BIOF3 组学数据分析

编程基础：R / Python / Bash

导出日期：2026年6月27日

编程基础：R / Python / Bash

做组学数据分析，编程不是目标，而是工具。**目标不是成为程序员**，是能：

- 读懂别人的脚本
- 修改参数让它跑自己的数据
- 看懂报错并找到原因
- 把分析过程整理成可重复的记录

这一篇给出 BioF3 推荐的最小学习路径。

为什么必须学编程

组学数据有三个特点，让点鼠标的工作流彻底失效：

- **数据量大**：表达矩阵、FASTQ、BAM、H5AD 文件动辄几个 GB，Excel 打不开
- **步骤多**：QC、标准化、降维、聚类、差异分析、富集分析...每一步都有参数要记
- **结果要可复现**：论文、报告、协作场景都需要"别人能跑一遍得到一样的图"

光靠点鼠标，下面这些问题答不出来：

- 这次分析用了哪些过滤阈值?
- 归一化和聚类参数是什么?
- 哪张图是哪一步生成的?
- 换一批样本能不能自动重跑?
- 半年后还能不能复现同样结果?

编程的价值就是把分析过程写下来，让它可以检查、重复、修改、共享。

BioF3 推荐工具栈

R：单细胞分析和统计可视化主力

BioF3 的单细胞实践主要用 R 生态。**先学 R 是最直接的路径。**

常见场景：

- Seurat 单细胞分析
- ggplot2 可视化
- DESeq2 / edgeR / limma 差异表达
- 统计检验和建模
- clusterProfiler 富集分析
- R Markdown / Quarto 报告

最低要求：

- 读写 CSV、TSV、RDS
- 用 data.frame / tibble

- 用 dplyr 做筛选、分组、排序、汇总
- 用 ggplot2 画常见图
- 安装和加载包
- 看懂函数参数和报错信息

Python：数据处理、机器学习和 Scanpy 生态

Python 适合处理大规模数据、机器学习、工程化流程。

常见场景：

- pandas / numpy 数据处理
- Scanpy / AnnData 单细胞分析
- scikit-learn 机器学习
- PyTorch 深度学习
- 自动化脚本和 API 调用
- 文件批处理

最低要求：

- 创建虚拟环境（venv / conda）
- 读写 CSV、TSV、JSON、H5AD
- 用 pandas 做表格处理
- 用 matplotlib / seaborn 画图
- 写简单函数
- 根据 traceback 定位错误

Bash：服务器和生信流程的入口

很多生信工具只有命令行版本。不会 Bash，就用不稳服务器和高通量流程。

常见场景：

- 查看和移动文件
- 解压、统计、合并文件
- 批量运行 FastQC、Cell Ranger、STAR、samtools
- 后台任务和日志检查
- 远程服务器操作
- 写 shell 管道和脚本

最低要求：

- `cd` / `ls` / `cp` / `mv` / `mkdir` / `rm`
- `head` / `tail` / `less` / `wc`
- `grep` / `awk` / `sed` 基本用法
- 写简单 `for` 循环
- 重定向输出和查看日志
- 用 `ssh` 登录服务器

AI：助教 + 排错助手 + 草稿生成器

适合让 AI 做的：解释陌生代码 / 把报错翻译成排查步骤 / 生成脚本草稿 / 改写重复代码 / 生成 README / 检查路径和变量名。

不适合完全交给 AI 的：决定实验分组 / 决定统计检验 / 判断 marker gene / 解释疾病机制 / 处理未脱敏临床数据 / 编造软件版本和文献依据。

更完整的工具介绍和提示词模板见 AI 辅助编程与智能体工具。

学习顺序

跟 组学入门 的 4 阶段呼应，编程层面这样切：

第 1 阶段（搭配 overview 第 1 阶段）：能跑、能改

新手最容易陷入"先系统学完整门语言"的误区。**优先目标是能跑通别人写的脚本**。

能做到这几件事就够：

- 打开 RStudio / Jupyter / 终端
- 运行一段教程代码
- 修改输入文件路径
- 修改过滤阈值
- 保存结果图和结果表

练习：

> 拿一份 BioF3 教程脚本，把输入文件路径改成自己的目录，
> 把输出目录改成 results/，跑一遍，确认产出在该出现的位置。

第 2 阶段（搭配 overview 第 2 阶段）：R 入门到可用

先掌握基础数据结构：

```r
# 向量
genes <- c("TP53", "BRCA1", "EGFR")
expr  <- c(5.2, 3.8, 7.1)

# 数据框
gene_data <- data.frame(gene = genes, expression = expr)

# 看一眼数据
head(gene_data)
str(gene_data)
summary(gene_data)

# 基础筛选
high_expr <- gene_data[gene_data$expression > 5, ]
```

再学 dplyr 风格：

```r
library(dplyr)

gene_data <- gene_data %>%
  mutate(log_expression = log2(expression + 1)) %>%
  arrange(desc(expression))
```

最后 ggplot2：

```r
library(ggplot2)

ggplot(gene_data, aes(x = gene, y = expression)) +
  geom_col(fill = "#3B82F6") +
  labs(x = "Gene", y = "Expression") +
  theme_classic()
```

R 阶段目标不是语法完美，是**能读懂 Seurat 教程里的对象、函数、参数。**

第 3 阶段（搭配 overview 第 2-3 阶段）：Python 入门到可用

Python 先从 pandas 开始：

```python
import pandas as pd
import numpy as np

data = pd.DataFrame({
    "gene": ["TP53", "BRCA1", "EGFR"],
    "expression": [5.2, 3.8, 7.1],
})

data["log_expression"] = np.log2(data["expression"] + 1)
high_expr = data[data["expression"] > 5]
print(data["expression"].mean())
```

画图先掌握 matplotlib + seaborn：

```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.barplot(data=data, x="gene", y="expression", color="#3B82F6")
plt.xlabel("Gene")
plt.ylabel("Expression")
plt.tight_layout()
plt.show()
```

Python 阶段目标是能处理表格数据、看懂 Scanpy / AnnData 对象、能写小型自动化脚本。

第 4 阶段（搭配 overview 第 3-4 阶段）：Bash 入门到可用

文件和目录：

```bash
pwd
ls -lh
mkdir -p results/qc
cp data/sample.csv results/
```

查看文件：

```
head expression.tsv
tail -n 20 run.log
wc -l expression.tsv
less run.log
```

批量处理：

```
mkdir -p qc_results

for file in data/*.fastq.gz; do
  echo "Processing $file"
  fastqc "$file" -o qc_results/
done
```

远程服务器：

```
ssh aliyun
scp results/report.html aliyun:/opt/project/results/
```

Bash 阶段目标是**能在服务器上找到文件、运行工具、检查日志、批量处理样本**。

一个最小可复现项目结构

从第一个项目开始，就用固定结构组织文件：

```
project/
├── data/
│   ├── raw/          # 原始数据，永远不动
│   └── processed/    # 中间产物
├── scripts/
│   ├── 01_qc.R
│   ├── 02_normalize.R
│   └── 03_plot.R
├── results/
│   ├── figures/      # 图
│   └── tables/       # 表
├── logs/             # 运行日志
└── README.md         # 顺序、版本、参数
```

为什么这种结构值得：

- `data/raw/` 永远不动 → 任何中间步骤跑出问题都能从原始数据重来
- 脚本编号 `01_` `02_` `03_` → 一眼看出执行顺序
- `results/figures/` 和 `results/tables/` 分开 → 写论文时能直接捞图
- README 列清"运行哪个脚本、需要什么版本、产出在哪" → 半年后自己看才认得这个项目

常见坑

坑 1：报错只看最后一行

R / Python 报错通常很长，新手习惯只看最后一行的红字，结果定位不到真实问题。**真实原因往往在 traceback 中段**。

避免：完整复制整段报错，从下往上读，找到第一个**你自己代码**的位置（不是 library 内部的）。

坑 2：路径写绝对路径，换机器就崩

`read.csv("/Users/zhangsan/data/sample.csv")` — 自己电脑能跑，发给同事就 `file not found`。

避免：用相对路径 `read.csv("data/sample.csv")`，配合 `setwd()` 或 RStudio Project 固定项目根目录。

坑 3：包版本不固定，半年后跑不出来

Seurat v4 → v5 函数签名大改，跑不动同一个脚本。

避免：每个项目里跑 `sessionInfo()` 或 `pip freeze` 把版本记进 README，关键时候用 renv（R）或 conda env（Python）锁版本。

坑 4：循环里改变量名重复了

```r
for (i in 1:length(samples)) {
  i <- samples[i]   # i 既是循环变量又是值，下次循环就崩
  ...
}
```

避免：循环变量和值用不同名字。`for (idx in seq_along(samples)) { sample <- samples[idx]; ... }`。

坑 5：把数据变量名跟函数名重叠

```r
data <- read.csv("foo.csv")    # data 是 R 内置函数，被覆盖了
mean <- mean(data$expr)         # mean 也是函数，再覆盖
```

后面再调 `data()` 或 `mean()` 就会困惑。

避免：用更具体的名字 — `expr_data`、`expr_mean`、`pbmc_counts` 等。

常见问题

没有基础，先学 R 还是 Python?

如果目标是尽快进入单细胞分析，**先学 R**。BioF3 单细胞主线工具用 R / Seurat。Python 可以等学到 Scanpy、机器学习、自动化时再补。

Bash 一定要学吗?

最基础的部分要学。不需要成为 Linux 专家，但要能在服务器上找到文件、运行命令、查看日志。否则很多上游流程（Cell Ranger、STAR）直接卡住。

可以全靠 AI 写代码吗?

不行。AI 可以写草稿，但**必须人工确认**输入数据、列名、统计方法、输出结果和生物学解释。涉及临床数据和未公开项目时，原始数据不要直接上传给外部 AI 服务。详见 AI 辅助编程与智能体工具 的"安全边界"和"5 个常见坑"。

报错时怎么处理?

按这个顺序：

1. 完整复制报错（不只看最后一行）
2. 确认对象是否存在、路径是否正确、包是否加载
3. 用小数据集复现问题

4. 搜索错误信息
5. 让 AI 根据代码、报错、环境信息给排查步骤
6. 修复后把原因记进 README 或 commit message

学到什么程度可以开始跑单细胞教程?

能做到这几件事就够：

- 运行 R 脚本
- 安装和加载 R 包
- 修改文件路径
- 查看 R 对象的基本信息（`str()` / `head()` / `dim()`）
- 保存图和表
- 根据报错做基础排查

下一步

接着深入（按推荐顺序读下去）：

1. 数据与环境准备 — 先把环境装好，避免每次跑教程脚本都报"找不到包"
2. R 数据整理与 ggplot2 可视化 — R 阶段最值得先精通的两件事
3. 单细胞实践 01：实践数据集与数据获取 — 用真实流程检验你学到的语法

横向延伸：

- Jupyter 与交互式分析环境 — 想用 Notebook 做探索性分析时
- 公共数据库与数据检索 — 想找练手数据时
- AI 辅助编程与智能体工具 — 用 AI 帮你打字（不替你思考）

编程能力不是背语法背出来的，是在真实任务里**练**出来的。最快的开始方式：拿一张表，完成"读 → 筛 → 画 → 存"四件事。
