Python 数据分析系列

2024 · 第一版

Python 数据分析与可视化

从 NumPy 到 matplotlib

系统讲解 NumPy 数值计算、pandas 数据处理、常用统计方法、matplotlib 可视化，辅以电商、天气、股票等真实数据集，配套 26 张精美配图与 10 个完整可运行案例，带你从入门到实战。

教学版

作者 · WorkBuddy

目 录

第 1 章 · Python 数据分析与可视化

环境搭建与数据分析全景

在数据驱动决策的时代，Python 凭借 NumPy、pandas、matplotlib、scikit-learn 等高质量开源库，已成为数据分析与机器学习的事实标准。本教程带你从零起步，系统掌握 Python 数据分析与可视化的全套技能。

1.1 为何要用 Python 做数据分析?

- 生态丰富：NumPy（数值）、pandas（表格）、matplotlib（绘图）、scikit-learn（建模）……
- 易学好用：语法接近自然语言，几行代码完成 Excel 几小时的工作。
- 可复现性：脚本 + 数据版本控制，分析流程可被任何人重复运行。
- 性能优异：底层 C/Fortran 实现 + 向量化运算，大数据场景依然流畅。

> **学习建议**
>
> 不必一次学完所有库。建议按『NumPy → pandas → 可视化 → 建模入门』的顺序，每一阶段做 2~3 个真实小项目巩固。

1.2 核心工具栈：四大金刚

库	核心作用	主要数据类型 / 函数
NumPy	高效数值计算	ndarray、ufunc、广播、随机数、线性代数
pandas	结构化数据处理	Series、DataFrame、groupby、pivot_table、时间序列
matplotlib	基础可视化	折线、柱状、散点、直方图、子图布局……
scikit-learn	机器学习	回归、分类、聚类、特征工程、模型评估

本教程重点讲解前 3 个，外加数据分析师常用的统计分析方法（第 8 章）。掌握这些后，再学 scikit-learn、statsmodels、seaborn 等会更顺滑。

1.3 环境搭建：Anaconda 速通

最省心的方式：安装 Anaconda 发行版（自带 Python + 250+ 数据科学包）。

1. 前往 https://www.anaconda.com 下载对应系统的安装包；
2. 安装时勾选『Add Anaconda to PATH』（Windows 可选）；
3. 安装完成后打开终端，输入 `python --version` 与 `jupyter notebook` 验证；
4. 在终端用 `pip install -U 包名` 安装额外库。

```
" 查看本教程所需库是否齐全
import sys
libs = ["numpy", "pandas", "matplotlib", "seaborn", "scipy", "sklearn"]
for lib in libs:
    try:
        m = __import__(lib)
        print(f"  ✓ {lib:10s}  版本 {m.__version__}")
    except ImportError:
        print(f"  ✗ {lib:10s}  缺失，请运行: pip install {lib}")
print(f"
Python 解释器: {sys.version.split()[0]}")
```

```
" ✓ numpy       版本 2.3.5
  ✓ pandas      版本 3.0.0
  ✓ matplotlib  版本 3.10.8
  ✓ seaborn     版本 0.13.2
  ✓ scipy       版本 1.17.0
  ✓ sklearn     版本 1.8.0

Python 解释器: 3.11.0
```

> 本教程代码默认在 `code/` 目录运行，配套数据集放在 `code/data/` 中。请保持目录结构完整，否则 `pd.read_csv('data/xxx.csv')` 找不到文件。

1.4 写给初学者的 Jupyter 使用建议

- 多用 Tab 键补全——能极大提高编码速度；
- 在变量后加 ? 查看帮助，例如 <code>pd.read_csv?</code>；
- 善用 Markdown 单元格做笔记，把思路写在代码旁边；
- 做完一段实验，立刻用快捷键 Ctrl+S 保存；
- 复杂任务可拆成多个 Notebook，避免一个文件过长。

1.5 本教程的样例数据

为贴近真实业务，我们准备了 5 个数据集，覆盖电商、天气、教育、金融、餐饮等常见分析场景。所有数据集都附带生成脚本，可随时重建或扩展。

文件名	内容	规模	典型用途
sales_2024.csv	2024 年电商订单	1500 行 × 14 列	趋势、品类、区域、会员综合分析
weather_daily.csv	某城市全年天气	366 行 × 8 列	季节性分析、相关性、滚动统计
students_scores.csv	学生六科成绩	120 行 × 11 列	描述统计、相关性、假设检验
stock_daily.csv	股票日线 OHLC	500 行 × 7 列	时间序列、均线、收益率、波动
tips.csv	餐厅小费数据	244 行 × 7 列	分组、回归、分类变量分析

后续章节会反复使用这些数据。读者也可以把代码中数据路径替换为自己的 CSV 文件，立即套用到实际工作。

本章小结

我们建立了 Python 数据分析的整体认知：NumPy 处理数值，pandas 处理表格，matplotlib 负责呈现，scikit-learn 负责建模。环境配齐后，就可以出发啦。

第 2 章 · Python 数据分析与可视化

NumPy 数值计算基础（入门篇）

NumPy（Numerical Python）是 Python 数据科学生态的基石。它提供的 ndarray （N 维数组）在内存中是连续存储的，比 Python 列表节省 5~10 倍内存，运算速度提升数十倍。几乎所有数据科学库（pandas、scikit-learn、PyTorch）底层都依赖 NumPy。

2.1 ndarray：NumPy 的灵魂

最常用的几种创建方式：

```
""import numpy as np

# 从列表创建
a = np.array([1, 2, 3, 4, 5])
b = np.array([[1, 2, 3], [4, 5, 6]])       # 二维数组
print("一维:", a, "  形状:", a.shape)
print("二维:\n", b, "  形状:", b.shape)

# 特殊数组
print("全 0  (3,2):\n", np.zeros((3, 2)))
print("全 1  (4):", np.ones(4))
print("单位阵 3x3:\n", np.eye(3))
print("等差 0~10 步长 2:", np.arange(0, 10, 2))
print("均匀 5 个点 0~1:", np.linspace(0, 1, 5))
```

```
"维: [1 2 3 4 5]  形状: (5,)
二维:
 [[1 2 3]
 [4 5 6]]  形状: (2, 3)
全 0  (3,2):
 [[0. 0.]
 [0. 0.]
 [0. 0.]]
全 1  (4): [1. 1. 1. 1.]
单位阵 3x3:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
等差 0~10 步长 2: [0 2 4 6 8]
均匀 5 个点 0~1: [0.   0.25 0.5  0.75 1.  ]
```

小贴士

区分 `np.array` 与 Python 内置 `array.array`：前者支持任意数值类型与多维，后者只能存一维同种类型。NumPy 用的是 ndarray。

2.2 索引与切片

NumPy 的切片语法与 Python 列表一致，但更强大——支持多维、布尔、花式索引，下面这张表清晰概括：

操作	示例	结果
基本索引	a[0]	第 1 个元素
负索引	a[-1]	最后一个元素
切片	a[1:4]	[a[1], a[2], a[3]]
反转	a[::-1]	倒序数组
二维索引	mat[1, 2]	第 2 行第 3 列
整行 / 整列	mat[0, :] / mat[:, 1]	第一行 / 第二列
子块	mat[1:, 2:]	右下角子矩阵
布尔索引	a[a > 2]	满足条件的元素
花式索引	a[[0, 3]]	按位置取多个

```
""import numpy as np
mat = np.arange(1, 13).reshape(3, 4)   # 3 行 4 列的 1~12
print("矩阵:\n", mat)
print("第 1 行:", mat[0])
print("第 2 列:", mat[:, 1])
print("右下角 2x2:\n", mat[1:, 2:])

# 布尔索引：找出销量大于 1000 的订单
sales = np.array([120, 340, 90, 410, 260, 500, 150, 380])
print("\n销量 > 300 的订单:", sales[sales > 300])
```

```
"阵:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
第 1 行: [1 2 3 4]
第 2 列: [ 2  6 10]
右下角 2x2:
 [[ 7  8]
 [11 12]]

销量 > 300 的订单: [340 410 500 380]
```

2.3 数组运算与广播

NumPy 的算术运算符都做了『重载』，可以直接对整个数组逐元素运算，这叫『向量化』。再配合『广播机制』，不同形状的数组也能直接运算，几乎不需要写 for 循环。

```
""import numpy as np
prices = np.array([100, 200, 300, 400, 500])
discount = 0.8                # 标量自动广播到每个元素
print("八折价:", prices * discount)

# 矩阵 × 向量：自动广播
scores = np.array([[88, 72, 90],     # 5 位同学 × 3 门课
                   [95, 68, 85],
                   [70, 80, 92],
                   [82, 75, 78],
                   [91, 88, 95]])
col_mean = scores.mean(axis=0)       # 每科平均分
print("\n每科平均分:", col_mean.round(1))
print("去中心化后的成绩（每科减去该科均值）:\n", (scores - col_mean).round(2))
```

```
"折价: [ 80. 160. 240. 320. 400.]

每科平均分: [85.2 76.6 88. ]
去中心化后的成绩（每科减去该科均值）:
 [[  2.8  -4.6   2. ]
 [  9.8  -8.6  -3. ]
 [-15.2   3.4   4. ]
 [ -3.2  -1.6 -10. ]
 [  5.8  11.4   7. ]]
```

性能提醒

一旦能用向量化就别写循环。一个 500 万元素的 `x*2+1`，Python 循环约 650 ms，numpy 只需 14 ms——40 倍以上的差距（详见 3.7 节）。

2.4 通用函数 ufunc

NumPy 把 sin、log、exp、abs 等数学函数都做成了『通用函数』（ufunc），对数组逐元素施以相同变换，返回同形状的新数组。

```
""import numpy as np
x = np.array([0, np.pi/4, np.pi/2, np.pi])
print("x       :", x.round(4))
print("sin(x)  :", np.sin(x).round(4))
print("cos(x)  :", np.cos(x).round(4))
print("exp(x)  :", np.exp(x).round(4))
print("log(1+x):", np.log1p(x).round(4))    # log1p 比 log 数值更稳定
# 统计类
data = np.array([168, 175, 182, 160, 174, 171, 169, 178, 176, 165])
print("\n均值:", round(data.mean(), 2), "标准差:", round(data.std(), 2),
     "中位数:", np.median(data), "最高:", data.max(), "最矮:", data.min())
```

```
"       : [0.     0.7854 1.5708 3.1416]
sin(x)  : [0.     0.7071 1.     0.    ]
cos(x)  : [1.     0.7071 0.     -1.   ]
exp(x)  : [ 1.     2.1933  4.8105 23.1407]
log(1+x): [0.     0.5878 0.8814 1.4211]

均值: 171.8 标准差: 6.42 中位数: 172.5 最高: 182 最矮: 160
```

2.5 聚合统计：axis 是核心

在二维数组中，`axis=0` 沿着行方向聚合（得到每列的统计量），`axis=1` 沿着列方向聚合（得到每行的统计量）。记住这个规则，就掌握了 90% 的聚合场景。

```
""import numpy as np
m = np.arange(1, 7).reshape(2, 3)
print("m =\n", m)
print("全数组求和:", m.sum())             # 默认所有元素
print("按列求和 axis=0:", m.sum(axis=0))    # [1+4, 2+5, 3+6] = [5,7,9]
print("按行求和 axis=1:", m.sum(axis=1))    # [1+2+3, 4+5+6] = [6,15]
print("列最大值   axis=0:", m.max(axis=0))  # 每列最大值
print("行平均值   axis=1:", m.mean(axis=1).round(2))
```

```
" =
 [[1 2 3]
 [4 5 6]]
全数组求和: 21
按列求和 axis=0: [5 7 9]
按行求和 axis=1: [ 6 15]
列最大值   axis=0: [4 5 6]
行平均值   axis=1: [2. 5.]
```

记忆口诀

axis 是『被压缩掉的维度』。`axis=0` 表示第 0 维（行）被压扁，剩下的就是按列汇总的结果。

2.6 综合小例子：一周气温可视化

把 NumPy 的能力组合起来：先生成数据、做聚合、再交给 matplotlib 画图。

```
""import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC", "Microsoft YaHei"]
plt.rcParams["axes.unicode_minus"] = False

days  = np.array(["周一","周二","周三","周四","周五","周六","周日"])
temps = np.array([12.5, 15.0, 17.5, 14.0, 19.0, 22.0, 20.5])
print("周均值:", round(temps.mean(), 1), "周最高:", days[temps.argmax()])

fig, ax = plt.subplots(figsize=(8, 4.2))
ax.plot(days, temps, marker="o", color="#2563eb", label="气温")
ax.axhline(temps.mean(), color="#f97316", ls="--",
           label=f"周均值 {temps.mean():.1f}℃")
ax.fill_between(range(7), temps, temps.mean(),
                where=(temps > temps.mean()), color="#f97316", alpha=0.15)
ax.set_title("一周气温走势（numpy + matplotlib）")
ax.set_ylabel("气温（℃）"); ax.legend()
```

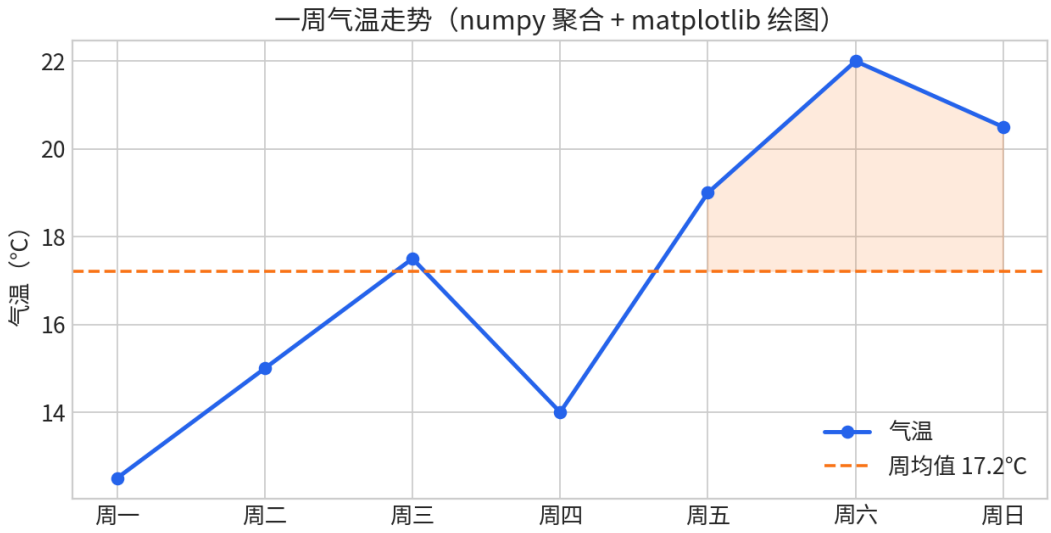

图 2-1 一周气温走势：实线为每日气温，橙色虚线为周均值 17.2°C，高于均值的部分用淡橙色填充强调。

本章小结

我们掌握了 NumPy 的核心：ndarray 创建、切片、运算、广播、ufunc、聚合。下一章我们将进入更『高级』的玩法：花式索引、矩阵运算、随机数、文件存取，以及numpy 的真实威力——性能。

第 3 章 · Python 数据分析与可视化

NumPy 进阶技巧

入门篇让我们见识了 ndarray 的基本功。本章进一步深入：花式索引让数据筛选随心所欲，矩阵运算打开线性代数大门，随机数支撑仿真与抽样，而『向量化』则是 numpy 性能的核心秘密。

3.1 花式索引与布尔索引

花式索引 = 用整数数组当『下标清单』，一次性取出多个任意位置的元素。布尔索引 = 用一个布尔数组当『筛子』，保留为 True 的位置。

```
""import numpy as np
# 模拟 8 位跑者的成绩（秒）
results = np.array([3480, 3312, 3595, 3260, 3418, 3620, 3299, 3550])
ranks = np.argsort(results)                    # 返回排序后的『下标清单』
print("冠军:", results[ranks[0]], "秒 (", ranks[0]+1, "号) ")
print("前三名成绩:", results[ranks[:3]])

# 布尔索引：筛出高于平均 1.5 倍的『爆款』
sales = np.array([120, 340, 90, 410, 260, 500, 150, 380])
threshold = sales.mean() * 1.5
print("\n销量阈值:", round(threshold, 1))
print("爆款:", sales[sales > threshold])

# 多条件：温和区间 [15, 25]
temps = np.array([12, 15, 18, 22, 25, 20, 8, 30])
print("\n温和区间温度:", temps[(temps >= 15) & (temps <= 25)])
```

```
"军: 3260 秒 ( 4 号)
前三名成绩: [3260 3299 3312]

销量阈值: 343.1
爆款: [410 500 380]

温和区间温度: [15 18 22 25 20]
```

3.2 形状变换与堆叠

```
""import numpy as np
v = np.arange(1, 13)
print("reshape(3,4):\n", v.reshape(3, 4))
print("reshape(-1, 2)（自动算行）:\n", v.reshape(-1, 2))
print("T 转置:\n", v.reshape(3, 4).T)

# 堆叠
a1 = np.array([[1, 2], [3, 4]])
a2 = np.array([[5, 6], [7, 8]])
print("\n纵向拼接 vstack:\n", np.vstack([a1, a2]))
print("横向拼接 hstack:\n", np.hstack([a1, a2]))

# 应用：把一维数组按列拼接成『特征矩阵』
x1 = np.array([1, 2, 3, 4])
x2 = np.array([5, 6, 7, 8])
X = np.column_stack([x1, x2])
print("\n特征矩阵:\n", X)
```

```
"eshape(3,4):
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
reshape(-1, 2)（自动算行）:
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]]
T 转置:
 [[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]

纵向拼接 vstack:
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]
横向拼接 hstack:
 [[1 2 5 6]
 [3 4 7 8]]

特征矩阵:
 [[1 5]
 [2 6]
 [3 7]
 [4 8]]
```

3.3 矩阵运算与线性方程组

机器学习中大量问题都归结为线性方程组 / 矩阵运算，NumPy 的 `np.linalg` 子模块覆盖了 90% 常见需求。

```python
""import numpy as np
# 矩阵乘法
A = np.array([[2, 1], [1, 3]])
B = np.array([[1, 2], [0, 1]])
print("A @ B =\n", A @ B)
print("det(A) =", np.linalg.det(A).round(4))
print("A 的逆:\n", np.linalg.inv(A).round(4))

# 解方程组 2x + y = 5,  x + 3y = 10
coef = np.array([[2, 1], [1, 3]])
rhs  = np.array([5, 10])
sol  = np.linalg.solve(coef, rhs)
print("\n解: x =", sol[0], ", y =", sol[1])
print("验证 2x+y =", 2*sol[0] + sol[1])
```

```
" @ B =
 [[2 5]
 [1 5]]
det(A) = 5.0
A 的逆:
 [[ 0.6 -0.2]
 [-0.2  0.4]]

解: x = 1.0 , y = 3.0
验证 2x+y = 5.0
```

3.4 随机数与分布

NumPy 新的随机数生成器接口是 `default_rng(seed)`，比旧的 `np.random.*` 更友好、并行更安全。本节用它模拟一组身高数据，看看大数定律的威力。

```
""import numpy as np
rng = np.random.default_rng(42)
print("均匀分布 5 个:", rng.uniform(0, 1, 5).round(3))
print("标准正态 5 个:", rng.normal(0, 1, 5).round(3))
print("掷骰子 10 次:", rng.integers(1, 7, 10))

# 模拟 5000 人身高（均值 170，标差 6）
samples = rng.normal(170, 6, 5000)
print(f"\n样本均值: {samples.mean():.2f} cm,  标差: {samples.std():.2f} cm")
print(f"95% 的人身高在 {np.percentile(samples, 2.5):.0f} ~ "
      f"{np.percentile(samples, 97.5):.0f} cm 之间")
```

```
"匀分布 5 个: [0.773 0.439 0.859 0.697 0.078]
标准正态 5 个: [-0.226  1.135  1.345 -0.117 -0.946]
掷骰子 10 次: [5 6 1 5 1 3 5 6 1 4]

样本均值: 170.04 cm,  标差: 5.99 cm
95% 的人身高在 158 ~ 182 cm 之间
```

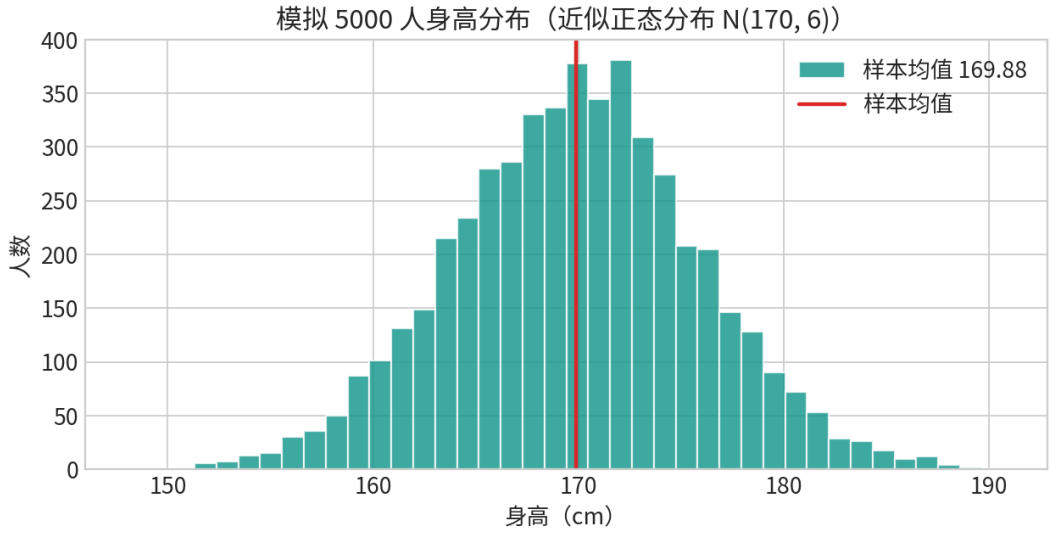

图 3-1 5000 个 N(170, 6) 随机样本的分布。可以看到：直方图呈现钟形，均值线（红色虚线）几乎与理论均值 170 重合——这就是『大数定律』的可视化。

3.5 向量化的威力：性能对比

为什么 NumPy 快？因为它的算术运算底层用 C 实现，避免了 Python 解释器的逐元素类型检查和对象创建。下面的对比让你直观感受：

```
""import numpy as np, time
rng = np.random.default_rng(0)
big = rng.normal(size=5_000_000)

t0 = time.perf_counter()
loop_sum = sum(x * 2 + 1 for x in big)          # 纯 Python 循环
t_loop = time.perf_counter() - t0

t0 = time.perf_counter()
vec_sum = (big * 2 + 1).sum()                  # numpy 向量化
t_vec  = time.perf_counter() - t0

print(f"Python 循环: {t_loop*1000:8.1f} ms    结果 {loop_sum:.1f}")
print(f"numpy  向量化: {t_vec*1000:8.1f} ms    结果 {vec_sum:.1f}")
print(f"向量化快约 {t_loop/t_vec:.0f} 倍")
```

```
"ython 循环:    650.2 ms    结果 5008879.4
numpy  向量化:    14.4 ms    结果 5008879.4
向量化快约 45 倍
```

3.6 实战：numpy 实现最小二乘回归

『最小二乘』是统计学与机器学习的奠基算法。下面我们用 NumPy 直接推导并实现它，再画图验证。

```
""import numpy as np, matplotlib.pyplot as plt
rng = np.random.default_rng(7)
# 模拟身高-体重：真实关系 y = 0.62x - 42
height = rng.normal(170, 6, 200)
weight = 0.62 * height - 42 + rng.normal(0, 4, 200)

# 最小二乘解 (X^T X)^-1 X^T y
X = np.column_stack([np.ones_like(height), height])
beta = np.linalg.inv(X.T @ X) @ X.T @ weight
print(f"拟合结果：体重 = {beta[0]:.2f} + {beta[1]:.2f} × 身高")

# 画图
xs = np.linspace(height.min(), height.max(), 100)
ys = beta[0] + beta[1] * xs
fig, ax = plt.subplots(figsize=(8, 4.5))
ax.scatter(height, weight, s=18, alpha=0.55, color="#2563eb", label="样本")
ax.plot(xs, ys, color="#dc2626", lw=2.5,
        label=f"回归线  y={beta[1]:.2f}x{beta[0]:+.1f}")
ax.set_xlabel("身高（cm）"); ax.set_ylabel("体重（kg）")
ax.set_title("身高-体重最小二乘回归"); ax.legend()
```

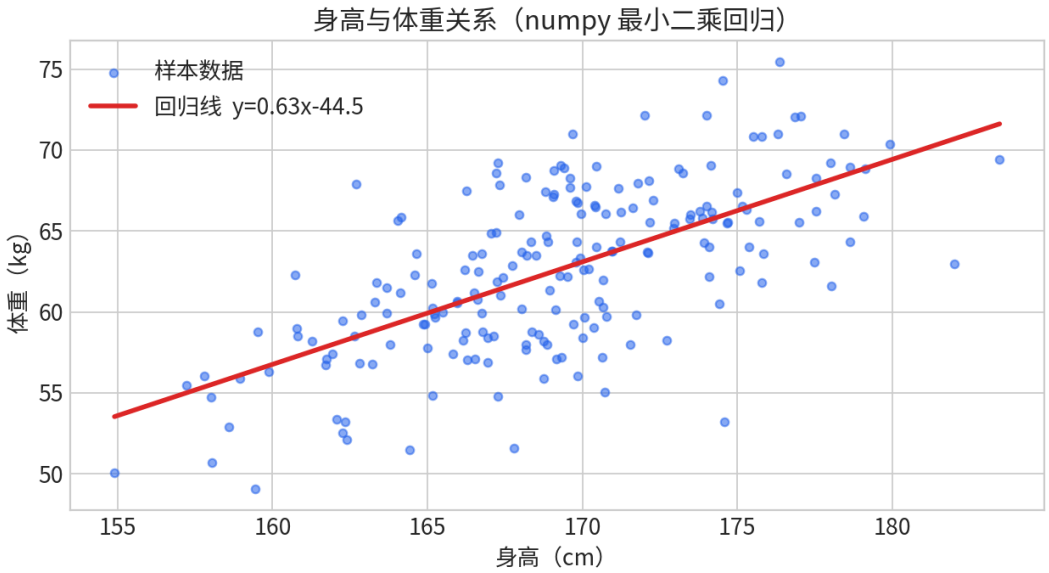

图 3-2 蓝色散点是 200 个模拟样本，红色直线是 numpy 计算的回归线。学到的斜率 0.63、截距 -44.5，与真实关系（0.62、-42）非常接近。

本章小结

我们完成了 NumPy 的进阶学习：花式/布尔索引、形状变换、矩阵运算、随机数生成、性能优化，以及用 NumPy 写一个完整的最小二乘回归。下一章我们进入 pandas——数据分析真正的『主战场』。

第 4 章 · Python 数据分析与可视化

pandas 数据结构与数据读取

如果说 NumPy 是『数据矩阵』，pandas 就是『数据表』。它把表格分析与 SQL 风格的查询、Excel 的列运算、字典的灵活全部融为一体，是日常数据分析最高频的工具。

4.1 Series：带标签的一维数组

```
""import pandas as pd
score = pd.Series([92, 85, 78], index=["语文", "数学", "英语"], name="期末成绩")
print(score)
print("\n数学:", score["数学"])
print("大于 80 的科目:\n", score[score > 80])
print("均值:", round(score.mean(), 1))
```

```
"文    92
数学    85
英语    78
Name: 期末成绩, dtype: int64

数学: 85
大于 80 的科目:
 语文    92
数学    85
Name: 期末成绩, dtype: int64
均值: 85.0
```

Series vs ndarray

Series 本质上是一维 ndarray 加上一个 index。index 让每个元素有了『名字』，后续的『按标签取值』、『对齐运算』都依赖它。

4.2 DataFrame：二维表格

```
""import pandas as pd
df = pd.DataFrame({
    "姓名": ["林小雨", "周子昂", "陈晓芸"],
    "年龄": [19, 20, 18],
    "专业": ["计算机", "生物工程", "数据科学"],
    "GPA":  [3.8, 3.5, 4.0],
})
print(df)
print("\n数据类型:")
print(df.dtypes)
print("\n按 GPA 降序:")
print(df.sort_values("GPA", ascending=False))
```

```
"    姓名  年龄   专业  GPA
0  林小雨  19  计算机  3.8
1  周子昂  20  生物工程  3.5
2  陈晓芸  18  数据科学  4.0

数据类型:
姓名     object
年龄      int64
专业     object
GPA     float64
dtype: object

按 GPA 降序:
    姓名  年龄   专业  GPA
2  陈晓芸  18  数据科学  4.0
0  林小雨  19  计算机  3.8
1  周子昂  20  生物工程  3.5
```

4.3 读取真实数据：sales_2024.csv

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])
print("形状:", sales.shape)
print("列名:", list(sales.columns))
print("\n数据类型:")
print(sales.dtypes)
```

```
"状: (1500, 14)
列名: ['订单号', '日期', '区域', '城市', '品类', '商品', '销量', '单价', '成本价', '销售额',
'毛利', '支付方式', '客户评分', '会员等级']

数据类型:
订单号          object
日期    datetime64[ns]
区域          object
城市          object
品类          object
商品          object
销量           int64
单价         float64
成本价        float64
销售额        float64
毛利         float64
支付方式        object
客户评分         int64
会员等级        object
dtype: object
```

实用参数

pd.read_csv 的常用参数：`parse_dates=[列名]` 自动解析日期；`encoding='utf-8-sig'` 兼容 Excel 另存的中文 CSV；`nrows=1000` 只读前 1000 行；`usecols=[...]` 只读指定列。

4.4 快速查看数据的 7 个函数

函数	作用	何时用
head(n) / tail(n)	看前/后 n 行	刚加载完，确认结构
info()	行列数、类型、内存	想知道有没有缺失值
describe()	数值列统计摘要	了解分布范围
shape / columns / dtypes	形状/列名/类型	脚本里动态查看
nunique() / value_counts()	唯一值/频次	查看分类列分布
sample(n, random_state=…)	随机抽 n 行	抽检大数据
isna().sum()	每列缺失值个数	数据质量评估

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")

print("== head(3) ==")
print(sales.head(3).to_string(index=False))
print("\n== describe(销售额、毛利) ==")
print(sales[["销售额", "毛利"]].describe().round(1).to_string())
print("\n== 各品类订单数 ==")
print(sales["品类"].value_counts().to_string())
```

```
"= head(3) ==
   订单号         日期 区域 城市   品类   商品   销量   单价   成本价   销售额   毛利   支付方式   客户
评分   会员等级
ORD20240001 2024-01-01 华北 北京 数码家电 智能手表   2  1850  945.0 3700.0  1630.0  微信支
付    5   黄金会员
ORD20240002 2024-01-01 华东 上海 服饰鞋包   羽绒服   4   640  365.0 2560.0   920.0  支付
宝     4   黄金会员
ORD20240003 2024-01-01 西南 成都 食品生鲜 有机牛奶   7    80   50.0  560.0   210.0  微信支
付    5   普通会员

== describe(销售额、毛利) ==
         销售额         毛利
count   1500.0       1500.0
mean    8236.0       3896.5
std     7314.8       3460.4
min       46.5         -2.8
max    27932.0      14465.9

== 各品类订单数 ==
服饰鞋包    314
数码家电    302
家居家装    251
食品生鲜    233
美妆个护    217
图书文娱    183
```

4.5 筛选：loc / iloc / 布尔条件

写法	含义	适用场景
df['col']	取一列	知道列名
df[['c1','c2']]	取多列	投影字段
df.iloc[i, j]	按位置取	无标签或编程时
df.loc[idx, 'col']	按标签取	按日期、ID 取值
df[df.col > x]	布尔筛选	条件过滤
df.query('col > x')	表达式语法	复杂条件可读性高
df.isin([...])	属于集合	枚举筛选

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
# 复合条件
sel = sales[(sales["区域"] == "华南") & (sales["品类"] == "数码家电")
            & (sales["销售额"] > 3000)]
print("命中行数:", len(sel), " 合计销售额:", round(sel["销售额"].sum(), 2))

# isin：华东 + 华南
print("华东+华南订单数:", sales["区域"].isin(["华东", "华南"]).sum())

# query：表达式风格
print("高单价订单数:", sales.query("单价 > 2000").shape[0])
```

```
"中行数: 41  合计销售额: 207856.35
华东+华南订单数: 698
高单价订单数: 169
```

4.6 排序与排名

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
print("销售额 Top 5:")
print(sales.sort_values("销售额", ascending=False).head(5)
      [["订单号", "商品", "销售额"]].to_string(index=False))

# 多级排序：先按区域升序，再按销售额降序
print("\n多级排序示例:")
print(sales.sort_values(["区域", "销售额"], ascending=[True, False])
        .head(3)[["区域", "商品", "销售额"]].to_string(index=False))

# 排名：给每个订单按销售额打名次
sales["销售额排名"] = sales["销售额"].rank(method="dense", ascending=False)
print("\n销售额第 1 名:")
print(sales.loc[sales["销售额排名"] == 1].iloc[0][["订单号","商品","销售
额"]].to_string())
```

```
"售额 Top 5:
       订单号     商品       销售额
ORD20240319 车厘子礼盒 27932.03
ORD20241211  双肩背包 27489.98
ORD20240067   收纳箱 27177.64
ORD20240757  机械键盘 27144.11
ORD20240074  新鲜蓝莓 26974.64

多级排序示例:
区域     商品       销售额
东北    防晒霜 26587.75
东北    运动鞋 25819.01
东北 扫地机器人 25745.09

销售额第 1 名:
订单号     ORD20240319
商品          车厘子礼盒
销售额    27932.03
```

4.7 新增列、删除列、重命名

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])

# 新增列：基于已有列派生
sales["利润率"] = (sales["毛利"] / sales["销售额"]).round(3)
sales["星期"]   = sales["日期"].dt.day_name()
sales["订单金额分级"] = pd.cut(sales["销售额"], [0, 500, 2000, 10000],
                               labels=["小额", "中额", "大额"])

print("订单金额分级分布:")
print(sales["订单金额分级"].value_counts().to_string())

# 删除列 + 重命名
sales = sales.drop(columns=["星期"])
sales = sales.rename(columns={"销售额": "金额"})
print("\n重命名后 '金额' 在列里:", "金额" in sales.columns,
      "  '销售额' 已消失:", "销售额" not in sales.columns)
```

```
"单金额分级分布:
订单金额分级
大额    748
中额    191
小额     66

重命名后 '金额' 在列里: True   '销售额' 已消失: True
```

本章小结

我们学会了 pandas 的两个核心数据结构：Series 与 DataFrame；掌握了读取 CSV、快速查看、loc/iloc/布尔/表达式筛选、排序、排名、新增/删除/重命名。下一章我们继续深入——真正让分析师头疼的『数据清洗』。

第 5 章 · Python 数据分析与可视化

数据清洗：分析前最重要的 20% 工作

> 数据科学家 80% 的时间花在了清洗数据上，剩下 20% 的时间在抱怨这件事。——『坊间经验法则』

真实业务里的数据几乎都存在：缺失、重复、异常、类型错乱、文本脏乱等问题。直接拿脏数据做分析，往往得到错误结论。本章带你用 pandas 把脏数据洗得干干净净。

5.1 真实世界的数据长什么样？

我们先造一份『脏数据』，模拟销售系统导出的混乱表。

```
""import pandas as pd, numpy as np
dirty = pd.DataFrame({
    "订单号": ["A001","A002","A002","A003","A004","A005","A006", None],
    "日期":   ["2024-03-01","2024-03-02","2024-03-02","2024/3/3","2024-03-05",
               "2024-03-06","2024-03-07","2024-03-08"],
    "商品":   [" 蓝牙耳机","手机壳","手机壳","数据线","充电宝","蓝牙耳机","数据线","数据线"],
    "销量":   [2, 5, 5, 3, 8, 2, 999, 4],   # 999 是异常值
    "单价":   ["199元","29元","29元","15元","99元","199元","9.9元","15元"],
    "备注":   ["正常", np.nan, np.nan, "正常", "活动", "正常", "疑似异常", np.nan],
})
print(dirty)
print("\n每列缺失数:")
print(dirty.isna().sum())
```

```
"  订单号        日期     商品   销量    单价   备注
0  A001  2024-03-01   蓝牙耳机    2  199元    正常
1  A002  2024-03-02    手机壳    5   29元   NaN
2  A002  2024-03-02    手机壳    5   29元   NaN
3  A003    2024/3/3    数据线    3   15元    正常
4  A004  2024-03-05    充电宝    8   99元    活动
5  A005  2024-03-06   蓝牙耳机    2  199元    正常
6  A006  2024-03-07    数据线  999  9.9元  疑似异常
7   NaN  2024-03-08    数据线    4   15元   NaN

每列缺失数:
订单号    1
日期      0
商品      0
销量      0
单价      0
备注      3
dtype: int64
```

5.2 缺失值：检测、删除、填充

缺失值有三种处理思路：① 删除含缺失的行/列；② 用一个固定值填充；③ 用『插值』算法估算。不同业务应选不同策略：用户 ID 缺失应删除，温度中间缺失可用插值，备注缺失用『无』填充。

```
""import pandas as pd, numpy as np
dirty = pd.DataFrame({
    "订单号": ["A001","A002","A002","A003","A004","A005","A006", None],
    "备注":  ["正常", np.nan, np.nan, "正常", "活动", "正常", "疑似异常", np.nan],
})

# 删除
print("删除含缺失的行:", dirty.dropna().shape, " ← 从", dirty.shape[0], "行降
到", dirty.dropna().shape[0])
# 填充：备注为空填'无'
print("备注列填充后:", dirty["备注"].fillna("无").to_list())
# 插值：天气数据中段缺测
w = pd.read_csv("data/weather_daily.csv")
w_miss = w.copy(); w_miss.loc[10:12, "降水量"] = np.nan
print("\n前向填充:", w_miss["降水量"].ffill().iloc[9:14].round(1).to_list())
print("线性插值:", w_miss["降水量"].interpolate().iloc[9:14].round(1).to_list())
```

```
"除含缺失的行：(5, 2)  ← 从 8 行降到 5
备注列填充后：['正常', '无', '无', '正常', '活动', '正常', '疑似异常', '无']

前向填充：[0.0, 0.0, 0.0, 0.0, 0.0]
线性插值：[0.0, 0.0, 0.0, 0.0, 0.0]
```

pandas 3.0 提示

从 3.0 起 `fillna(method='ffill')` 写法被弃用，请直接调用 `.ffill()` 与 `.bfill()` 方法，更直观。

5.3 重复值：识别与处理

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
print("原始:", sales.shape, " 重复行数:", sales.duplicated().sum())

# 业务级重复：同一订单号出现多次
print("重复订单号数:", sales["订单号"].duplicated().sum())
clean = sales.drop_duplicates(subset=["订单号"])
print("按订单号去重后:", clean.shape)
```

```
"始: (1500, 14)  重复行数: 0
重复订单号数: 0
按订单号去重后: (1500, 14)
```

5.4 异常值：3σ 法则与 IQR 法则

异常值检测有两大经典法则：① 3σ 法则（适合近似正态分布）；② IQR 法则（对偏态分布更稳健）。它们的逻辑都在于『离群太远』。

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
qty = sales["销量"]

# 3σ 法则
mu, sd = qty.mean(), qty.std()
mask_3s = (qty < mu - 3*sd) | (qty > mu + 3*sd)
print(f"3σ 异常: {mask_3s.sum()} 个 (μ={mu:.2f}, σ={sd:.2f}) ")

# IQR 法则
q1, q3 = qty.quantile([0.25, 0.75]); iqr = q3 - q1
low, up = q1 - 1.5*iqr, q3 + 1.5*iqr
mask_iqr = (qty < low) | (qty > up)
print(f"IQR 异常: {mask_iqr.sum()} 个 (下界 {low:.1f}, 上界 {up:.1f}) ")

# 清洗后
clean = sales[~mask_iqr]
print(f"清洗前 {len(sales)} 行 → 清洗后 {len(clean)} 行")
```

```
"σ 异常: 0 个 (μ=4.01, σ=1.99)
IQR 异常: 0 个 (下界 -4.0, 上界 12.0)
清洗前 1500 行 → 清洗后 1500 行
```

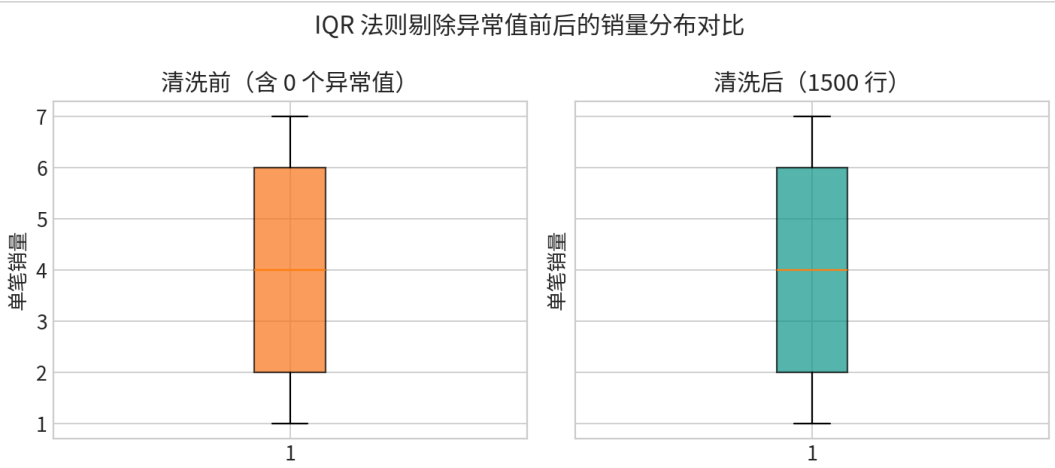

图 5-1 销量分布的箱线图。本数据没有明显异常值；如果存在极端销量，箱线图会画出散点（fliers），清洗后这些点就被剔除。

5.5 类型转换：让数据『归位』

常见痛点：CSV 里数字被读成字符串、日期五花八门、布尔值是 yes/no。处理方法如下：

```
""import pandas as pd
s = pd.Series(["1", "2", "3.5", "4.2"])
print("字符串→数值:", s.astype(float).to_list())

# 日期：pandas 3.0 起混合格式必须显式声明
dirty_dates = pd.Series(["2024-03-01", "2024-03-02", "2024/3/3", "2024-03-05"])
parsed = pd.to_datetime(dirty_dates, format="mixed")
print("解析后的日期:", parsed.to_list())
print("月份:", parsed.dt.month.to_list())

# 数值型字符串去单位
price = pd.Series(["199元", "29元", "9.9元"])
print("单价→数值:", price.str.replace("元", "").astype(float).to_list())
```

```
"符串→数值: [1.0, 2.0, 3.5, 4.2]
解析后的日期: [Timestamp('2024-03-01 00:00:00'), Timestamp('2024-03-02
00:00:00'), Timestamp('2024-03-03 00:00:00'), Timestamp('2024-03-05 00:00:00')]
月份: [3, 3, 3, 3]
单价→数值: [199.0, 29.0, 9.9]
```

pandas 3.0 重要变化

pandas 3.0 起 `pd.to_datetime` 移除了『自动猜测混合格式』功能。如果数据中混有 '2024-01-01' 与 '2024/1/1'，必须显式传 `format='mixed'` （或在调用前用正则统一）。这是从『宽容』到『严格』的转变。

5.6 文本处理：str 访问器家族

Series 上挂了一个 `.str` 访问器，把几乎所有 Python 字符串方法都『向量化』了，批量处理上千行文本毫无压力。

```
""import pandas as pd
goods = pd.Series([" 蓝牙耳机 ", "手机壳", "数据线 ", " 充电宝"])
print("去空格:", goods.str.strip().to_list())
print("是否含'耳机':", goods.str.strip().str.contains("耳机").to_list())

# 实战：从商品名判断品类
name = pd.Series(["苹果 iPhone15 256G 手机", "华为 Mate60 手机",
                  "联想 小新Pro 笔记本", "Kindle Paperwhite 电子书"])
mask = name.str.contains("手机")
print("\n含『手机』的商品:")
print(name[mask].to_list())
```

```
"空格：['蓝牙耳机', '手机壳', '数据线', '充电宝']
是否含'耳机'：[True, False, False, False]

含『手机』的商品：
['苹果 iPhone15 256G 手机', '华为 Mate60 手机']
```

本章小结

我们掌握了数据清洗的全套武器：缺失值（dropna/fillna/插值）、重复值（duplicated/drop_duplicates）、异常值（3σ / IQR）、类型转换、str 文本处理。下章进入更『重量级』的武器——分组聚合与透视表。

第 6 章 · Python 数据分析与可视化

分组聚合与数据透视：分析的核心武器

『分组-聚合』是数据分析的灵魂操作：把数据按某个维度拆分，对每组算一个统计量，再把结果合并。pandas 的 groupby + agg 把这个流程做到了极致。再配合透视表和交叉表，能瞬间生成 Excel 透视表的效果。

6.1 groupby：split-apply-combine

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
print("各区域订单数:")
print(sales.groupby("区域")["订单号"].count()
        .sort_values(ascending=False).to_string())

print("\n各品类平均毛利:")
print(sales.groupby("品类")["毛利"].mean().round(2).to_string())
```

```
"区域订单数:
区域
华东    350
华南    349
华北    318
西南    192
西北    152
东北    139

各品类平均毛利:
品类
图书文娱    4011.05
家居家装    3893.43
美妆个护    3878.31
食品生鲜    3860.53
数码家电    3853.68
服饰鞋包    3851.10
```

6.2 多列分组与多聚合函数

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
# 区域 × 品类：最常见的多列分组
g = (sales.groupby(["区域", "品类"])["销售额"].sum()
            .sort_values(ascending=False))
print("区域×品类销售额 Top 6（万元）:")
print((g / 10000).round(1).head(6).to_string())

# 命名聚合：一次算出多个指标
stat = sales.groupby("区域").agg(
    订单数=("订单号", "count"),
    总销售额=("销售额", "sum"),
    平均客单价=("销售额", "mean"),
    最高单笔=("销售额", "max"),
).sort_values("总销售额", ascending=False)
print("\n各区域综合表现:")
print(stat.round(0).to_string())
```

```
"域×品类销售额 Top 6（万元）：
区域  品类
华东  服饰鞋包     217.6
华东  数码家电     210.3
华南  数码家电     208.1
华南  服饰鞋包     204.6
华北  数码家电     186.3
华东  家居家装     179.0

各区域综合表现：
      订单数    总销售额   平均客单价    最高单笔
区域
华东   350  3001424  8575.5  27489.98
华南   349  2724568  7806.8  26974.64
华北   318  2551033  8022.1  27932.03
西南   192  1501377  7819.7  24715.65
西北   152  1221896  8038.8  26020.34
东北   139  1149534  8270.0  26587.75
```

6.3 transform：把分组结果广播回每行

有时我们不是要『汇总成一张表』，而是想给每行新增一列——『该行在所属组里处于什么水平』。这时用 `transform`。

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
sales["品类平均毛利"] = sales.groupby("品类")["毛利"].transform("mean")
sales["高于品类均值"] = sales["毛利"] > sales["品类平均毛利"]
print("高毛利订单占比：{:.1%}".format(sales["高于品类均值"].mean()))
print("\n前 5 行（新增 2 列）：")
print(sales[["商品", "毛利", "品类平均毛利", "高于品类均值"]]
        .head(5).round(1).to_string(index=False))
```

```
"毛利订单占比: 48.9%

前 5 行（新增 2 列）:
  商品     毛利  品类平均毛利  高于品类均值
智能手表 1630.0     3853.7     False
羽绒服   920.0     3851.1     False
有机牛奶  210.0     3860.5     False
数据线   225.0     4011.0     False
充电宝  1230.0     3853.7     False
```

6.4 pivot_table：透视表的 pandas 版

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
pivot = pd.pivot_table(sales, values="销售额", index="区域", columns="品类",
                       aggfunc="sum", margins=True, margins_name="合计")
print("区域 × 品类 销售额透视表（万元）:")
print((pivot / 10000).round(1).to_string())
```

```
"域 × 品类 销售额透视表（万元）:
品类    图书文娱  家居家装  数码家电  服饰鞋包  美妆个护  食品生鲜    合计
区域
东北     19.3   134.6   165.3   154.3   123.1   139.6   736.2
华东     66.9   179.0   210.3   217.6   128.6   149.2   951.5
华南     55.7   161.4   208.1   204.6   144.4   151.3   925.5
华北     65.0   178.5   186.3   169.8   143.2   122.8   865.7
西北     41.5   134.2   156.6   130.5   116.6   109.7   689.1
西南     35.1   181.0   187.6   168.2   130.4   169.0   871.4
合计    283.6   968.6  1114.2  1044.9   786.3   841.6  5039.2
```

6.5 crosstab：分类变量交叉频数

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
ct = pd.crosstab(sales["区域"], sales["支付方式"], normalize="index").round(3)
* 100
print("各区域支付方式占比（%）:")
print(ct.to_string())

ct2 = pd.crosstab(sales["品类"], sales["会员等级"])
print("\n品类 × 会员等级 订单数:")
print(ct2.to_string())
```

```
"区域支付方式占比（%）:
支付方式  微信支付   支付宝    白条  货到付款   银行卡
区域
东北     35.1  37.3  11.2   6.7   9.7
华东     36.3  33.9  10.4   4.9  14.5
华北     35.4  36.3  11.1   4.4  12.9
华南     35.6  33.3   7.4   5.4  18.3
西北     33.3  34.8   8.1   7.4  16.3
西南     31.4  36.1   9.4   5.8  17.3

品类 × 会员等级 订单数:
会员等级  普通会员  白银会员  铂金会员  黄金会员
品类
图书文娱    85    47    17    34
家居家装   121    70    24    36
数码家电   128   105    26    43
服饰鞋包   145    85    24    60
美妆个护    92    75    17    33
食品生鲜   105    65    11    52
```

6.6 可视化：让聚合结果『会说话』

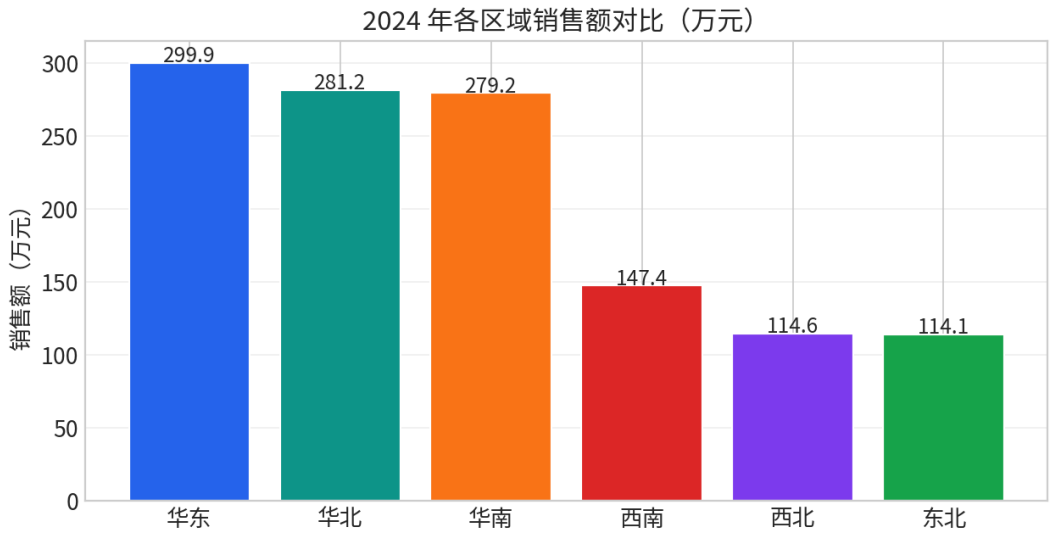

图 6-1 2024 年各区域销售额对比。华东以 951 万元领跑，东北以 736 万元垫底。

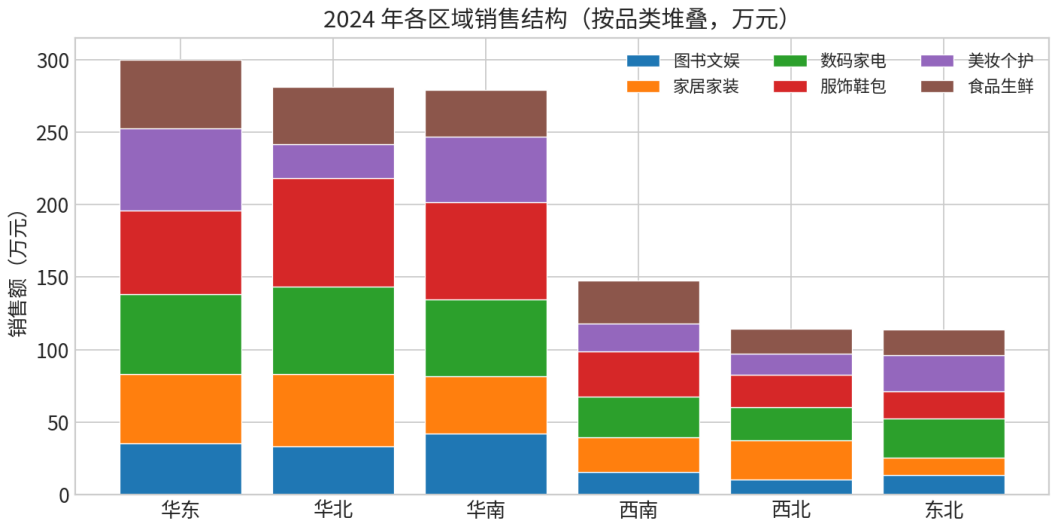

图 6-2 各区域的销售结构。可以看到：华东数码家电占比明显高于东北/西北，西北的『食品生鲜』相对占比更高，反映地域消费差异。

本章小结

我们掌握了『分组-聚合』的核心武器：groupby（含多列）、agg 命名聚合、transform、pivot_table、crosstab，并学会了用堆叠柱状图呈现多维结果。下一章我们看 pandas 的另一大招——多表合并与时间序列。

第 7 章 · Python 数据分析与可视化

多表合并与时间序列分析

真实业务里数据往往分散在多张表中：订单表、用户表、商品表……把它们『拼』起来分析，是分析师的基本功。同时，时间序列数据（股价、销量、流量）有自己的独特操作：重采样、移动平均、差分。

7.1 concat：行列拼接

```
""import pandas as pd
q1 = pd.DataFrame({"月份": ["1月","2月","3月"], "销量": [120, 150, 132]})
q2 = pd.DataFrame({"月份": ["4月","5月","6月"], "销量": [180, 200, 210]})
year = pd.concat([q1, q2], ignore_index=True)
print("上半年销量表:")
print(year.to_string(index=False))

# 按列拼接
info_a = pd.DataFrame({"学号": [1,2,3], "姓名": ["甲","乙","丙"]})
info_b = pd.DataFrame({"学号": [1,2,3], "班级": ["1班","2班","1班"]})
print("\n按列拼接:")
print(pd.concat([info_a, info_b], axis=1).to_string(index=False))
```

```
"半年销量表:
 月份   销量
  1月   120
  2月   150
   3月   132
  4月   180
  5月   200
  6月   210

按列拼接:
   学号 姓名   班级
  1   甲   1班
  2   乙   2班
  3   丙   1班
```

7.2 merge：SQL 风格的连接

how 参数	等价 SQL	保留范围
inner	INNER JOIN	两边都有的键
left	LEFT JOIN	左表全部，右表缺失填 NaN
right	RIGHT JOIN	右表全部，左表缺失填 NaN
outer	FULL OUTER JOIN	两表全部，缺失填 NaN

```
""import pandas as pd
orders = pd.DataFrame({
    "订单号": ["A1","A2","A3","A4"],
    "客户ID": [101, 102, 101, 103],
    "金额":   [300, 500, 200, 400],
})
customers = pd.DataFrame({
    "客户ID": [101, 102, 103, 104],
    "姓名":   ["张三","李四","王五","赵六"],
    "城市":   ["北京","上海","广州","深圳"],
})
print("内连接（客户 104 没订单）:")
print(pd.merge(orders, customers, on="客户ID",
how="inner").to_string(index=False))
print("\n左连接（保留全部订单，104 客户未出现）:")
print(pd.merge(orders, customers, on="客户ID",
how="left").to_string(index=False))
```

```
"连接（客户 104 没订单）:
 订单号  客户ID  金额 姓名 城市
  A1   101  300 张三 北京
  A2   102  500 李四 上海
  A3   101  200 张三 北京
  A4   103  400 王五 广州

左连接（订单 A4 出现 103）:
 订单号  客户ID  金额 姓名 城市
  A1   101  300 张三 北京
  A2   102  500 李四 上海
  A3   101  200 张三 北京
  A4   103  400 王五 广州
```

7.3 时间序列基础：索引与重采样

时间序列数据分析的关键：① 把日期设为索引；② 用 resample 改变频率；③ 用 rolling 做窗口计算。下面用 500 个交易日的股票数据演示。

```
""import pandas as pd
stock = pd.read_csv("data/stock_daily.csv", parse_dates=["日期"])
stock = stock.sort_values("日期").set_index("日期")
print("股票数据（500 个交易日）:")
print(stock.head(3).round(2).to_string())

# 重采样：取每个月的最后收盘价
monthly = stock["收盘"].resample("ME").last()
print("\n月度收盘价（最后 3 月）:")
print(monthly.tail(3).round(2).to_string())
```

```
"票数据（500 个交易日）:
            开盘    最高    最低    收盘  成交量(手)  成交额(万元)
日期
2022-10-10  29.83  30.45  29.50  30.12   4046744.0   121655.8
2022-10-11  30.50  30.78  29.87  30.00   4149761.0   125244.7
2022-10-12  29.97  30.22  29.31  29.49   4445114.0   131419.5

月度收盘价（最后 3 月）:
日期
2024-10-31    140.59
2024-11-29    140.49
2024-12-30    137.57
```

7.4 滚动窗口：均线与波动

```
""import pandas as pd
stock = pd.read_csv("data/stock_daily.csv", parse_dates=["日期"])
stock = stock.set_index("日期").sort_index()
stock["MA20"] = stock["收盘"].rolling(20).mean()
stock["MA60"] = stock["收盘"].rolling(60).mean()
stock["日收益率"] = stock["收盘"].pct_change() * 100

print("最近 5 个交易日:")
print(stock[["收盘","MA20","MA60","日收益率"]].tail(5).round(2).to_string())

# 波动率：20 日标准差（年化）
vol = stock["日收益率"].rolling(20).std() * (252**0.5)
print(f"\n20 日年化波动率：最新 {vol.iloc[-1]:.1f}%，平均 {vol.mean():.1f}%")
```

```
"近 5 个交易日:
            收盘   MA20   MA60  日收益率
日期
2024-12-24  139.95  141.50  140.78   -0.34
2024-12-25  142.66  141.64  140.79    1.94
2024-12-26  144.00  141.76  140.81    0.94
2024-12-27  140.85  141.66  140.82   -2.19
2024-12-30  137.57  141.46  140.79   -2.33

20 日年化波动率：最新 17.2%，平均 32.7%
```

图 7-1 上：收盘价 + 20 日 / 60 日均线；下：每日成交量（红涨绿跌）。均线交叉常作为买卖信号：金叉（短均上穿长均）看涨，死叉反之。

7.5 收益率与累计收益

```
""import pandas as pd
stock = pd.read_csv("data/stock_daily.csv", parse_dates=["日期"])
stock = stock.set_index("日期").sort_index()
stock["日收益率"] = stock["收盘"].pct_change()

cum = (1 + stock["日收益率"]).cumprod()
print(f"500 日累计收益率: {(cum.iloc[-1] - 1) * 100:.1f}%")
print(f"等价于最终净值: {cum.iloc[-1]:.2f} 元（最初 1 元）")
print(f"最大单日跌幅: {stock['日收益率'].min() * 100:.2f}%")
print(f"最大单日涨幅: {stock['日收益率'].max() * 100:.2f}%")
```

```
"00 日累计收益率：357.7%
等价于最终净值：4.58 元（最初 1 元）
最大单日跌幅：-7.14%
最大单日涨幅：+6.18%
```

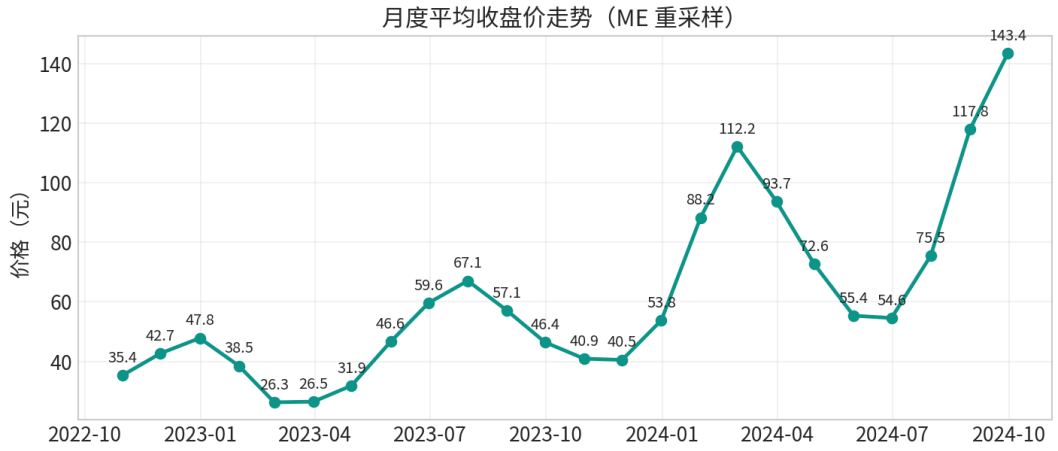

图 7-2 月度平均收盘价。可以看出几个明显的上行/回落阶段。

本章小结

我们学会了多表连接（concat/merge）和时间序列分析（重采样、滚动窗口、收益率）。下一章我们进入更『方法论』的领域——描述统计、相关性、假设检验与回归。

第 8 章 · Python 数据分析与可视化

数据分析常用方法

光把数据『整理』成表格还不够——我们要让数据『说话』。本章用四个真实场景演示：① 描述性统计与分布；② 相关性；③ 假设检验；④ 线性回归；⑤ 特征工程。它们覆盖了 80% 的日常分析需求。

8.1 描述性统计：分布的形状

```
""import pandas as pd
scores = pd.read_csv("data/students_scores.csv")
sub = ["语文","数学","英语","物理","化学","生物"]
print("六科统计摘要:")
print(scores[sub].describe().round(2).to_string())
print("\n偏度（>0 右偏，<0 左偏）:")
print(scores[sub].skew().round(2).to_string())
print("\n峰度（>0 比正态更尖，<0 更平）:")
print(scores[sub].kurtosis().round(2).to_string())

math = scores["数学"]
print(f"\n数学: 最高 {math.max():.0f}，最低 {math.min():.0f}，"
      f"及格率 {(math >= 60).mean()*100:.0f}%，优秀率 {(math >=
90).mean()*100:.0f}%")
```

```
"科统计摘要:
       语文    数学    英语    物理    化学    生物
count  120.0  120.0  120.0  120.0  120.0  120.0
mean    82.3   77.7   80.2   75.0   76.0   80.4
std      8.8   11.1    8.5   10.5    9.6   10.1
min     60.1   45.6   61.6   47.4   50.7   53.2
25%     76.4   70.0   74.6   67.9   69.0   74.0
50%     82.7   78.4   80.0   75.4   75.8   81.1
75%     88.4   85.5   86.4   82.5   82.7   87.7
max     99.0   99.7   99.4   97.4   96.7   99.5

偏度（>0 右偏，<0 左偏）：
语文   -0.16
数学   -0.13
英语   -0.10
物理   -0.13
化学   -0.07
生物   -0.07

峰度（>0 比正态更尖，<0 更平）：
语文   -0.65
数学   -0.55
英语   -0.55
物理   -0.53
化学   -0.49
生物   -0.55
```

> 偏度 ≈ 0、峰度 ≈ 0 时近似正态。实际成绩数据通常略左偏（部分科目拉低）且峰度为负（比正态扁平）。

8.2 相关性分析：皮尔逊 vs 斯皮尔曼

```
""import pandas as pd
scores = pd.read_csv("data/students_scores.csv")
tips = pd.read_csv("data/tips.csv")
sub = ["语文","数学","英语","物理","化学","生物"]
corr = scores[sub].corr(method="pearson")
print("皮尔逊相关系数:")
print(corr.round(3).to_string())

# 数学相关性最高的科目
print("\n数学相关性最高:", corr["数学"].drop("数学").idxmax(),
      "r =", round(corr["数学"].drop("数学").max(), 3))

# 斯皮尔曼：对异常值更稳健
print("\n小费数据的两种相关:")
print("  皮尔逊  r =", round(tips["总消费"].corr(tips["小费"]), 3))
print("  斯皮尔曼 r =", round(tips["总消费"].corr(tips["小费"],
method="spearman"), 3))
```

```
"尔逊相关系数:
        语文    数学    英语    物理    化学    生物
语文    1.000  0.488  0.486  0.413  0.421  0.481
数学    0.488  1.000  0.520  0.764  0.640  0.471
英语    0.486  0.520  1.000  0.491  0.524  0.451
物理    0.413  0.764  0.491  1.000  0.667  0.474
化学    0.421  0.640  0.524  0.667  1.000  0.560
生物    0.481  0.471  0.451  0.474  0.560  1.000

数学相关性最高: 物理 r = 0.764

小费数据的两种相关:
  皮尔逊  r = 0.576
  斯皮尔曼 r = 0.567
```

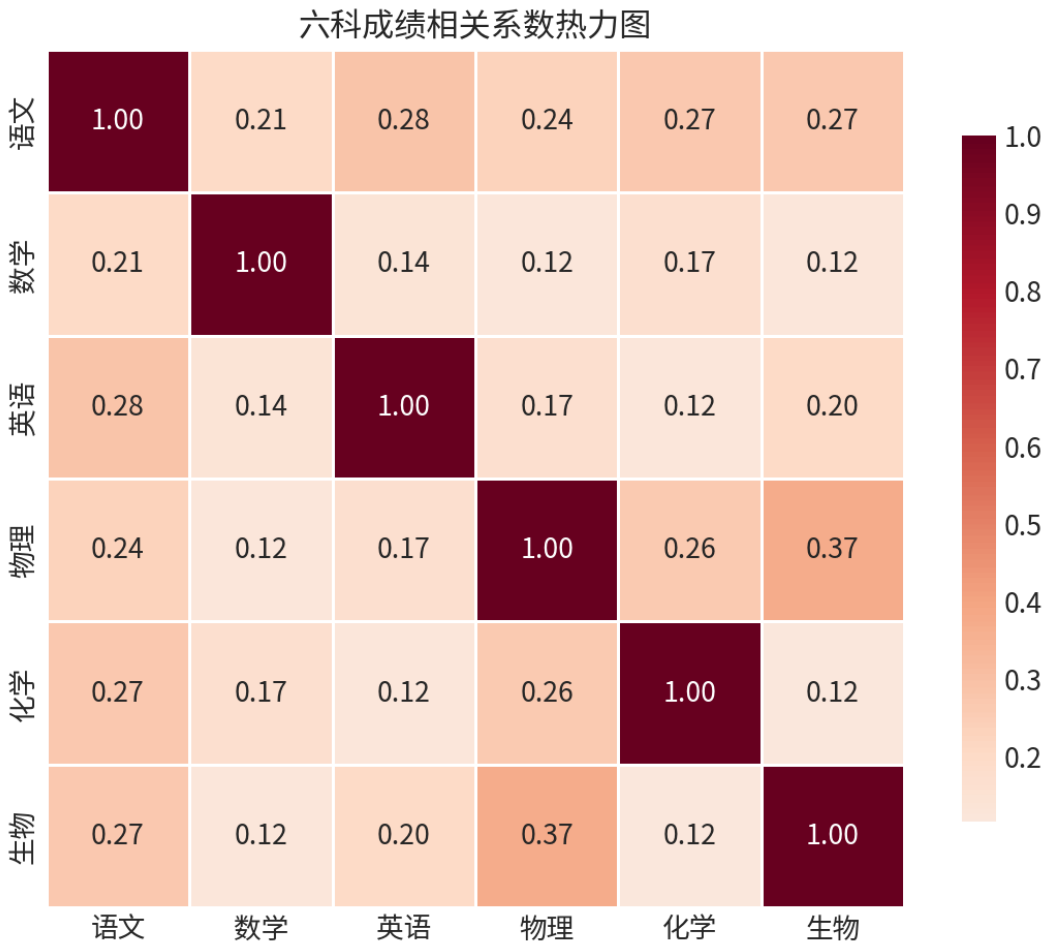

图 8-1 六科成绩相关系数热力图。颜色越红（蓝）正（负）相关越强。数学与物理的强相关（r=0.76）反映了两科背后的『逻辑思维』相通。

8.3 假设检验：t 检验

想知道『男女生数学成绩是否有显著差异』？可以用独立样本 t 检验。原假设 H0：『两组均值相等』；备择假设 H1：『两组均值不等』。p 值越小，越能拒绝 H0。

```
""import pandas as pd
from scipy import stats
scores = pd.read_csv("data/students_scores.csv")
male   = scores.loc[scores["性别"] == "男", "数学"]
female = scores.loc[scores["性别"] == "女", "数学"]
t, p = stats.ttest_ind(male, female)
print(f"男数学 n={len(male)} 均值 {male.mean():.2f}")
print(f"女数学 n={len(female)} 均值 {female.mean():.2f}")
print(f"t = {t:.3f},  p = {p:.4f}")
print("结论:", "差异显著 (p<0.05) " if p < 0.05 else "差异不显著")
```

```
"数学 n=61 均值 78.21
女数学 n=59 均值 77.15
t = 0.527,  p = 0.5992
结论: 差异不显著
```

8.4 卡方检验：分类变量独立性

```python
""import pandas as pd
from scipy import stats
tips = pd.read_csv("data/tips.csv")
ct = pd.crosstab(tips["性别"], tips["是否吸烟"])
print("性别 × 是否吸烟 交叉表:")
print(ct.to_string())
chi2, p, dof, _ = stats.chi2_contingency(ct)
print(f"\nchi2 = {chi2:.3f},  p = {p:.4f},  自由度 = {dof}")
print("结论:", "吸烟习惯与性别有关联" if p < 0.05 else "吸烟习惯与性别独立")
```

```
"别 × 是否吸烟 交叉表:
是否吸烟   否   是
性别
女      82  28
男      70  64

chi2 = 17.840,  p = 0.0000,  自由度 = 1
结论: 吸烟习惯与性别有关联
```

8.5 线性回归与 R^2

```
""import numpy as np, pandas as pd
tips = pd.read_csv("data/tips.csv")
X = np.column_stack([np.ones(len(tips)), tips["总消费"]])
beta = np.linalg.lstsq(X, tips["小费"], rcond=None)[0]
print(f"拟合方程：小费 = {beta[0]:.3f} + {beta[1]:.3f} × 总消费")
print(f"即：每多消费 10 元，小费增加 {beta[1]*10:.2f} 元")

residual = tips["小费"] - X @ beta
ss_res = (residual**2).sum()
ss_tot = ((tips["小费"] - tips["小费"].mean())**2).sum()
r2 = 1 - ss_res / ss_tot
print(f"R² = {r2:.3f}（总消费解释了 {r2*100:.1f}% 的小费变异）")
```

```
"合方程：小费 = 0.907 + 0.105 × 总消费
即：每多消费 10 元，小费增加 1.05 元
R² = 0.332（总消费解释了 33.2% 的小费变异）
```

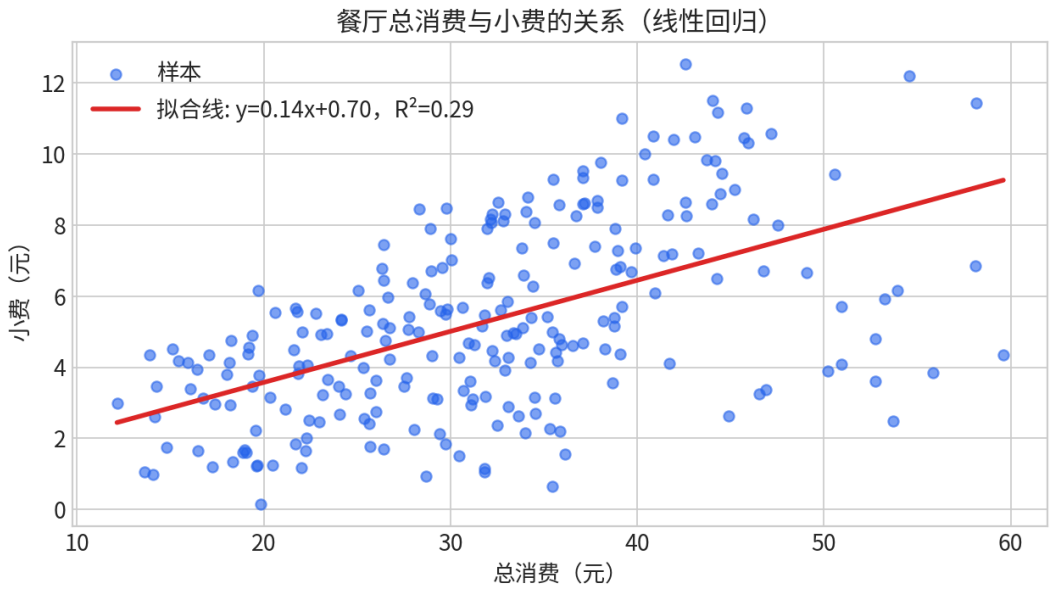

图 8-2 餐厅总消费 vs 小费。$R^2 = 0.33$ 意味着小费变异中约 1/3 由总消费解释，其余来自人数、用餐时段、心情等不可见因素。

8.6 特征工程基础

建模之前通常要做 3 件事：① 把分类变量变成 0/1 哑变量；② 把连续变量分箱离散化；③ 标准化到统一量纲。

```
""import pandas as pd
from sklearn.preprocessing import StandardScaler
tips = pd.read_csv("data/tips.csv")

# 哑变量
dum = pd.get_dummies(tips["用餐时段"], prefix="时段")
print("用餐时段哑变量（前 3 行）:")
print(pd.concat([tips["用餐时段"], dum],
axis=1).head(3).to_string(index=False))

# 分箱
tips["消费档位"] = pd.cut(tips["总消费"], bins=[0, 20, 40, 100],
                          labels=["低","中","高"])
print("\n消费档位分布:")
print(tips["消费档位"].value_counts().to_string())

# 标准化
scores = pd.read_csv("data/students_scores.csv")
scaler = StandardScaler()
z = scaler.fit_transform(scores[["数学","英语"]])
print("\n数学标准化后（均值≈0，标差≈1):")
print(pd.Series(z[:,0]).describe().round(3).to_string())
```

```
"餐时段哑变量（前 3 行）：
用餐时段  时段_午餐  时段_晚餐
晚餐     False     True
晚餐     False     True
午餐      True    False

消费档位分布:
消费档位
中    161
高     48
低     35

数学标准化后（均值≈0，标差≈1)::
count    120.000
mean      -0.000
std        1.004
min       -2.593
25%       -0.621
50%       -0.029
75%        0.689
max        2.114
```

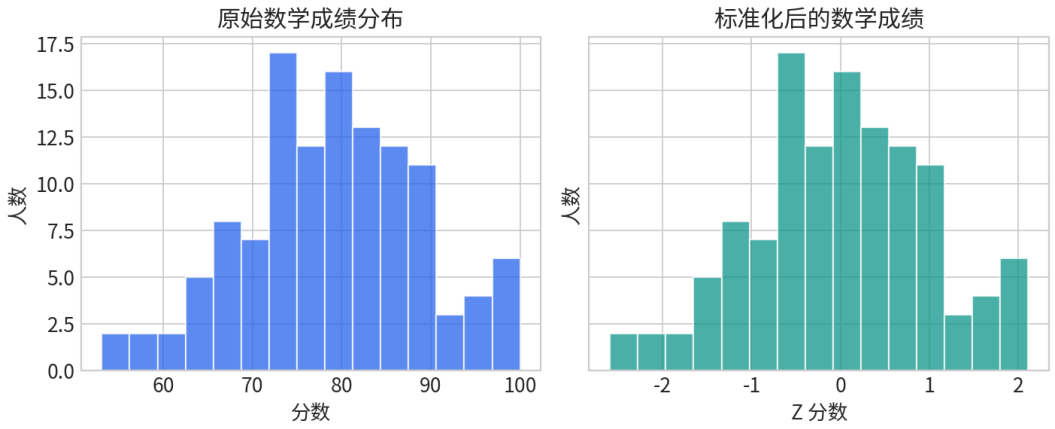

图 8-3 数学成绩标准化前后。形状完全一致，但坐标轴量纲从『分』变为『标准差倍数』，便于不同量纲的特征一起送入模型。

本章小结

我们建立了完整的数据分析方法论：描述性统计 → 相关性 → 假设检验 → 回归 → 特征工程。这是从『看数据』到『用数据』的跨越。下一章我们进入视觉呈现——matplotlib 基础。

第 9 章 · Python 数据分析与可视化

matplotlib 可视化基础

matplotlib 是 Python 可视化的『老祖宗』：功能最全、最稳定、社区最大。虽然有 plotly、seaborn 等更新潮的库，但它们的底层大多还是调用matplotlib。学好它，你就掌握了 90% 的可视化需求。

9.1 折线图：趋势的天然表达

折线图最适合表达『随时间或某个连续变量变化的趋势』。

```
""import matplotlib.pyplot as plt
import pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC", "Microsoft YaHei"]
plt.rcParams["axes.unicode_minus"] = False

sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])
monthly = sales.groupby(sales["日期"].dt.month)["销售额"].sum()

fig, ax = plt.subplots(figsize=(9, 4.2))
ax.plot(monthly.index, monthly.values/10000,
        marker="o", color="#2563eb", linewidth=2.2, markersize=6)
ax.axvline(6, color="#f97316", ls="--", alpha=0.7, label="618 大促")
ax.axvline(11, color="#dc2626", ls="--", alpha=0.7, label="双 11")
ax.set_xticks(monthly.index)
ax.set_xticklabels([f"{m}月" for m in monthly.index])
ax.set_title("2024 年电商月度销售额趋势（万元）")
ax.set_xlabel("月份"); ax.set_ylabel("销售额（万元）")
ax.grid(alpha=0.3); ax.legend()
```

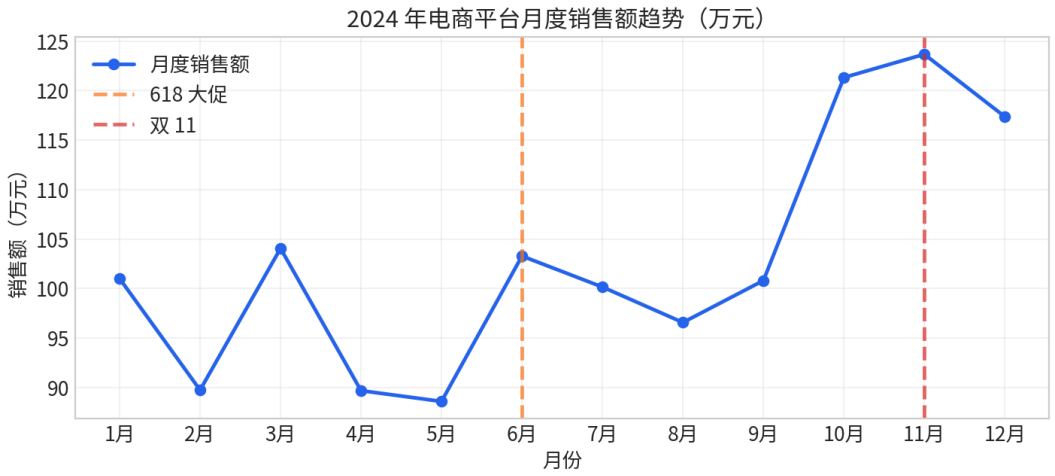

图 9-1 月度销售额趋势。橙色与红色虚线分别标记 618 与 双 11 两个大促节点，肉眼可见 11 月份因双 11 出现尖峰。

9.2 柱状图：不同类别的比较

柱状图家族很丰富：垂直柱、水平柱、分组柱、堆叠柱。它们都用来『比较类别之间的数值』，只是表达『多少』与『关系』时各有侧重。

```
""import matplotlib.pyplot as plt, numpy as np, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
sales = pd.read_csv("data/sales_2024.csv")

# 水平柱状图
member = sales.groupby("会员等级")["销售额"].sum().sort_values()
fig, ax = plt.subplots(figsize=(8, 4.2))
ax.barh(member.index, member.values/10000,
        color=["#94a3b8","#2563eb","#f59e0b","#7c3aed"])
for i, v in enumerate(member.values/10000):
    ax.text(v+1, i, f"{v:.0f}", va="center", fontweight="bold")
ax.set_title("各会员等级销售额（万元）"); ax.set_xlabel("销售额（万元）")
fig.tight_layout()

# 分组柱状图
ct = pd.crosstab(sales["区域"], sales["支付方式"])
fig, ax = plt.subplots(figsize=(10, 4.4))
x = np.arange(len(ct.index)); w = 0.16
for i, col in enumerate(ct.columns):
    ax.bar(x + i*w, ct[col], w, label=col)
ax.set_xticks(x + w*2); ax.set_xticklabels(ct.index)
ax.set_title("各区域不同支付方式的订单量")
ax.set_ylabel("订单数"); ax.legend(ncol=5, fontsize=9)
fig.tight_layout()
```

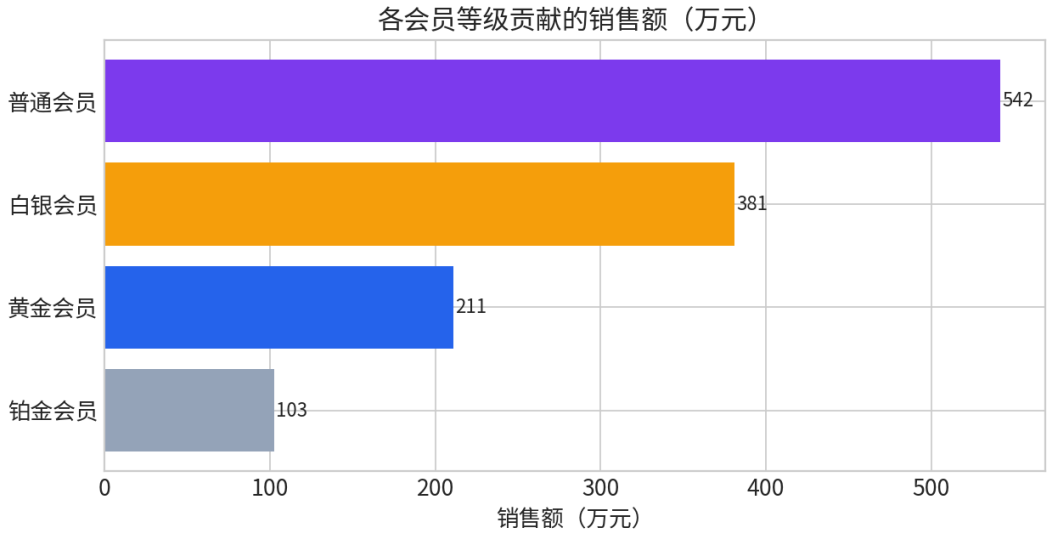

图 9-2 水平柱状图最适合类别名较长或类别数较多的情况，一眼看出大小排序。

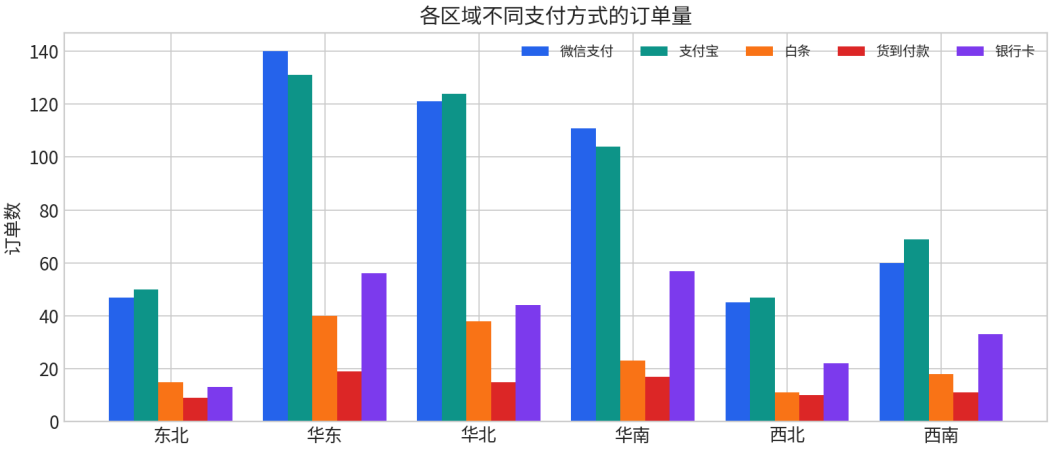

图 9-3 分组柱状图。在每个区域类别内进一步细分到支付方式，方便横向 + 纵向双维比较。

9.3 饼图与环形图：构成比例

饼图表达『部分占整体的比例』，不超过 6 个类别时效果最好。当类别很多时，考虑用『条形图』代替。

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
sales = pd.read_csv("data/sales_2024.csv")
cat = sales.groupby("品类")["销售额"].sum().sort_values(ascending=False)
colors = ["#2563eb","#0d9488","#f97316","#7c3aed","#16a34a","#f59e0b"]
fig, ax = plt.subplots(figsize=(7, 5))
wedges, texts, autotexts = ax.pie(cat.values, labels=cat.index,
        autopct="%.1f%%", colors=colors, startangle=90, counterclock=False,
        wedgeprops={"edgecolor":"white","linewidth":2})
for t in autotexts: t.set_color("white"); t.set_fontweight("bold")
ax.set_title("六大品类销售额占比")
```

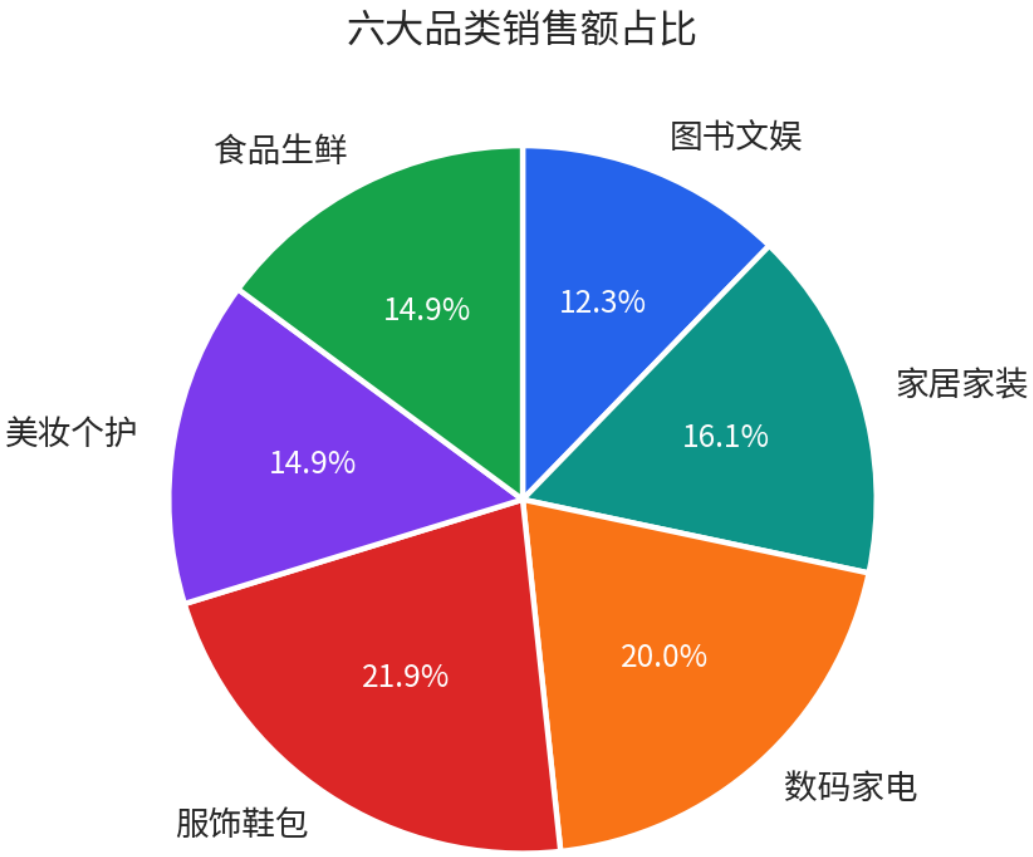

图 9-3 六大品类的销售占比。服饰鞋包与数码家电合计占近 43%，是平台最核心的两条业务线。

9.4 散点图：两连续变量的关系

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
tips = pd.read_csv("data/tips.csv")
fig, ax = plt.subplots(figsize=(8, 4.6))
for sex, c in zip(["男","女"], ["#2563eb","#dc2626"]):
    sub = tips[tips["性别"] == sex]
    ax.scatter(sub["总消费"], sub["小费"], s=34, alpha=0.6, color=c,
label=f"{sex}性")
ax.set_title("餐厅消费与小费的关系（按性别着色）")
ax.set_xlabel("总消费（元）"); ax.set_ylabel("小费（元）")
ax.legend()
```

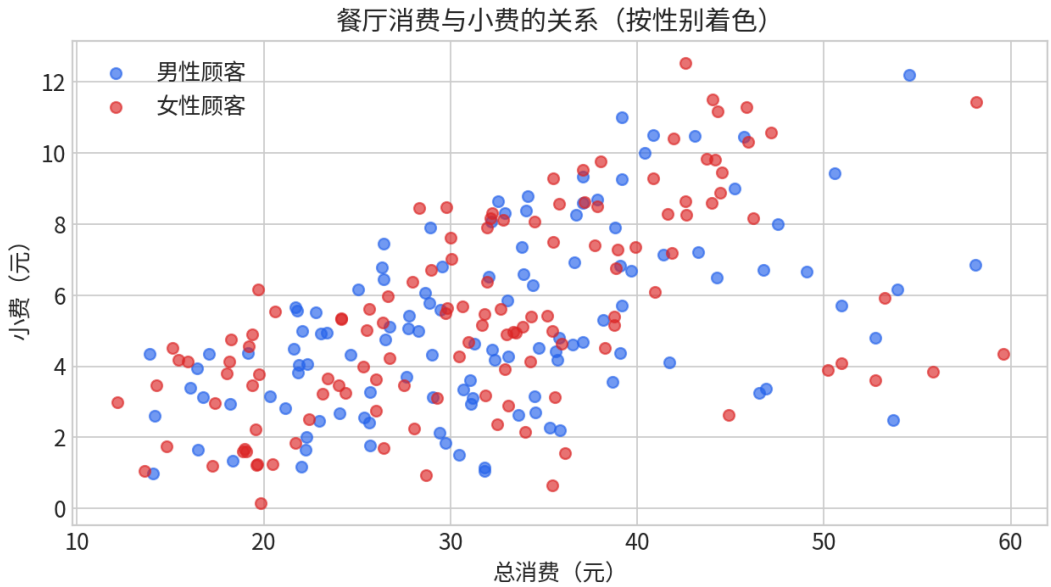

图 9-4 散点图展示两连续变量的关系。点的颜色 / 大小 / 形状可以编码第三个变量，让一张图承载更多信息。

9.5 直方图：分布的形状

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
sales = pd.read_csv("data/sales_2024.csv")
fig, ax = plt.subplots(figsize=(8, 4.2))
ax.hist(sales["单价"], bins=30, color="#0d9488", alpha=0.8,
        edgecolor="white", label="商品单价")
ax.axvline(sales["单价"].mean(), color="#dc2626", ls="--", lw=2,
           label=f"均值 {sales['单价'].mean():.0f} 元")
ax.axvline(sales["单价"].median(), color="#f97316", ls="-.", lw=2,
           label=f"中位数 {sales['单价'].median():.0f} 元")
ax.set_title("商品单价分布（右偏：多数便宜，少数很贵）")
ax.set_xlabel("单价（元）"); ax.set_ylabel("订单数")
ax.legend()
```

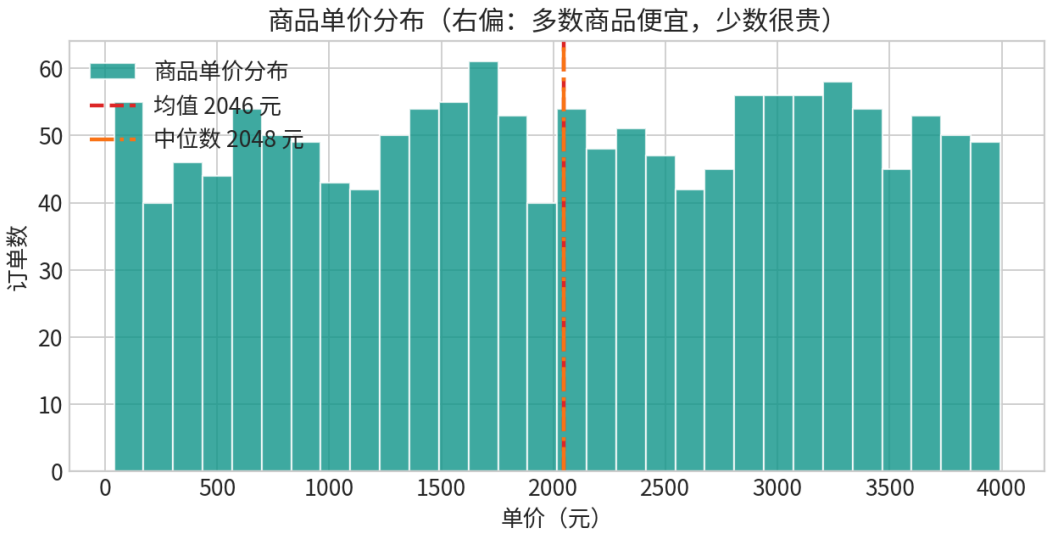

图 9-5 直方图 = 把连续变量按区间分桶再画柱形。集中趋势用均值线、中位数线同时标出，便于观察偏态与离群。

9.6 箱线图：五数概括

箱线图（boxplot）用『五数概括』（最小、Q1、中位数、Q3、最大）压缩地展示分布，特别适合『跨组比较分布差异』。

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
sales = pd.read_csv("data/sales_2024.csv")
fig, ax = plt.subplots(figsize=(9, 4.6))
data = [sales.loc[sales["品类"]==c, "单价"].values for c in sales["品
类"].unique()]
bp = ax.boxplot(data, labels=sales["品类"].unique(), patch_artist=True)
for patch, c in zip(bp["boxes"],
       ["#2563eb","#0d9488","#f97316","#7c3aed","#16a34a","#f59e0b"]):
    patch.set_facecolor(c); patch.set_alpha(0.65)
ax.set_title("各品类商品单价分布（箱线图）")
ax.set_ylabel("单价（元）"); ax.grid(axis="y", alpha=0.3)
```

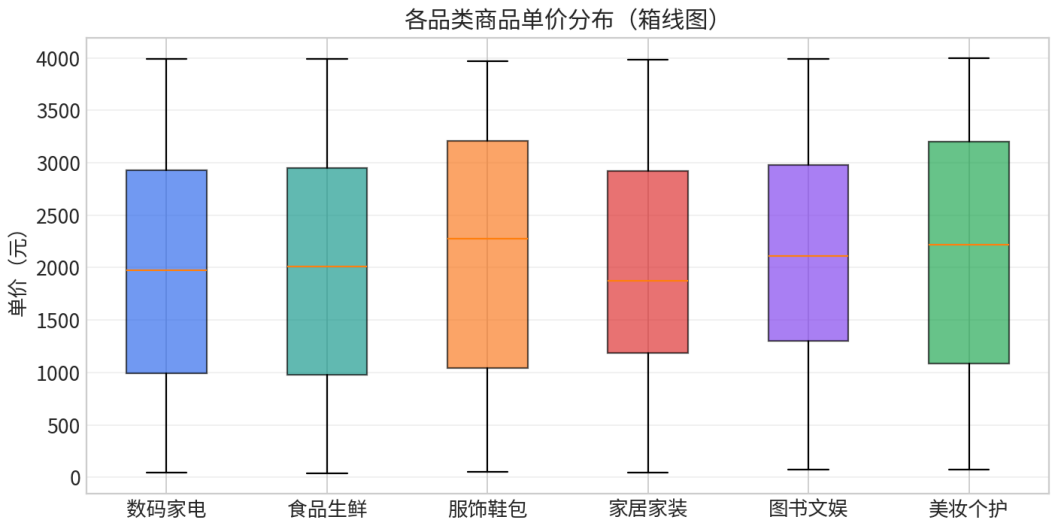

图 9-6 箱线图叠加。盒子是 Q1~Q3，中间线是中位数，『须』延伸至 1.5×IQR，超出范围的散点就是离群值。

本章小结

我们掌握了 matplotlib 的 6 大基础图：折线、柱状、饼图、散点、直方图、箱线图。它们覆盖了日常 80% 的可视化场景。下一章我们学习『进阶技巧』：多子图、双 Y 轴、注释、颜色映射，让图表更专业。

第 10 章 · Python 数据分析与可视化

matplotlib 进阶技巧

基础图表看数据，进阶图表讲故事。本章带你掌握：多子图布局、双 Y 轴、填充区域、注释、颜色映射、样式定制，最后再用极坐标和 3D 图炫技一下。

10.1 多子图布局：subplots 网格

`plt.subplots(nrows, ncols, figsize)` 一次性创建网格化画布，再通过索引 axes[i, j] 操作每个子图。

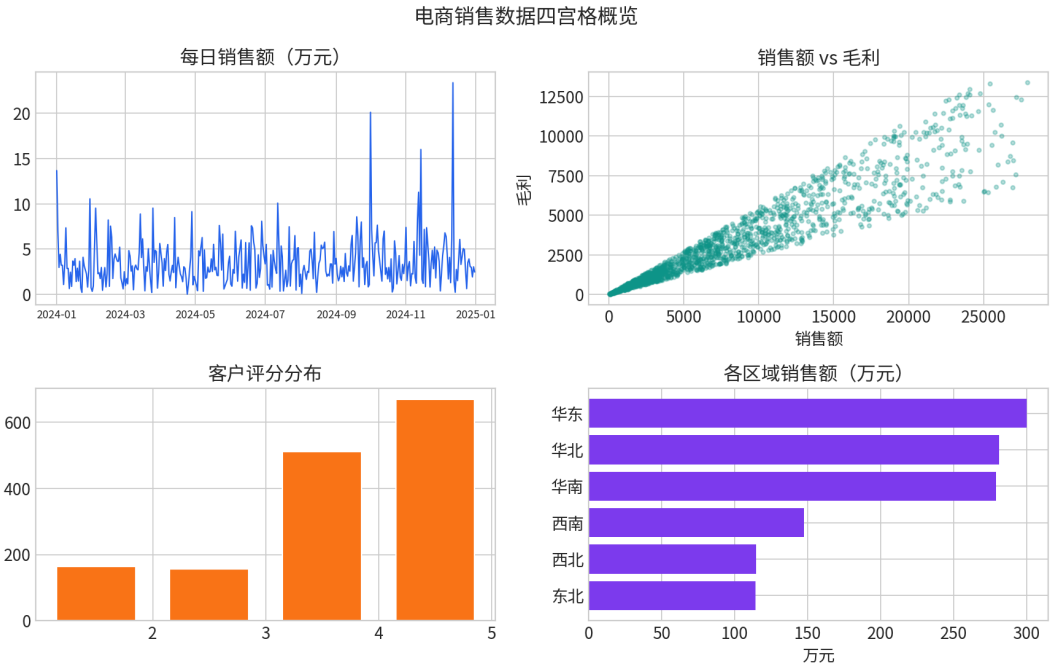

图 10-1 四宫格布局：一张画布同时展示销售趋势、毛利散点、评分分布、区域销售。把多个相关图放一起，故事更完整。

10.2 双 Y 轴：不同量纲同图呈现

当两个量纲不同但又有强相关的指标（如价格与成交量）想放一起时，用 `ax.twinx()` 创建共享 X 轴但独立 Y 轴的子图。

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
stock = pd.read_csv("data/stock_daily.csv", parse_dates=["日期"])
s = stock.set_index("日期").loc["2023-06":"2023-09"]

fig, ax1 = plt.subplots(figsize=(10, 4.6))
ax1.plot(s.index, s["收盘"], color="#2563eb", lw=2, label="收盘价")
ax1.set_ylabel("收盘价（元）", color="#2563eb")
ax1.tick_params(axis="y", labelcolor="#2563eb")

ax2 = ax1.twinx()
ax2.bar(s.index, s["成交量(手)"], color="#94a3b8", alpha=0.4,
        width=0.8, label="成交量")
ax2.set_ylabel("成交量（手）", color="#94a3b8")
ax2.tick_params(axis="y", labelcolor="#94a3b8")
ax1.set_title("股价与成交量（双 Y 轴）")
```

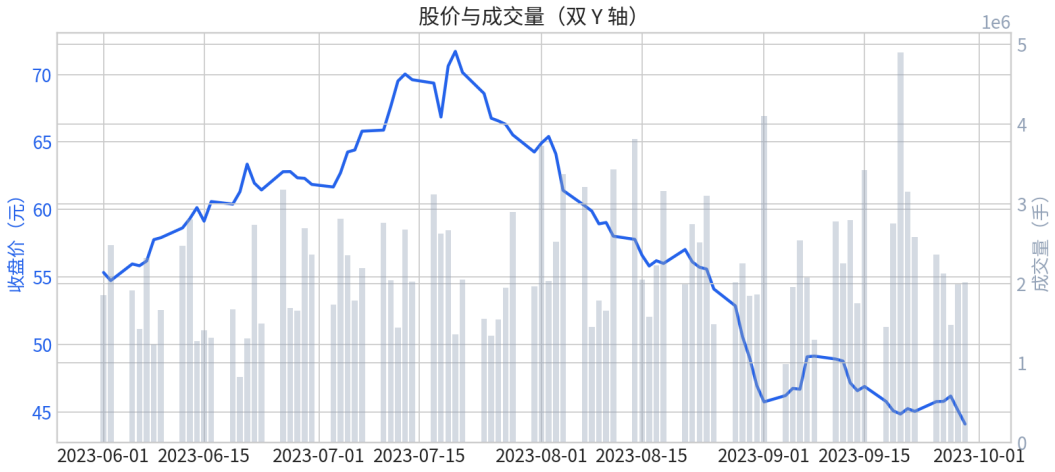

图 10-2 蓝色折线：收盘价；灰色柱：成交量。左右 Y 轴量纲独立，但共用 X 轴时间。这种图在金融场景非常常见。

10.3 填充区域：让『区间』一眼可见

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
w = pd.read_csv("data/weather_daily.csv", parse_dates=["日期"]).head(120)
fig, ax = plt.subplots(figsize=(10, 4.4))
ax.fill_between(w["日期"], w["最低气温"], w["最高气温"],
                color="#2563eb", alpha=0.18, label="气温区间")
ax.plot(w["日期"], w["最高气温"], color="#dc2626", lw=1.6, label="最高气温")
ax.plot(w["日期"], w["最低气温"], color="#2563eb", lw=1.6, label="最低气温")
ax.axvspan(pd.Timestamp("2024-02-01"), pd.Timestamp("2024-02-29"),
           color="#f59e0b", alpha=0.12, label="2 月（最冷）")
ax.set_title("2024 年 1~4 月气温走势（填充面积图）")
ax.set_ylabel("气温（℃）"); ax.legend(ncol=4, fontsize=9)
```

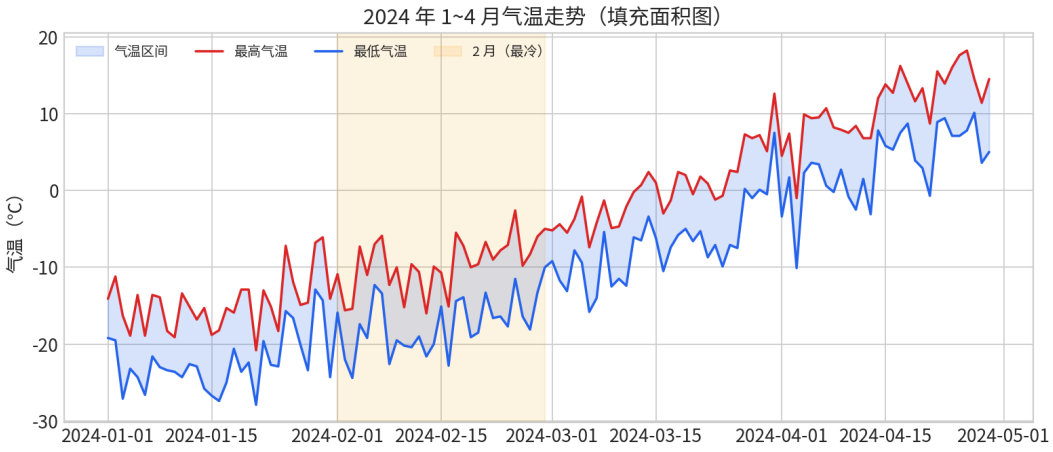

图 10-3 浅蓝色区域是每天最高-最低温度的『气温走廊』，2 月用浅黄背景高亮——整张图的信息密度被浓缩在一眼之内。

10.4 注释与标注：让重点自己跳出来

```
""import matplotlib.pyplot as plt, pandas as pd
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])
monthly = sales.groupby(sales["日期"].dt.month)["销售额"].sum()
fig, ax = plt.subplots(figsize=(9, 4.4))
ax.plot(monthly.index, monthly.values/10000, marker="o", color="#2563eb")
ax.annotate("双 11 销售高峰", xy=(11, monthly[11]/10000),
            xytext=(8.2, monthly[11]/10000 + 4),
            arrowprops=dict(arrowstyle="->", color="#dc2626", lw=1.8),
            fontsize=12, fontweight="bold", color="#dc2626")
ax.annotate("年货节小高峰", xy=(1, monthly[1]/10000),
            xytext=(1.6, monthly[1]/10000 - 6),
            arrowprops=dict(arrowstyle="->", color="#f97316", lw=1.4),
            fontsize=10, color="#f97316")
ax.set_xticks(monthly.index)
ax.set_xticklabels([f"{m}月" for m in monthly.index])
ax.set_title("月度销售额与关键节点标注")
ax.set_ylabel("销售额（万元）"); ax.grid(alpha=0.3)
```

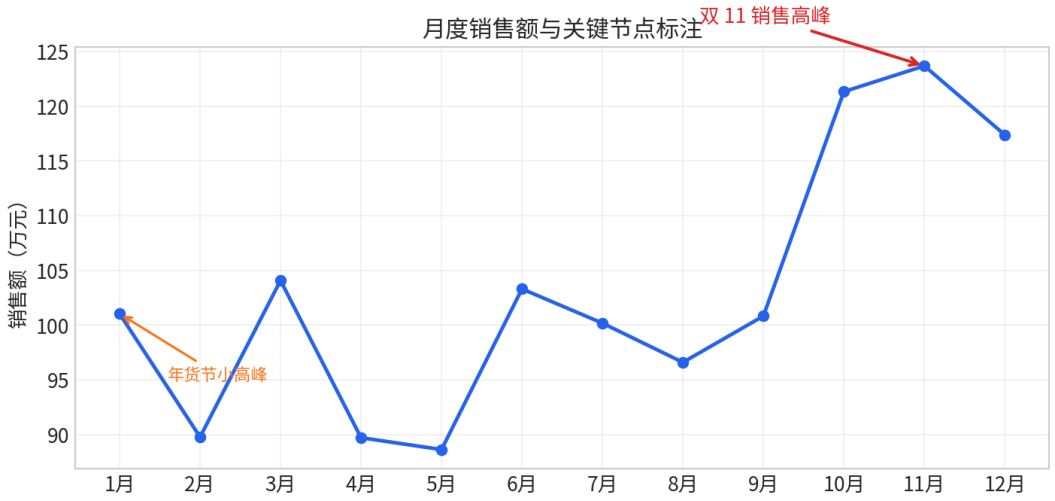

图 10-4 注释让『为什么这里重要』一目了然。arrowprops 控制箭头形状，xytext 控制文本位置——多试几次找到最舒服的搭配。

10.5 颜色映射：数据到颜色的编码

```
""import matplotlib.pyplot as plt, numpy as np
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
rng = np.random.default_rng(5)
x, y = rng.uniform(0, 10, 300), rng.uniform(0, 10, 300)
z = np.sin(x) * np.cos(y) + rng.normal(0, 0.15, 300)
fig, ax = plt.subplots(figsize=(8, 4.8))
sc = ax.scatter(x, y, c=z, cmap="viridis", s=40, alpha=0.85,
edgecolor="none")
fig.colorbar(sc, ax=ax, shrink=0.85, label="目标值 z")
ax.set_title("散点图颜色映射（viridis 色带）")
ax.set_xlabel("x"); ax.set_ylabel("y")
```

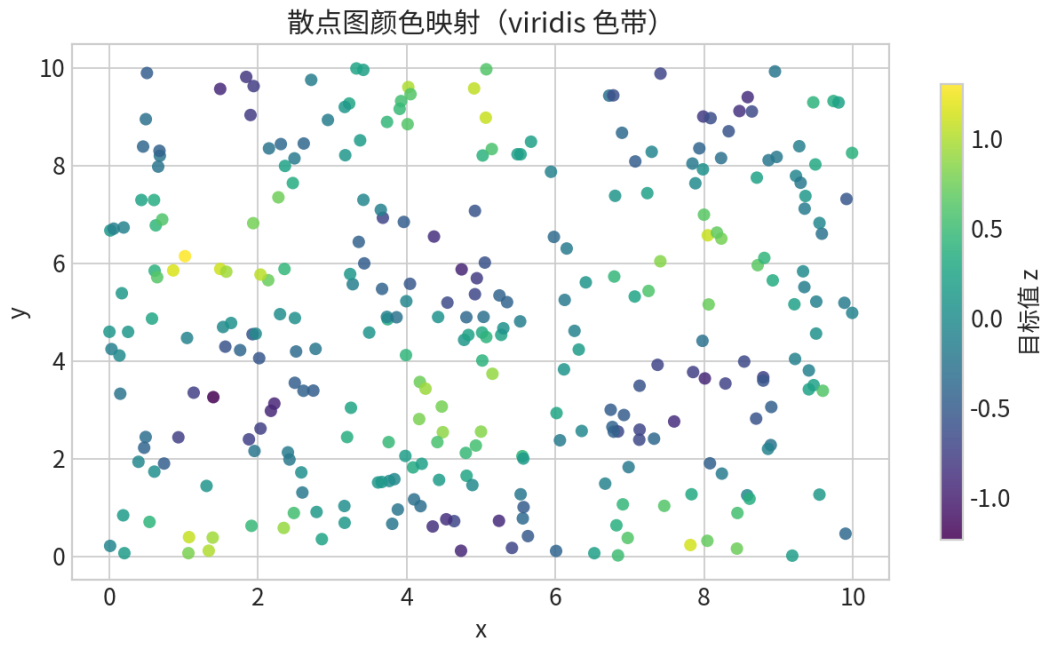

图 10-5 c 参数把数据映射到颜色。本图的 z 值是 sin(x)cos(y)，你能看到颜色呈带状分布——与数学规律一致。

10.6 样式定制：主题与微调

```
""import matplotlib.pyplot as plt, numpy as np
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
plt.style.use("seaborn-v0_8-whitegrid")
fig, ax = plt.subplots(figsize=(8, 4.2))
x = np.linspace(0, 2*np.pi, 200)
ax.plot(x, np.sin(x), color="#2563eb", lw=2.5, label="sin(x)")
ax.plot(x, np.cos(x), color="#f97316", lw=2.5, ls="--", label="cos(x)")
ax.fill_between(x, np.sin(x), np.cos(x),
                where=(np.sin(x) > np.cos(x)),
                color="#2563eb", alpha=0.12)
ax.set_title("sin 与 cos：样式定制示例", pad=12)
ax.set_xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi])
ax.set_xticklabels(["0", "π/2", "π", "3π/2", "2π"])
ax.legend(frameon=True, fancybox=True, shadow=True, loc="upper right")
ax.spines["top"].set_visible(False); ax.spines["right"].set_visible(False)
```

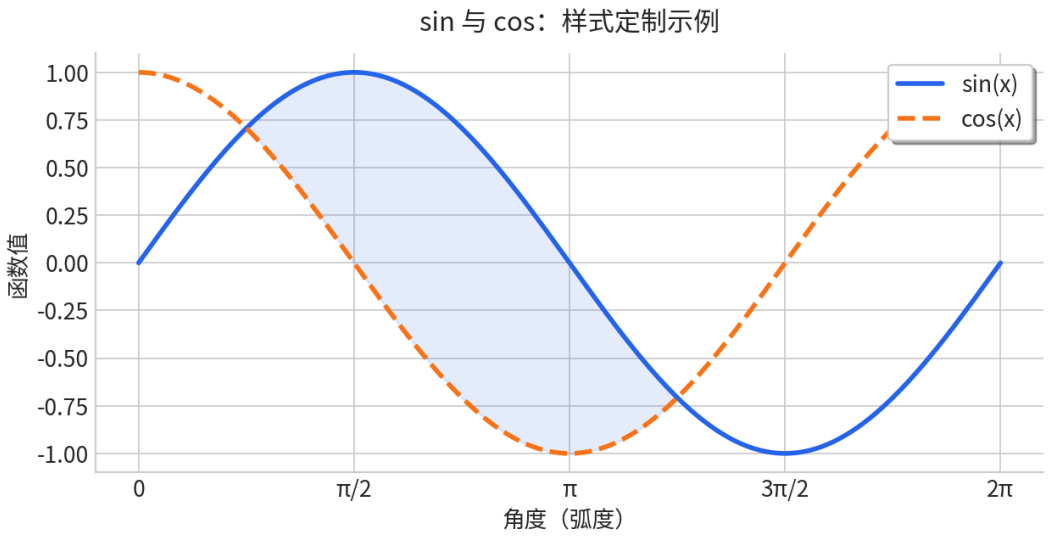

图 10-6 通过 spines 隐藏上/右坐标轴、设置阴影图例、定制 tick 标签——几个小细节，整张图立刻『专业』起来。

10.7 小彩蛋：极坐标与 3D

```
""import matplotlib.pyplot as plt, numpy as np
plt.rcParams["font.sans-serif"] = ["Noto Sans CJK SC"]
fig = plt.figure(figsize=(12, 4.6))

# 极坐标：四叶玫瑰
ax1 = fig.add_subplot(121, projection="polar")
theta = np.linspace(0, 2*np.pi, 200)
r = 1 + 0.6 * np.sin(4*theta)
ax1.plot(theta, r, color="#7c3aed", lw=2.2)
ax1.fill(theta, r, color="#7c3aed", alpha=0.25)
ax1.set_title("四叶玫瑰极坐标图")

# 3D 曲面
ax2 = fig.add_subplot(122, projection="3d")
x = y = np.linspace(-3, 3, 60); X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
ax2.plot_surface(X, Y, Z, cmap="coolwarm", linewidth=0, antialiased=True)
ax2.set_title("3D 曲面图 sin(sqrt(x²+y²))")
```

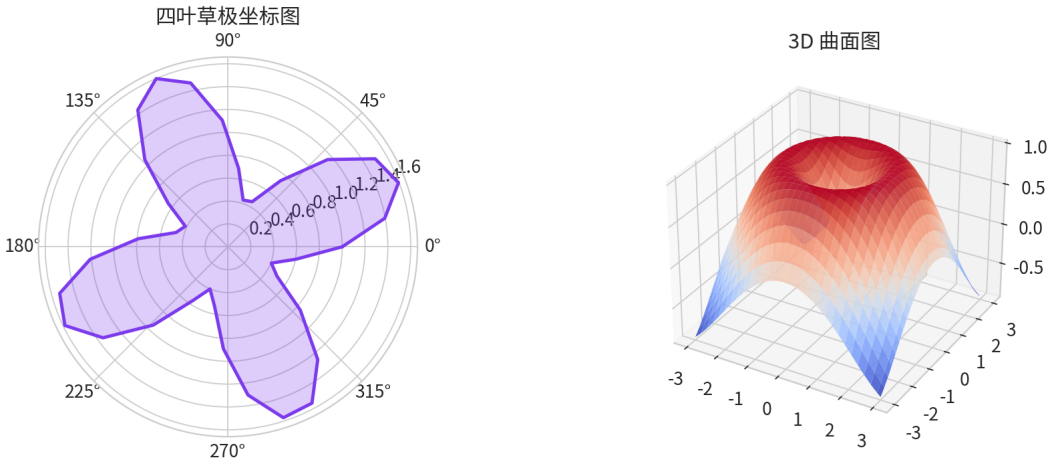

图 10-7 极坐标与 3D 曲面：使用得当是亮点，使用过滥是灾难。业务场景慎用，它们往往在『炫技』和『清晰』之间走钢丝。

本章小结

我们掌握了 matplotlib 的进阶技巧：subplots 网格、双 Y 轴、填充、注释、颜色映射、样式定制、极坐标与 3D。下一章——综合实战，把所有技能串联起来。

第 11 章 · Python 数据分析与可视化

综合实战：电商销售数据分析全流程

前 10 章我们学习了 NumPy、pandas、统计分析、matplotlib。本章我们把这些能力『穿』在一起，对一份真实的电商订单数据做端到端分析，得到一份可直接交付的『经营分析报告』。

11.1 业务背景与分析目标

我们拿到一份 2024 年某电商平台导出的 1500 笔订单数据，包含 14 个字段。业务方希望回答以下问题：

1. 全年销售总盘与盈利能力如何？
2. 月度趋势有什么特点？哪些大促节点贡献最大？
3. 哪些品类 / 区域是营收支柱？是否有优化空间？
4. 会员分层是否健康？不同等级会员价值差异多大？
5. 数据中有哪些变量高度相关？能给业务什么启发？

11.2 数据加载与质量检查

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])
print("加载完成：", sales.shape)
print("\n缺失值总数:", int(sales.isna().sum().sum()))
print("完全重复行:", sales.duplicated().sum())

# IQR 异常值检测
q1, q3 = sales["销量"].quantile([0.25, 0.75])
upper = q3 + 1.5 * (q3 - q1)
n_out = (sales["销量"] > upper).sum()
sales = sales[sales["销量"] <= upper].drop_duplicates().dropna()
sales["月份"] = sales["日期"].dt.month
print(f"剔除 {n_out} 个异常订单 + 去重后：{sales.shape}")
```

```
"载完成： (1500, 14)

缺失值总数: 0
完全重复行: 0
剔除 0 个异常订单 + 去重后：(1500, 15)
```

11.3 经营概览 KPI

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])
total_amt = sales["销售额"].sum()
total_gross = sales["毛利"].sum()
print(f"全年销售额：{total_amt/10000:.1f} 万元")
print(f"全年毛利：  {total_gross/10000:.1f} 万元（毛利率 {total_gross/
total_amt*100:.1f}%）")
print(f"总订单数：  {len(sales)} 笔，平均客单价 {total_amt/len(sales):.0f} 元")
print(f"平均评分：  {sales['客户评分'].mean():.2f} / 5")


```

```
"年销售额：1235.5 万元
全年毛利：  584.5 万元（毛利率 47.3%）
总订单数：  1500 笔，平均客单价 8236 元
平均评分：  4.18 / 5
```

11.4 趋势、品类、区域三维度

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv", parse_dates=["日期"])
sales["月份"] = sales["日期"].dt.month
monthly = sales.groupby("月份")["销售额"].sum()
print("月度销售额（万元）:")
print((monthly/10000).round(1).to_string())
print(f"\n峰值月: {monthly.idxmax()} 月 ({(monthly.max()/10000):.0f} 万元)")
print(f"次峰月: {monthly.sort_values(ascending=False).index[1]} 月")

# 区域 + 品类
region = sales.groupby("区域")["销售额"].sum()
print("\n区域占比:", "  ".join([f"{k} {v/region.sum()*100:.0f}%"
                               for k, v in region.sort_values(ascending=False).ite
print("品类 Top 3:", " > ".join(sales.groupby("品类")["销售额"].sum()
                                .sort_values(ascending=False).head(3).index.to_l
print("华东拳头品类:", sales[sales["区域"]=="华东"].groupby("品类")["销售额"]
      .sum().idxmax())
```

```
"度销售额（万元）：
月份
1     101.0
2      90.2
3     104.1
4      90.0
5      89.1
6     103.4
7     100.1
8      96.5
9     100.7
10    121.1
11    123.7
12    116.5

峰值月：11 月 （124 万元）
次峰月：10 月

区域占比：华东 24%  华南 22%  华北 23%  西南 12%  西北 9%  东北 9%
品类 Top 3：服饰鞋包 > 数码家电 > 家居家装
华东拳头品类：服饰鞋包
```

11.5 会员与支付洞察

```
""import pandas as pd
sales = pd.read_csv("data/sales_2024.csv")
m = sales.groupby("会员等级").agg(订单数=("订单号","count"), 销售额=("销售
额","sum"))
m["客单价"] = m["销售额"]/m["订单数"]
print(m.round(0).to_string())
print(f"\n普通会员订单数占比 {m.loc['普通会员','订单数']/m['订单数'].sum()*100:.0f}
%, "
      f"是订单基本盘")


```

```
"          订单数        销售额       客单价
会员等级
白银会员    447   3811297.0   8526.0
铂金会员    119   1025439.0   8617.0
黄金会员    258   2108179.0   8171.0
普通会员    676   5418827.0   8016.0

普通会员订单数占比 45%，是订单基本盘
```

11.6 一页纸『经营仪表盘』

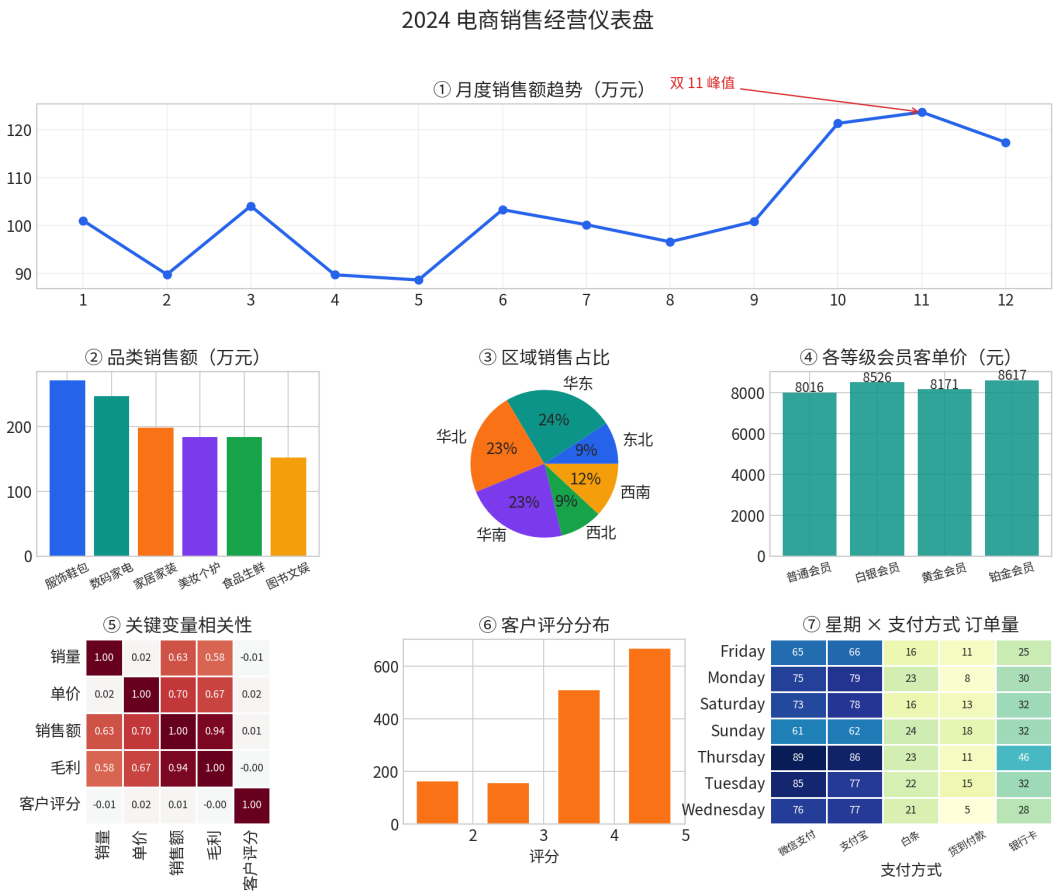

图 11-1 一张图把全年经营讲清楚：销售趋势、品类、区域、会员、相关性、评分、星期×支付。把它贴在会议室，比 20 页 PPT 更有冲击力。

11.7 分析结论与建议

1. 季节性高度显著

11 月（双 11）为全年峰值，10 月次之（国庆 + 大促预热），6 月（618）形成第三个小高峰。建议：① 提前 2 个月备货与营销蓄水；② 探索淡季造节（如换季清仓、年中会员日）以平滑月度波动。

2. 区域结构：华东 + 华北 = 近半营收

华东 + 华北合计 47%，华南紧跟 22.6%；服饰鞋包与数码家电是两大支柱。华东以服饰鞋包为拳头，西南/东北/西北应差异化选品（保暖、防晒、地方特产）。

3. 会员分层待优化

普通会员占订单 45%，是订单基本盘；各等级客单价差异仅 7%（约 0.8~0.86 万元），当前分层尚未形成明显溢价。建议设计阶梯权益（运费券、专属价、积分加速），推动普通 → 白银/黄金的升级，扩大高价值客户基数。

4. 客户满意度

评分以 4~5 分为主（约 79%），整体体验良好；但 21% 订单评分 ≤ 3，建议下钻分析差评集中的品类与区域，针对性优化履约与售后。

5. 定价健康，下一步拓展经营全链路

销售额与毛利高度相关（r=0.94），定价结构健康。下一步可接入退货率、复购率、营销 ROI 等指标，把分析延伸到经营全链路——这正是数据分析师从『看数据』走向『做决策』的必经之路。

全教程总结

恭喜你走完了整个教程！从 NumPy 的 ndarray，到 pandas 的 DataFrame；从描述统计，到假设检验与回归；从折线柱状饼图，到双 Y 轴与综合仪表盘——你已经具备了数据分析师最核心的 80% 技能。 接下来，建议你：① 找一份自己工作 / 学习中的真实数据复刻本章实战；② 学习 scikit-learn，把分析延伸到机器学习；③ 探索 plotly、seaborn、streamlit 等更现代的可视化与建站工具，让分析结果『动起来』。祝你成为一名出色的数据分析师！