pandas 中，时间序列分析的主要方法包括：

- 创建时间序列：可以通过 pandas.date_range() 方法创建一个时间范围，或者将字符串转换为时间序列对象。
- 时间序列索引：可以使用时间序列作为 DataFrame 的索引，从而方便地进行时间序列分析。
- 时间序列的切片和索引：可以使用时间序列的标签或位置进行切片和索引。
- 时间序列的重采样：可以将时间序列转换为不同的时间间隔，例如将日频率的数据转换为月频率的数据。
- 移动窗口函数：可以对时间序列数据进行滑动窗口操作，计算滑动窗口内的统计指标，例如均值、方差等。
- 时间序列的分组操作：可以将时间序列数据按照时间维度进行分组，从而进行聚合操作，例如计算每月的平均值、最大值等。
- 时间序列的聚合操作：可以对时间序列数据进行聚合操作，例如计算每周、每月、每季度的总和、平均值等。
- 时间序列的可视化：可以使用 Pandas、Matplotlib、Seaborn、Plotly 等库对时间序列数据进行可视化，例如绘制线形图、散点图、直方图等。

30.2 缺失值

缺失值 (missing value) 指的是数据集中的某些值缺失或未被记录的情况。它们可能是由于测量设备故障、记录错误、样本丢失或数据清洗不完整等原因导致的。缺失值可能在数据分析和建模中产生严重的影响，因为它们会导致数据样本的大小不一致，使得数据的统计分布和关系不准确或无法得出。另外，许多机器学习算法无法处理缺失值，必须对其进行处理或者删除。

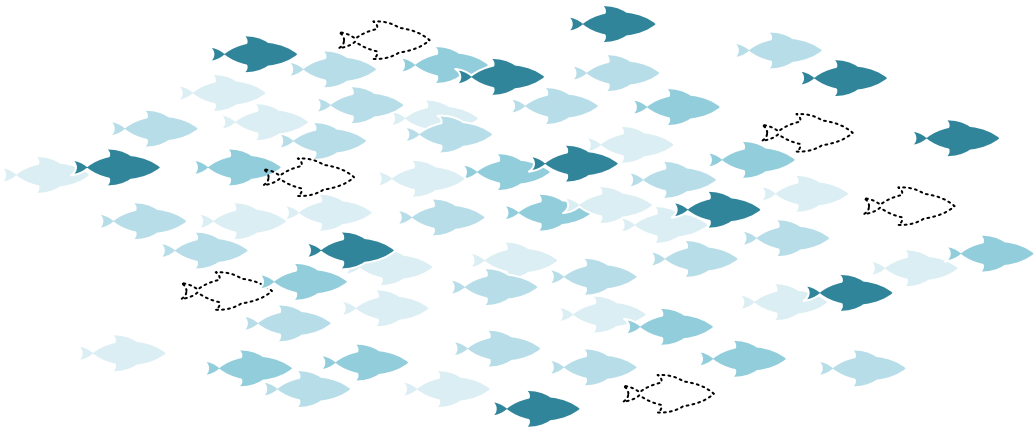


图 2. 缺失值

在数据处理中，通常需要对缺失值进行识别、处理或删除。一些处理缺失值的方法包括：

- 删除带有缺失值的样本或变量。
- 使用常量填充缺失值，例如用零、平均值、中位数等常量填充。
- 使用回归模型、插值方法等技术，对缺失值进行预测和填充。
- 对于分类变量，可以创建一个新的类别来表示缺失值。

在选择处理方法时，需要根据具体情况和数据分析的目的来决定。

图 1 中的缺失值则对应非营业日，比如周六日、节假日等。将这些缺失值删除之后，我们便得到图 3 所示的趋势。为了醒目地观察每年趋势，我们绘制了图 4。

鸢尾花书《数据有道》将介绍处理缺失值的各种方法。

图 3. 标普 500 数据，删除缺失值

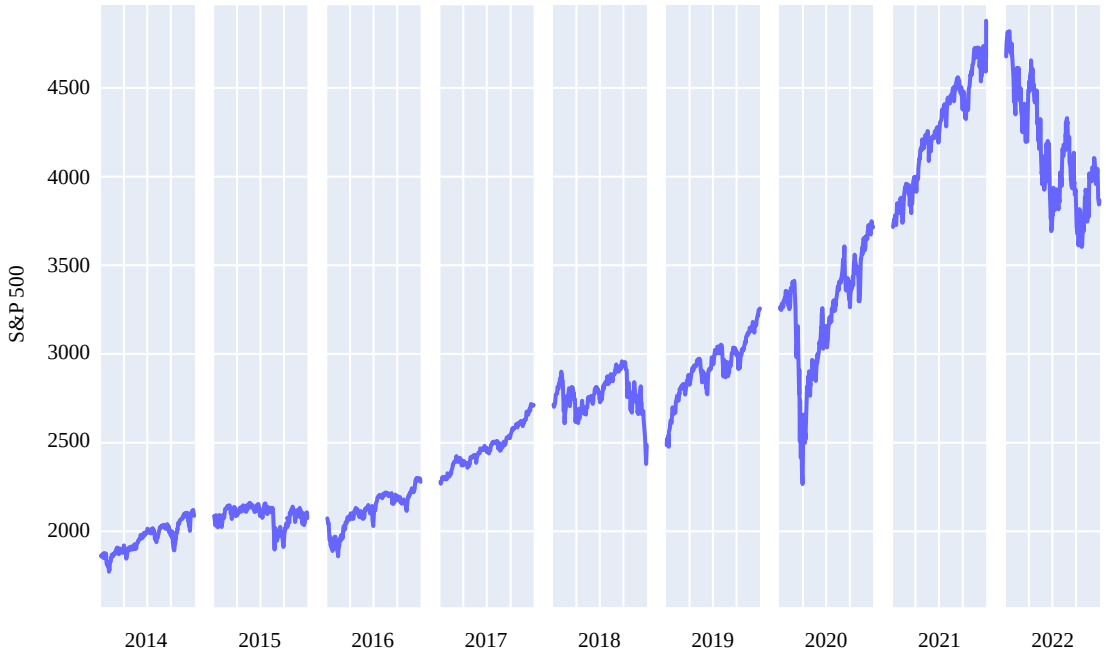


图 4. 标普 500 数据，按年观察趋势

什么是离群值？

在统计学和数据分析中，离群值 (outlier) 指的是在数据集中与其他数据值显著不同的异常值。它们可能是由于测量误差、实验异常、录入错误、样本损坏或数据处理错误等因素导致的。离群值具有比其他数据点更大或更小的数值，与其他数据点之间的差异通常非常显著。

离群值会对数据分析结果产生影响，比如对平均值、方差、相关性等统计指标的计算都会受到其影响。因此，在数据分析和建模中，需要对离群值进行识别、处理或删除。常见的方法包括使用箱线图或 3σ 准则等方法来识别离群值，并根据具体情况进行处理或删除。如果离群值确实是数据中真实存在的异常值，则可能需要对其进行单独分析或建立针对其的模型。

本章不会介绍如何处理离群值，相关内容请参考《数据有道》。

30.3 移动平均

时间序列的移动平均 (moving average, MA) 是一种常用的平滑技术，用于去除序列中的噪声和波动，以便更好地观察和分析序列的长期趋势。

移动平均通过计算序列中一段固定长度（通常称为窗口）内数据点的平均值来平滑序列。窗口的大小决定了平滑的程度，较大的窗口将平滑更多的波动，但可能会导致较长的滞后。

具体步骤如下：

- 1) 选择窗口的大小，例如 10 个数据点。
- 2) 从序列的起始位置开始，计算窗口内数据点的平均值。
- 3) 将该平均值作为移动平均的第一个数据点，记录下来。
- 4) 移动窗口向后滑动一个数据点的位置。
- 5) 重复步骤 2 至 4，计算新窗口内的平均值，并记录下来。
- 6) 继续滑动窗口直到到达序列的末尾，得到一系列移动平均值。

移动平均的计算可以使用简单移动平均 (Simple Moving Average, SMA) 或加权移动平均 (Weighted Moving Average, WMA) 来进行。简单移动平均对窗口内的每个数据点赋予相等的权重，而加权移动平均则可以根据需求赋予不同的权重，以更强调某些数据点的重要性。

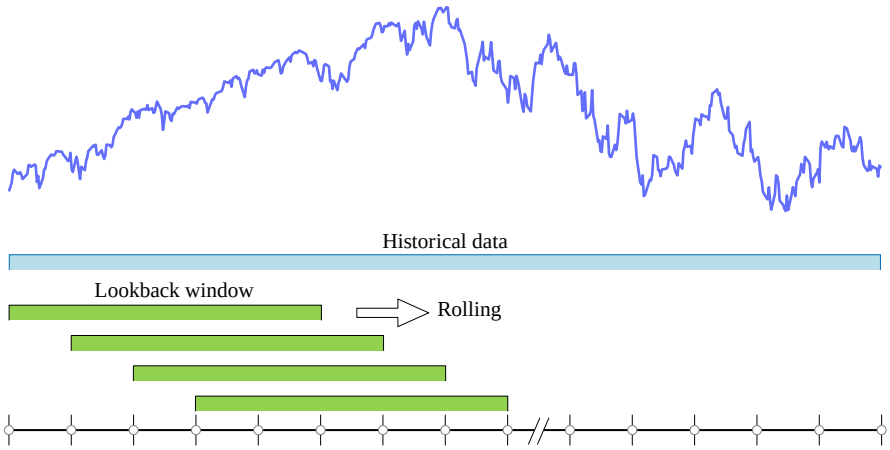


图 5. 移动窗口

通过计算移动平均，时间序列中的短期波动可以平滑，从而更容易观察到长期趋势和周期性变化。移动平均在金融分析、经济预测和数据分析等领域得到广泛应用。

图 6. 标普 500 数据，移动平均

30.4 收益率

为了量化股票市场的每日涨跌，我们需要计算股票的日收益率。计算当日收益率时需要知道两个关键数据点：股票的当日收盘价、前一日收盘价。

日收益率的计算公式为：日收益率 = (当日收盘价 − 前一日收盘价) / 前一日收盘价。将这个公式应用于具体的股票数据，就可以计算出每个交易日的日收益率。图 7 所示为标普 500 的日收益率。

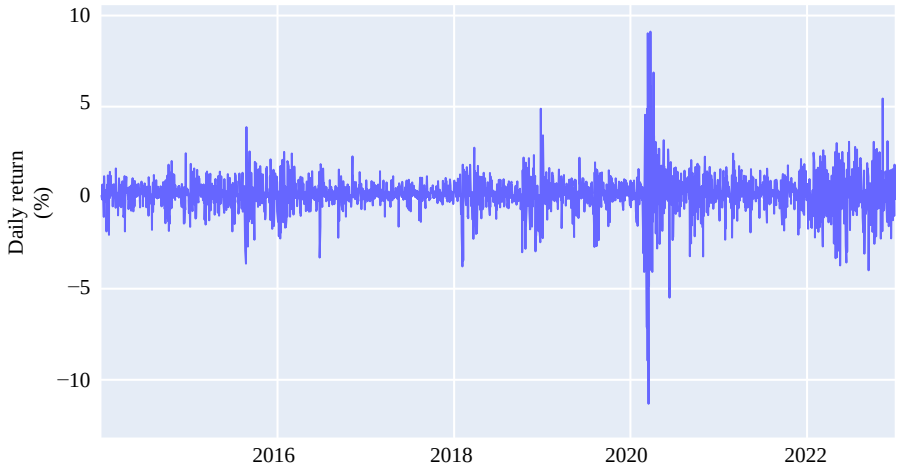


图 7. 标普 500 数据日收益率

为了更方便观察每年涨跌情况，我们绘制了图 8。

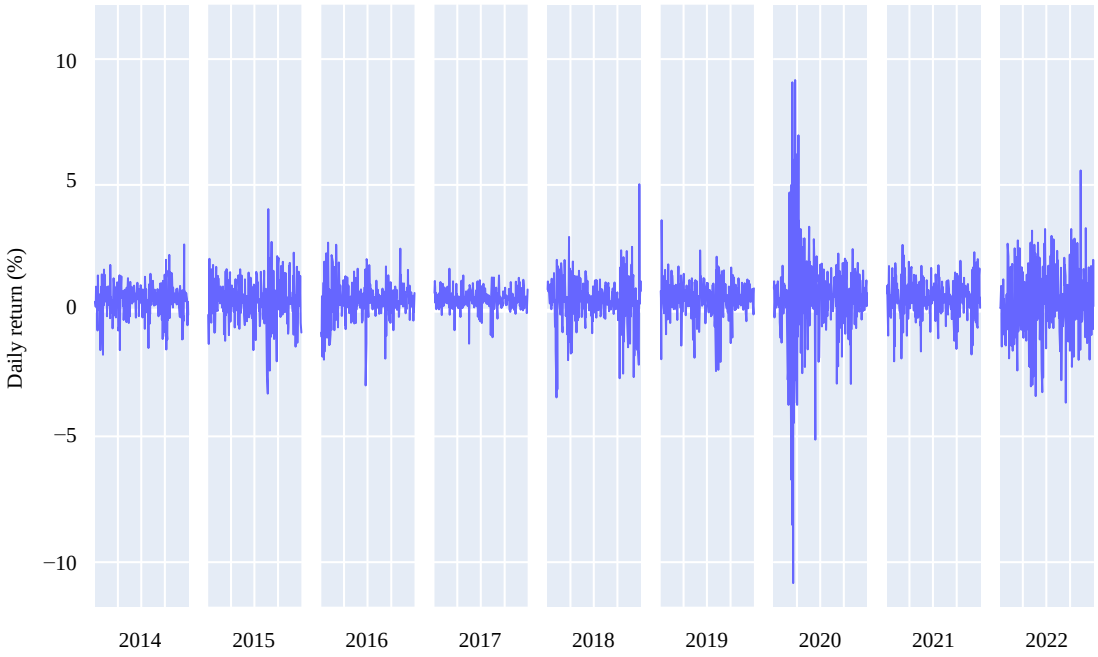


图 8. 标普 500 数据日收益率，按年观察趋势

30.5 统计分析

市场涨跌越越剧烈，曲线波动越剧烈。图 8 这些曲线类似随机行走，为了发现规律，我们需要借助统计工具。

年度分布

图 9 所示为下载所有数据计算得到日收益率绘制的分布图。大家可以从分布中计算得到均值和标准差。这个任务交给大家自行完成。图 10 所示为年度日收益率分布变化情况。

为了更好的量化股票的波动情况，我们需要一个指标——波动率 (volatility)。波动率是衡量其价格变动幅度的指标，常用的量化方法为历史波动率 (historical volatility)。历史波动率本质上就是一定回望窗口内收益率样本数据的标准差。

图 11 所示为利用水平柱状图可视化日收益率的年度均值、波动率 (标准差)。

此外，我们还可以使用山脊图 (ridgeline plot) 可视化每年收益率的分布情况，具体如图 12 所示。Joypy 是一个 Python 库，用于创建山脊图。山脊图是一种可视化工具，用于展示多个连续变量在一个维度上的分布，并且能够显示不同组之间的比较。

山脊图的特点是将多个曲线图，通常是核密度估计曲线，沿着一个共享的垂直轴线堆叠显示，形成一座山脉状的图形。每个曲线代表一个组或类别，可以通过颜色或其他视觉属性进行区分。要使用 Joypy 绘制山脊图，需要首先安装 Joypy 库，并导入 joyplot 模块。

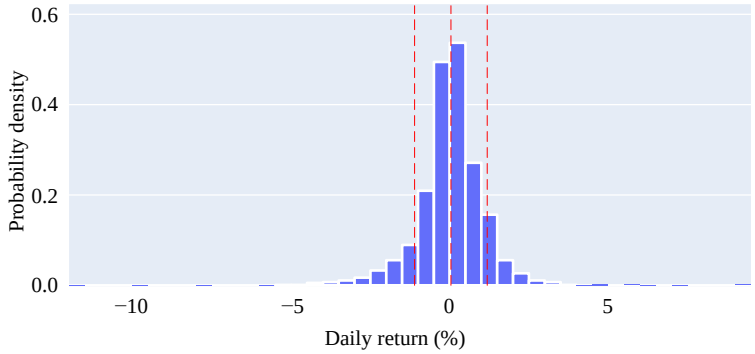


图 9. 所有下载历史数据日收益率分布

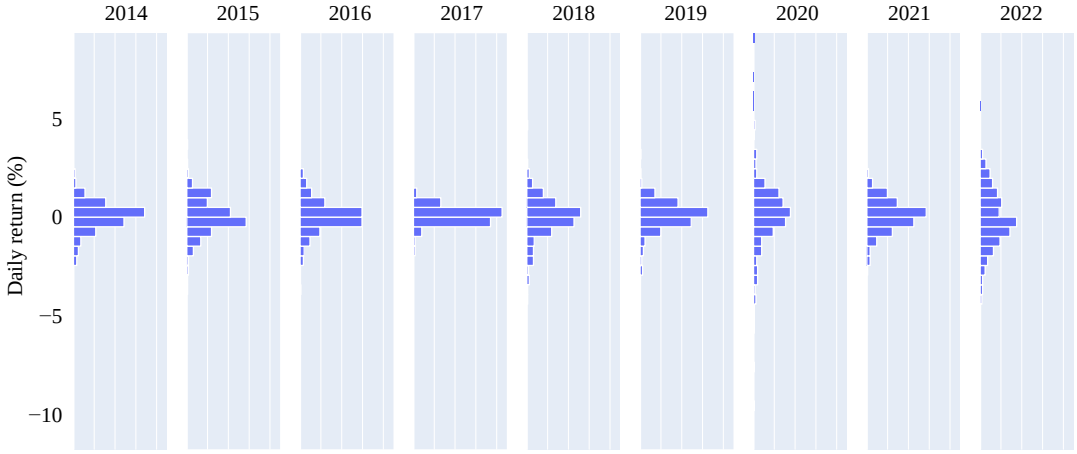


图 10. 日收益率分布，按年

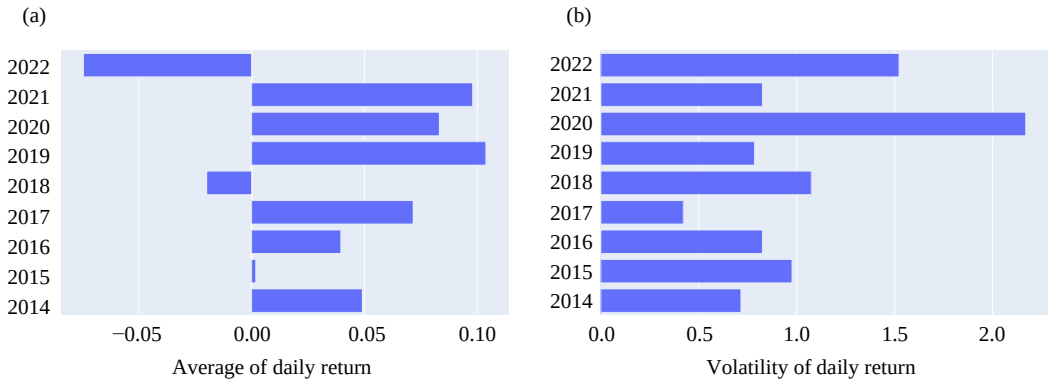


图 11. 水平柱状图可视化收益率均值、标准差 (波动率)，按年

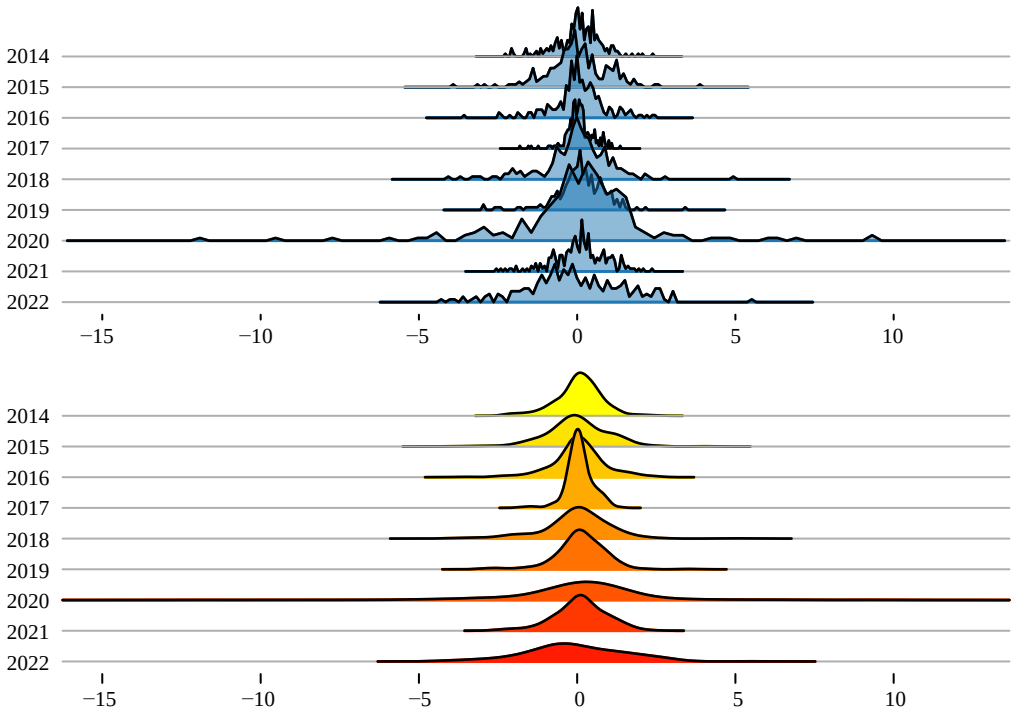


图 12. 山脊图，按年

季度分布

当然，我们也可以按季度分析收益率。图 13 所示为每个季度收益率的均值、标准差的柱状图。图 14 所示为每个季度收益率的山脊图。这幅图我们把纵轴的时间隐去。

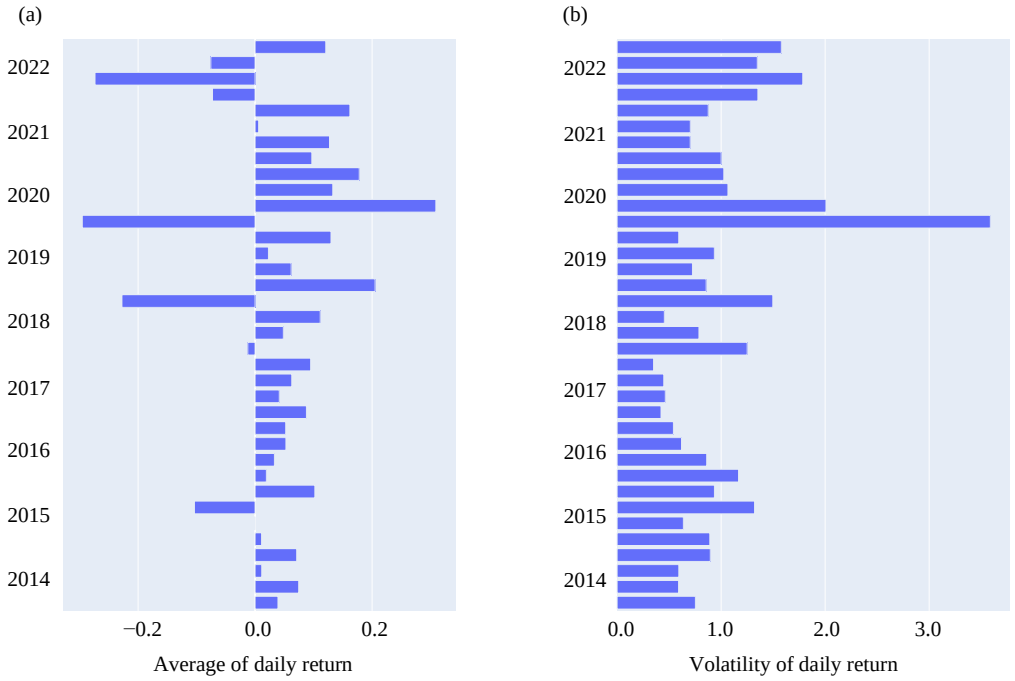


图 13. 水平柱状图可视化收益率均值、标准差 (波动率)，按季度

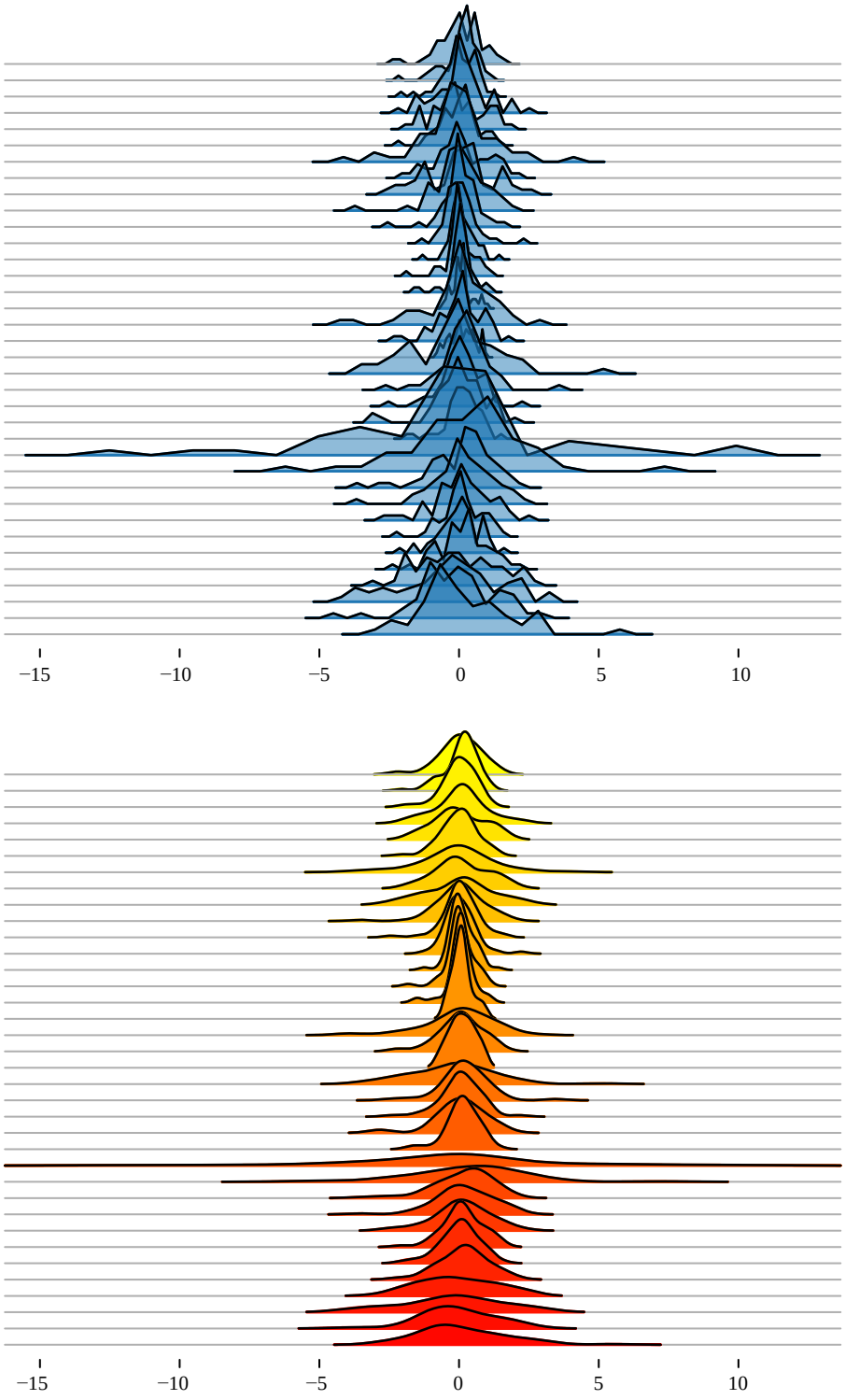


图 14. 山脊图，按季度

移动波动率

准确来说，历史波动率是根据过去一段时间内的股票价格数据计算得出的波动率。

可以选择一个时间窗口，例如 20 天 (一个月)、60 天 (一个季度)、125 或 126 天 (半年)、250 或 252 天 (一年)，计算每个交易日的收益率，然后求得其标准差，最终得到历史波动率。当这个回望窗口移动时，我们便得到移动波动率的时间序列数据。图 15 所示的移动波动率的回望窗口长度为 250 天 (营业日)。请大家自己修改回望窗口长度 (营业日数量)，比较移动波动率曲线。

《数据有道》还会专门介绍指数加权移动平均 EWMA 方法计算的均值和波动率。

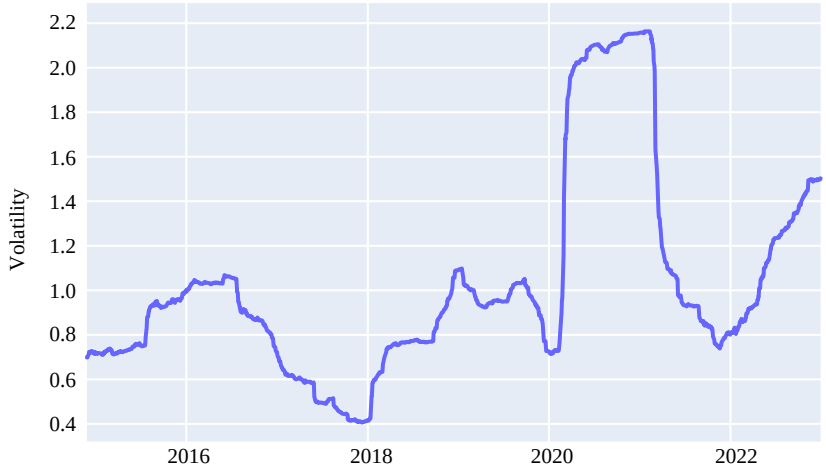


图 15. 移动波动率

30.6 相关性

几个不同时间序列之间肯定也会存在相关性。图 16 所示为标普 500 日收益率和三个汇率收益率之间的相关性系数矩阵热图。

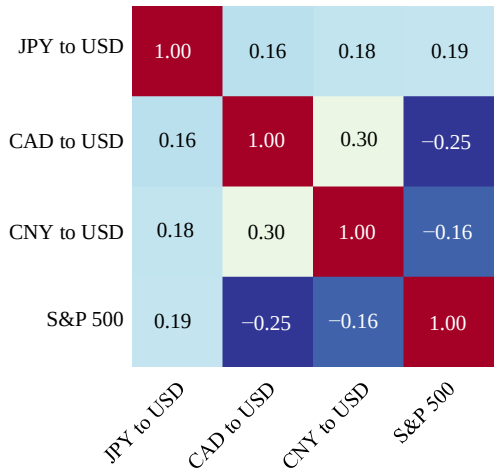


图 16. 相关性系数矩阵

相关性并不是一成不变的，也是随时间不断变化。如图 17 所示，当我们指定具体的移动窗口长度，也可以计算移动相关性。

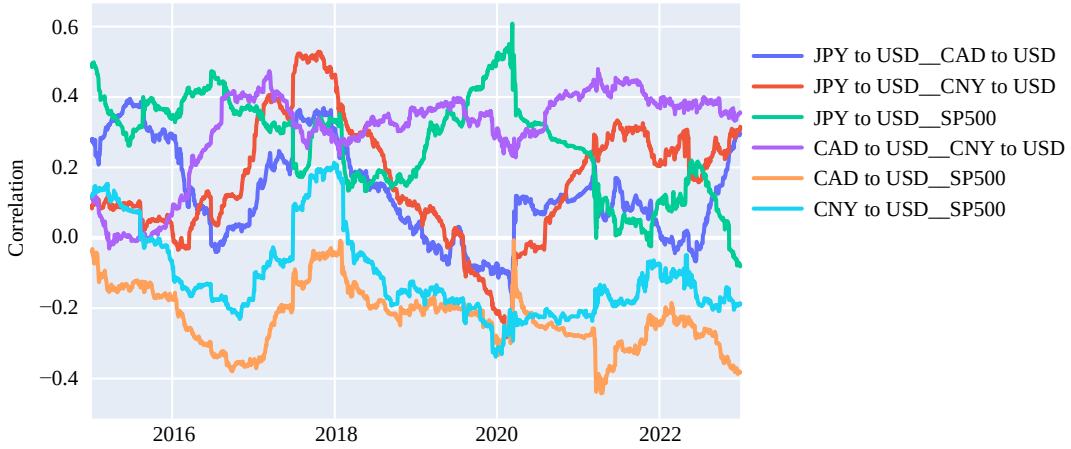


图 17. 移动相关性

对时间序列历史数据完成分析后自然少不了预测这个环节。本书不会展开讲解，请大家参考《数据有道》。

请大家完成下面这道题目。

Q1. 请大家把本章配套代码中历史数据截止时间修改为最近日期，重新下载数据逐步完成本章前文时间序列分析。

* 本章不提供答案。

有关 Pandas 中时间序列更多用法，请大家参考：

https://pandas.pydata.org/docs/user_guide/timeseries.html

此外，Statsmodels 有大量时间序列分析工具：

https://www.statsmodels.org/stable/user-guide.html#time-series-analysis

表 1. 编辑模式，常用快捷键

快捷键组合	功能
esc	进入“命令”模式；鼠标左键单击任何 cell 返回，或单击 enter 返回编辑模式
ctrl + M	进入“命令”模式
ctrl + S	保存；尽管 JupyterLab 会自动保存，建议大家还是要养成边写边存的好习惯
shift + enter	执行 + 跳转；运行当前 cell 中的代码，光标跳转到下一 cell
ctrl + enter	执行；运行当前 cell 中的代码
alt + enter	执行 + 创建 cell；运行当前 cell 中的代码，并在下方创建一个新 cell
ctrl + shift + -	分割；在光标所在位置将代码/文本分割成两个 cells
ctrl + /	注释/撤销注释；对所在行，或选中行进行注释/撤销注释操作
ctrl + [	向右缩进；行首加四个空格
ctrl +]	向左缩进；行首减四个空格
ctrl + A	全选；全选当前 cell 内容
ctrl + Z	撤销；撤销上一个键盘操作
ctrl + shift + Z	重做：恢复刚才撤销命令对应操作，相当于“撤销撤销”
ctrl + C	复制；复制选中的代码或文本
ctrl + X	剪切；剪切选中的代码或文本
ctrl + V	粘贴；粘贴复制/剪切的代码或文本
ctrl + F	查询；实际上就是浏览器的搜索
home	跳到某一行开头
end	跳到某一行结尾
ctrl + home	跳到多行 cell 第一行开头
ctrl + end	跳到多行 cell 最后一行结尾
tab	代码补齐；忘记函数拼写时，可以给出前一两个字母，按 tab 键得到提示
shift + tab	对键入的函数提供帮助文档
ctrl + B	展开/关闭左侧 sidebar

表 2. 命令模式，常用快捷键

快捷键组合	功能
esc	编辑模式下，进入“命令”模式；鼠标左键单击任何 cell 返回，或单击 enter 返回编辑模式
esc → M	在按下 esc 进入编辑模式后，将当前 cell 从代码 markdown 转成文本
esc → Y	将当前 cell 从文本 markdown 转成代码
enter	从命令模式进入编辑模式，或者鼠标左键单击任何 cell
esc → A	插入；在当前 cell 上方插入新 cell
esc → B	插入；在当前 cell 下方插入新 cell
esc → D → D	删除；在按下 esc 进入编辑模式后，连续按两下 D，删除当前 cell
esc → 0 → 0	重启 kernel；在按下 esc 进入编辑模式后，连续按两下零 0，重启 kernel
esc → ctrl + B	展开/关闭左侧 sidebar
esc → ctrl + A	选中所有 cells
esc → shift + ▲	选中当前和上方 cell，不断按 ctrl + ▲不断选中更上一层 cell
esc → shift + ▼	选中当前和下方 cell，不断按 ctrl + ▼不断选中更下一层 cell
shift + M	合并；将所有选中的 cells 合并；如果没有多选 cell，则将当前 cell 和下方 cell 合并
shift + enter	执行 + 跳转；运行当前 cell 中的代码，光标跳转到下一 cell；和编辑模式一致
ctrl + enter	执行；运行当前 cell 中的代码；和编辑模式一致
alt + enter	执行 + 创建 cell；运行当前 cell 中的代码，并在下方创建一个新 cell；和边际模式一致
esc → 1	一级标题，等同于 markdown 状态下 #
esc → 2	二级标题，等同于 markdown 状态下 ##
esc → 3	三级标题，等同于 markdown 状态下 ###，以此类推

“鸢尾花书”的整体布局

数学

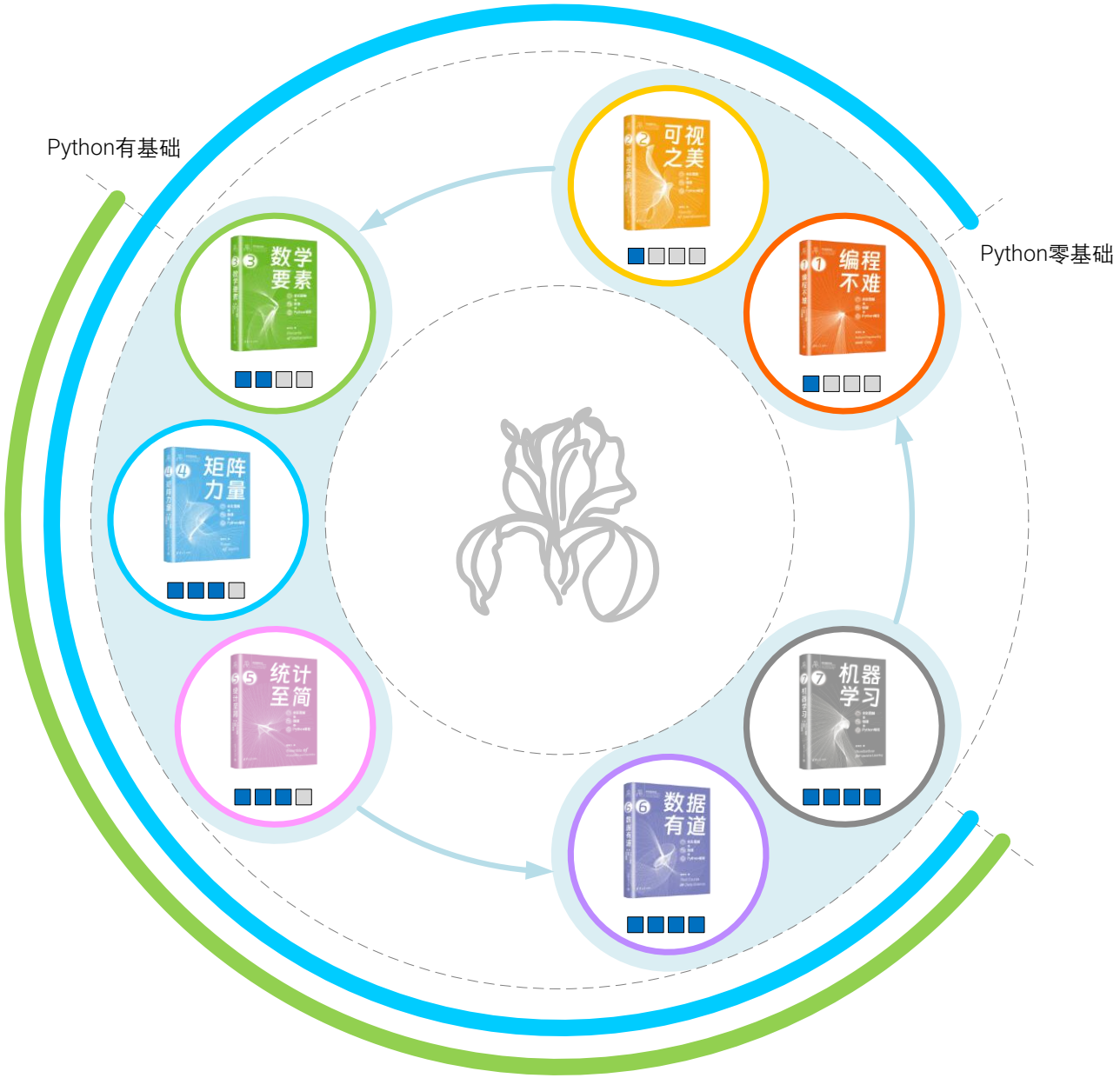
数学基础　线性代数　概率统计

Python编程　数据可视化　回归、降维　分类、聚类

工具　实践

数学 + Python编程 + 可视化 + 机器学习实践

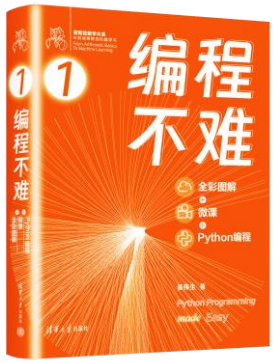
“鸢尾花书”的学习顺序



分册进度状态

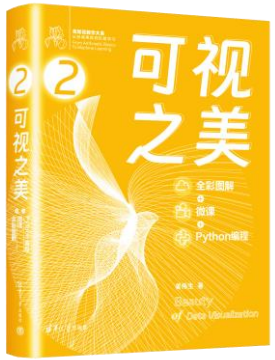
	草稿、Python	打磨、视频	清华社五审五校	上架
1 编程不难 	~50%			
2 可视之美 	~100%	2023, 10		
3 数学要素 	100%	完成	完成	完成
4 矩阵力量 	100%	完成	完成	完成
5 统计至简 	100%	2023, 07	2023, 08	2023, 09
6 数据有道 	80%	2024年初		
7 机器学习 	80%	2024年初		

Book 1《编程不难》



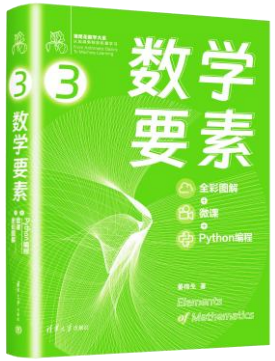
- 编程不难
  - 预备
    - 介绍
    - JupyterLab
    - Spyder
  - 语法
    - 基础
    - 数据类型
    - 运算
    - 循环
    - 函数
    - 面向对象编程
  - 绘图
    - 基础
    - 二维
    - 三维
  - 数组
    - 基础
    - 索引和切片
    - 常见运算
    - 变形
    - 数组操作
    - 线性代数
  - 符号数学
    - 代数
    - 微积分
    - 线性代数进阶
  - 数值计算
    - 代数进阶
    - 概率统计
    - 算法
  - 数据科学
    - 数据帧
    - 索引和切片
    - 常见运算
    - 规整
    - 时间序列
    - 统计分析
  - 机器学习
    - 综述
    - 回归
    - 降维
    - 分类
    - 聚类
    - 神经网络

Book 2《可视之美》



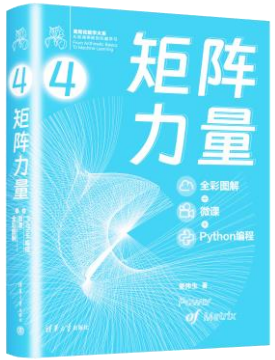
- 《可视之美》
  - 图说
    - 数学 + 艺术
    - 说图
  - 美化
    - 布局
    - 装饰
    - 风格
    - 媒介
  - 彩色
    - 色彩
    - 色彩空间
    - 色谱
  - 二维
    - 平面散点
    - 平面线图
    - 平面填充
    - 极坐标
    - 各种网格
    - 平面等高线
    - 热图
    - 非矢量图片
  - 三维
    - 三维散点
    - 三维线图
    - 网格曲面
    - 三维等高线
    - 箭头图
    - 三角网格
  - 数学之美
    - 平面几何
    - 立体几何
    - 距离
    - 平面几何变换
    - 立体几何变换
    - 贝塞尔曲线
    - 函数
    - 二次型
    - 复数
    - 分形
    - 网络图
    - 数据
    - 模式 + 随机

Book 3《数学要素》



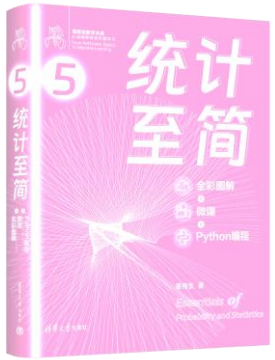
- 数学要素
  - 基础
    - 算数
    - 几何
    - 代数
  - 坐标系
    - 二维
    - 三维
  - 解析几何
    - 距离
    - 圆锥曲线
  - 函数
    - 可视化
    - 代数函数
    - 超越函数
    - 二元函数
    - 数列
  - 微积分
    - 导数
    - 偏导数
    - 微分
    - 积分
    - 优化入门
  - 概率统计
    - 概率
    - 统计
  - 线性代数
    - 向量
    - 鸡兔同笼三部曲

Book 4《矩阵力量》



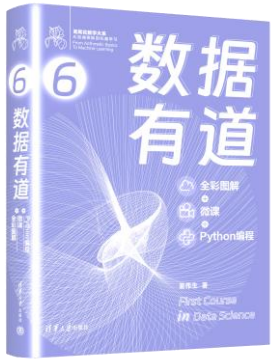
- 矩阵力量
  - 向量
    - 定义
    - 运算
    - 范数
  - 矩阵
    - 定义
    - 运算
    - 分块矩阵
  - 向量空间
    - 定义
    - 几何变换
    - 投影
    - 数据投影
  - 矩阵分解
    - 各种分解
    - Cholesky分解
    - 特征值分解
    - 奇异值分解
  - 微积分
    - 多元微分
    - 拉格朗日乘子法
  - 空间几何
    - 直线到超平面
    - 曲线
    - 曲面
  - 数据
    - 多元统计入门
    - 数据空间
    - 数据分解
    - 数据应用

Book 5《统计至简》



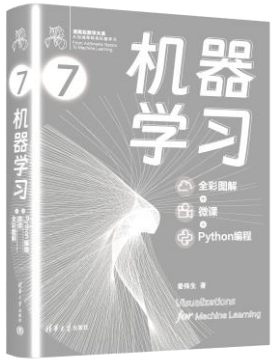
- 统计至简
  - 统计
    - 概率统计全景
    - 统计描述
  - 概率
    - 古典概率模型
    - 离散随机变量
    - 离散分布
    - 连续随机变量
    - 连续分布
    - 条件概率
  - 高斯
    - 一元
    - 二元
    - 多元
    - 条件
    - 协方差矩阵
  - 随机
    - 随机变量的函数
    - 蒙特卡洛模拟
  - 频率派
    - 频率统计推断
    - 概率密度估计
  - 贝叶斯派
    - 贝叶斯分类
    - 贝叶斯统计推断
    - 马尔科夫链
  - 椭圆
    - 马氏距离
    - 线性回归
    - 主成分分析

Book 6《数据有道》



- 数据有道
  - 数据处理
    - 缺失值
    - 离群值
    - 数据转换
    - 插值
  - 时间数据
    - 时间数据
    - 移动窗口
    - 随机过程入门
  - 回归
    - 回归分析
    - 多元线性回归
    - 正则化
    - 贝叶斯回归
    - 非线性回归
  - 降维
    - 主成分分析
    - 正交回归
    - 主元回归
    - 典型相关分析

Book 7《机器学习》


7
机器学习
全彩图解
微课
Python编程
Visualizations for Machine Learning


机器学习
有监督学习
k近邻
距离
朴素贝叶斯
高斯朴素贝叶斯
高斯判别分析
支持向量机
核技巧
决策树
高斯过程
回归
无监督学习
k均值聚类
高斯混合模型
最大期望算法
层次聚类
密度聚类
谱聚类
降维

开源资源

PDF书稿、代码：https://github.com/Visualize-ML

微课视频：https://space.bilibili.com/513194466

信息发布：https://www.zhihu.com/people/jamestong-xue

专属邮箱：jiang.visualize.ml@gmail.com