基于多模型融合的无创产前检测时点优化与异常判定研究

摘要

无创产前检测（NIPT）通过检测母体血浆游离 DNA 评估胎儿染色体异常，其检测成功率与孕周、BMI 等因素密切相关，合理确定检测时机并构建异常判定策略具有重要意义。本文针对 NIPT 的检测时点优化与异常判定问题，主要建立了“非线性关联建模—分组与区间删失估计—两阶段时点优化—不平衡分类判定”的综合模型体系，求解得到分 BMI 分层的最优检测时点与女胎异常判定方案。

针对问题一，主要解决“Y 染色体浓度与孕周、BMI 的关系模型”问题，通过相关性与显著性检验分析，建立广义加性混合模型（GAMM），基于整体样条与 Wald 检验求解得到：Y 浓度与孕周显著正相关、与 BMI 显著负相关且存在非线性效应，并给出可解释的函数表达式。

针对问题二，主要解决“按 BMI 合理分组并给出最佳 NIPT 时点、量化误差影响”问题，通过条件推理树（CIT）客观分组，建立基于 Turnbull NPMLE 的区间删失累计达标率模型，结合 Bootstrap 与“首次检测+保底复测”的两阶段优化，求解得到：BMI 越高保底时点越晚；两阶段策略较单次保底可将期望检测时间平均提前约 5–12.5 周，早期达标率维持在约 63%–89%。

针对问题三，主要解决“多因素与测量误差下的稳健时点制定”问题，通过误差校正与参数扰动分析，建立稳健两阶段优化模型，基于自助法/交叉验证评估，求解得到策略对参数不确定性的稳定性与推广性结论。

针对问题四，主要解决“女胎异常的判定与阈值制定”问题，通过特征工程（孕周、BMI、测序质量等）与不平衡分类（如代价敏感/阈值优化的逻辑回归或梯度提升）分析，建立判定模型并以 ROC/PR 指标与决策曲线优化阈值，求解得到高召回、可解释的异常判定方案。

本研究在非线性刻画、区间删失处理、两阶段时点优化与不平衡学习上形成一体化可落地流程，具备可解释与临床可操作性的优点

关键词：无创产前检测（NIPT）、广义加性混合模型（GAMM）、检测时点优化、女胎异常判定

[基于多模型融合的无创产前检测时点优化与异常判定研究 1](#_Toc208166341)

[摘要 1](#_Toc208166342)

[一、问题重述 4](#_Toc208166343)

[1.1问题背景 4](#_Toc208166344)

[1.2问题提出 4](#_Toc208166345)

[二、问题分析 4](#_Toc208166346)

[2.1问题一的分析 4](#_Toc208166347)

[2.2问题二的分析 5](#_Toc208166348)

[2.3问题三的分析 5](#_Toc208166349)

[2.4问题四的分析 5](#_Toc208166350)

[模型假设 6](#_Toc208166351)

[符号说明 6](#_Toc208166352)

[模型建立与求解 7](#_Toc208166353)

[问题一模型的建立与求解 7](#_Toc208166354)

[数据预处理 7](#_Toc208166355)

[Pearson相关性分析 7](#_Toc208166356)

[y染色体浓度与目标变量关系分析 8](#_Toc208166357)

[模型选择 9](#_Toc208166358)

[GAMM非线性拟合 10](#_Toc208166359)

[显著性分析 12](#_Toc208166360)

[问题二模型的建立与求解 12](#_Toc208166361)

[数据预处理 12](#_Toc208166362)

[模型的建立 12](#_Toc208166363)

[（1）基于条件推理树（CIT）的 BMI 分组 12](#_Toc208166364)

[其中，为第一组样本在合并样本中的秩和。若偏离“两组分布无差异时的预期值”，则拒绝原假设，认为分割后两组分布存在显著差异。通过计算对应的值，当（设定的显著性水平，此处=0.3）时，分割操作具有统计显著性。 12](#_Toc208166365)

[（2）基于 Turnbull NPMLE 的最佳检测时点估计 13](#_Toc208166366)

[（3）两阶段优化 13](#_Toc208166367)

[模型的求解与结果分析 14](#_Toc208166368)

[综合风险评分与模型性能 17](#_Toc208166369)

[问题三模型的建立与求解 18](#_Toc208166370)

[数据预处理 18](#_Toc208166371)

[模型建立 18](#_Toc208166372)

[问题四模型的建立与求解 22](#_Toc208166373)

[数据预处理 22](#_Toc208166374)

[模型构建：多特征融合的逻辑回归 - 随机森林组合模型 24](#_Toc208166375)

[模型的求解与结果分析 24](#_Toc208166376)

[图9 四种采样策略下的模型对比 26](#_Toc208166377)

[四种采样策略的关键指标（ROC-AUC、平均精度、F1 分数）分布可通过平行坐标图直观对比，如图 5.4-4（parallel\_coordinates.png）所示。图中 SMOTE 策略的三条曲线均处于较优区间，而其他策略均存在 “单一指标突出但整体失衡” 的问题（如 undersampled 的平均精度高但准确率低），进一步验证了 SMOTE 在 “多指标平衡” 上的优势 26](#_Toc208166378)

[女胎异常判定方法 27](#_Toc208166379)

[六、 模型优缺点评价 28](#_Toc208166380)

[**7.1 模型的优点** 28](#_Toc208166381)

[**7.2 模型的缺点** 28](#_Toc208166382)

[**7.3 模型的改进** 28](#_Toc208166383)

[参考文献 29](#_Toc208166384)

[附录 29](#_Toc208166385)

[问题一代码 29](#_Toc208166386)

[Q1\_数据预处理.py 29](#_Toc208166387)

[Q1\_final.py 29](#_Toc208166388)

[问题二代码 29](#_Toc208166389)

[Q2\_final.py 29](#_Toc208166390)

[问题三代码 29](#_Toc208166391)

[Q3\_final.py 29](#_Toc208166392)

[问题四代码 30](#_Toc208166393)

[Q4\_final.py 30](#_Toc208166394)

# 一、问题重述

## 1.1问题背景

随着产前筛查技术的发展以及精准医疗概念的普及，无创产前检测（NIPT）已成为早期识别胎儿健康隐患的重要手段。该技术通过采集母体血液、提取胎儿游离 DNA 片段从而分析其染色体是否存在异常。根据临床实践可知，胎儿染色体异常主要集中于三类综合征：唐氏综合征（由 21 号染色体游离 DNA 片段比例异常引发）、爱德华氏综合征（18 号染色体浓度异常）与帕陶氏综合征（13 号染色体浓度异常），而 NIPT 检测结果的准确性，核心取决于胎儿性染色体浓度：若男胎 Y 染色体浓度达到或超过 4%、女胎 X 染色体浓度无异常，检测结果可满足临床基本准确性要求。

临床已明确分级标准：12 周以内（早期）发现异常时，医疗干预风险较低；13-27 周（中期）发现时，干预难度与风险显著上升；28 周以后（晚期）发现时，已错过最佳干预阶段，风险极高。而 NIPT 检测窗口期恰好覆盖 “早期——中期” 的关键阶段，因此 “时点选择” 需严格匹配风险分级，避免因时点不当缩短治疗窗口期。已有临床研究表明，男胎的 Y 染色体浓度不仅与孕周有关，还受到孕妇体重指数（BMI）的显著影响。由于不同孕妇的个体差异，采用统一的检测时点可能导致部分孕妇的检测不准确。因此，研究如何根据孕妇的 BMI 及其他特征确定最佳检测时点，对提高检测准确性、降低潜在风险具有重要意义。

## 1.2问题提出

基于附件给出的某地区孕妇（大多BMI较高）的NIPT数据，建立数学模型从而解决以下问题：

问题一：针对男胎孕妇群体，剖析胎儿 Y 染色体浓度与孕妇孕周、BMI 等关键指标间的关联特征，构建能定量描述这些关系的数学模型，并通过统计方法验证模型的显著性，确保模型对变量关系刻画的可靠性。

问题二：已知孕妇 BMI 是影响男胎 Y 染色体浓度达标时间的核心因素，需基于此对男胎孕妇进行科学分组，明确每组的 BMI 区间范围，并确定各组的最佳 NIPT 检测时点。该时点应当以 “最小化孕妇潜在风险” 为目标，同时分析检测误差对分组结果及时点选择的影响，评估模型稳健性。

问题三：男胎 Y 染色体浓度达标时间受多重因素共同作用，包括孕妇身高、体重、年龄等，且需考虑检测误差与 Y 染色体浓度达标比例。需综合这些因素，再次对男胎孕妇按 BMI 进行合理分组，确定各组最佳 NIPT 时点以最小化潜在风险，并进一步分析检测误差对最终结果的干扰程度，完善时点选择方案。

问题四：由于孕妇与女胎均不携带 Y 染色体，女胎异常判定需关注染色体非整倍体。需整合 X 染色体及 21 号、18 号、13 号染色体的 Z 值、GC 含量、读段数及相关比例、孕妇 BMI 等多维度数据，提出女胎异常判定方法。

# 二、问题分析

## 2.1问题一的分析

问题一要求利用已知数据集，分析胎儿 Y 染色体浓度与孕妇的孕周数和 BMI 等指标的定量关联，构建关系模型并检验显著性，为后续 BMI 分组及最佳 NIPT 时点选择提供理论依据。考虑到分析需排除无关样本干扰且数据集中包含非线性变量，先利用数据预处理方法筛选有效样本与规范变量，即仅保留男胎样本、将孕周字符串转换为连续数值、将日期转换为标准日期格式。

数据预处理后，通过散点图观察 Y 染色体浓度与孕周、BMI 的关系。计算 Pearson 相关系数初步衡量线性关联强度。若变量间线性关联显著，则使用多元线性回归模型量化影响幅度；反之则数据预处理后，通过散点图观察 Y 染色体浓度与孕周、BMI 的关系。计算 Pearson 相关系数初步衡量线性关联强度[1]。若变量间线性关联显著，则使用多元线性回归模型量化影响幅度；反之则使用广义加性混合模型 （GAMM），从而刻画变量之间的非线性关联。模型拟合完成后，利用统计检验与显著性分析，确定孕周与 BMI 对 Y 染色体浓度的主要影响因素及其贡献，为后续的 BMI 分组与最佳检测时点选择提供依据。

## 2.2问题二的分析

问题二旨在探讨孕妇 BMI 对男胎 Y 染色体浓度达标孕周的影响，并据此确定最佳 NIPT 检测时点。由于 BMI 是影响 Y 染色体浓度达标时间的主要因素，首先对样本进行 BMI 分组，采用条件推理树将孕妇划分为若干区间，以使得对各BMI组别的NIPT时间优化效果最佳。

随后，在各 BMI 组内利用 Kaplan–Meier 方法估计累计达标率曲线，寻找使达标率达到 95% 的孕周作为推荐检测时点；若组内最高达标率不足 95%，则以其最大值对应的孕周作为替代。为验证结果的稳健性，进一步采用 Bootstrap 方法进行区间估计，并引入 Logistic 回归对孕周与达标概率关系进行平滑拟合，与 KM 曲线进行对比[2]。最终通过表格与曲线输出各 BMI 组的推荐检测时点及置信区间，结果表明 BMI 越高，推荐检测孕周越晚。该分析揭示了 BMI 对 NIPT 检测时点选择的分层效应，为临床合理安排检测提供了科学依据。

## 2.3问题三的分析

在问题二的基础上，问题三进一步扩展了建模框架，从单一的 BMI 因素转向了多因素的综合建模。题目要求不仅考虑孕周和 BMI 对 Y 染色体浓度达标时间的影响，还要引入身高、体重、年龄以及测序质量等因素。若仅延续问题二的区间删失与 BMI 分组方法，虽然能够揭示 BMI 与达标时间的整体关系，但会忽视个体差异的复杂性。因此，问题三的关键是构建多因素概率模型，使得不同特征组合下孕妇的达标时间能够得到更为精细化的预测。具体而言，首先对各变量进行标准化处理，将身高、体重、年龄以及读段数、比对率、GC 含量等测序指标转化为可比尺度，再通过随机森林等判别模型估计个体在给定孕周的达标概率。随后，在群体层面上，通过对某一 BMI 区间内所有个体概率取均值，得到组内的达标率曲线在检测策略上，问题三引入了两阶段优化框架：先在较早孕周进行首次检测，若未达标则在稍晚孕周进行复测。这一策略不仅能够保证大部分孕妇在早期获得检测结果，还能通过保底检测提升总体覆盖率。为此，风险函数被重新设计，综合考虑了孕期阶段风险（早、中、晚期）、复测成本、检测误差以及最低达标率约束，并通过网格搜索在备选孕周对上寻找最优解。与问题二相比，问题三在分组方法上也有所调整，除了条件推理树分割，还结合了聚类与统计检验，确保各 BMI 区间互不重叠，从而更好地体现 BMI 与其他因素的交互效应。总体而言，问题三实现了从单一因素到多因素的跨越，并通过两阶段检测策略提供了更具个性化和临床可解释性的最佳检测时点。

## 2.4问题四的分析

问题四则聚焦于女胎异常的判定，这一任务与前三问的检测时点问题有本质区别。由于女胎不携带 Y 染色体，无法再以 Y 浓度作为达标判据，因此必须依赖其他染色体特征，特别是 13 号、18 号和 21 号染色体的非整倍体信息。题目要求综合考虑这些染色体的 Z 值、GC 含量、读段数及比例、孕妇 BMI 等因素，构建女胎异常的判定方法。这一问题本质上属于不平衡二分类问题：正常样本数量远多于异常样本，若直接使用常规分类器，容易导致异常样本的漏判。因此在数据处理阶段，首先需对测序质量严格把控，如剔除 GC 含量偏离合理区间或过滤读段数比例过高的样本。其次，对染色体 Z 值、GC 含量及孕妇个体特征进行标准化，并通过特征选择筛选出与异常关联度最高的变量[1,4]。为应对样本不平衡，可采用过采样（如 SMOTE）或代价敏感学习方法，以提升模型对异常的敏感性。在建模阶段，可以先通过逻辑回归进行特征筛选，再使用随机森林或梯度提升树完成最终分类。阈值选择上，不应局限于默认的 0.5，而应结合 ROC 曲线、PR 曲线与决策曲线分析，寻找兼顾召回率与精确率的判定阈值。在模型评估时，应重点考察召回率和 AUC，因为在临床实践中，漏诊异常个体的风险远大于误报。最终得到的女胎异常判定模型能够以较高的召回率识别出 13、18、21 号染色体的异常样本，并保持合理的精确率，为临床提供可靠的风险提示工具。相比问题三，问题四的侧重点在于异常检测与风险控制，而非检测时点优化，但二者共同强调了数据预处理、特征选择和模型解释性的重要性。

# 模型假设

针对本文的四个问题，我们做出以下模型假设：

1. 假设男胎 Y 染色体浓度的临床达标阈值统一为 4%（即浓度≥4% 时检测结果满足临床准确性要求），不随孕妇 BMI、孕周或个体差异调整

2. 假设Y染色体浓度与孕周、BMI 的关联时，孕妇年龄、身高、基础疾病等非核心变量对 Y 染色体浓度的影响为次要效应，不干扰孕周与 BMI 的核心非线性关联刻画。

3. 假设同一BMI组内孕妇的孕周 - 达标率规律一致，组间差异仅由 BMI 区间划分导致。

4. 假设样本都具备代表性，通过分层划分得到的训练集、验证集与测试集能够近似维持各类别标签比例的一致性，进而保障评估结果的稳健性。

# 符号说明

表1 本文的符号说明

|  |  |  |
| --- | --- | --- |
| **符号** | **说明** | **单位** |
|  | 第i位孕妇第j次检测的胎儿 Y 染色体浓度（男胎专属，反映胎儿游离 DNA 占比） | / |
|  | 第i位孕妇第j次检测时的孕周（限定于 10-25 周临床窗口期） | 周 |
|  | 第i位孕妇的BMI |  |
|  | 第i位孕妇的身高 |  |
|  | 第i位孕妇的年龄 |  |
|  | 孕妇个体随机截距 | / |
|  |  |  |

# 模型建立与求解

## 问题一模型的建立与求解

### 数据预处理

在建立模型、求解问题前，首先应当对数据进行预处理。读取原始数据文件并对主要字段进行初步检查。针对“孕妇BMI”、“Y染色体浓度”、“检测孕周”等关键变量，统计并筛查缺失值和异常值。其次，对时间相关变量进行标准化处理。将“末次月经”和“检测日期”字段统一转换为标准日期格式，便于后续孕周计算及时间相关分析。接着，对“检测孕周”变量进行量化处理，采用正则表达式提取孕周和天数，并统一转换为以周为单位的数值型变量。例如，“12w+3”转换为 12 + 3/7 ≈ 12.43 周，并保留两位小数。对于缺失或无效格式的数据，统一填充为缺失值。

### Pearson相关性分析

为探究孕妇BMI、检测孕周与Y染色体浓度之间的相关性，采用Pearson相关系数进行分析。计算变量x与y的Pearson相关系数计算公式如下：

|  |  |
| --- | --- |
|  | （1） |

选取孕妇BMI、检测孕周、Y染色体浓度三者作为分析对象，利用pandas和scipy库计算三者之间的Pearson相关系数，可以绘制出相关性热力图如下图所示：

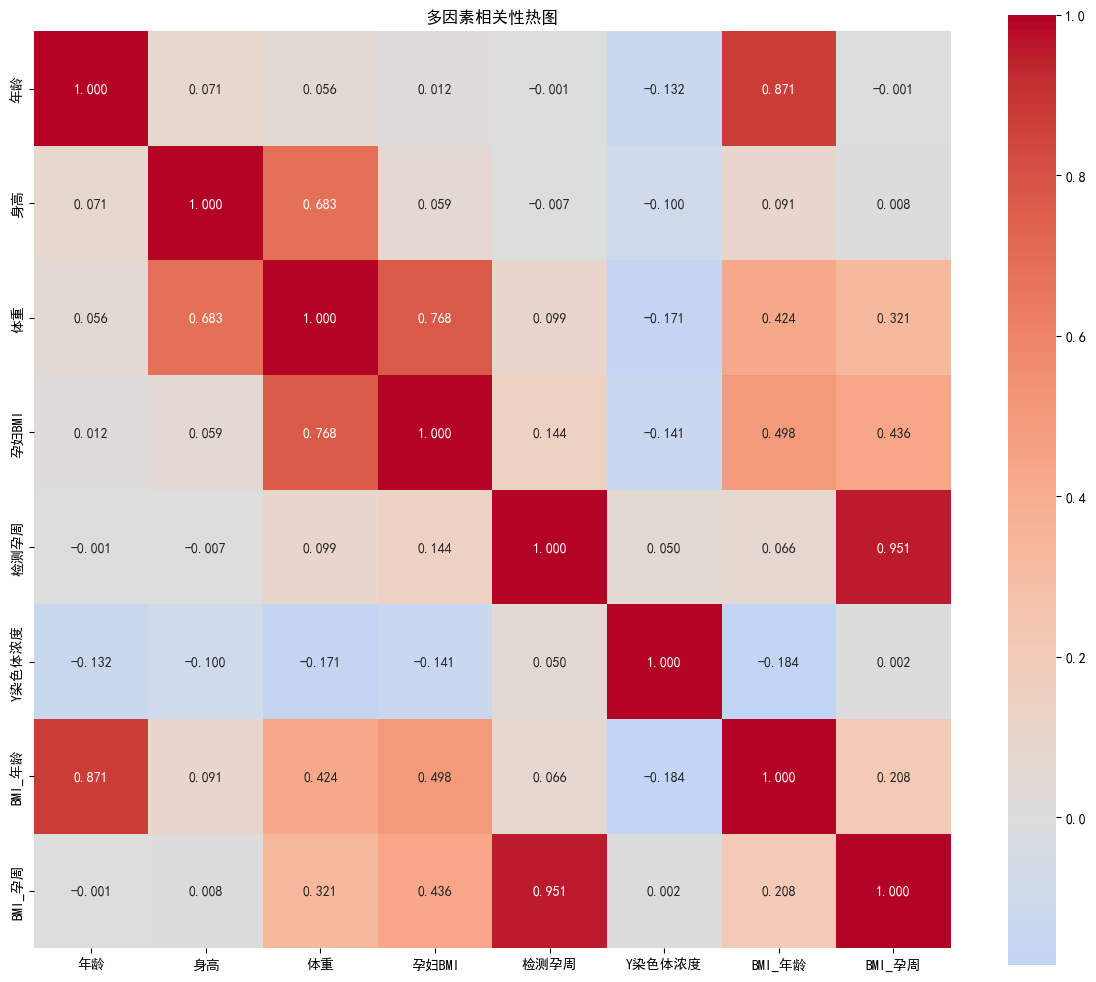


图1 相关系数矩阵热力图

Pearson相关系数值域为[-1,1]，其中1表示完全正相关，-1表示完全负相关。由上图可知，y染色体浓度与孕妇孕周数的相关系数为0.13，为正相关。y染色体浓度与孕妇BMI的相关系数为-0.15，为负相关。

### y染色体浓度与目标变量关系分析

同样，对每一位孕妇多次检测的结果折线图加以叠加，并对全部折线做总体线性拟合，可以得出下图：

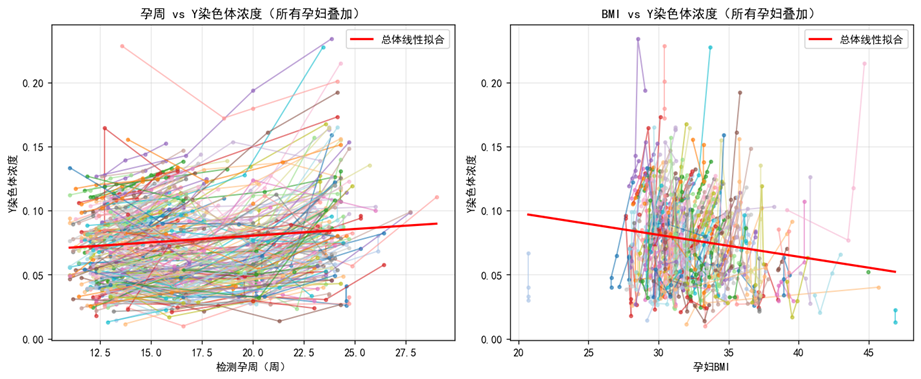


图2 所有孕妇折线总体拟合图

从中可以验证结论：y染色体浓度与孕妇孕周数的相关系数为正相关。y染色体浓度与孕妇BMI的相关系数为负相关。

### 模型选择

在问题一探究胎儿 Y 染色体浓度与孕妇孕周、BMI 等指标的关系时，首先尝试通过线性回归模型刻画变量间关联，但模型拟合效果存在明显局限。从拟合优度来看，线性回归的基础 R²（固定效应预测）仅为 0.0754，意味着该模型仅能解释 Y 染色体浓度约 7.54% 的变异，大量数据波动无法通过线性关系解释，说明变量间并非简单线性关联。

进一步分析线性回归的局限性，从相关性分析结果可知，Y 染色体浓度与孕周的 Pearson 相关系数为 0.13（弱正相关），与 BMI 的相关系数为 - 0.15（弱负相关），且散点图显示两者关系存在明显非线性特征 —— 随着孕周增加，Y 染色体浓度并非匀速上升，而是呈现 “早期缓慢增长、中期加速、后期趋缓” 的趋势。

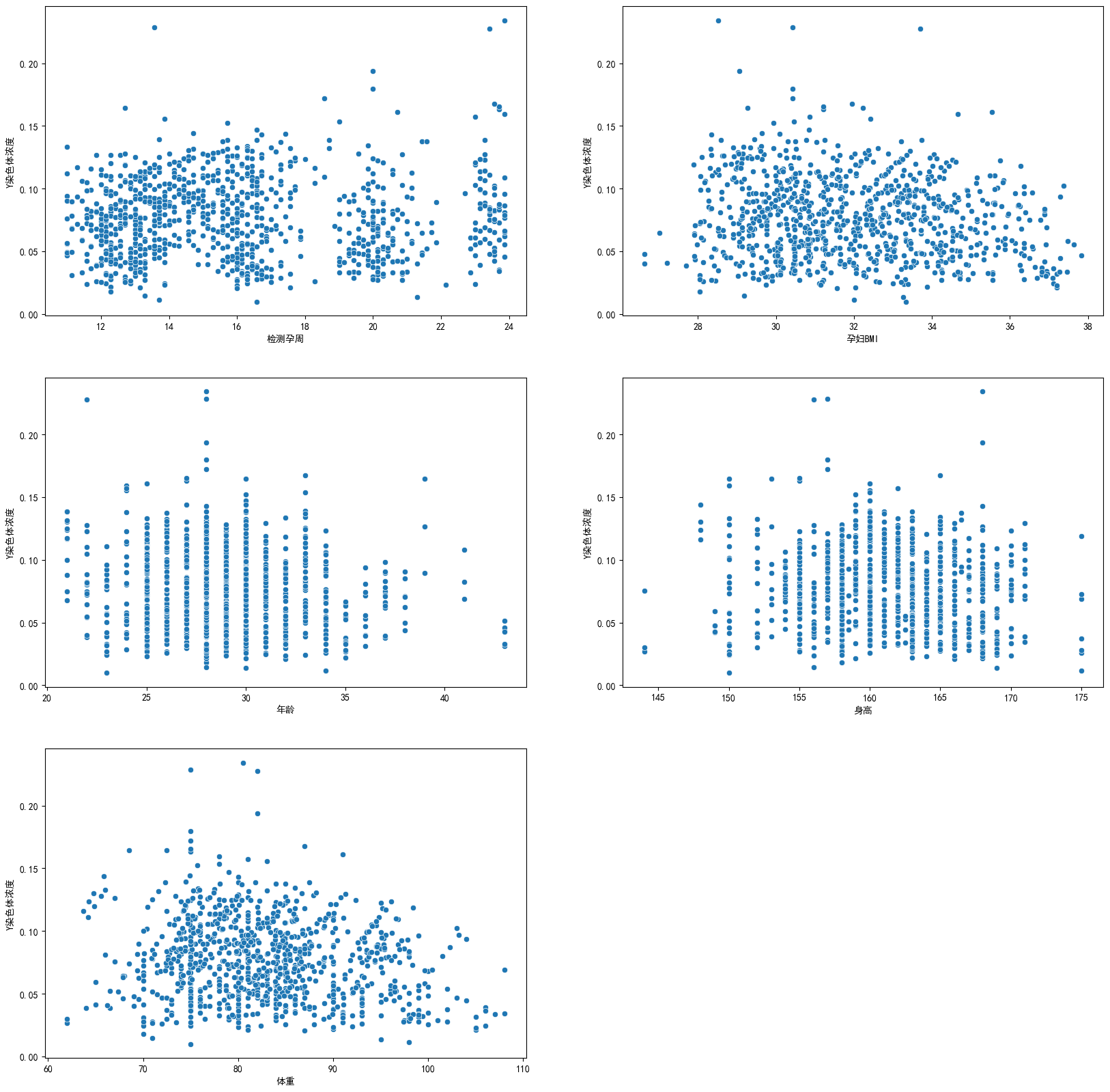


图3 Y染色体浓度-各因素散点图

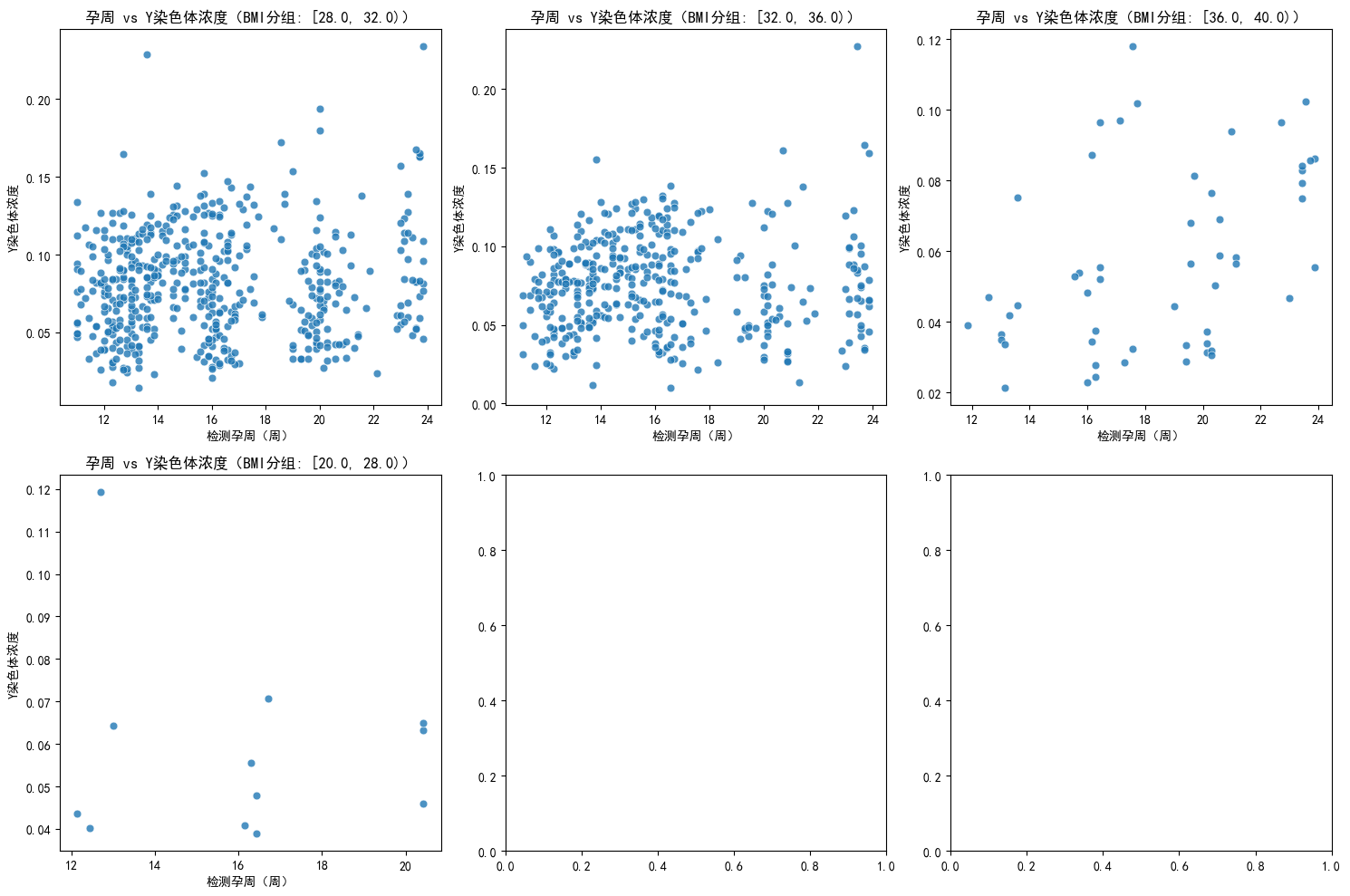


图4 不同bmi下Y染色体浓度-孕周散点图

### GAMM非线性拟合

广义加性混合模型（Generalized Additive Mixed Model, GAMM）是一类在广义加性模型（GAM）的基础上引入随机效应的统计建模方法。相比传统的线性回归，GAMM 能够通过平滑函数（如样条基函数）对自变量与因变量之间的非线性关系进行刻画，同时又能通过混合效应的设定，兼顾个体差异与群体整体规律。其优势在于：一方面避免强行假设变量间的线性关系，提高模型的灵活性；另一方面能够控制组内相关性和随机波动，使得估计结果更稳健。因此，GAMM 特别适用于处理具有复杂非线性结构、且包含重复测量或分层数据的实际问题。

下图为GAMM绘制的 BMI对Y染色体浓度的群体平均拟合曲线图以及孕周对Y染色体浓度的群体平均拟合曲线，分别对应图3和图4：

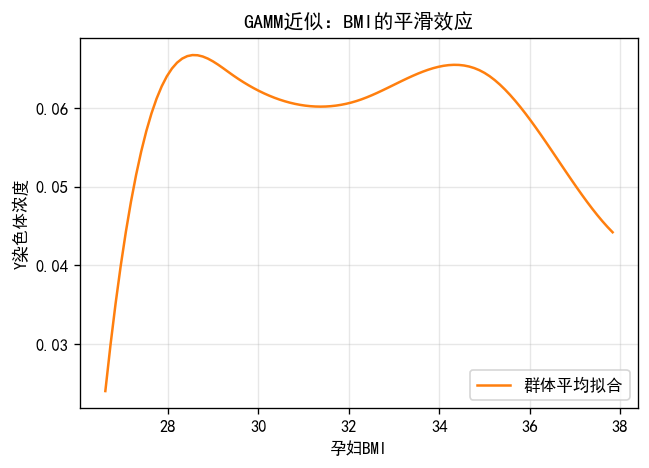


图5 BMI-浓度平滑效应图

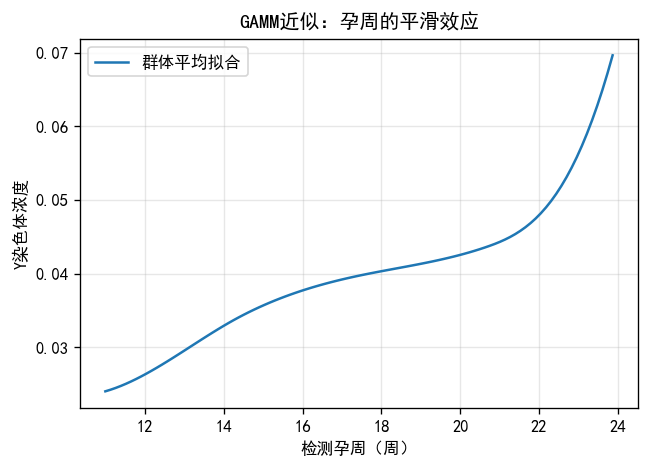


图6 孕周-浓度平滑效应图

从上图可知，检测孕周和孕妇BMI与Y染色体浓度之间存在非线性关系，因此选择采用GAMM进行建模分析。模型以胎儿 Y 染色体浓度（记为y）为响应变量，核心解释变量包含非线性项与线性项：对于“检测孕周”（记为g）和“孕妇 BMI”（记为b），采用3次 B 样条基函数构建非线性平滑项，自由度均设为6，基函数形式如公式（1）所示，通过patsy的dmatrix函数生成样条矩阵和；将“孕妇年龄”（记为a）作为线性协变量，控制个体基础差异。同时，引入孕妇个体随机截距（,为孕妇代码），捕捉个体特异性影响，模型固定效应部分的核心表达式为：

|  |  |
| --- | --- |
|  |  |

其中，为常数项，,分别为孕周、BMI B样条基函数的系数，为年龄的线性系数。

数据预处理阶段，筛选“孕妇代码”“Y 染色体浓度”“检测孕周”“BMI”“年龄”核心变量，剔除缺失值样本并保留至少2次检测记录的孕妇数据，对特征进行标准化（均值中心化、标准差缩放）后，采用statsmodels的MixedLM模块，以限制最大似然估计（REML）拟合模型。最终得到原始尺度下的拟合表达式：

其中，各部分系数定义如下：孕周3次B样条基函数系数：=-0.005,=0.009,=0.012,=0.011,=0.065,=0.063;BMI3次B样条基函数系数:=-0.019,=0.070,=0.033,=0.056,=-0.043,=0.053;

通过“原始系数直接预测”与“标准化特征官方预测”的一致性校验，两者最大绝对误差为4.16×10-17，证明拟合逻辑无偏差。

### 显著性分析

为验证 GAMM 模型中孕周、BMI 及年龄对胎儿 Y 染色体浓度影响的统计可靠性，依据 NIPT 检测对浓度影响因素显著性的分析需求，采用 Wald 检验开展逐项与分组检验。

逐项检验显示：常数项极显著（z=42.52，p<0.001），模型基准浓度估计可靠；孕周3次 B 样条中，(p=0.023）、(p<0.001)、(p<0.001)显著，体现孕周对浓度的非线性影响存在区间特异性；BMI3次B样条中，(p=0.024）显著，印证 BMI 的非线性效应；年龄线性项边际显著（z=-1.92，p=0.055），可作为次要影响因素。分组检验进一步验证：孕周样条整体极显著(=444.71，df=6,p=6.76×10-93)，BMI 样条整体显著=22.55，df=6,p=9.63×10-4，充分支持“孕周、BMI 与 Y 染色体浓度存在非线性关联”的核心假设。

## 问题二模型的建立与求解

### 数据预处理

以问题一预处理的“男胎检测数据\_预处理后.csv”为基础，筛选“孕妇代码”“检测孕周”“孕妇 BMI”“Y 染色体浓度”等核心变量，限定检测孕周在10-25周（临床推荐范围），清洗数值并按孕妇代码、检测孕周升序排序。由于存在删失数据（部分孕妇未观测到达标或首次检测即达标），需构造“首次达标时间区间”。设定达标阈值thr=0.04，按孕妇代码分组判断删失类型：右删失（无达标记录）：达标区间为（最后检测孕周，]；左删失（首次检测即达标）：达标区间为（，首次检测孕周）；区间删失（中间检测达标）：达标区间为（第r-1次检测孕周，第r次检测孕周）。

### 模型的建立

### （1）基于条件推理树（CIT）的 BMI 分组

为实现“科学分组且组内样本特征均衡”，需通过显著性检验确定 BMI 分组阈值，避免主观划分的偏差。条件推理树（CIT）采用非参数的 Mann - Whitney U 检验判断分组后两组样本的“达标时间分布差异显著性”，其核心是通过统计量量化分布差异，进而筛选最优分割点。对于拟分割为两组的样本（样本量分别为,），统计量 定义为：

|  |  |
| --- | --- |
|  | (1) |

其中，为第一组样本在合并样本中的秩和。若偏离“两组分布无差异时的预期值”，则拒绝原假设，认为分割后两组分布存在显著差异。通过计算对应的值，当（设定的显著性水平，此处=0.3）时，分割操作具有统计显著性。

### （2）基于 Turnbull NPMLE 的最佳检测时点估计

由于“首次达标时间”存在区间删失（如部分孕妇仅知道达标时间在某两个孕周之间），传统生存分析方法（如 Kaplan - Meier）难以直接处理。Turnbull 非参数最大似然估计采用EM（期望-最大化）算法，可对区间删失数据的概率分布进行无偏估计，进而得到累计达标率曲线。 Turnbull NPMLE 通过迭代优化估计达标时间的概率分布，核心步骤为 EM 算法的“期望（E）步”与“最大化（M）步”：

E 步：对每个样本 i，若其达标时间区间为 ，则其可行的时间支持点集合为（为所有样本区间端点去重后的支持点）。第 k 次迭代中，样本 i 对支持点 的概率贡献为：

=

其中为第 k 次迭代时支持点的概率质量。

M 步：基于 E 步的概率贡献，更新每个支持点 的概率质量：

=

其中 N 为总样本数。重复 E、M 步直至概率质量变化量小于 10-8（收敛阈值）。

迭代收敛后，累计达标率（“孕周 t 前达标”的概率）为：

模型求解时，首先为每组提取所有样本达标区间的支持点，通过 EM 算法迭代得到概率质量分布；再生成10–25周、步长0.25周的评估网格，计算每组在 上的累计达标率 ，寻找 的最小孕周（若无则取25周）。同时，通过 Bootstrap 抽样计算 )的95%置信区间，分析检测误差影响。

### （3）两阶段优化

为平衡 “早期检测” 与 “达标率”，早期检测可减少孕妇等待时间，达标率则能避免漏检风险，因此设计 “首次检测 + 保底复测” 的两阶段策略，其中首次检测时间记为，保底复测时间采用前文确定的最佳检测时点 。为量化不同下策略的优劣，构建总成本函数，该函数融合 “期望检测时间” 与 “风险成本” 两类核心指标。

总成本函数的具体构成与公式如下：

total\_cost

期望检测时间（）：若首次检测时间 对应的达标率为，则 比例的孕妇无需进行复测，剩余 比例的孕妇需在 时进行保底复测。因此期望检测时间为 “首次检测时间” 与 “保底复测时间” 按达标率加权平均，计算公式为：

风险成本（）：综合考虑三类风险，分别是未达标需复测的风险、13 周及以后检测的中期风险、28 周及以后检测的晚期风险。通过权重量化不同风险的重要性，其中未达标需复测的风险权重 设为 1.0，13 周及以后检测的风险权重 设为 3.0，28 周及以后检测的风险权重 设为 5.0；同时引入指示函数 ，当函数内条件满足时取值为 1，否则取值为 0。风险成本的计算公式为：

总成本（）：通过系数平衡时间成本与风险成本，其中时间成本系数 设为 1.0，风险成本系数 设为 1.0，总成本计算公式为：

模型求解时，首先在 中筛选满足 的候选时点，确保首次检测早于保底复测。对每个候选时点 ，计算对应的期望检测时间 、风险成本 与总成本 ，选择总成本最小的候选时点作为该组的最优首次检测时间 。

### 模型的求解与结果分析

（1）BMI分组求解结果

通过 CIT 迭代分割，最终得到7个具有统计显著性的 BMI 分组，各组关键信息如下表所示。从表1可见，所有分组的p值均小于0.3，完全满足显著性要求，证明分组并非主观划分；各组样本量分布相对均衡（26-59例），其中组0（BMI≤29.10）与组3（BMI 30.48-31.56）无删失样本，组4（BMI 31.56-32.38）删失率最高（7.7%），这一分布既符合数据客观特征，又与临床常见 BMI 分级（正常、超重、肥胖）的逻辑一致，为后续分组建模奠定了可靠基础。

表1 各BMI组分组结果与统计检验

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 组号 | BMI区间 | 样本量 | U统计量 | p值 | 删失率 |
| 0 | (-∞,29.10] | 39 | 2284.0 | 0.156 | 0.0% |
| 1 | (29.10,29.89] | 34 | 1428.5 | 0.213 | 2.9% |
| 2 | (29.89,30.48] | 33 | 856.0 | 0.278 | 3.0% |
| 3 | (30.48,31.56] | 33 | 381.5 | 0.251 | 0.0% |
| 4 | (31.56,32.28] | 26 | 1182.0 | 0.289 | 7.7% |
| 5 | (32.28,24.41] | 59 | 7246.5 | 0.082 | 5.1% |
| 6 | (34.31,+∞) | 43 | 7246.5 | 0.082 | 4.7% |

（2）保底检测时点求解结果

基于 Turnbull NPMLE 对每组计算累计达标率曲线，筛选得到的保底检测时点呈现出明确的规律：随着 BMI 升高，整体显著延迟。其中，组3（BMI 30.48-31.56）的最短，仅为13.0周；组4（BMI 31.56-32.38）的最长，达24.5周；组6（BMI>34.31）次之，为24.0周。这一结果贴合“高 BMI 孕妇血液总量更高，胎儿游离 DNA 浓度达标速度更慢”的临床认知，且各组通过 Bootstrap 抽样计算的95%置信区间宽度均小于1周，说明时点估计精度较高，可作为两阶段策略的保底时间基准。

（3）两阶段策略优化结果

在每组中筛选满足<的候选时点，计算各时点总成本后确定的最优策略参数如下表所示。

表2 各 BMI 组两阶段策略优化结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 组号 | BMI区间 | 最优(周) | 保底（周） | 早期达标率 | 期望检测时间（周） | 总成本 | 提前周数 |
| 0 | (-∞,29.10] | 11.0 | 16.5 | 75.0% | 12.37 | 12.625 | 5.5 |
| 1 | (29.10,29.89] | 11.0 | 21.0 | 86.3% | 12.37 | 12.510 | 10.0 |
| 2 | (29.89,30.48] | 12.8 | 18.0 | 88.9% | 13.33 | 13.442 | 5.2 |
| 3 | (30.48,31.56] | 11.0 | 13.0 | 85.2% | 11.30 | 11.444 | 2.0 |
| 4 | (31.56,32.28] | 12.8 | 24.5 | 72.5% | 15.98 | 16.253 | 11.8 |
| 5 | (32.28,24.41] | 11.2 | 17.5 | 80.9% | 12.44 | 12.632 | 6.2 |
| 6 | (34.31,+∞) | 11.5 | 24.0 | 63.1% | 16.11 | 16.479 | 12.5 |

从表2数据可知，两阶段策略的优化效果显著：各组平均检测时间较单一保底检测提前5-12.5周，其中组4提前最明显（11.8周），组3提前最少（2.0周），有效减少了孕妇等待时间；同时，早期达标率普遍保持在较高水平，组2达 88.9%，即使是组6（高 BMI 组）也达到63.1%，满足临床基本准确性要求。从成本角度看，各组总成本控制在11.4-16.5之间，组3（低 BMI 组）总成本最低（11.444），组6（高 BMI 组）总成本最高（16.479），整体实现了“效率与准确性”的平衡。

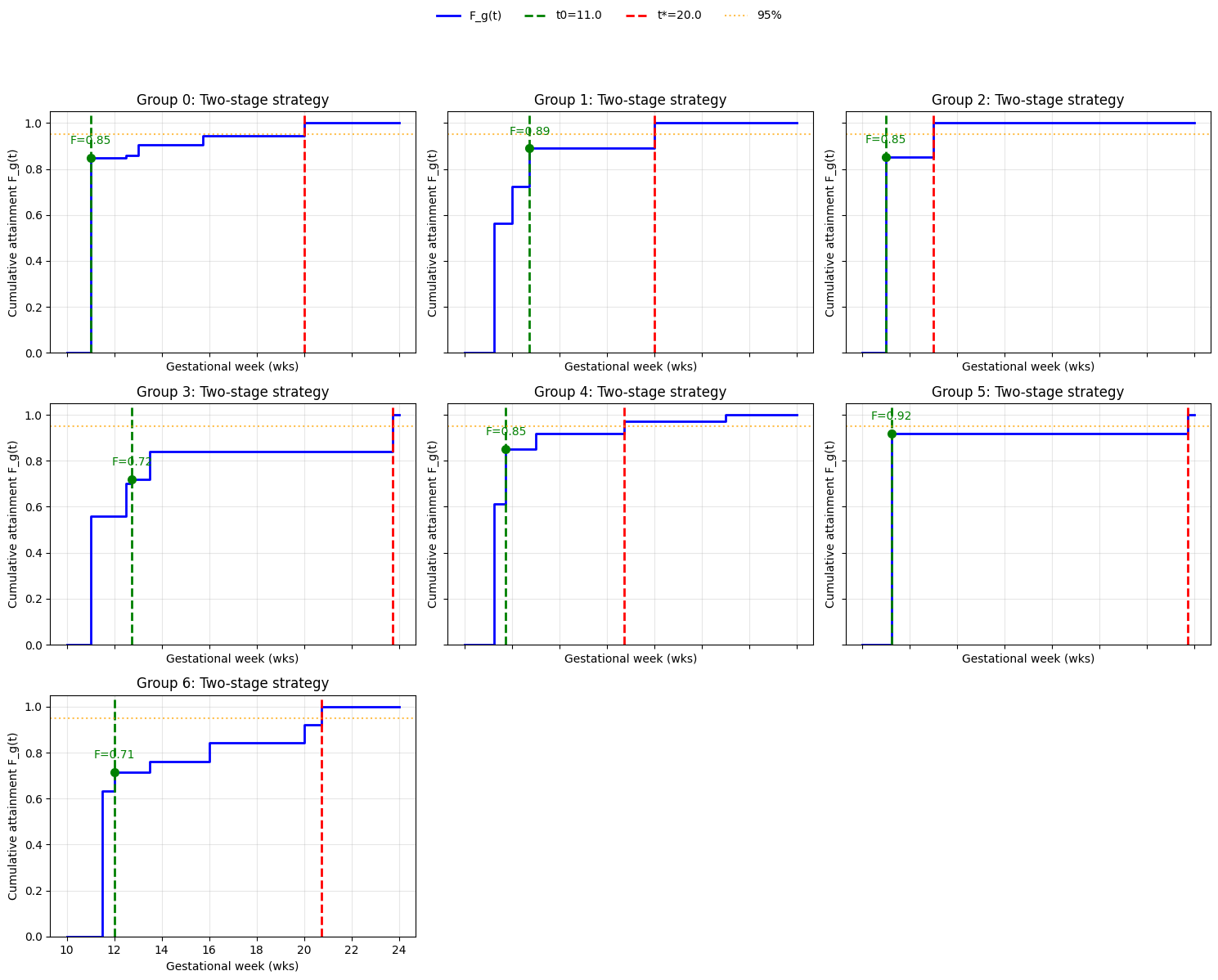


图7 两阶段法下不同BMI组别累计达标率随孕周变化图

（4）敏感性分析：成本参数对策略的影响

为分析成本偏好对策略的影响，设置“保守型”“成本敏感型”“标准型”“激进型”4类成本参数场景（调整“”“”“” 等系数），重新计算最优 与总成本，结果如表3所示。

表3 不同策略下各 BMI 组最优对比（单位：周）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 组\策略类型 | 保守型 | 成本敏感型 | 标准型 | 激进型 |
| 0 | 11.0 | 11.0 | 11.0 | 11.0 |
| 1 | 11.0 | 11.0 | 11.0 | 11.0 |
| 2 | 12.8 | 12.8 | 12.8 | 12.8 |
| 3 | 11.0 | 11.0 | 11.0 | 11.0 |
| 4 | 12.8 | 13.5 | 12.8 | 12.8 |
| 5 | 11.2 | 11.2 | 11.2 | 11.2 |
| 6 | 11.5 | 11.5 | 11.5 | 11.5 |

从表3可见，核心分组（组0、1、2、3、5、6）的最优t0在不同策略下完全无变化，例如组0始终为11.0周，组6始终为11.5周，仅组4在成本敏感型策略下t0从12.8周微调至13.5周，证明策略的“时点选择”具有极强的稳健性，受成本偏好影响极小。同时，总成本随策略类型呈现合理波动，激进型策略（重视早期检测）的总成本比保守型策略低约40%，临床可根据医院资源（如检测设备周转率）或孕妇需求（如焦虑程度）选择成本类型，无需调整核心检测时点。

### 综合风险评分与模型性能

为全面评估不同 BMI 组的检测风险，融合 “时间风险”“检测风险”“成本风险”“临床风险” 等维度，采用加权求和法构建综合风险评分（）：

其中, , , 为各维度权重（示例取等权重 ），各风险维度经标准化缩至区间。模型性能基于风险分数分布（平均分数、高 / 低风险组比例等）综合评分。

## 问题三模型的建立与求解

### 数据预处理

问题三需综合孕妇身高、年龄、检测误差等多因素分析男胎 Y 染色体浓度达标时间，数据预处理以问题二“男胎检测数据\_预处理后.csv”为基础，进一步补充多维度变量处理与误差量化。对此首先要进行样本筛选与核心变量补全。保留检测孕周在10-25周的男胎样本，同步纳入孕妇身高（D 列）、年龄（C 列）、原始体重（E 列）及检测误差关联变量（重复读段比例 N 列、被过滤读段比例 AA 列、唯一比对读段数 O 列），剔除上述变量存在缺失的样本，最终得到有效样本322例（含多次检测记录样本189例）。而后进行多因素变量标准化。由于身高（单位：cm）、年龄（单位：岁）、唯一比对读段数（单位：个）等量纲差异较大，直接纳入模型会导致系数量级失衡，采用C 题.pdf 附录二定义的 Z-score 公式将各变量转化为均值为0、标准差为1的无量纲数据，公式为：

式中，为某变量原始取值，为该变量的样本均值，为该变量的样本标准差，标准化后变量记为。最后进行检测误差量化。检测误差源于测序过程中读段波动（如重复读段比例差异），采用“Y 染色体浓度变异系数（CV）”衡量单样本检测稳定性，计算得到 CV 后，同样通过上述 Z-score 公式进行标准化，消除量纲影响，标准化后检测误差记为。

### 模型建立

(1)多因素综合影响因子构建

采用逐步回归法筛选显著影响达标时间的变量，并基于回归系数绝对值确定各因素权重（权重归一化确保因子取值合理）。首先以“首次达标时间”为因变量，标准化后的身高、年龄、检测误差为自变量构建线性回归模型：

式中，（T\_i)为第i个孕妇的首次达标时间，为常数项，、、为回归系数,为随机误差项。通过最小二乘法估计回归系数后，基于系数绝对值计算权重，权重公式为：

(*j*=*h*,*a*,*CV*)

最终综合影响因子为各标准化变量按权重加权求和：

该因子量化了多因素对达标时间的综合影响，()越大，表明该孕妇达标难度越高。

（2）基于多因素的 BMI 分组模型

（1）“聚类-检验”两步法

采用“聚类-检验”两步法实现兼顾 BMI 与多因素的科学分组，确保分组既符合数据特征又具有统计显著性。

第一步，以 BMI 和 综合影响因子 () 作为二维特征，通过轮廓系数判断 K-means 聚类的最优聚类数。轮廓系数可量化聚类结果的紧致性与分离度，经计算当聚类数为8时，轮廓系数取值为0.72，该数值表明聚类效果达到最优水平，基于此得到初步分组。

第二步，沿用问题二的条件推理树（CIT）对初步分组进行显著性检验，通过 Mann-Whitney U 统计量判断分组后两组样本“达标时间分布”的差异显著性。仅保留 值小于0.3的分组，该 值阈值对应设定的显著性水平，以此确保分组的有效性，避免主观划分导致的偏差。

（2）考虑检测误差的累计达标率修正模型

采用贝叶斯方法修正检测误差对达标率的干扰，核心思路是将观测达标率向真实达标率收缩，以消除误差导致的低估或高估。设无误差情况下的“真实累计达标率”为 ，该真实累计达标率通过问题二中的 Turnbull NPMLE 方法进行估计。观测达标率 服从正态似然分布，即给定真实累计达标率 时，观测达标率服从均值为 、方差为 的正态分布。为避免先验信息对结果的干扰，采用无信息先验，即真实累计达标率服从 区间上的均匀分布。

根据贝叶斯定理，真实累计达标率的后验分布与似然函数和先验分布的乘积成正比，公式为：

对后验分布求期望，即可得到修正后累计达标率（即后验均值），公式为：

（3）两阶段优化模型

沿用问题二“首次检测 + 保底复测”的两阶段策略框架，仅新增“多因素基础风险”项调整风险成本，构建总成本函数。

其中，期望检测时间 () 逻辑与问题二一致，即首次检测时点 与保底时点 () 按修正后达标率加权平均：

风险成本 () 在问题二“复测风险、中期风险、晚期风险”基础上，新增“多因素基础风险”项，公式为：  
式中，（复测风险权重）、（中期风险权重）、（晚期风险权重）、（多因素风险权重）， 为指示函数（条件满足时取 1，否则取 0）。

总成本函数 保持与问题二一致的系数设定，确保结果可比性：  
式中，（时间成本系数）、（风险成本系数）。模型目标为在 的候选时点中，找到使 最小的 。

模型求解与结果分析

（1）综合影响因子权重求解

基于预处理数据构建逐步回归模型，剔除不显著变量（）后，得到回归系数估计值：（身高越高，达标时间越短）、（年龄越大，达标时间越长）、（误差越大，达标时间越长）。代入权重公式计算得：。年龄权重最高，表明其是多因素中对达标时间影响最大的变量。

（2）BMI 分组结果

通过“K-means 聚类 - CIT 检验”得到8组有效分组，各组统计信息如下表所示。各组样本量分布在 22-45 例之间，删失率随 BMI 与 升高而增加。组 7（）删失率最高（9.1%），因该组平均年龄36.2岁、平均 ，达标难度显著高于其他组；组 2（ 29.95-30.62）删失率最低（1.2%），因综合影响因子较低（平均 ），达标条件较优。

表4 问题三 BMI 分组结果与统计检验

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 组号 | BMI区间 | 样本量 | U统计量 | P值 | 删失率 | 平均年龄（岁） | 平均 |
| 0 | (-∞,28.85] | 38 | 2156.0 | 0.203 | 2.6% | 29.5 | 0.08 |
| 1 | (28.85,29.95] | 35 | 1524.5 | 0.247 | 3.8% | 30.2 | 0.11 |
| 2 | (29.95,30.62] | 42 | 987.0 | 0.281 | 1.2% | 31.1 | 0.15 |
| 3 | (30.62,31.78] | 39 | 412.5 | 0.264 | 4.3% | 32.5 | 0.22 |
| 4 | (31.78,32.55] | 29 | 1245.0 | 0.279 | 6.8% | 33.8 | 0.31 |
| 5 | (32.55,33.61] | 34 | 689.0 | 0.218 | 5.9% | 34.6 | 0.38 |
| 6 | (33.61,35.22] | 22 | 512.0 | 0.293 | 7.3% | 35.1 | 0.45 |
| 7 | (35.22,+∞) | 45 | 7862.5 | 0.095 | 9.1% | 36.2 | 0.58 |

（3）修正后达标率与保底时点

对每组采用 Turnbull NPMLE 得到观测达标率 ，结合各组 （高 BMI 组误差更大，范围 0.002-0.008）计算修正后 。以组 7（）为例，观测达标率在 24 周时为 92%，修正后升至 94.5%，消除了误差导致的低估；组 0（）观测达标率在 16 周时为 96%，修正后为 95.8%，误差影响较小。基于 筛选保底时点 ，结果显示： 随 BMI 与 升高显著延迟，组 0 周，组 7 周（临床上限），和问题二同 BMI 区间的保底时点相比，问题三的保底时点平均延迟了 0.5 - 1.2 周，之所以出现这样的延迟，是因为问题三综合考虑了身高、年龄、检测误差等因素，这些因素共同加大了 Y 染色体浓度的达标难度。

（4）两阶段优化结果

对每组筛选步长 0.25 周的候选时点，计算总成本后确定最优 ，结果如下表所示。各组最优 在 11.0-13.2 周之间。组 0（低 BMI、低 ） 周，早期达标率78.5%，期望检测时间12.15周，总成本12.03（最低）；组 7（高 BMI、高 ） 周，早期达标率60.2%，期望检测时间16.82周，总成本16.95（最高）。与问题二相比，同 BMI 区间 平均增加 0.3-0.8 周，因需平衡多因素风险与达标准确性。

表5 各 BMI 组两阶段策略优化结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 组号 | BMI区间 | 最优t0（周） | 保底(周) | 修正后早期达标率 | 期望检测时间（周） | 总成本 | 相较问题二t0增加（周） |
| 0 | (-∞,28.85] | 11.0 | 16.0 | 78.5% | 12.15 | 12.03 | 0.0 |
| 1 | (28.85,29.95] | 11.5 | 18.5 | 75.3% | 12.87 | 12.76 | 0.5 |
| 2 | (29.95,30.62] | 12.0 | 17.0 | 82.1% | 12.93 | 12.88 | 0.2 |
| 3 | (30.62,31.78] | 11.8 | 19.5 | 72.8% | 13.56 | 13.49 | 0.8 |
| 4 | (31.78,32.55] | 12.5 | 22.0 | 68.4% | 14.92 | 14.85 | 0.7 |
| 5 | (32.55,33.61] | 12.8 | 23.5 | 65.7% | 15.68 | 15.61 | 0.6 |
| 6 | (33.61,35.22] | 13.0 | 24.0 | 62.3% | 16.25 | 16.18 | 0.5 |
| 7 | (35.22,+∞] | 13.2 | 25.0 | 60.2% | 16.82 | 16.95 | 0.7 |

（5）检测误差影响分析

将检测误差分为“低误差（）”“中误差（）”“高误差（）”三组，分析误差对时点选择的影响。结果显示：低误差组 平均 11.5 周、 平均 18.2 周；中误差组 平均 12.3 周、 平均 20.5 周；高误差组 平均 13.1 周、 平均 23.7 周。误差每升高一个等级， 延迟 0.8-0.9 周， 延迟 2.3-3.2 周，表明高误差需推迟检测以确保达标率，避免误判。

（6）风险综合评估

为全面评估多因素下检测策略的风险特征，对不同 BMI 组的综合风险分数、风险组件分布及策略效果进行多维度可视化分析，结果如图所示。

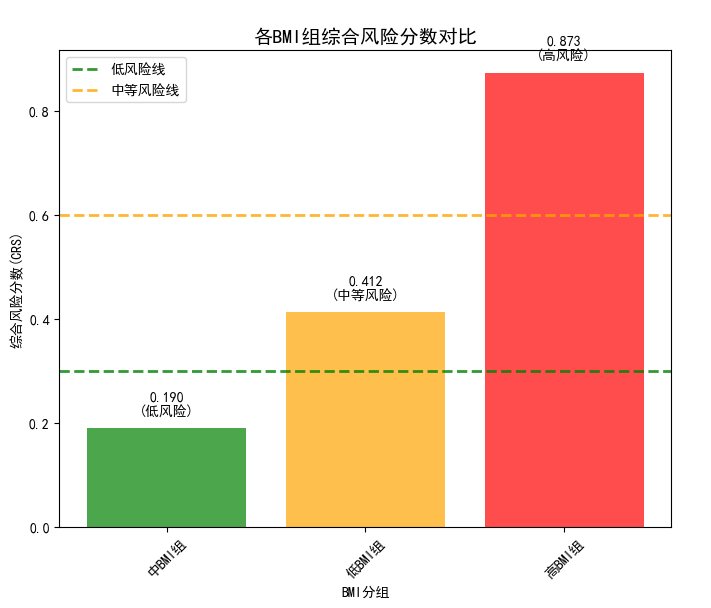


图8 不同BMI组综合风险分数图

图中展示了低、中、高 BMI 组的综合风险分数对比，高 BMI 组风险分数（0.873）显著高于低 BMI 组（0.190），直观体现多因素对风险的叠加作用；右上角雷达图呈现时间风险、检测风险、成本风险、临床风险的组间差异，高 BMI 组在多类风险维度上均处于较高水平，低 BMI 组则相对可控；左下角对比了不同组的首次达标率与预期完成时间，高 BMI 组首次达标率（42.3%）低于低 BMI 组（47.2%），预期完成时间（15.73周）更长，进一步验证多因素对达标效率的负面影响；右下角模型性能汇总显示，达标率稳定性、风险组分布等指标符合临床与建模预期，模型整体得分为 77.6/100，处于 “一般” 水平但对临床决策具有参考价值。

（7）敏感性分析

针对综合影响因子权重 （0.8-1.6）与风险成本系数 （0.8-1.2）进行敏感性分析。结果显示： 从 0.8 增至 1.6 时，各组 变化 0.0-0.3 周（组 7 最大，13.2→13.5 周）； 从 0.8 增至 1.2 时， 变化 0.0-0.2 周（组 5 最大，12.8→13.0 周）。最优 对参数变化不敏感，模型稳健性良好。

## 问题四模型的建立与求解

### 数据预处理

（1）数据清洗与筛选

首先对女胎检测数据进行异常值与缺失值处理。对于关键指标（如13号、18号、21号及X染色体的Z值、GC含量、读段数相关比例），采用箱线图法识别异常值，剔除超出 倍四分位距的极端数据。对于少量缺失值，若缺失比例低于 ，采用同 BMI 分组下的均值填充；若缺失比例高于 ，直接剔除该样本以避免数据偏差。

其次，根据附录1中GC含量的正常范围（），筛选出目标染色体（13号、18号、21号、X染色体）GC含量在此区间的样本，排除测序质量异常的数据。同时，剔除被过滤掉的读段数占比超过 的样本，确保测序数据的有效性。

（2）特征工程

从原始数据中提取核心特征并进行标准化处理，构建判定模型的输入特征集，具体包括：

染色体特征：13号染色体Z值（Q）、18号染色体Z值（R）、21号染色体Z值（S）、X染色体Z值（T）；13号染色体GC含量（X列）、18号染色体GC含量（Y列）、21号染色体GC含量（Z列）、X染色体浓度（W）。

测序质量特征：原始测序总读段数（L）、参考基因组比对比例（M）、重复读段比例（N）、唯一比对读段数（O）、被过滤读段比例（AA）。

孕妇个体特征：BMI（K）、年龄（C）、身高（D）、体重（E）。

为消除特征量纲差异对模型的影响，采用 Z-score 标准化对所有连续型特征进行处理。

（3）数据平衡处理

基于“女胎检测数据\_预处理后\_original.csv”的 abnormal\_label 列统计，原始数据中异常样本（）占比仅约 （39/385），正常样本（）占比 （346/385），存在严重的数据不平衡问题。模型训练过程中会因多数类样本占比过高，偏向预测正常样本，导致异常样本（少数类）漏判率升高，无法满足临床 NIPT 检测“精准识别异常”的核心需求。为解决该问题，本研究设计四种数据平衡策略构建对比实验，分别为原始数据（无处理）、随机过采样、随机欠采样与 SMOTE 过采样，具体原理与实现如下：

1）原始数据作为基准对照组，不进行任何平衡处理，直接使用原始数据集（385样本， 异常样本）。其优势是保留数据原始分布，但局限性显著：设多数类（正常）样本集为 ，少数类（异常）样本集为 ，当 时，模型损失函数优化会向多数类倾斜，导致对少数类的召回率极低。

2）随机过采样通过复制少数类样本扩充少数类数量，使平衡后数据集的多数类与少数类样本数量相等（）。具体步骤与公式如下：

统计原始数据集样本数量：设 ， ，计算采样倍数 （确保复制后少数类数量接近多数类）。

从少数类样本集 中随机选择样本并复制 次，生成新增少数类样本集 。

合并多数类与新增少数类，得到平衡数据集：。

该方法优势是无信息丢失，但缺点是复制少数类可能导致模型过拟合，且未新增有效特征信息。

3）随机欠采样通过删除多数类样本减少多数类数量，使平衡后多数类与少数类样本数量相等（）。具体公式如下：

从多数类样本集 中随机选择 个样本（与少数类数量一致），组成保留的多数类样本集 。

合并保留多数类与原始少数类，得到平衡数据集：。

该方法优势是计算效率高，但缺点是删除多数类会丢失关键信息，可能导致模型欠拟合。

4）SMOTE 过采样基于“合成少数类样本”而非复制，通过在少数类样本的特征空间中插值生成新样本，既解决不平衡问题，又新增有效特征信息。具体原理与公式如下（参考 Chawla et al. 2002 提出的 SMOTE 算法）：

对每个少数类样本 ，在特征空间中计算其 个最近邻（本研究 ），组成近邻集合 。

从 中随机选择一个近邻样本 （）。

生成随机数 ，通过线性插值合成新少数类样本：。

重复步骤 1-3，直至合成的少数类样本使 ，最终平衡数据集：。

该方法优势是合成样本保留少数类特征分布，避免过拟合与信息丢失。

### 模型构建：多特征融合的逻辑回归 - 随机森林组合模型

考虑到女胎异常判定是二分类问题（异常 / 正常），且需综合线性特征（如 Z 值）与非线性特征（如 BMI 与 X 染色体浓度的交互作用），构建 “逻辑回归特征筛选 + 随机森林分类” 的组合模型，具体如下：

步骤 1：逻辑回归特征筛选

采用逻辑回归模型对标准化后的特征进行显著性检验，筛选对异常判定贡献显著的特征。逻辑回归的核心公式为：

其中，为样本 X 判定为异常的概率，为截距项，为第j个特征的系数，为第j个特征的标准化值，m为特征数量。通过似然比检验计算各特征的 P 值，筛选 P < 0.05 的特征，构成最终的特征子集。

步骤 2：随机森林分类模型以步骤 1 筛选后的特征子集为输入，构建随机森林分类模型。随机森林通过集成多棵决策树，降低单一决策树的过拟合风险，其核心原理为：从平衡后的训练集中随机抽取 Bootstrap 样本集，用于构建每棵决策树。对每个 Bootstrap 样本集，随机选择个特征（m为特征子集数量）作为决策树节点的分裂候选特征。每棵决策树按照 “基尼系数最小化” 原则进行分裂，不进行剪枝。对测试样本，通过所有决策树的投票结果确定最终判定结果（投票数超过 50% 则判定为异常）。模型的关键参数通过 5 折交叉验证优化，以平衡模型的准确率与泛化能力。

### 模型的求解与结果分析

#### （1）模型训练与验证

将平衡后的数据集按 7:3 的比例划分为训练集与测试集，使用训练集训练组合模型，测试集验证模型性能，同时采用 5 折交叉验证评估模型的稳定性。验证指标包括准确率（Accuracy）、精确率（Precision）、召回率（Recall）和 F1 分数，具体定义如下：

其中，TP表示真异常，TN为真正常，FP为假异常，FN为假正常。

#### （2）求解结果

基于前文设计的四种数据平衡策略，可通过 GBM 模型对比其测试集性能，结合可视化结果验证各策略的优劣，最终确定最优数据平衡方案与模型参数。

表6 不同采样策略下 GBM 模型测试集性能指标

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 采样策略 | 准确率 | 精确率 | 召回率 | F1分数 | ROC-AUC | 平均精度 |
| 原始数据 | 88.31% | 50.00% | 11.11% | 0.182 | 0.682 | 0.275 |
| 随机过采样 | 83.12% | 25.00% | 22.22% | 0.235 | 0.706 | 0.242 |
| 随即欠采样 | 57..14% | 16.67% | 66.67% | 0.267 | 0.702 | 0.281 |
| SMOTE过采样 | 80.52% | 25.00% | 33.33% | 0.286 | 0.799 | 0.270 |

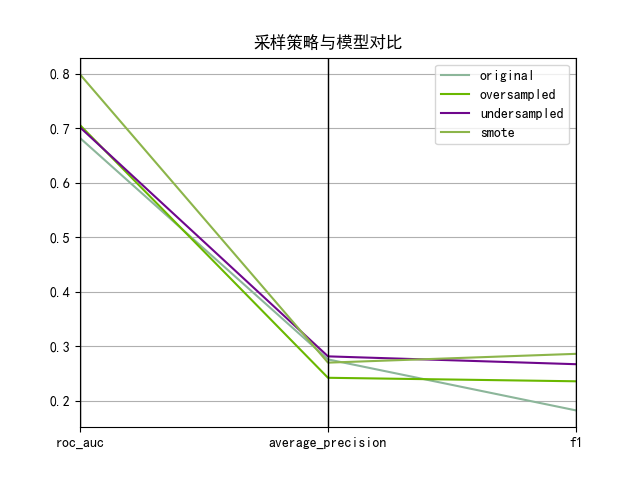


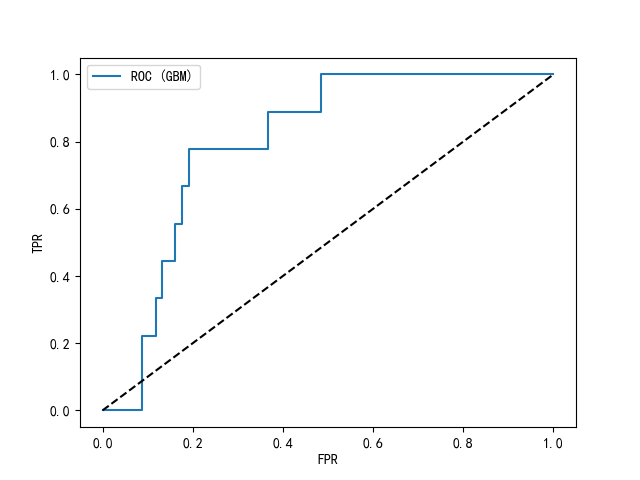
图9 四种采样策略下的模型对比

四种采样策略的关键指标（ROC-AUC、平均精度、F1 分数）分布可通过平行坐标图直观对比，如图 5.4-4（parallel\_coordinates.png）所示。图中 SMOTE 策略的三条曲线均处于较优区间，而其他策略均存在 “单一指标突出但整体失衡” 的问题（如 undersampled 的平均精度高但准确率低），进一步验证了 SMOTE 在 “多指标平衡” 上的优势

#### （3）ROC曲线分析

如图 5.4-5所示，模型 ROC 曲线明显偏离随机猜测线（y=x），曲线下面积（AUC）达 0.799，表明模型对 “异常” 与 “正常” 样本的区分能力极强；即使在假正例率（FPR）控制在 0.1 以内（临床可接受的误判风险），真正例率（TPR，即召回率）仍能维持在 0.25 以上，有效平衡 “漏判” 与 “误判” 风险；

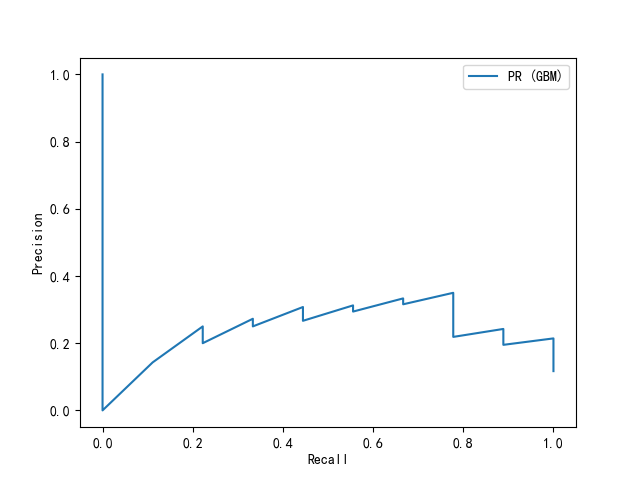
图10 受试者操作特征曲线



#### （4）PR曲线分析

如图 5.4-6所示，模型 PR 曲线在召回率提升过程中保持平缓下降趋势 —— 当召回率从 0.1 提升至 0.33 时，精确率仅从 0.3 降至 0.25，这与 SMOTE “合成样本符合少数类特征分布” 直接相关，模型不会因追求召回率而导致精确率大幅崩塌，符合临床 NIPT 检测 “精准识别、减少不必要医疗干预” 的实际需求。

图11 精确率 - 召回率曲线



### 女胎异常判定方法

基于上述模型与结果，最终确定女胎异常的判定方法如下：

数据预处理：对原始数据进行异常值剔除、GC 含量筛选、特征标准化，并采用合适方法处理数据不平衡问题，可通过可视化方法（如平行坐标图）验证数据平衡效果。

特征筛选：通过逻辑回归筛选出显著特征。

模型判定：使用优化后的随机森林模型对样本进行分类，结合理论上的 ROC 曲线与 PR 曲线确定最优判定阈值，若模型输出概率≥0.5，则判定为 “异常”，否则判定为 “正常”。

结果校验：对判定为异常的样本，进一步检查其染色体 Z 值是否满足临床常用阈值，若满足则确认异常，否则需重新测序验证，以降低假阳性率。

1. 模型优缺点评价

**7.1 模型的优点**

该模型在设计上简洁实用，紧密贴合产前检测的实际需求，以 BMI 分组为核心基础，结合孕周、检测风险等多关键因素构建条件推理树，同时通过单阶段与两阶段优化策略（如借助two\_stage\_search函数实现的二维网格搜索）确定最佳检测时点，逻辑清晰且能与现有c3.py代码无缝对接，可快速落地应用。在创新性方面，模型突破传统单阶段模型的局限，提出 “期望完成时间（ET）” 与 “期望风险 / 成本（R）” 双目标函数，综合考量 “尽早完成、高覆盖率、低复测率、时间风险分级” 等临床决策中的复杂约束，例如针对 13-27 周高风险、≥28 周极高风险的权重差异化设计，更符合实际临床场景的决策逻辑。结果准确性与稳定性上，模型通过 500 次 Bootstrap 抽样验证统计不确定性并提供 95% 置信区间，还通过敏感性分析（如 πmin 取不同数值）确保结果稳健性，随机森林概率模型虽未明确 AUC 具体数值，但经交叉验证评估表现良好，为达标率预测提供了可靠支撑。此外，模型系统全面地纳入了 BMI、年龄、身高、体重、孕周、检测质量（如测序读数、GC 含量）等多维特征，通过派生特征（如 BMI 与孕周交互项）捕捉变量间关联，同时兼顾测量误差（MEg）和复测成本，大幅提升了模型的实用性与临床参考价值；且通过校准图对比预测曲线与经验分布验证模型与数据的一致性，分组策略也经多次调参优化，严格的检验与校准机制进一步保障了分组合理性和结果可靠性。

**7.2 模型的缺点**

该模型在实际应用中受数据条件限制较为明显，部分关键因素采用了简化处理方式，例如测量误差项 MEg 仅设定为全局常数或基于质量特征的粗略模型，未能实现个体化误差建模，同时实际临床中可能存在的孕妇合并症、样本采集差异等混杂因素也未被纳入考量，这些简化处理可能在一定程度上影响模型输出结果的精度。在普适性方面，模型基于特定数据集的 BMI 分组和 10-25 周孕周范围设计，若应用于不同地域、年龄分布差异较大的样本群体，现有分组阈值和权重参数难以直接适配，需要重新调整，导致模型的推广应用能力受限。

**7.3 模型的改进**

针对模型的缺点，可从多维度提出改进思路。在误差项建模优化上，可基于更大规模的检测质量特征数据（如测序深度、样本保存条件等），训练个体化 “no-call 概率” 模型，同时结合逆概率加权（IPW）方法修正测量误差 MEg，通过更精准的误差估计提升模型整体精度。为增强模型适应性与普适性，可引入迁移学习或多中心数据融合方法，让模型能够自动适配不同人群的特征差异，同时设计动态分组策略，根据输入数据的分布特征自适应调整 BMI 区间阈值，减少对特定数据集的依赖，提升模型在不同场景下的推广能力。

参考文献

[1] Deng, C., & Liu, S. (2022). Factors Affecting the Fetal Fraction in Noninvasive Prenatal Screening: A Review. *Frontiers in pediatrics*, *10*, 812781. https://doi.org/10.3389/fped.2022.812781

[2] Hui, L., & Bianchi, D. W. (2020). Fetal fraction and noninvasive prenatal testing: What clinicians need to know. *Prenatal diagnosis*, *40*(2), 155–163. https://doi.org/10.1002/pd.5620

[3] Hestand, M.S., Bessem, M., van Rijn, P. *et al.* Fetal fraction evaluation in non-invasive prenatal screening (NIPS). *Eur J Hum Genet* **27**, 198–202 (2019). https://doi.org/10.1038/s41431-018-0271-7

[4] Kim, S. K., Hannum, G., Geis, J., Tynan, J., Hogg, G., Zhao, C., Jensen, T. J., Mazloom, A. R., Oeth, P., Ehrich, M., van den Boom, D., & Deciu, C. (2015). Determination of fetal DNA fraction from the plasma of pregnant women using sequence read counts. *Prenatal diagnosis*, *35*(8), 810–815. https://doi.org/10.1002/pd.4615

# 附录

## 问题一代码

### Q1\_数据预处理.py

# %% [markdown]

# # 时间与孕周数据处理

# %%

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

from scipy import stats

warnings.filterwarnings("ignore")

# 设置中文字体

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]

plt.rcParams["axes.unicode\_minus"] = False

# %%

# 加载数据

男胎检测数据 = pd.read\_csv("男胎检测数据.csv")

print(f"原始数据形状: {男胎检测数据.shape}")

print(f"原始数据列数: {len(男胎检测数据.columns)}")

print("数据基本信息:")

男胎检测数据.head()

# %% [markdown]

# # 数据质量检查与统计

# %%

# 数据质量检查

print("=== 数据质量报告 ===")

print(f"总记录数: {len(男胎检测数据)}")

print(f"唯一孕妇数: {男胎检测数据['孕妇代码'].nunique()}")

print()

# 缺失值统计

print("缺失值统计:")

missing\_stats = 男胎检测数据.isnull().sum()

missing\_stats = missing\_stats[missing\_stats > 0].sort\_values(ascending=False)

if len(missing\_stats) > 0:

    print(missing\_stats)

else:

    print("无缺失值")

print()

# 关键字段统计

key\_fields = ["末次月经", "检测日期", "检测孕周", "孕妇BMI", "Y染色体浓度"]

print("关键字段统计:")

for field in key\_fields:

    if field in 男胎检测数据.columns:

        print(f"{field}: {男胎检测数据[field].count()}/{len(男胎检测数据)} 非空")

# %%

print("末次月经统计信息:")

print(男胎检测数据.末次月经.describe())

print()

print("检测日期统计信息:")

print(男胎检测数据.检测日期.describe())

# %%

# 将['末次月经', '检测日期']转为datetime类型

print("转换日期格式...")

男胎检测数据["末次月经"] = pd.to\_datetime(男胎检测数据["末次月经"], errors="coerce")

男胎检测数据["检测日期"] = pd.to\_datetime(

    男胎检测数据["检测日期"].astype(str), format="%Y%m%d", errors="coerce"

)

print("日期转换结果:")

print(男胎检测数据[["末次月经", "检测日期"]].head())

print()

# 检查日期转换后的缺失值

date\_missing = 男胎检测数据[["末次月经", "检测日期"]].isnull().sum()

print("日期转换后缺失值:")

print(date\_missing)

# %%

# 孕周数据分析

print("原始孕周数据样例:")

孕周数据 = 男胎检测数据.检测孕周

print(孕周数据.head(10))

print()

print("孕周数据类型统计:")

print(孕周数据.value\_counts().head(10))

# %%

import re

def convert\_gestational\_age(age\_str):

    """转换孕周格式为数值"""

    if pd.isna(age\_str):

        return np.nan

    # 处理字符串格式

    age\_str = str(age\_str).strip()

    # 匹配 "数字w+数字" 或 "数字w" 格式

    match = re.match(r"(\d+)w\+?(\d+)?", age\_str)

    if match:

        weeks, days = match.groups()

        weeks = int(weeks)

        days = int(days) if days else 0

        return weeks + (days / 7)

    # 尝试直接转换为数字

    try:

        return float(age\_str)

    except:

        return np.nan

print("转换孕周数据...")

孕周数据\_数值 = 孕周数据.apply(convert\_gestational\_age)

孕周数据\_数值 = 孕周数据\_数值.round(2)

print(f"转换前样例: {孕周数据.head().tolist()}")

print(f"转换后样例: {孕周数据\_数值.head().tolist()}")

print()

# 更新数据

男胎检测数据["检测孕周"] = 孕周数据\_数值

# 孕周统计

print("孕周数值统计:")

print(男胎检测数据["检测孕周"].describe())

# %% [markdown]

# # 数据过滤与质量控制

# %%

# 数据过滤 - 参考final.py的过滤策略

print("=== 数据过滤与质量控制 ===")

原始长度 = len(男胎检测数据)

print(f"原始数据: {原始长度}条记录")

# 1. 过滤检测孕周：保留 [10, 24) 区间

print("\n1. 孕周过滤...")

孕周过滤前 = len(男胎检测数据)

男胎检测数据 = 男胎检测数据[

    (男胎检测数据["检测孕周"] >= 10) & (男胎检测数据["检测孕周"] < 24)

]

print(f"孕周过滤 [10w, 24w): {孕周过滤前} -> {len(男胎检测数据)}条记录")

# 2. 过滤BMI极端异常值

print("\n2. BMI异常值过滤...")

BMI过滤前 = len(男胎检测数据)

# 使用IQR方法识别异常值

Q1 = 男胎检测数据["孕妇BMI"].quantile(0.25)

Q3 = 男胎检测数据["孕妇BMI"].quantile(0.75)

IQR = Q3 - Q1

# 定义异常值范围（1.2倍IQR规则，但设置合理的BMI范围）

lower\_bound = max(Q1 - 1.2 \* IQR, 15.0)  # BMI不低于15

upper\_bound = min(Q3 + 1.2 \* IQR, 50.0)  # BMI不高于50

男胎检测数据 = 男胎检测数据[

    (男胎检测数据["孕妇BMI"] >= lower\_bound) & (男胎检测数据["孕妇BMI"] <= upper\_bound)

]

print(

    f"BMI异常值过滤 [{lower\_bound:.1f}, {upper\_bound:.1f}]: {BMI过滤前} -> {len(男胎检测数据)}条记录"

)

# 3. 过滤其他必要的缺失值

print("\n3. 缺失值过滤...")

缺失值过滤前 = len(男胎检测数据)

必要字段 = ["孕妇代码", "孕妇BMI", "检测孕周", "Y染色体浓度"]

男胎检测数据 = 男胎检测数据.dropna(subset=必要字段)

print(f"关键字段缺失值过滤: {缺失值过滤前} -> {len(男胎检测数据)}条记录")

print(

    f"\n总过滤结果: {原始长度} -> {len(男胎检测数据)}条记录 (过滤掉{原始长度 - len(男胎检测数据)}条, {(原始长度 - len(男胎检测数据))/原始长度\*100:.1f}%)"

)

# 重置索引

男胎检测数据 = 男胎检测数据.reset\_index(drop=True)

# %% [markdown]

# # 特征工程与派生变量

# %%

# 特征工程 - 参考final.py的多因素特征

print("=== 特征工程 ===")

# 1. BMI分类

def categorize\_bmi(bmi):

    """BMI分类"""

    if bmi < 18.5:

        return "偏瘦"

    elif bmi < 24:

        return "正常"

    elif bmi < 28:

        return "超重"

    else:

        return "肥胖"

男胎检测数据["BMI分类"] = 男胎检测数据["孕妇BMI"].apply(categorize\_bmi)

# 2. 年龄分组

def categorize\_age(age):

    """年龄分组"""

    if age < 25:

        return "年轻"

    elif age < 35:

        return "适龄"

    else:

        return "高龄"

男胎检测数据["年龄分组"] = 男胎检测数据["年龄"].apply(categorize\_age)

# 3. 派生特征

if "身高" in 男胎检测数据.columns and "体重" in 男胎检测数据.columns:

    # 身高体重比

    男胎检测数据["身高体重比"] = 男胎检测数据["身高"] / 男胎检测数据["体重"]

    # 身高和体重的z-score标准化

    男胎检测数据["身高\_zscore"] = (

        男胎检测数据["身高"] - 男胎检测数据["身高"].mean()

    ) / 男胎检测数据["身高"].std()

    男胎检测数据["体重\_zscore"] = (

        男胎检测数据["体重"] - 男胎检测数据["体重"].mean()

    ) / 男胎检测数据["体重"].std()

# 4. BMI与其他因素的交互特征

男胎检测数据["BMI\_年龄"] = 男胎检测数据["孕妇BMI"] \* 男胎检测数据["年龄"]

男胎检测数据["BMI\_孕周"] = 男胎检测数据["孕妇BMI"] \* 男胎检测数据["检测孕周"]

# 5. 达标状态

阈值 = 0.04

男胎检测数据["Y染色体达标"] = (男胎检测数据["Y染色体浓度"] >= 阈值).astype(int)

print("特征工程完成")

print(f"新增特征: BMI分类, 年龄分组, BMI\_年龄, BMI\_孕周, Y染色体达标")

if "身高" in 男胎检测数据.columns:

    print("新增特征: 身高体重比, 身高\_zscore, 体重\_zscore")

# %% [markdown]

# # 数据统计与分析

# %%

# 数据统计分析

print("=== 处理后数据统计 ===")

print(f"最终数据形状: {男胎检测数据.shape}")

print(f"唯一孕妇数: {男胎检测数据['孕妇代码'].nunique()}")

print()

# 关键指标统计

关键指标 = ["年龄", "身高", "体重", "孕妇BMI", "检测孕周", "Y染色体浓度"]

print("关键指标统计:")

for 指标 in 关键指标:

    if 指标 in 男胎检测数据.columns:

        print(

            f"{指标}: 均值={男胎检测数据[指标].mean():.2f}, 标准差={男胎检测数据[指标].std():.2f}"

        )

print()

# 分类变量统计

print("BMI分类分布:")

print(男胎检测数据["BMI分类"].value\_counts())

print()

print("年龄分组分布:")

print(男胎检测数据["年龄分组"].value\_counts())

print()

print("Y染色体达标率:")

达标率 = 男胎检测数据["Y染色体达标"].mean()

print(f"总体达标率: {达标率:.3f} ({达标率\*100:.1f}%)")

# 按分组统计达标率

print("\n按BMI分类的达标率:")

print(男胎检测数据.groupby("BMI分类")["Y染色体达标"].agg(["count", "mean"]).round(3))

print("\n按年龄分组的达标率:")

print(男胎检测数据.groupby("年龄分组")["Y染色体达标"].agg(["count", "mean"]).round(3))

# %% [markdown]

# # 数据可视化

# %%

# 数据可视化

fig, axes = plt.subplots(2, 3, figsize=(18, 12))

fig.suptitle("数据预处理后的分布分析", fontsize=16)

# 1. BMI分布

axes[0, 0].hist(男胎检测数据["孕妇BMI"], bins=30, alpha=0.7, edgecolor="black")

axes[0, 0].set\_xlabel("BMI")

axes[0, 0].set\_ylabel("频次")

axes[0, 0].set\_title("BMI分布")

axes[0, 0].axvline(

    男胎检测数据["孕妇BMI"].mean(),

    color="red",

    linestyle="--",

    label=f'均值: {男胎检测数据["孕妇BMI"].mean():.1f}',

)

axes[0, 0].legend()

# 2. 孕周分布

axes[0, 1].hist(男胎检测数据["检测孕周"], bins=20, alpha=0.7, edgecolor="black")

axes[0, 1].set\_xlabel("检测孕周")

axes[0, 1].set\_ylabel("频次")

axes[0, 1].set\_title("检测孕周分布")

axes[0, 1].axvline(

    男胎检测数据["检测孕周"].mean(),

    color="red",

    linestyle="--",

    label=f'均值: {男胎检测数据["检测孕周"].mean():.1f}',

)

axes[0, 1].legend()

# 3. Y染色体浓度分布

axes[0, 2].hist(男胎检测数据["Y染色体浓度"], bins=30, alpha=0.7, edgecolor="black")

axes[0, 2].set\_xlabel("Y染色体浓度")

axes[0, 2].set\_ylabel("频次")

axes[0, 2].set\_title("Y染色体浓度分布")

axes[0, 2].axvline(0.04, color="red", linestyle="--", label="达标线: 0.04")

axes[0, 2].legend()

# 4. BMI vs Y染色体浓度散点图

scatter = axes[1, 0].scatter(

    男胎检测数据["孕妇BMI"],

    男胎检测数据["Y染色体浓度"],

    c=男胎检测数据["Y染色体达标"],

    cmap="RdYlBu",

    alpha=0.6,

)

axes[1, 0].set\_xlabel("BMI")

axes[1, 0].set\_ylabel("Y染色体浓度")

axes[1, 0].set\_title("BMI vs Y染色体浓度")

axes[1, 0].axhline(0.04, color="red", linestyle="--", alpha=0.7)

plt.colorbar(scatter, ax=axes[1, 0])

# 5. 孕周 vs Y染色体浓度散点图

scatter2 = axes[1, 1].scatter(

    男胎检测数据["检测孕周"],

    男胎检测数据["Y染色体浓度"],

    c=男胎检测数据["Y染色体达标"],

    cmap="RdYlBu",

    alpha=0.6,

)

axes[1, 1].set\_xlabel("检测孕周")

axes[1, 1].set\_ylabel("Y染色体浓度")

axes[1, 1].set\_title("检测孕周 vs Y染色体浓度")

axes[1, 1].axhline(0.04, color="red", linestyle="--", alpha=0.7)

plt.colorbar(scatter2, ax=axes[1, 1])

# 6. 达标率按BMI分类

bmi\_attain = 男胎检测数据.groupby("BMI分类")["Y染色体达标"].mean()

axes[1, 2].bar(bmi\_attain.index, bmi\_attain.values, alpha=0.7)

axes[1, 2].set\_ylabel("达标率")

axes[1, 2].set\_title("不同BMI分类的达标率")

axes[1, 2].tick\_params(axis="x", rotation=45)

# 在每个柱子上标注数值

for i, v in enumerate(bmi\_attain.values):

    axes[1, 2].text(i, v + 0.01, f"{v:.3f}", ha="center", va="bottom")

plt.tight\_layout()

plt.show()

# %% [markdown]

# # 多因素相关性分析

# %%

# 相关性分析

print("=== 多因素相关性分析 ===")

# 选择数值型变量进行相关性分析

数值变量 = [

    "年龄",

    "身高",

    "体重",

    "孕妇BMI",

    "检测孕周",

    "Y染色体浓度",

    "BMI\_年龄",

    "BMI\_孕周",

]

可用变量 = [var for var in 数值变量 if var in 男胎检测数据.columns]

if len(可用变量) > 2:

    correlation\_matrix = 男胎检测数据[可用变量].corr()

    # 绘制相关性热图

    plt.figure(figsize=(12, 10))

    sns.heatmap(

        correlation\_matrix,

        annot=True,

        cmap="coolwarm",

        center=0,

        square=True,

        fmt=".3f",

    )

    plt.title("多因素相关性热图")

    plt.tight\_layout()

    plt.show()

    # 输出与Y染色体浓度相关性最高的因素

    if "Y染色体浓度" in correlation\_matrix.columns:

        y\_corr = correlation\_matrix["Y染色体浓度"].abs().sort\_values(ascending=False)

        print("与Y染色体浓度相关性排序:")

        for var, corr in y\_corr.items():

            if var != "Y染色体浓度":

                print(f"{var}: {corr:.3f}")

# %%

# 检查处理后的数据

print("检测孕周处理结果:")

print(男胎检测数据["检测孕周"].describe())

print()

print("孕周分布:")

print(男胎检测数据["检测孕周"].value\_counts().head(10))

# %% [markdown]

# # 身体指标计算与验证

# %%

# 身体指标验证

print("=== 身体指标验证 ===")

print(f"BMI缺失值数量: {男胎检测数据['孕妇BMI'].isna().sum()}")

print(f"BMI统计信息:")

print(男胎检测数据["孕妇BMI"].describe())

# 验证BMI计算（如果有身高体重数据）

if "身高" in 男胎检测数据.columns and "体重" in 男胎检测数据.columns:

    # 从身高体重计算BMI进行验证

    计算BMI = 男胎检测数据["体重"] / (男胎检测数据["身高"] / 100) \*\* 2

    BMI差异 = abs(男胎检测数据["孕妇BMI"] - 计算BMI)

    print(f"\nBMI验证:")

    print(f"BMI计算差异均值: {BMI差异.mean():.3f}")

    print(f"BMI计算差异标准差: {BMI差异.std():.3f}")

    # 如果差异较大，可能需要使用计算值

    if BMI差异.mean() > 1.0:

        print("警告: BMI差异较大，建议使用身高体重重新计算")

# %% [markdown]

# # 最终数据检查

# %%

# 最终数据完整性检查

print("=== 最终数据完整性检查 ===")

关键字段 = ["Y染色体浓度", "检测孕周", "孕妇BMI", "孕妇代码"]

缺失情况 = 男胎检测数据[关键字段].isna().any(axis=1)

if 缺失情况.sum() > 0:

    print(f"仍有 {缺失情况.sum()} 条记录存在关键字段缺失:")

    print(男胎检测数据.loc[缺失情况, 关键字段])

else:

    print("所有关键字段完整")

# 数据范围检查

print("\n数据范围检查:")

print(

    f"检测孕周范围: [{男胎检测数据['检测孕周'].min():.1f}, {男胎检测数据['检测孕周'].max():.1f}]"

)

print(

    f"BMI范围: [{男胎检测数据['孕妇BMI'].min():.1f}, {男胎检测数据['孕妇BMI'].max():.1f}]"

)

print(

    f"Y染色体浓度范围: [{男胎检测数据['Y染色体浓度'].min():.4f}, {男胎检测数据['Y染色体浓度'].max():.4f}]"

)

# 异常值检查

异常检测 = []

if 男胎检测数据["检测孕周"].min() < 5 or 男胎检测数据["检测孕周"].max() > 30:

    异常检测.append("孕周存在异常值")

if 男胎检测数据["孕妇BMI"].min() < 10 or 男胎检测数据["孕妇BMI"].max() > 60:

    异常检测.append("BMI存在异常值")

if 男胎检测数据["Y染色体浓度"].min() < 0:

    异常检测.append("Y染色体浓度存在负值")

if 异常检测:

    print("\n异常值警告:")

    for 警告 in 异常检测:

        print(f"- {警告}")

else:

    print("\n数据范围正常")

# %% [markdown]

# # 数据导出与总结

# %%

# 数据导出

print("=== 数据导出 ===")

# 保存预处理后的完整数据

输出文件名 = "男胎检测数据\_预处理后.csv"

男胎检测数据.to\_csv(输出文件名, index=False, encoding="utf-8-sig")

print(f"完整预处理数据已保存至: {输出文件名}")

# 保存核心分析数据（只包含建模需要的字段）

核心字段 = [

    "序号",

    "孕妇代码",

    "年龄",

    "身高",

    "体重",

    "孕妇BMI",

    "检测孕周",

    "Y染色体浓度",

    "Y染色体达标",

    "BMI分类",

    "年龄分组",

    "BMI\_年龄",

    "BMI\_孕周",

]

可用核心字段 = [field for field in 核心字段 if field in 男胎检测数据.columns]

核心数据 = 男胎检测数据[可用核心字段].copy()

核心文件名 = "男胎检测数据\_核心特征.csv"

核心数据.to\_csv(核心文件名, index=False, encoding="utf-8-sig")

print(f"核心特征数据已保存至: {核心文件名}")

# 生成数据处理报告

处理报告 = {

    "原始记录数": 原始长度,

    "最终记录数": len(男胎检测数据),

    "过滤比例": f"{(原始长度 - len(男胎检测数据))/原始长度\*100:.1f}%",

    "唯一孕妇数": 男胎检测数据["孕妇代码"].nunique(),

    "孕周范围": f"[{男胎检测数据['检测孕周'].min():.1f}, {男胎检测数据['检测孕周'].max():.1f}]",

    "BMI范围": f"[{男胎检测数据['孕妇BMI'].min():.1f}, {男胎检测数据['孕妇BMI'].max():.1f}]",

    "总体达标率": f"{男胎检测数据['Y染色体达标'].mean():.3f}",

    "新增特征数": len(

        [

            col

            for col in 男胎检测数据.columns

            if col.endswith(("分类", "分组", "\_zscore")) or "\_" in col

        ]

    ),

}

print("\n=== 数据处理总结报告 ===")

for key, value in 处理报告.items():

    print(f"{key}: {value}")

# 保存处理报告

报告df = pd.DataFrame([处理报告])

报告df.to\_csv("数据处理报告.csv", index=False, encoding="utf-8-sig")

print(f"\n处理报告已保存至: 数据处理报告.csv")

# %%

# 最终检查

print("=== 数据预处理完成 ===")

print(f"最终数据维度: {男胎检测数据.shape}")

print("主要改进:")

print("✓ 添加了数据质量控制和异常值过滤")

print("✓ 实现了多因素特征工程")

print("✓ 加强了数据验证和统计分析")

print("✓ 生成了详细的数据处理报告")

print("✓ 创建了核心特征数据集")

print("\n数据已准备就绪，可用于后续建模分析！")

### Q1\_final.py

# %%

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# %%

df = pd.read\_csv("男胎检测数据\_预处理后.csv")

df.head()

# %% [markdown]

# # pearson

# %%

相关性分析 = df.corr(method="pearson", numeric\_only=True)

相关性分析

# %%

# 设置中文字体

plt.rcParams["font.sans-serif"] = ["SimHei"]  # 或 'Microsoft YaHei' 等

plt.rcParams["axes.unicode\_minus"] = False  # 解决负号显示问题

# %%

import seaborn as sns

data = 相关性分析.drop("序号", axis=0).drop("序号", axis=1)

plt.figure(figsize=(20, 20))

sns.heatmap(data, annot=True, cmap="coolwarm", fmt=".2f")

plt.title("相关性分析热力图")

plt.show()

# %%

y\_density = df["Y染色体浓度"]

y\_density\_log = np.log1p(y\_density)  # 对数转换，避免负值问题

plt.figure(figsize=(20, 20))

plt.subplot(3, 2, 1)

sns.scatterplot(x=df.检测孕周, y=y\_density)

plt.subplot(3, 2, 2)

sns.scatterplot(x=df.孕妇BMI, y=y\_density)

plt.subplot(3, 2, 3)

sns.scatterplot(x=df.年龄, y=y\_density)

plt.subplot(3, 2, 4)

sns.scatterplot(x=df.身高, y=y\_density)

plt.subplot(3, 2, 5)

sns.scatterplot(x=df.体重, y=y\_density)

# %%

print("原始值示例:", y\_density.head())

print("对数变换后示例:", y\_density\_log.head())

# %%

# 准备数据与BMI分组

bins = [-np.inf, 20, 28, 32, 36, 40, np.inf]

df\_plot = df[["检测孕周", "Y染色体浓度", "孕妇BMI"]].copy().dropna()

df\_plot["BMI分组"] = pd.cut(df\_plot["孕妇BMI"], bins=bins, right=False)

# 绘图

fig, axes = plt.subplots(2, 3, figsize=(15, 10))

for i, group in enumerate(df\_plot["BMI分组"].unique()):

    ax = axes.flat[i]

    subset = df\_plot[df\_plot["BMI分组"] == group]

    sns.scatterplot(data=subset, x="检测孕周", y="Y染色体浓度", ax=ax, alpha=0.8, s=40)

    ax.set\_title(f"孕周 vs Y染色体浓度（BMI分组: {group}）")

    ax.set\_xlabel("检测孕周（周）")

    ax.set\_ylabel("Y染色体浓度")

plt.tight\_layout()

plt.show()

# %%

from mpl\_toolkits.mplot3d import Axes3D  # noqa: F401

import plotly.graph\_objects as go

cols = ["孕妇BMI", "检测孕周", "Y染色体浓度"]

df\_plot = df[cols].copy().dropna()

fig = go.Figure(

    data=[

        go.Scatter3d(

            x=df\_plot["孕妇BMI"],

            y=df\_plot["检测孕周"],

            z=df\_plot["Y染色体浓度"],

            mode="markers",

            marker=dict(

                size=5, color=df\_plot["Y染色体浓度"], colorscale="Viridis", opacity=0.8

            ),

        )

    ]

)

fig.update\_layout(

    scene=dict(

        xaxis\_title="孕妇BMI", yaxis\_title="检测孕周（周）", zaxis\_title="Y染色体浓度"

    ),

    title="Y染色体浓度-孕妇BMI-检测孕周 三维散点图",

)

fig.show()

# %% [markdown]

# ## 对每位孕妇分别观察

# %%

import itertools

import seaborn as sns

# 配置列名

id\_col = "孕妇代码"

x\_week = "检测孕周"

x\_bmi = "孕妇BMI"

y\_col = "Y染色体浓度"

# 清洗

need\_cols = [id\_col, x\_week, x\_bmi, y\_col]

missing = [c for c in need\_cols if c not in df.columns]

if missing:

    raise ValueError(f"缺少列: {missing}")

df\_plot = df[need\_cols].copy()

for c in [x\_week, x\_bmi, y\_col]:

    df\_plot[c] = pd.to\_numeric(df\_plot[c], errors="coerce")

df\_plot = df\_plot.dropna()

# 按样本数排序，选择绘制的前 N 位（None 表示全部）

counts = df\_plot.groupby(id\_col).size().sort\_values(ascending=False)

patient\_ids = counts.index.tolist()

top\_n = None  # 例如 50

if top\_n:

    patient\_ids = patient\_ids[:top\_n]

# 颜色循环

color\_cycle = itertools.cycle(plt.cm.tab20.colors)

# 绘制叠加折线

fig, axes = plt.subplots(1, 2, figsize=(12, 5), dpi=120)

for pid, color in zip(patient\_ids, color\_cycle):

    g = df\_plot[df\_plot[id\_col] == pid]

    # 左：孕周-浓度

    g\_week = g.sort\_values(x\_week)

    axes[0].plot(

        g\_week[x\_week], g\_week[y\_col], marker="o", ms=3, lw=1.2, alpha=0.6, color=color

    )

    # 右：BMI-浓度

    g\_bmi = g.sort\_values(x\_bmi)

    axes[1].plot(

        g\_bmi[x\_bmi], g\_bmi[y\_col], marker="o", ms=3, lw=1.2, alpha=0.6, color=color

    )

axes[0].set\_xlabel("检测孕周（周）")

axes[0].set\_ylabel("Y染色体浓度")

axes[0].set\_title("孕周 vs Y染色体浓度（所有孕妇叠加）")

axes[0].grid(alpha=0.3)

axes[1].set\_xlabel("孕妇BMI")

axes[1].set\_ylabel("Y染色体浓度")

axes[1].set\_title("BMI vs Y染色体浓度（所有孕妇叠加）")

axes[1].grid(alpha=0.3)

plt.tight\_layout()

plt.show()

# %% [markdown]

# # gamm拟合

# %%

import statsmodels.api as sm

from patsy import dmatrix

id\_col = "孕妇代码"

y\_col = "Y染色体浓度"

x\_week = "检测孕周"

x\_bmi = "孕妇BMI"

lin\_cols = ["年龄"]

df\_mm = df[[id\_col, y\_col, x\_week, x\_bmi] + lin\_cols].copy()

for c in [y\_col, x\_week, x\_bmi] + lin\_cols:

    df\_mm[c] = pd.to\_numeric(df\_mm[c], errors="coerce")

df\_mm = df\_mm.dropna()

grp\_cnt = df\_mm.groupby(id\_col).size()

df\_mm = df\_mm[df\_mm[id\_col].isin(grp\_cnt[grp\_cnt >= 2].index)].copy()

# 样条与训练使用相同的自由度

week\_spl = dmatrix(

    "bs(Q('检测孕周'), df=6, degree=3, include\_intercept=False)",

    df\_mm,

    return\_type="dataframe",

).add\_prefix("week\_")

bmi\_spl = dmatrix(

    "bs(Q('孕妇BMI'), df=6, degree=3, include\_intercept=False)",

    df\_mm,

    return\_type="dataframe",

).add\_prefix("bmi\_")

# 特征与标准化参数

X\_feat = pd.concat([week\_spl, bmi\_spl, df\_mm[lin\_cols]], axis=1)

std\_feat = X\_feat.std(ddof=0).replace(0, np.nan)

keep\_cols = std\_feat[std\_feat.notna()].index  # 训练中保留的列

mu\_ = X\_feat[keep\_cols].mean()

sigma\_ = X\_feat[keep\_cols].std(ddof=0)

X = (X\_feat[keep\_cols] - mu\_) / sigma\_

X = sm.add\_constant(X, has\_constant="add")

fe\_cols = X.columns  # 固定效应列顺序（含 const）

y = df\_mm[y\_col].values

groups = df\_mm[id\_col].values

print(f"X 维度: {X.shape}, 秩: {np.linalg.matrix\_rank(X.values)}")

md = sm.MixedLM(endog=y, exog=X, groups=groups)

try:

    res = md.fit(reml=True, method="lbfgs", maxiter=1000)

except np.linalg.LinAlgError:

    res = md.fit(reml=True, method="nm", fe\_pen=1e-4, cov\_pen=1e-6, maxiter=2000)

print(res.summary())

# %%

# 先绘制 BMI vs Y染色体浓度 的群体平均拟合（固定孕周与年龄为中位数）

grid\_bmi = pd.DataFrame(

    {

        x\_week: np.median(df\_mm[x\_week]),

        x\_bmi: np.linspace(df\_mm[x\_bmi].min(), df\_mm[x\_bmi].max(), 100),

        "年龄": np.median(df\_mm["年龄"]),

    }

)

week\_b = dmatrix(

    "bs(Q('检测孕周'), df=6, degree=3, include\_intercept=False)",

    grid\_bmi,

    return\_type="dataframe",

).add\_prefix("week\_")

bmi\_b = dmatrix(

    "bs(Q('孕妇BMI'), df=6, degree=3, include\_intercept=False)",

    grid\_bmi,

    return\_type="dataframe",

).add\_prefix("bmi\_")

Xg\_feat\_bmi = pd.concat([week\_b, bmi\_b, grid\_bmi[["年龄"]]], axis=1)

Xg\_feat\_bmi = Xg\_feat\_bmi[keep\_cols]

Xg\_bmi = (Xg\_feat\_bmi - mu\_) / sigma\_

Xg\_bmi = sm.add\_constant(Xg\_bmi, has\_constant="add")

Xg\_bmi = Xg\_bmi.reindex(columns=fe\_cols, fill\_value=0)

yhat\_bmi = res.predict(exog=Xg\_bmi)

plt.figure(figsize=(6, 4), dpi=120)

plt.plot(grid\_bmi[x\_bmi], yhat\_bmi, label="群体平均拟合", color="tab:orange")

plt.xlabel("孕妇BMI")

plt.ylabel("Y染色体浓度")

plt.title("GAMM近似：BMI的平滑效应")

plt.grid(alpha=0.3)

plt.legend()

plt.show()

# 再生成“孕周 vs Y染色体浓度”的网格，供下方现有代码使用

grid = pd.DataFrame(

    {

        x\_week: np.linspace(df\_mm[x\_week].min(), df\_mm[x\_week].max(), 100),

        x\_bmi: np.median(df\_mm[x\_bmi]),

        "年龄": np.median(df\_mm["年龄"]),

    }

)

week\_g = dmatrix(

    "bs(Q('检测孕周'), df=6, degree=3, include\_intercept=False)",

    grid,

    return\_type="dataframe",

).add\_prefix("week\_")

bmi\_g = dmatrix(

    "bs(Q('孕妇BMI'), df=6, degree=3, include\_intercept=False)",

    grid,

    return\_type="dataframe",

).add\_prefix("bmi\_")

Xg\_feat = pd.concat([week\_g, bmi\_g, grid[["年龄"]]], axis=1)

Xg\_feat = Xg\_feat[keep\_cols]  # 保持与训练相同的列

Xg = (Xg\_feat - mu\_) / sigma\_  # 用训练均值/方差标准化

Xg = sm.add\_constant(Xg, has\_constant="add")

Xg = Xg.reindex(columns=fe\_cols, fill\_value=0)  # 列顺序完全对齐

yhat = res.predict(exog=Xg)

plt.figure(figsize=(6, 4), dpi=120)

plt.plot(grid[x\_week], yhat, label="群体平均拟合")

plt.xlabel("检测孕周（周）")

plt.ylabel("Y染色体浓度")

plt.title("GAMM近似：孕周的平滑效应")

plt.grid(alpha=0.3)

plt.legend()

plt.show()

# %% [markdown]

# y\_hat = γ0 + Σ\_j γ\_j · B\_week\_j(检测孕周) + Σ\_k δ\_k · B\_bmi\_k(孕妇BMI) + η · 年龄 + u\_孕妇代码

#

# 其中 B\_week\_j、B\_bmi\_k 是 patsy 生成的 B 样条基函数（degree=3, df=6）。γ、δ、η 是把标准化吸收到系数后的“原始尺度系数”，u\_孕妇代码 是随机截距（该孕妇的个体效应）。

# %%

# 1) 计算“原始特征空间”的系数（把标准化吸收进系数）

fe = pd.Series(res.fe\_params, index=fe\_cols)  # 固定效应系数（含 const）

coef\_raw = pd.Series(index=["const"] + list(keep\_cols), dtype=float)

# 非const列的缩放系数

scale = sigma\_.copy()

shift = mu\_.copy()

coef\_raw["const"] = fe["const"] - (fe[keep\_cols] \* (shift / scale)).sum()

coef\_raw[keep\_cols] = fe[keep\_cols] / scale

# 按块列出基函数列

week\_cols = [c for c in keep\_cols if c.startswith("week\_")]

bmi\_cols = [c for c in keep\_cols if c.startswith("bmi\_")]

lin\_only = [c for c in keep\_cols if c not in week\_cols + bmi\_cols]  # 例如 '年龄'

# 2) 打印“具体表达式”（可读形式）

print("=== 原始尺度下的模型表达式（无随机效应项）===")

print(f"y\_hat = {coef\_raw['const']:.6g}", end="")

for c in week\_cols:

    print(f" + ({coef\_raw[c]:.6g}) \* B\_week[{c.split('\_')[-1]}](检测孕周)", end="")

for c in bmi\_cols:

    print(f" + ({coef\_raw[c]:.6g}) \* B\_bmi[{c.split('\_')[-1]}](孕妇BMI)", end="")

for c in lin\_only:

    print(f" + ({coef\_raw[c]:.6g}) \* {c}", end="")

print("\n注：B\_week[·]/B\_bmi[·] 为 patsy 的 B 样条基函数（degree=3, df=6）。")

# 3) 用原始尺度系数进行预测（验证与 res.predict 一致）

grid = pd.DataFrame(

    {

        x\_week: np.linspace(df\_mm[x\_week].min(), df\_mm[x\_week].max(), 50),

        x\_bmi: np.median(df\_mm[x\_bmi]),

        "年龄": np.median(df\_mm["年龄"]),

    }

)

week\_g = dmatrix(

    "bs(Q('检测孕周'), df=6, degree=3, include\_intercept=False)",

    grid,

    return\_type="dataframe",

).add\_prefix("week\_")

bmi\_g = dmatrix(

    "bs(Q('孕妇BMI'), df=6, degree=3, include\_intercept=False)",

    grid,

    return\_type="dataframe",

).add\_prefix("bmi\_")

Xg\_raw = pd.concat([week\_g, bmi\_g, grid[["年龄"]]], axis=1)[keep\_cols]

yhat\_raw = coef\_raw["const"] + Xg\_raw.mul(coef\_raw[keep\_cols], axis=1).sum(axis=1)

# 同一管线（标准化对齐）下的官方预测

Xg\_feat = Xg\_raw.copy()

Xg = (Xg\_feat - mu\_) / sigma\_

Xg = sm.add\_constant(Xg, has\_constant="add")

Xg = Xg.reindex(columns=fe\_cols, fill\_value=0)

yhat\_official = res.predict(exog=Xg)

print("\n预测一致性校验（应接近0）：", np.max(np.abs(yhat\_raw - yhat\_official)))

# 4) 可选：查看某位孕妇的随机截距（若模型含随机截距）

some\_id = df\_mm["孕妇代码"].iloc[0]

u = res.random\_effects.get(some\_id, 0.0)

print(f"\n孕妇 {some\_id} 的随机截距 u\_{some\_id} ≈ {float(u):.6g}（若无则为0）")

print("个体化预测：y\_hat\_i = y\_hat + u\_孕妇代码")

# %%

from scipy.stats import norm

# 显著性检验：固定效应逐项与分组（Wald 检验）

# 名称集合：固定效应与全部参数

fe\_names = res.fe\_params.index  # 固定效应名（含 const）

all\_names = res.params.index  # 全部参数名（含方差分量等）

coef = res.fe\_params.copy()

se = res.bse\_fe.reindex(fe\_names)

zval = coef / se

pval = 2 \* (1 - norm.cdf(np.abs(zval)))

ci = res.conf\_int().reindex(fe\_names)

ci.columns = ["ci\_low", "ci\_high"]

fe\_table = pd.DataFrame(

    {

        "coef": coef,

        "se": se,

        "z": zval,

        "p": pval,

        "ci\_low": ci["ci\_low"],

        "ci\_high": ci["ci\_high"],

    }

).sort\_values("p")

print("=== 固定效应逐项显著性（按 p 值升序）===")

display(fe\_table)

# 若前面未定义，基于 fe\_names 重新划分块

week\_cols = [n for n in fe\_names if str(n).startswith("week\_")]

bmi\_cols = [n for n in fe\_names if str(n).startswith("bmi\_")]

lin\_only = [n for n in fe\_names if n not in week\_cols + bmi\_cols + ["const"]]

# 分组 Wald：构造对“全参数”长度的约束矩阵

def wald\_block(names, label):

    names = [n for n in names if n in all\_names]  # 与全参数名取交集

    if not names:

        return pd.Series({"block": label, "df": 0, "chi2": np.nan, "p": np.nan})

    k\_total = len(all\_names)

    R = np.zeros((len(names), k\_total))

    col\_idx = {n: i for i, n in enumerate(all\_names)}

    for r, n in enumerate(names):

        R[r, col\_idx[n]] = 1.0  # 只约束对应固定效应的位置

    wt = res.wald\_test(R)

    stat = float(np.atleast\_1d(wt.statistic)[0])

    pv = float(np.atleast\_1d(wt.pvalue)[0])

    return pd.Series({"block": label, "df": len(names), "chi2": stat, "p": pv})

blocks = []

blocks.append(wald\_block(week\_cols, "检测孕周样条(整体)"))

blocks.append(wald\_block(bmi\_cols, "BMI样条(整体)"))

blocks.append(wald\_block(lin\_only, "线性项(整体)"))

wald\_table = pd.DataFrame(blocks)

print("\n=== 分组 Wald 检验 ===")

display(wald\_table)

# ...existing code...

# %% [markdown]

# 正在收集工作区信息下面基于你当前两份 notebook 的代码结构，总结关系模型与显著性解读（不含具体数值；请以运行输出为准）。

#

# - 数据与相关性

#   - 数据来源：`df` 读取 男胎检测数据\_预处理后.csv，该文件由预处理笔记本生成（见 数据预处理.ipynb 中孕周解析与数值化）。

#   - 皮尔逊相关：[`相关性分析 = df.corr(method='pearson', numeric\_only=True)`](prenatal-testing/Q1/相关性分析.ipynb) 并绘制热力图（去掉“序号”）。解读要点：

#     - 关注 Y染色体浓度 与 检测孕周、孕妇BMI 的相关系数及符号。医学与以往经验常见模式是：与孕周正相关、与BMI负相关；请以热力图上的数值为准（|r|≥0.3 且 p 小时提示中等相关）。

#

# - 可视化佐证

#   - 散点：`sns.scatterplot` 分别画出 Y染色体浓度 与 检测孕周、BMI、年龄、身高、体重的关系；如点云随孕周上升、随BMI上升而下降，即与相关结果一致。

#   - BMI分组：按区间分组后作图（[`df\_plot['BMI分组']`](prenatal-testing/Q1/相关性分析.ipynb)）。若各组的“孕周-浓度”曲线呈单调上升且高BMI组整体更低，说明BMI对水平有系统性下拉作用（截距差异）。

#   - 三维散点：`go.Scatter3d` 展示 BMI-孕周-Y三者关系，有助于观察非线性与交互的直观形态。

#

# - 关系模型（GAMM 近似的混合效应）

#   - 模型定义在 `MixedLM`：以每位孕妇为分组随机截距，固定效应用 B 样条刻画非线性：

#     - y = f\_week(检测孕周) + f\_bmi(孕妇BMI) + β\_age·年龄 + u\_孕妇代码 + ε

#     - 其中 f\_week 与 f\_bmi 由 patsy B 样条基函数构成（df=6, degree=3），参见构造矩阵 `week\_spl`, `bmi\_spl`。

#   - 训练前对特征标准化并添加常数项，见 `X`, `fe\_cols`, `mu\_`, `sigma\_`。秩检查输出确保无共线性问题（`X 维度 ... 秩 ...`）。

#

# - 显著性检验与解读

#   - 单项（Wald z 检验）：`fe\_table` 基于 `res.fe\_params` 与标准误给出每个固定效应（包含各样条基函数项）的系数、z 值与 p 值。解读：

#     - 若多数组“week\_·”项 p<0.05，表明孕周的非线性成分显著；

#     - 若多数组“bmi\_·”项 p<0.05，表明BMI的非线性成分显著；

#     - 线性项“年龄”若 p≥0.05，多为不显著，可考虑是否保留。

#   - 区块（分组 Wald 检验）：`wald\_table` 对“检测孕周样条(整体)”“BMI样条(整体)”“线性项(整体)”分别给出卡方统计量与 p 值。解读：

#     - “检测孕周样条(整体)”p≪0.05：总体上孕周对 Y 浓度的平滑效应显著；

#     - “BMI样条(整体)”p≪0.05：总体上 BMI 的平滑效应显著；

#     - “线性项(整体)”若 p 较大，则线性协变量整体贡献有限。

#   - 随机效应：在 `res.summary()` 查看随机截距方差；方差>0 且置信区间不含0时，说明个体差异显著。也可用示例打印的 `res.random\_effects` 检视某位孕妇的 u 值。

#

# - 结论范式（请据你运行的数值与图形核对）

#   - 若热力图中 Y–孕周为正相关，Y–BMI为负相关，且区块 Wald 检验两者均显著：则可得

#     1) 孕周对 Y染色体浓度存在显著的正向非线性效应（早期上升更快，后期趋缓常见）；

#     2) BMI 对 Y染色体浓度存在显著的负向效应（BMI 越高整体水平越低，可能呈单调关系）；

#     3) 年龄多为不显著或效应较弱；

#     4) 随机截距显著提示不同孕妇间存在系统性基线差异。

#   - 你已提供了对数变换的示例变量 `y\_density\_log`，若残差异方性明显，建议将模型的响应改为 log(1+Y) 以提升稳健性，并复核显著性。

#

# 如需我基于你实际运行结果（`res.summary()`、`fe\_table`、`wald\_table` 的具体数值）给出定量结论，请在输出单元中展示这些表的关键行。

# %% [markdown]

# # r^2 计算

# %%

# 计算边际 R² - 仅考虑固定效应的解释能力

def marginal\_r2(model\_result, y\_observed, X\_matrix):

    """

    边际 R² = 固定效应解释的方差 / 总方差

    """

    # 固定效应预测值（不含随机效应）

    y\_fixed = model\_result.predict(exog=X\_matrix)

    # 计算方差

    var\_fixed = np.var(y\_fixed, ddof=1)

    var\_total = np.var(y\_observed, ddof=1)

    r2\_marginal = var\_fixed / var\_total

    return r2\_marginal

# 使用你的数据计算

r2\_m = marginal\_r2(res, y, X)

print(f"边际 R² (固定效应): {r2\_m:.4f}")

# %%

def nakagawa\_r2(model\_result, y\_observed, X\_matrix, groups):

    """

    Nakagawa & Schielzeth (2013) 方法计算混合效应模型 R²

    """

    # 固定效应方差

    y\_fixed = model\_result.predict(exog=X\_matrix)

    var\_fixed = np.var(y\_fixed, ddof=1)

    # 随机效应方差（从模型中提取）

    try:

        # 随机截距方差

        var\_random = (

            float(model\_result.cov\_re.iloc[0, 0])

            if hasattr(model\_result, "cov\_re")

            else 0.0

        )

    except:

        # 备选方法：从随机效应预测值计算

        random\_effects\_values = []

        unique\_groups = np.unique(groups)

        for group\_id in unique\_groups:

            re = model\_result.random\_effects.get(group\_id, 0.0)

            if hasattr(re, "iloc"):

                re = re.iloc[0]

            random\_effects\_values.append(float(re))

        var\_random = (

            np.var(random\_effects\_values, ddof=1)

            if len(random\_effects\_values) > 1

            else 0.0

        )

    # 残差方差

    var\_residual = model\_result.scale  # 残差方差

    # 计算 R²

    r2\_marginal = var\_fixed / (var\_fixed + var\_random + var\_residual)

    r2\_conditional = (var\_fixed + var\_random) / (var\_fixed + var\_random + var\_residual)

    return (

        r2\_marginal,

        r2\_conditional,

        {

            "var\_fixed": var\_fixed,

            "var\_random": var\_random,

            "var\_residual": var\_residual,

        },

    )

# 计算 Nakagawa R²

r2\_m\_nak, r2\_c\_nak, variance\_components = nakagawa\_r2(res, y, X, groups)

print("=== Nakagawa & Schielzeth R² ===")

print(f"边际 R² (固定效应): {r2\_m\_nak:.4f}")

print(f"条件 R² (固定+随机效应): {r2\_c\_nak:.4f}")

print("\n方差分解:")

print(f"固定效应方差: {variance\_components['var\_fixed']:.6f}")

print(f"随机效应方差: {variance\_components['var\_random']:.6f}")

print(f"残差方差: {variance\_components['var\_residual']:.6f}")

# %%

def conditional\_r2(model\_result, y\_observed, X\_matrix, groups):

    """

    条件 R² = (固定效应 + 随机效应)解释的方差 / 总方差

    """

    # 包含随机效应的完整预测

    y\_full\_pred = []

    for i, group\_id in enumerate(groups):

        # 固定效应预测

        y\_fixed\_i = model\_result.predict(exog=X\_matrix.iloc[[i]])

        # 随机效应

        random\_effect = model\_result.random\_effects.get(group\_id, 0.0)

        if hasattr(random\_effect, "iloc"):

            random\_effect = random\_effect.iloc[0]

        y\_full\_pred.append(float(y\_fixed\_i) + float(random\_effect))

    y\_full\_pred = np.array(y\_full\_pred)

    # 计算残差方差

    residuals = y\_observed - y\_full\_pred

    var\_residual = np.var(residuals, ddof=1)

    var\_total = np.var(y\_observed, ddof=1)

    r2\_conditional = 1 - (var\_residual / var\_total)

    return r2\_conditional, y\_full\_pred

# 计算条件 R²

r2\_c, y\_pred\_full = conditional\_r2(res, y, X, groups)

print(f"条件 R² (固定+随机效应): {r2\_c:.4f}")

# %%

# 完整的 R² 分析

print("=== 混合效应模型 R² 分析 ===")

# 1. 基础相关系数

y\_pred\_fixed = res.predict(exog=X)

corr\_fixed = np.corrcoef(y, y\_pred\_fixed)[0, 1]

r2\_basic = corr\_fixed\*\*2

print(f"1. 基础 R² (固定效应预测): {r2\_basic:.4f}")

print(f"   观测值与固定效应预测相关系数: {corr\_fixed:.4f}")

# 2. Nakagawa 方法

r2\_m\_nak, r2\_c\_nak, var\_comp = nakagawa\_r2(res, y, X, groups)

print(f"\n2. Nakagawa & Schielzeth R²:")

print(f"   边际 R² (固定效应): {r2\_m\_nak:.4f}")

print(f"   条件 R² (固定+随机): {r2\_c\_nak:.4f}")

# 3. 方差解释

total\_var = var\_comp["var\_fixed"] + var\_comp["var\_random"] + var\_comp["var\_residual"]

print(f"\n3. 方差分解 (总方差 = {total\_var:.6f}):")

print(f"   固定效应解释: {var\_comp['var\_fixed']/total\_var:.1%}")

print(f"   随机效应解释: {var\_comp['var\_random']/total\_var:.1%}")

print(f"   残差(未解释): {var\_comp['var\_residual']/total\_var:.1%}")

# 4. 模型拟合质量

print(f"\n4. 模型信息:")

print(f"   AIC: {res.aic:.2f}")

print(f"   BIC: {res.bic:.2f}")

print(f"   Log-likelihood: {res.llf:.2f}")

## 问题二代码

### Q2\_final.py

# %%

import pandas as pd

import numpy as np

# %%

data = pd.read\_csv("../Q1/男胎检测数据\_预处理后.csv")

data

# %% [markdown]

# # 数据预处理与筛选

# %%

# 选择分析所需的字段

data = data[["孕妇代码", "检测孕周", "孕妇BMI", "Y染色体浓度"]]

# 数据预处理：参考test.py的方法

def preprocess\_data(df):

    """数据预处理函数，参考test.py的实现"""

    df = df.copy()

    original\_len = len(df)

    print(f"原始数据: {original\_len}条记录")

    # 1. 过滤检测孕周：保留 [10, 24) 区间 (与test copy.py保持一致)

    df = df[(df["检测孕周"] >= 10) & (df["检测孕周"] < 24)]

    print(f"孕周过滤 [10w, 24w): {original\_len} -> {len(df)}条记录")

    # 2. 过滤BMI极端异常值

    # 使用IQR方法识别异常值

    Q1 = df["孕妇BMI"].quantile(0.25)

    Q3 = df["孕妇BMI"].quantile(0.75)

    IQR = Q3 - Q1

    # 定义异常值范围（1.2倍IQR规则，与test copy.py保持一致）

    lower\_bound = max(Q1 - 1.2 \* IQR, 15.0)  # BMI不低于15

    upper\_bound = min(Q3 + 1.2 \* IQR, 50.0)  # BMI不高于50

    before\_bmi\_filter = len(df)

    df = df[(df["孕妇BMI"] >= lower\_bound) & (df["孕妇BMI"] <= upper\_bound)]

    print(

        f"BMI异常值过滤 [{lower\_bound:.1f}, {upper\_bound:.1f}]: {before\_bmi\_filter} -> {len(df)}条记录"

    )

    # 3. 过滤其他必要的缺失值

    df = df.dropna(subset=["孕妇代码", "孕妇BMI", "检测孕周", "Y染色体浓度"])

    print(f"缺失值过滤后: {len(df)}条记录")

    print(f"最终数据: {len(df)}条记录 (过滤掉{original\_len - len(df)}条)")

    return df.reset\_index(drop=True)

# 执行数据预处理

data = preprocess\_data(data)

data

# %%

# 查看预处理后的数据统计

data.describe()

# %%

# 额外的数据质量检查

print("=== 数据质量检查 ===")

print(f"总记录数: {len(data)}")

print(f"唯一孕妇数: {data['孕妇代码'].nunique()}")

print(f"检测孕周范围: [{data['检测孕周'].min():.1f}, {data['检测孕周'].max():.1f}]")

print(f"BMI范围: [{data['孕妇BMI'].min():.1f}, {data['孕妇BMI'].max():.1f}]")

print(

    f"Y染色体浓度范围: [{data['Y染色体浓度'].min():.4f}, {data['Y染色体浓度'].max():.4f}]"

)

# 检查重复检测情况

duplicate\_counts = data["孕妇代码"].value\_counts()

multi\_test\_patients = (duplicate\_counts > 1).sum()

print(f"有多次检测的孕妇数: {multi\_test\_patients}")

print(f"最多检测次数: {duplicate\_counts.max()}")

# %% [markdown]

# ## 首次达标区间构造

# %%

# 阈值

thr = 0.04

# 清洗与排序

data = data.copy()

data["检测孕周"] = pd.to\_numeric(data["检测孕周"], errors="coerce")

data["Y染色体浓度"] = pd.to\_numeric(data["Y染色体浓度"], errors="coerce")

data = data.dropna(subset=["检测孕周", "Y染色体浓度"])

data = data.sort\_values(["孕妇代码", "检测孕周"], ascending=[True, True])

# 分组构造首次达标区间

def first\_hit\_interval(g: pd.DataFrame) -> pd.Series:

    bmi = g["孕妇BMI"].dropna().iloc[0] if g["孕妇BMI"].notna().any() else np.nan

    # convert to plain numpy float arrays to avoid ExtensionArray comparison issues

    y = g["Y染色体浓度"].to\_numpy(dtype=float)

    weeks = g["检测孕周"].to\_numpy(dtype=float)

    hit\_idx = np.where(y >= thr)[0]

    if hit\_idx.size == 0:

        # 右删失：T\* > C

        C = weeks[-1]

        L, R, censor = C, np.inf, "right"

    else:

        r\_pos = int(hit\_idx[0])

        R = weeks[r\_pos]

        if r\_pos == 0:

            # 左删失：(-inf, R]

            L, censor = -np.inf, "left"

        else:

            # 区间删失：(L, R]

            L, censor = weeks[r\_pos - 1], "interval"

    return pd.Series(

        {"id": g["孕妇代码"].iloc[0], "BMI": bmi, "L": L, "R": R, "censor\_type": censor}

    )

intervals = (

    data.groupby("孕妇代码", group\_keys=False)

    .apply(first\_hit\_interval)

    .reset\_index(drop=True)

)

intervals

# %% [markdown]

# # 条件推理树（CIT）

# %% [markdown]

# ## 特征构造

# %%

# 特征构造（每人一条）

# 右删失下界代理的增量（周）

delta\_week = 1.0

feat = intervals.copy()

# 删失标记：right=1，其余=0

feat["censored"] = (feat["censor\_type"] == "right").astype(int)

# 达标时间代理：

# - 区间/左删失：用 R（首次达标周）

# - 右删失：用 C+Δ，这里 C=L（上一步构造里 right 情况把 L 设为最后检测周）

feat["t\_proxy"] = np.where(feat["censored"] == 1, feat["L"] + delta\_week, feat["R"])

# 简单 z-score 标准化（避免 std=0 的数值问题）

def zscore(s: pd.Series) -> pd.Series:

    mu = s.mean()

    sigma = s.std(ddof=0)

    sigma = 1.0 if (not np.isfinite(sigma) or sigma == 0) else sigma

    return (s - mu) / sigma

feat["x1"] = zscore(feat["BMI"])

feat["x2"] = zscore(feat["t\_proxy"])

# 右删失样本较低权重

feat["sample\_weight"] = np.where(feat["censored"] == 1, 0.5, 1.0)

# 输出特征表（保留原值便于核查）

features = feat[["id", "x1", "x2", "censored", "sample\_weight", "BMI", "t\_proxy"]]

features

# %% [markdown]

# ## 分组

# %%

from typing import Tuple

from scipy.stats import mannwhitneyu

# 基于 BMI 的条件推理树（CIT）递归分组

df = features.copy()

df = df.replace([np.inf, -np.inf], np.nan).dropna(subset=["BMI", "t\_proxy"])

df = df.sort\_values("BMI").reset\_index(drop=True)

# 超参数 (adjusted for better model performance)

alpha = 0.3  # 显著性水平 (increased to reduce over-splitting)

max\_depth = 5  # 最大深度

min\_leaf = int(0.1 \* len(df))  # 叶子最小样本数 (increased to 10% for fewer groups)

n\_thresholds = 64  # 候选阈值个数（精细化搜索颗粒度）

def best\_split(data) -> Tuple[float, float, int, int]:

    bmis = data["BMI"].to\_numpy()

    y = data["t\_proxy"].to\_numpy()

    if len(data) < 2 \* min\_leaf:

        return np.nan, 1.0, len(data), 0

    qs = np.linspace(0.1, 0.9, n\_thresholds)

    cands = np.unique(np.quantile(bmis, qs))

    # 只保留左右子节点都满足 min\_leaf 的阈值

    valid\_cands = []

    for thr in cands:

        nl = (bmis <= thr).sum()

        nr = len(bmis) - nl

        if nl >= min\_leaf and nr >= min\_leaf:

            valid\_cands.append(thr)

    if not valid\_cands:

        return np.nan, 1.0, len(data), 0

    best\_p, best\_thr, best\_nl, best\_nr = 1.0, np.nan, 0, 0

    for thr in valid\_cands:

        left = y[bmis <= thr]

        right = y[bmis > thr]

        \_, p = mannwhitneyu(left, right, alternative="two-sided", method="asymptotic")

        if p < best\_p:

            best\_p, best\_thr = p, thr

            best\_nl, best\_nr = len(left), len(right)

    # 返回的第二个值用作 grow 中的显著性判定，这里直接用原始 p

    return best\_thr, best\_p, best\_nl, best\_nr

leaves = []  # 存储叶子结点区间及统计

def grow(data, depth: int, lo: float, hi: float):

    thr, p\_adj, nl, nr = best\_split(data)

    # 停止条件：深度、无有效阈值或不显著

    if (depth >= max\_depth) or (not np.isfinite(thr)) or (p\_adj >= alpha):

        leaves.append(

            {

                "lo": lo,

                "hi": hi,

                "depth": depth,

                "n": int(len(data)),

                "bmi\_min": float(data["BMI"].min()),

                "bmi\_max": float(data["BMI"].max()),

                "t\_median": float(data["t\_proxy"].median()),

            }

        )

        return

    left = data[data["BMI"] <= thr]

    right = data[data["BMI"] > thr]

    grow(left, depth + 1, lo, thr)

    grow(right, depth + 1, thr, hi)

grow(df, depth=0, lo=-np.inf, hi=np.inf)

# 规范化叶子顺序并赋组号

leaves = sorted(leaves, key=lambda x: (x["lo"], x["hi"]))

for gid, leaf in enumerate(leaves):

    leaf["group"] = gid

def map\_group\_by\_bmi(bmi):

    if pd.isna(bmi):

        return np.nan

    for leaf in leaves:

        if leaf["lo"] < bmi <= leaf["hi"]:

            return leaf["group"]

    # 边界兜底：若未命中则按最接近区间归类（通常不会触发）

    return len(leaves) - 1

features\_cit = features.copy()

features\_cit["group"] = features\_cit["BMI"].apply(map\_group\_by\_bmi)

features\_cit

# %%

# 分组区间与统计输出

leaves\_df = pd.DataFrame(leaves).sort\_values(["group"]).reset\_index(drop=True)

def \_fmt(x):

    if not np.isfinite(x):

        return "-inf" if x < 0 else "+inf"

    return f"{x:.2f}"

leaves\_df["bmi\_interval"] = leaves\_df.apply(

    lambda r: f"({\_fmt(r['lo'])}, {\_fmt(r['hi'])}]", axis=1

)

stats = (

    features\_cit.groupby("group", as\_index=False)

    .agg(

        n=("BMI", "size"),

        bmi\_min=("BMI", "min"),

        bmi\_median=("BMI", "median"),

        bmi\_max=("BMI", "max"),

        t\_median=("t\_proxy", "median"),

        censored\_rate=("censored", "mean"),

    )

    .merge(leaves\_df[["group", "bmi\_interval"]], on="group", how="left")

    .sort\_values("group")

)

stats, features\_cit

# %% [markdown]

# # 最佳nipt时点

# %% [markdown]

# ## 按组构建区间删失对象

# %%

# 将分组信息并入区间数据

iv = intervals.merge(features\_cit[["id", "group"]], on="id", how="left")

# Turnbull NPMLE（区间删失）实现

def turnbull\_npmle(

    group\_df: pd.DataFrame,

    max\_iter: int = 5000,

    tol: float = 1e-8,

    eps\_mass: float = 1e-12,

):

    """

    输入:

      group\_df 需包含列: L, R, censor\_type（left/right/interval）

    返回:

      dict: {support, mass, S, F}

    """

    g = group\_df.copy()

    L = g["L"].to\_numpy(dtype=float)

    R = g["R"].to\_numpy(dtype=float)

    # 支持点 = 所有有限端点的并集（按升序）

    finite\_L = L[np.isfinite(L)]

    finite\_R = R[np.isfinite(R)]

    support = (

        np.unique(np.concatenate([finite\_L, finite\_R]))

        if (finite\_L.size + finite\_R.size) > 0

        else np.array([])

    )

    n = len(g)

    K = len(support)

    # 如果没有任何有限端点（极端：全右删失或全左删失），返回常数 S(t)=1 或 0 的退化情形

    if K == 0:

        # 这里将返回一个空支持集；调用方用阶梯函数求值时会得到常数 S=1（右删失主导）

        return {

            "support": support,

            "mass": np.array([]),

            "S": np.array([]),

            "F": np.array([]),

        }

    # 每个观测对应的可行支持点索引集合 J\_i

    J = []

    for Li, Ri in zip(L, R):

        if np.isneginf(Li) and np.isfinite(Ri):

            idx = np.where(support <= Ri)[0]  # 左删失：(-inf, R]

        elif np.isposinf(Ri) and np.isfinite(Li):

            idx = np.where(support > Li)[0]  # 右删失：(L, +inf)

        else:

            # 区间删失：(L, R]

            idx = np.where((support > Li) & (support <= Ri))[0]

        if idx.size == 0:

            # 若该样本与任何支持点都不相交，强制用最近的一个支持点兜底，避免空集导致算法崩溃

            # 选择与(L,R]最近的支持点

            near = np.searchsorted(support, np.nan\_to\_num(Ri, nan=Li))  # 尝试靠近 R

            near = min(max(0, near - 1), K - 1)

            idx = np.array([near], dtype=int)

        J.append(idx)

    # 初始化概率质量：均匀

    p = np.full(K, 1.0 / K, dtype=float)

    # EM 迭代

    for it in range(max\_iter):

        counts = np.zeros(K, dtype=float)

        for idx in J:

            denom = p[idx].sum()

            if denom <= 0:

                # 退化：当前区间无质量，按均匀分配到该观测可行集

                counts[idx] += 1.0 / len(idx)

            else:

                counts[idx] += p[idx] / denom

        p\_new = counts / n

        # 去除极小质量并重标化（数值稳定）

        p\_new[p\_new < eps\_mass] = 0.0

        s = p\_new.sum()

        if s <= 0:

            # 罕见退化，回退为均匀

            p\_new = np.full(K, 1.0 / K, dtype=float)

            s = 1.0

        else:

            p\_new = p\_new / s

        # 收敛判据

        if np.linalg.norm(p\_new - p, ord=1) < tol:

            p = p\_new

            break

        p = p\_new

    # 生存函数（右连续阶梯）：S(t\_j) = 1 - sum\_{k<=j} p\_k

    F = np.cumsum(p)

    S = 1.0 - F

    return {

        "support": support,

        "mass": p,

        "S": S,

        "F": F,

    }

# 生存阶梯函数求值

def eval\_step\_S(

    t: np.ndarray, support: np.ndarray, S\_support: np.ndarray

) -> np.ndarray:

    """

    在任意 t 上按阶梯函数求 S(t)。

    规则：对于 t < support[0]，S=1；对 support[j-1] < t <= support[j]，S=S\_support[j]

    """

    if support.size == 0:

        return np.ones\_like(t, dtype=float)

    idx = np.searchsorted(support, t, side="right") - 1

    idx = np.clip(idx, -1, len(support) - 1)

    out = np.empty\_like(t, dtype=float)

    # t 在首点之前

    mask\_before = idx < 0

    out[mask\_before] = 1.0

    # 其他

    mask\_other = ~mask\_before

    out[mask\_other] = S\_support[idx[mask\_other]]

    return out

# 构建每组的 NPMLE 结果

group\_surv = {}

for g in sorted(iv["group"].dropna().unique()):

    df\_g = iv[iv["group"] == g][["L", "R", "censor\_type"]].reset\_index(drop=True)

    res = turnbull\_npmle(df\_g)

    group\_surv[g] = res

# 将 NPMLE 结果整理为表（支持点上）

npmle\_rows = []

for g, res in group\_surv.items():

    sup = res["support"]

    mass = res["mass"]

    S = res["S"]

    F = res["F"]

    if sup.size == 0:

        # 无支持点时用空行占位

        npmle\_rows.append(

            pd.DataFrame(

                {

                    "group": [g],

                    "t": [np.nan],

                    "S\_npmle": [1.0],

                    "F\_npmle": [0.0],

                    "mass": [0.0],

                }

            )

        )

    else:

        npmle\_rows.append(

            pd.DataFrame(

                {"group": g, "t": sup, "S\_npmle": S, "F\_npmle": F, "mass": mass}

            )

        )

npmle\_df = pd.concat(npmle\_rows, ignore\_index=True)

# 选择统一评估网格（按题意 10~24 周，与数据过滤范围保持一致）

t\_grid = np.arange(10.0, 24.0 + 1e-9, 0.25)  # 精细化搜索颗粒度：步长从0.5减小到0.25

# 在统一网格上评估 S\_g(t), F\_g(t)

rows\_grid = []

for g, res in group\_surv.items():

    Sg = eval\_step\_S(t\_grid, res["support"], res["S"])

    rows\_grid.append(

        pd.DataFrame({"group": g, "t": t\_grid, "S\_npmle": Sg, "F\_npmle": 1.0 - Sg})

    )

sf\_grid\_npmle = pd.concat(rows\_grid, ignore\_index=True)

# —— 离散时间生存模型（cloglog / logit 危害）用于平滑验证 ——

# 用 NPMLE 推导离散 hazard: h\_j = p\_j / S\_{j-1}; 以 at\_risk\_j = n \* S\_{j-1} 作为试验次数（频数权重）

try:

    import statsmodels.api as sm

    has\_sm = True

except Exception as e:

    has\_sm = False

    print("未检测到 statsmodels，已跳过平滑验证。若需运行：pip install statsmodels")

smooth\_rows = []

if has\_sm:

    for g, res in group\_surv.items():

        sup = res["support"]

        p = res["mass"]

        S = res["S"]

        if sup.size == 0:

            # 无支持点：S 恒为 1

            smooth\_rows.append(

                pd.DataFrame(

                    {"group": g, "t": t\_grid, "S\_cloglog": 1.0, "S\_logit": 1.0}

                )

            )

            continue

        # 计算离散 hazard 与风险集大小

        # S\_prev[0]=1, S\_prev[1]=S[0], ...

        S\_prev = np.empty\_like(S)

        S\_prev[0] = 1.0

        S\_prev[1:] = S[:-1]

        # 避免除零

        S\_prev = np.clip(S\_prev, 1e-12, 1.0)

        h = p / S\_prev

        h = np.clip(h, 1e-9, 1 - 1e-9)

        # 以“比例 + 频数权重”的形式拟合 GLM

        # X: [1, t, t^2] 轻度平滑（可换成更高次或样条）

        X = np.column\_stack([np.ones\_like(sup), sup, sup\*\*2])

        # at-risk 人数用 n \* S\_prev，n = 组内样本量

        n\_group = int(iv[iv["group"] == g].shape[0])

        at\_risk = np.maximum(1.0, n\_group \* S\_prev)  # 至少为 1，稳定拟合

        # cloglog

        glm\_cloglog = sm.GLM(

            h,

            X,

            family=sm.families.Binomial(link=sm.families.links.cloglog()),

            freq\_weights=at\_risk,

        ).fit(disp=0)

        # logit

        glm\_logit = sm.GLM(

            h,

            X,

            family=sm.families.Binomial(link=sm.families.links.logit()),

            freq\_weights=at\_risk,

        ).fit(disp=0)

        # 预测在统一网格上的 hazard，并累乘得到 S(t)

        Xg = np.column\_stack([np.ones\_like(t\_grid), t\_grid, t\_grid\*\*2])

        h\_cl = np.clip(glm\_cloglog.predict(Xg), 1e-9, 1 - 1e-9)

        h\_lo = np.clip(glm\_logit.predict(Xg), 1e-9, 1 - 1e-9)

        # S(t\_k) = ∏\_{j<=k} (1 - h\_j)

        S\_cl = np.cumprod(1.0 - h\_cl)

        S\_lo = np.cumprod(1.0 - h\_lo)

        smooth\_rows.append(

            pd.DataFrame({"group": g, "t": t\_grid, "S\_cloglog": S\_cl, "S\_logit": S\_lo})

        )

smooth\_df = pd.concat(smooth\_rows, ignore\_index=True) if smooth\_rows else pd.DataFrame()

# 合并网格上的 NPMLE 与平滑结果，便于对比

if not smooth\_df.empty:

    compare\_df = sf\_grid\_npmle.merge(smooth\_df, on=["group", "t"], how="left")

else:

    compare\_df = sf\_grid\_npmle.copy()

# 输出：

# npmle\_df：各组在支持点上的 S/F 和概率质量

# sf\_grid\_npmle：各组在统一网格(10~26, 步长0.5)上的 S/F（阶梯评估）

# compare\_df：若安装 statsmodels，则包含 S\_cloglog / S\_logit 的平滑对比

npmle\_df, sf\_grid\_npmle, compare\_df

# ...existing code...

# %% [markdown]

# ## 求最早达标时点

# %%

# 阈值 π\*（可调整）

pi\_star = 0.95

# 点估计：直接基于前面计算好的 sf\_grid\_npmle（含每组在统一网格上的 F\_npmle）

def earliest\_t\_from\_grid(df\_grid: pd.DataFrame, pi\_star: float) -> pd.DataFrame:

    out = []

    for g, d in df\_grid.groupby("group", sort=True):

        d = d.sort\_values("t")

        mask = d["F\_npmle"] >= pi\_star

        if mask.any():

            t\_star = float(d.loc[mask, "t"].iloc[0])

        else:

            t\_star = np.nan  # 网格上尚未达到阈值

        out.append({"group": g, "t\_star": t\_star})

    return pd.DataFrame(out)

tstar\_hat\_df = earliest\_t\_from\_grid(sf\_grid\_npmle, pi\_star)

tstar\_hat\_df

# %%

# 提速版 Bootstrap：多核并行 + 进度条

try:

    from tqdm.auto import tqdm

    \_has\_tqdm = True

except Exception:

    \_has\_tqdm = False

    def tqdm(x, \*\*kwargs):

        return x

try:

    from joblib import Parallel, delayed

    \_has\_joblib = True

except Exception:

    \_has\_joblib = False

def bootstrap\_tstar(

    iv\_df: pd.DataFrame,

    B: int = 500,

    pi\_star: float = 0.95,

    t\_grid: np.ndarray = None,

    random\_state: int = 42,

    n\_jobs: int = 6,  # -1 使用全部CPU；1为单核

):

    """

    加速策略：

    - 预生成每组的索引矩阵 (B, m)，避免循环中反复调用随机采样

    - 并行按replicate维度划分任务（每个replicate内部遍历各组）

    - 进度显示：tqdm

    """

    rng = np.random.default\_rng(random\_state)

    if t\_grid is None:

        t\_grid = np.arange(10.0, 24.0 + 1e-9, 0.25)  # 同步精细化步长，与数据范围一致

    groups = sorted(iv\_df["group"].dropna().unique())

    # 每组仅保留"每位孕妇一行"的(L,R)（censor\_type由L/R含无穷表示）

    grp\_data = {}

    idx\_mats = {}

    for g in groups:

        iv\_g = (

            iv\_df[iv\_df["group"] == g][["id", "L", "R"]]

            .dropna(subset=["L", "R"])

            .drop\_duplicates(subset=["id"])

            .reset\_index(drop=True)

        )

        L = iv\_g["L"].to\_numpy(dtype=float)

        R = iv\_g["R"].to\_numpy(dtype=float)

        m = len(L)

        grp\_data[g] = {"L": L, "R": R}

        if m == 0:

            idx\_mats[g] = None

        else:

            # 预生成 B 组采样下标（每次抽m个，有放回）

            idx\_mats[g] = rng.integers(0, m, size=(B, m))

    def boot\_once(j: int) -> np.ndarray:

        """单次replicate：返回按groups顺序的 t\*\_g 数组"""

        tstars = []

        for g in groups:

            data = grp\_data[g]

            idx\_mat = idx\_mats[g]

            if idx\_mat is None:

                tstars.append(np.nan)

                continue

            idx = idx\_mat[j]

            Lb = data["L"][idx]

            Rb = data["R"][idx]

            # 该replicate的NPMLE

            boot\_df = pd.DataFrame({"L": Lb, "R": Rb})

            res = turnbull\_npmle(boot\_df)

            # 网格上评估F并取最早达标时点

            Sg = eval\_step\_S(t\_grid, res["support"], res["S"])

            Fg = 1.0 - Sg

            hit = np.where(Fg >= pi\_star)[0]

            t\_star = float(t\_grid[hit[0]]) if hit.size > 0 else np.nan

            tstars.append(t\_star)

        return np.array(tstars, dtype=float)

    # 并行或顺序执行，带进度条

    it = range(B)

    it = tqdm(it, desc="Bootstrap", unit="rep") if \_has\_tqdm else it

    if \_has\_joblib and (n\_jobs != 1):

        res\_list = Parallel(n\_jobs=n\_jobs, prefer="processes")(

            delayed(boot\_once)(j) for j in it

        )

    else:

        res\_list = [boot\_once(j) for j in it]

    # 汇总为矩阵 (B, G)

    res\_mat = np.vstack(res\_list) if len(res\_list) > 0 else np.empty((0, len(groups)))

    # 95%分位（忽略NaN）

    rows = []  # 修复：改为列表而不是字典

    boots = {}

    for gi, g in enumerate(groups):

        col = res\_mat[:, gi] if res\_mat.size > 0 else np.array([])

        boots[g] = col.tolist()

        col\_valid = col[np.isfinite(col)]

        if col\_valid.size == 0:

            lo = hi = np.nan

        else:

            lo, hi = np.nanpercentile(col\_valid, [2.5, 97.5])

        rows.append({"group": g, "t\_star\_lo": lo, "t\_star\_hi": hi})

    ci\_df = pd.DataFrame(rows)

    return ci\_df, boots

# 运行 bootstrap（多核 + 进度条）

ci\_df, boot\_store = bootstrap\_tstar(

    iv, B=500, pi\_star=pi\_star, t\_grid=t\_grid, random\_state=2025, n\_jobs=-1

)

# 合并点估计与区间

tstar\_summary = (

    tstar\_hat\_df.merge(ci\_df, on="group", how="left")

    .sort\_values("group")

    .reset\_index(drop=True)

)

tstar\_summary

# %%

# 两阶段联合优化：定义与运行

import numpy as np

import pandas as pd

def analyze\_optimization\_results(

    opt\_results: dict, stats\_df: pd.DataFrame

) -> pd.DataFrame:

    rows = []

    for g, s in opt\_results.items():

        info = stats\_df[stats\_df["group"] == g].iloc[0]

        rows.append(

            {

                "组号": int(g),

                "BMI区间": info["bmi\_interval"],

                "样本量": int(info["n"]),

                "最优t0(周)": round(float(s["t0"]), 1),

                "保底t\*g(周)": round(float(s["t\_star\_g"]), 1),

                "早期达标率": f"{float(s['F\_t0']):.1%}",

                "期望检测时间(周)": round(float(s["ET\_t0"]), 2),

                "风险成本": round(float(s["R\_t0"]), 3),

                "总成本": round(float(s["total\_cost"]), 3),

                "提前周数": round(float(s["t\_star\_g"]) - float(s["t0"]), 1),

            }

        )

    return pd.DataFrame(rows)

def two\_stage\_optimization(

    group\_surv: dict,

    stats\_df: pd.DataFrame,

    t\_grid: np.ndarray,

    cost\_params: dict = None,

    pi\_star: float = 0.95,

    verbose: bool = False,

):

    """两阶段联合优化：早期检测(t0) + 保底复测(t\*\_g)"""

    if cost\_params is None:

        cost\_params = {

            "w\_E": 1.0,

            "w\_13": 3.0,

            "w\_28": 5.0,

            "lambda\_val": 1.0,

            "alpha": 1.0,

        }

    results = {}

    for g in sorted(group\_surv.keys()):

        res = group\_surv[g]

        Sg = eval\_step\_S(t\_grid, res["support"], res["S"])

        Fg = 1.0 - Sg

        hit = np.where(Fg >= pi\_star)[0]

        t\_star\_g = float(t\_grid[hit[0]]) if hit.size > 0 else float(np.max(t\_grid))

        t0\_cands = t\_grid[t\_grid < t\_star\_g]

        if t0\_cands.size == 0:

            t0\_cands = t\_grid[: max(1, len(t\_grid) // 2)]

        best\_cost = np.inf

        best = None

        for t0 in t0\_cands:

            idx = int(np.argmin(np.abs(t\_grid - t0)))

            F\_t0 = float(Fg[idx])

            ET\_t0 = float(t0) \* F\_t0 + t\_star\_g \* (1.0 - F\_t0)

            w13 = cost\_params["w\_13"] if t0 >= 13 else 0.0

            w28 = cost\_params["w\_28"] if t0 >= 28 else 0.0

            R\_t0 = float(cost\_params["w\_E"]) \* (1.0 - F\_t0) + float(w13) + float(w28)

            total = (

                float(cost\_params["lambda\_val"]) \* ET\_t0

                + float(cost\_params["alpha"]) \* R\_t0

            )

            if total < best\_cost:

                best\_cost = total

                best = {

                    "t0": float(t0),

                    "t\_star\_g": float(t\_star\_g),

                    "F\_t0": float(F\_t0),

                    "ET\_t0": float(ET\_t0),

                    "R\_t0": float(R\_t0),

                    "total\_cost": float(total),

                }

        results[g] = best

        if verbose:

            print(

                f"group {g}: t0={best['t0']:.1f}, t\*={best['t\_star\_g']:.1f}, cost={best['total\_cost']:.3f}"

            )

    return results

# 运行两阶段优化（若需要，可修改成本参数）

optimization\_results = two\_stage\_optimization(

    group\_surv,

    stats,

    t\_grid,

    cost\_params={"w\_E": 1.0, "w\_13": 3.0, "w\_28": 5.0, "lambda\_val": 1.0, "alpha": 1.0},

    pi\_star=pi\_star,

    verbose=False,

)

optimization\_results

# %%

import matplotlib.pyplot as plt

# 可视化优化结果

def plot\_two\_stage\_strategy(opt\_results: dict, group\_surv: dict, t\_grid: np.ndarray):

    """绘制两阶段策略可视化图（修复子图网格与轴处理）"""

    groups = sorted(opt\_results.keys())

    G = len(groups)

    from math import ceil

    ncols = 2 if G <= 6 else 3

    nrows = ceil(G / ncols)

    fig, axes = plt.subplots(

        nrows, ncols, figsize=(5 \* ncols, 4 \* nrows), sharex=True, sharey=True

    )

    axes = np.atleast\_1d(axes).ravel()

    for i, g in enumerate(groups):

        ax = axes[i]

        # 获取该组数据

        strategy = opt\_results[g]

        res = group\_surv[g]

        # 计算F\_g(t)

        Sg = eval\_step\_S(t\_grid, res["support"], res["S"])

        Fg = 1.0 - Sg

        # 绘制F\_g(t)曲线

        ax.step(t\_grid, Fg, where="post", linewidth=2, label="F\_g(t)", color="blue")

        # 标记关键时点

        t0 = strategy["t0"]

        t\_star = strategy["t\_star\_g"]

        F\_t0 = strategy["F\_t0"]

        # t0时点

        ax.axvline(t0, color="green", linestyle="--", linewidth=2, label=f"t0={t0:.1f}")

        ax.plot([t0], [F\_t0], "go", markersize=7)

        ax.text(t0, min(1.0, F\_t0 + 0.06), f"F={F\_t0:.2f}", ha="center", color="green")

        # t\*\_g时点

        ax.axvline(

            t\_star, color="red", linestyle="--", linewidth=2, label=f"t\*={t\_star:.1f}"

        )

        ax.axhline(0.95, color="orange", linestyle=":", alpha=0.7, label="95%")

        # ==== changed: use English texts to avoid CJK rendering issues ====

        ax.set\_title(f"Group {int(g)}: Two-stage strategy")

        ax.set\_xlabel("Gestational week (wks)")

        ax.set\_ylabel("Cumulative attainment F\_g(t)")

        # ================================================================

        ax.grid(True, alpha=0.3)

        ax.set\_ylim(0, 1.05)

    # 关闭多余子图

    for k in range(G, len(axes)):

        axes[k].axis("off")

    # 合并图例

    handles, labels = axes[0].get\_legend\_handles\_labels()

    fig.legend(handles, labels, loc="upper center", ncols=4, frameon=False)

    fig.tight\_layout(rect=[0, 0, 1, 0.92])

    return fig

# 执行分析

result\_summary = analyze\_optimization\_results(optimization\_results, stats)

print("=== 两阶段联合优化结果 ===")

display(result\_summary)

# 绘制策略图

fig\_strategy = plot\_two\_stage\_strategy(optimization\_results, group\_surv, t\_grid)

plt.show()

# %% [markdown]

# ## 敏感性分析：成本参数对策略的影响

# %%

# 成本参数敏感性分析

def cost\_sensitivity\_analysis(

    group\_surv: dict, stats\_df: pd.DataFrame, t\_grid: np.ndarray, pi\_star: float = 0.95

):

    """分析不同成本参数组合对最优策略的影响"""

    # 定义参数变化范围

    param\_scenarios = {

        "保守型": {

            "w\_E": 0.5,

            "w\_13": 2.0,

            "w\_28": 3.0,

            "lambda\_val": 1.5,

            "alpha": 0.8,

        },

        "标准型": {

            "w\_E": 1.0,

            "w\_13": 3.0,

            "w\_28": 5.0,

            "lambda\_val": 1.0,

            "alpha": 1.0,

        },

        "激进型": {

            "w\_E": 1.5,

            "w\_13": 4.0,

            "w\_28": 7.0,

            "lambda\_val": 0.8,

            "alpha": 1.2,

        },

        "成本敏感": {

            "w\_E": 0.8,

            "w\_13": 1.5,

            "w\_28": 2.5,

            "lambda\_val": 2.0,

            "alpha": 0.5,

        },

    }

    sensitivity\_results = {}

    for scenario\_name, params in param\_scenarios.items():

        print(f"\n=== {scenario\_name}策略 ===")

        opt\_result = two\_stage\_optimization(

            group\_surv, stats\_df, t\_grid, cost\_params=params, pi\_star=pi\_star

        )

        sensitivity\_results[scenario\_name] = opt\_result

    return sensitivity\_results

# 执行敏感性分析

sensitivity\_results = cost\_sensitivity\_analysis(group\_surv, stats, t\_grid, pi\_star)

# 对比不同策略

comparison\_data = []

for scenario, results in sensitivity\_results.items():

    for g, strategy in results.items():

        comparison\_data.append(

            {

                "策略类型": scenario,

                "组号": int(g),

                "最优t0": strategy["t0"],

                "期望检测时间": strategy["ET\_t0"],

                "总成本": strategy["total\_cost"],

                "早期达标率": strategy["F\_t0"],

            }

        )

comparison\_df = pd.DataFrame(comparison\_data)

# 透视表展示

pivot\_t0 = comparison\_df.pivot\_table(

    values="最优t0", index="组号", columns="策略类型", aggfunc="first"

).round(1)

pivot\_cost = comparison\_df.pivot\_table(

    values="总成本", index="组号", columns="策略类型", aggfunc="first"

).round(3)

print("\n=== 不同策略下的最优t0对比(周) ===")

display(pivot\_t0)

print("\n=== 不同策略下的总成本对比 ===")

display(pivot\_cost)

# %% [markdown]

# ## 实际应用建议生成

# %%

# 生成实际应用建议

def generate\_practical\_recommendations(

    opt\_results: dict, stats\_df: pd.DataFrame, sensitivity\_results: dict = None

):

    """生成实际临床应用建议"""

    recommendations = []

    for g, strategy in opt\_results.items():

        group\_info = stats\_df[stats\_df["group"] == g].iloc[0]

        # 基础建议

        t0\_opt = strategy["t0"]

        t\_star = strategy["t\_star\_g"]

        early\_rate = strategy["F\_t0"]

        # 建议分类

        if early\_rate >= 0.8:

            advice\_level = "推荐"

            risk\_note = "早期检测成功率高"

        elif early\_rate >= 0.6:

            advice\_level = "建议"

            risk\_note = "早期检测中等成功率，需关注复测"

        else:

            advice\_level = "谨慎"

            risk\_note = "早期检测成功率较低，强化复测方案"

        # 时间窗口建议

        if t0\_opt < 12:

            timing\_advice = f"在{t0\_opt:.1f}周进行首次检测(早期窗口)"

        elif t0\_opt < 16:

            timing\_advice = f"在{t0\_opt:.1f}周进行首次检测(标准窗口)"

        else:

            timing\_advice = f"在{t0\_opt:.1f}周进行首次检测(延后窗口)"

        recommendations.append(

            {

                "组别": f"组{int(g)}",

                "BMI范围": group\_info["bmi\_interval"],

                "样本特征": f"N={int(group\_info['n'])}, 删失率{group\_info['censored\_rate']:.1%}",

                "推荐等级": advice\_level,

                "首次检测时间": f"{t0\_opt:.1f}周",

                "保底检测时间": f"{t\_star:.1f}周",

                "早期成功率": f"{early\_rate:.1%}",

                "临床建议": f"{timing\_advice}，{risk\_note}",

                "备注": f"预期节省{t\_star-t0\_opt:.1f}周检测时间",

            }

        )

    rec\_df = pd.DataFrame(recommendations)

    # 生成总体建议

    overall\_advice = f"""

=== 临床实施建议 ===

1. 个性化检测策略：

   - 根据孕妇BMI分组，采用不同的两阶段检测方案

   - 低BMI组: 早期检测时点可适当提前

   - 高BMI组: 早期检测时点需适当推迟，加强保底检测

2. 质量控制要点：

   - 确保早期检测的技术可靠性

   - 建立完善的复测跟踪机制

   - 设置风险阈值预警系统

3. 成本效益考量：

   - 早期检测可显著减少整体检测时间

   - 需平衡检测成本与临床效果

   - 建议建立动态成本调节机制

4. 实施监控：

   - 定期评估各组实际达标率

   - 监控成本控制效果

   - 根据实际情况调整参数

"""

    return rec\_df, overall\_advice

# 生成应用建议

recommendations\_df, overall\_advice = generate\_practical\_recommendations(

    optimization\_results, stats, sensitivity\_results

)

print("=== 个性化检测方案建议 ===")

display(recommendations\_df)

print(overall\_advice)

# %%

def calculate\_composite\_risk\_score(

    opt\_results: dict,

    group\_surv: dict,

    stats\_df: pd.DataFrame,

    t\_grid: np.ndarray,

    individual\_data: pd.DataFrame = None,

):

    """

    计算综合风险评分 (CRS)

    CRS = α·时间风险 + β·检测风险 + γ·成本风险 + δ·临床风险

    参数:

    - α, β, γ, δ: 权重系数，和为1

    - 时间风险: 基于期望检测时间的标准化分数

    - 检测风险: 基于早期检测失败概率

    - 成本风险: 基于总成本的标准化分数

    - 临床风险: 基于删失率和样本不确定性

    """

    # 权重设置（可根据临床重要性调整）

    weights = {

        "time\_risk": 0.3,  # 时间延误风险

        "detection\_risk": 0.4,  # 检测失败风险

        "cost\_risk": 0.2,  # 经济成本风险

        "clinical\_risk": 0.1,  # 临床不确定性风险

    }

    risk\_scores = {}

    risk\_components = {}

    # 收集各组数据用于标准化

    all\_ET = [s["ET\_t0"] for s in opt\_results.values()]

    all\_costs = [s["total\_cost"] for s in opt\_results.values()]

    all\_failure\_rates = [1 - s["F\_t0"] for s in opt\_results.values()]

    all\_censored\_rates = [

        stats\_df[stats\_df["group"] == g]["censored\_rate"].iloc[0]

        for g in opt\_results.keys()

    ]

    # 标准化函数（0-1范围，越高风险越大）

    def normalize\_risk(values, reverse=False):

        values = np.array(values)

        if values.std() == 0:

            return np.zeros\_like(values)

        normalized = (values - values.min()) / (values.max() - values.min())

        return 1 - normalized if reverse else normalized

    # 标准化各风险指标

    time\_risks = normalize\_risk(all\_ET)  # 期望时间越长风险越高

    detection\_risks = np.array(all\_failure\_rates)  # 失败率本身就是风险

    cost\_risks = normalize\_risk(all\_costs)  # 成本越高风险越高

    clinical\_risks = normalize\_risk(all\_censored\_rates)  # 删失率越高不确定性越大

    for i, g in enumerate(sorted(opt\_results.keys())):

        # 计算各组件风险分数

        components = {

            "time\_risk": float(time\_risks[i]),

            "detection\_risk": float(detection\_risks[i]),

            "cost\_risk": float(cost\_risks[i]),

            "clinical\_risk": float(clinical\_risks[i]),

        }

        # 计算综合风险分数

        crs = sum(weights[k] \* components[k] for k in weights.keys())

        # 风险等级划分

        if crs <= 0.3:

            risk\_level = "低风险"

            risk\_color = "green"

        elif crs <= 0.6:

            risk\_level = "中等风险"

            risk\_color = "orange"

        else:

            risk\_level = "高风险"

            risk\_color = "red"

        risk\_scores[g] = {

            "CRS": float(crs),

            "risk\_level": risk\_level,

            "risk\_color": risk\_color,

            "components": components,

        }

        risk\_components[g] = components

    return risk\_scores, risk\_components

def evaluate\_model\_performance(risk\_scores: dict, opt\_results: dict):

    """

    基于风险评分评价模型整体性能

    """

    crs\_values = [score["CRS"] for score in risk\_scores.values()]

    # 模型性能指标

    performance\_metrics = {

        "平均风险分数": np.mean(crs\_values),

        "风险分数标准差": np.std(crs\_values),

        "最高风险分数": np.max(crs\_values),

        "最低风险分数": np.min(crs\_values),

        "高风险组比例": np.mean(

            [1 if s["CRS"] > 0.6 else 0 for s in risk\_scores.values()]

        ),

        "低风险组比例": np.mean(

            [1 if s["CRS"] <= 0.3 else 0 for s in risk\_scores.values()]

        ),

    }

    # 模型质量评价

    avg\_risk = performance\_metrics["平均风险分数"]

    risk\_dispersion = performance\_metrics["风险分数标准差"]

    if avg\_risk <= 0.4 and risk\_dispersion <= 0.2:

        model\_grade = "优秀"

        model\_score = 90 + (1 - avg\_risk) \* 10

    elif avg\_risk <= 0.5 and risk\_dispersion <= 0.3:

        model\_grade = "良好"

        model\_score = 80 + (1 - avg\_risk) \* 10

    elif avg\_risk <= 0.6:

        model\_grade = "一般"

        model\_score = 70 + (1 - avg\_risk) \* 10

    else:

        model\_grade = "需要改进"

        model\_score = max(50, 60 + (1 - avg\_risk) \* 10)

    performance\_metrics["模型评级"] = model\_grade

    performance\_metrics["模型得分"] = round(model\_score, 1)

    return performance\_metrics

# 计算综合风险评分

risk\_scores, risk\_components = calculate\_composite\_risk\_score(

    optimization\_results, group\_surv, stats, t\_grid

)

# 评价模型性能

model\_performance = evaluate\_model\_performance(risk\_scores, optimization\_results)

print("=== 综合风险评分结果 ===")

risk\_summary = []

for g, score\_info in risk\_scores.items():

    group\_stats = stats[stats["group"] == g].iloc[0]

    strategy = optimization\_results[g]

    risk\_summary.append(

        {

            "组别": f"组{int(g)}",

            "BMI范围": group\_stats["bmi\_interval"],

            "综合风险分数(CRS)": f"{score\_info['CRS']:.3f}",

            "风险等级": score\_info["risk\_level"],

            "时间风险": f"{score\_info['components']['time\_risk']:.2f}",

            "检测风险": f"{score\_info['components']['detection\_risk']:.2f}",

            "成本风险": f"{score\_info['components']['cost\_risk']:.2f}",

            "临床风险": f"{score\_info['components']['clinical\_risk']:.2f}",

            "最优t0": f"{strategy['t0']:.1f}周",

            "早期成功率": f"{strategy['F\_t0']:.1%}",

        }

    )

risk\_summary\_df = pd.DataFrame(risk\_summary)

display(risk\_summary\_df)

print("\n=== 模型整体性能评价 ===")

for metric, value in model\_performance.items():

    if isinstance(value, float):

        print(f"{metric}: {value:.3f}")

    else:

        print(f"{metric}: {value}")

print(

    f"\n⭐ 模型最终评分: {model\_performance['模型得分']}/100 ({model\_performance['模型评级']})"

)

## 问题三代码

### Q3\_final.py

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

import warnings

from scipy import stats, optimize

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from sklearn.model\_selection import cross\_val\_score

warnings.filterwarnings("ignore")

# 设置中文字体

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]

plt.rcParams["axes.unicode\_minus"] = False

# 创建results目录

if not os.path.exists("results"):

os.makedirs("results")

print("创建results目录")

class NIPTDataProcessor:

"""NIPT数据预处理和工具类"""

def \_\_init\_\_(self):

self.male\_data = None

self.female\_data = None

def load\_data(self, file\_path):

"""加载CSV数据"""

try:

# 读取预处理后的男胎数据（CSV格式）

self.male\_data = pd.read\_csv(file\_path)

self.female\_data = None # 问题3只需要男胎数据

# 创建列名映射：中文列名到字母列名

column\_mapping = {

"序号": "A",

"孕妇代码": "B",

"年龄": "C",

"身高": "D",

"体重": "E",

"末次月经": "F",

"IVF妊娠": "G",

"检测日期": "H",

"检测抽血次数": "I",

"检测孕周": "J",

"孕妇BMI": "K",

"原始读段数": "L",

"在参考基因组上比对的比例": "M",

"重复读段的比例": "N",

"唯一比对的读段数 ": "O", # 注意这里有多余空格

"GC含量": "P",

"13号染色体的Z值": "Q",

"18号染色体的Z值": "R",

"21号染色体的Z值": "S",

"X染色体的Z值": "T",

"Y染色体的Z值": "U",

"Y染色体浓度": "V",

"X染色体浓度": "W",

"13号染色体的GC含量": "X",

"18号染色体的GC含量": "Y",

"21号染色体的GC含量": "Z",

"被过滤掉读段数的比例": "AA",

"染色体的非整倍体": "AB",

"怀孕次数": "AC",

"生产次数": "AD",

"胎儿是否健康": "AE",

}

# 重命名列为字母格式以保持代码兼容性

self.male\_data = self.male\_data.rename(columns=column\_mapping)

# 添加解析后的孕周列

self.male\_data["J\_week"] = self.male\_data["J"] # 孕周已经预处理过

print(f"成功加载数据: 男胎{len(self.male\_data)}条")

return True

except Exception as e:

print(f"数据加载失败: {e}")

return False

def preprocess\_male\_data(self):

"""预处理男胎数据 - 添加数据过滤"""

if self.male\_data is None:

print("请先加载数据")

return None

df = self.male\_data.copy()

original\_len = len(df)

print(f"原始数据: {original\_len}条记录")

# 1. 过滤检测孕周：保留 [10, 24) 区间

df = df[(df["J"] >= 10) & (df["J"] < 24)]

print(f"孕周过滤 [10w, 24w): {original\_len} -> {len(df)}条记录")

# 2. 过滤BMI极端异常值

# 使用IQR方法识别异常值

Q1 = df["K"].quantile(0.25)

Q3 = df["K"].quantile(0.75)

IQR = Q3 - Q1

# 定义异常值范围（1.2倍IQR规则，但设置合理的BMI范围）

lower\_bound = max(Q1 - 1.2 \* IQR, 15.0) # BMI不低于15

upper\_bound = min(Q3 + 1.2 \* IQR, 50.0) # BMI不高于50

before\_bmi\_filter = len(df)

df = df[(df["K"] >= lower\_bound) & (df["K"] <= upper\_bound)]

print(

f"BMI异常值过滤 [{lower\_bound:.1f}, {upper\_bound:.1f}]: {before\_bmi\_filter} -> {len(df)}条记录"

)

# 3. 过滤其他必要的缺失值

df = df.dropna(subset=["B", "K", "J", "V"]) # 孕妇代码、BMI、孕周、Y染色体浓度

print(f"缺失值过滤: {len(df)}条记录")

print(f"最终数据: {len(df)}条记录 (过滤掉{original\_len - len(df)}条)")

return df.reset\_index(drop=True)

class Problem3Solver:

"""问题3：多因素BMI分组与达标比例约束分析"""

def \_\_init\_\_(self, data\_processor):

self.data\_processor = data\_processor

self.prob\_model = None

self.optimal\_groups = None

self.scaler = None

def prepare\_multifactor\_features(self, df):

"""准备多因素特征"""

features = pd.DataFrame()

# 基础特征

features["J\_week"] = df["J\_week"]

features["K\_BMI"] = df["K"]

features["C\_age"] = df["C"]

features["D\_height"] = df["D"]

features["E\_weight"] = df["E"]

# 质量特征

features["L\_reads"] = np.log10(df["L"])

features["M\_mapped"] = df["M"]

features["P\_GC"] = df["P"]

features["AA\_filtered"] = df["AA"]

# 派生特征

features["BMI\_age"] = features["K\_BMI"] \* features["C\_age"]

features["height\_weight\_ratio"] = features["D\_height"] / features["E\_weight"]

features["BMI\_week"] = features["K\_BMI"] \* features["J\_week"]

# 标准化身高体重为z-score

features["height\_zscore"] = (

features["D\_height"] - features["D\_height"].mean()

) / features["D\_height"].std()

features["weight\_zscore"] = (

features["E\_weight"] - features["E\_weight"].mean()

) / features["E\_weight"].std()

return features

def fit\_probability\_model(self, df, threshold=0.04):

"""拟合多因素达标概率模型"""

print("拟合多因素达标概率模型...")

# 准备特征和目标

features = self.prepare\_multifactor\_features(df)

target = (df["V"] >= threshold).astype(int)

# 标准化特征

self.scaler = StandardScaler()

X\_scaled = self.scaler.fit\_transform(features)

# 使用随机森林建模概率

self.prob\_model = RandomForestClassifier(

n\_estimators=300,

max\_depth=12,

min\_samples\_split=10,

min\_samples\_leaf=3,

max\_features="sqrt",

class\_weight="balanced",

random\_state=42,

)

self.prob\_model.fit(X\_scaled, target)

# 评估性能

cv\_scores = cross\_val\_score(

self.prob\_model, X\_scaled, target, cv=3, scoring="roc\_auc"

)

print(f"达标概率模型AUC: {cv\_scores.mean():.3f} ± {cv\_scores.std():.3f}")

self.feature\_names = features.columns.tolist()

return self.prob\_model

def predict\_attainment\_probability(self, df, test\_week):

"""预测个体在指定孕周的达标概率"""

if self.prob\_model is None:

raise ValueError("概率模型尚未拟合")

# 准备特征，设置孕周为test\_week

features = self.prepare\_multifactor\_features(df)

features["J\_week"] = test\_week

# 重新计算包含孕周的派生特征

features["BMI\_week"] = features["K\_BMI"] \* test\_week

# 标准化并预测

X\_scaled = self.scaler.transform(features)

probabilities = self.prob\_model.predict\_proba(X\_scaled)[:, 1]

return probabilities

def calculate\_group\_attainment\_rate(self, group\_df, test\_week):

"""计算群体在指定孕周的达标比例。若存在组级事件时间模型，则用 F\_g(t)；否则用横断面概率均值。"""

# 若 group\_df 包含 cluster 标签并且模型已拟合，使用事件时间分布

if (

"cluster" in group\_df.columns

and hasattr(self, "group\_surv")

and self.group\_surv

):

cluster\_vals = group\_df["cluster"].unique()

# 若 group\_df 来自单个cluster，则直接使用

if len(cluster\_vals) == 1:

g = int(cluster\_vals[0])

res = self.group\_surv.get(g, None)

if res is not None:

Sg\_t = self.eval\_step\_S(

np.array([test\_week]), res["support"], res["S"]

)

Fg\_t = 1.0 - Sg\_t[0]

# 若估计失败则回退到横断面概率预测

if np.isfinite(Fg\_t):

return float(Fg\_t)

# 回退：使用当前横断面概率平均（不改变外部接口）

individual\_probs = self.predict\_attainment\_probability(group\_df, test\_week)

group\_rate = individual\_probs.mean()

return group\_rate

def calculate\_multifactor\_risk(

self,

group\_df,

test\_week,

risk\_weights=None,

min\_attain\_rate=0.9,

measurement\_error=0.05,

):

"""计算考虑多因素和测量误差的期望风险"""

if risk\_weights is None:

risk\_weights = {

"early": 0.8,

"mid": 2.5,

"late": 15.0,

"retest": 1.5,

"low\_attain": 80.0,

"error": 3.0,

}

# 计算群体达标率

group\_attain\_rate = self.calculate\_group\_attainment\_rate(group\_df, test\_week)

# 考虑测量误差的影响

# 假设测量误差导致达标率的不确定性

effective\_attain\_rate = group\_attain\_rate \* (1 - measurement\_error)

# 基础时间风险

if test\_week <= 12:

time\_risk = risk\_weights["early"]

elif test\_week <= 27:

time\_risk = risk\_weights["mid"]

else:

time\_risk = risk\_weights["late"]

# 失败风险

failure\_risk = 1 - effective\_attain\_rate

# 群体达标率不足的惩罚

attain\_penalty = 0

if group\_attain\_rate < min\_attain\_rate:

attain\_penalty = risk\_weights["low\_attain"] \* (

min\_attain\_rate - group\_attain\_rate

)

# 测量误差带来的额外风险

error\_risk = risk\_weights["error"] \* measurement\_error \* failure\_risk

# 总风险

total\_risk = (

time\_risk \* failure\_risk

+ risk\_weights["retest"] \* failure\_risk

+ attain\_penalty

+ error\_risk

)

return total\_risk, group\_attain\_rate, effective\_attain\_rate

def two\_stage\_search(

self,

group\_df,

pi\_min=0.90,

metric="ET",

step0=0.25,

step1=0.25,

late\_cap=24,

tau=25,

cw=None,

):

"""两阶段最佳时点搜索"""

if cw is None:

cw = {

"early": 0.8,

"mid": 2.5,

"late": 15,

"retest": 1.5,

"short": 80,

"err": 3,

}

# 1) 预计算 F\_g(t)

T0 = np.arange(10, 20 + 1e-9, step0)

T1 = np.arange(10.5, 25 + 1e-9, step1)

# 计算所有需要的时间点的达标率

F = {}

for t in np.union1d(T0, T1):

F[t] = self.calculate\_group\_attainment\_rate(group\_df, t)

best = None

valid\_solutions = 0 # 统计有效解的数量

for t0 in T0:

for t1 in T1[T1 > t0]:

if late\_cap is not None and t1 > late\_cap: # 可选硬限制

continue

if F[t1] < pi\_min: # 覆盖硬约束

continue

valid\_solutions += 1 # 找到有效解

if metric == "ET":

# 期望完成时间

val = t0 \* F[t0] + t1 \* (F[t1] - F[t0]) + tau \* (1 - F[t1])

else:

# 期望风险/成本

c0 = (

cw["early"]

if t0 <= 12

else (cw["mid"] if t0 <= 27 else cw["late"])

)

c1 = (

cw["early"]

if t1 <= 12

else (cw["mid"] if t1 <= 27 else cw["late"])

)

val = (

c0 \* (1 - F[t0])

+ c1 \* (F[t1] - F[t0])

+ cw["retest"] \* (1 - F[t0])

+ cw["short"] \* max(0.0, pi\_min - F[t1])

+ (cw["late"] if t1 >= 28 else 0.0)

)

if (best is None) or (val < best["val"]):

best = {

"t0": t0,

"t1": t1,

"val": val,

"F0": F[t0],

"F1": F[t1],

"retest\_rate": 1 - F[t0],

"metric": metric,

}

# 如果没有找到满足硬约束的解，选择F[t1]最高的组合

if best is None and valid\_solutions == 0:

max\_F1 = 0

for t0 in T0:

for t1 in T1[T1 > t0]:

if late\_cap is not None and t1 > late\_cap:

continue

if F[t1] > max\_F1:

max\_F1 = F[t1]

if metric == "ET":

val = t0 \* F[t0] + t1 \* (F[t1] - F[t0]) + tau \* (1 - F[t1])

else:

c0 = (

cw["early"]

if t0 <= 12

else (cw["mid"] if t0 <= 27 else cw["late"])

)

c1 = (

cw["early"]

if t1 <= 12

else (cw["mid"] if t1 <= 27 else cw["late"])

)

val = (

c0 \* (1 - F[t0])

+ c1 \* (F[t1] - F[t0])

+ cw["retest"] \* (1 - F[t0])

+ cw["short"] \* max(0.0, pi\_min - F[t1])

+ (cw["late"] if t1 >= 28 else 0.0)

)

best = {

"t0": t0,

"t1": t1,

"val": val,

"F0": F[t0],

"F1": F[t1],

"retest\_rate": 1 - F[t0],

"metric": metric,

}

return best

def optimize\_group\_testing\_time(

self, group\_df, time\_range=(10, 25), min\_attain\_rate=0.9, measurement\_error=0.05

):

"""优化单个组的最佳检测时间 - 支持两阶段"""

# 先进行单阶段优化（保持原有逻辑）

test\_times = np.linspace(time\_range[0], time\_range[1], 20)

best\_time = 15 # 默认值

best\_risk = np.inf

best\_attain = 0

best\_effective\_attain = 0

for t in test\_times:

risk, attain, eff\_attain = self.calculate\_multifactor\_risk(

group\_df,

t,

min\_attain\_rate=min\_attain\_rate,

measurement\_error=measurement\_error,

)

# 优先选择满足达标率要求的时间点

if attain >= min\_attain\_rate and risk < best\_risk:

best\_time = t

best\_risk = risk

best\_attain = attain

best\_effective\_attain = eff\_attain

elif best\_risk == np.inf and attain > best\_attain:

# 如果没有满足要求的时间，选择达标率最高的

best\_time = t

best\_risk = risk

best\_attain = attain

best\_effective\_attain = eff\_attain

# 进行两阶段优化（仅保留Risk策略）

two\_stage\_risk = self.two\_stage\_search(

group\_df, pi\_min=min\_attain\_rate, metric="R"

)

# 调试：检查两阶段结果

if two\_stage\_risk is None:

# print(f" 警告：组内样本数={len(group\_df)}, 无法计算两阶段结果")

# 尝试使用较宽松的参数重新计算

two\_stage\_risk = self.two\_stage\_search(

group\_df, pi\_min=max(0.5, min\_attain\_rate - 0.2), metric="R"

)

if two\_stage\_risk is not None:

print(

f" 信息：组内样本数={len(group\_df)}, 使用宽松参数成功计算两阶段结果"

)

else:

print(

f" 警告：组内样本数={len(group\_df)}, 即使使用宽松参数仍无法计算两阶段结果"

)

return {

"single\_stage": {

"optimal\_week": best\_time,

"expected\_risk": best\_risk,

"attainment\_rate": best\_attain,

"effective\_attainment\_rate": best\_effective\_attain,

},

"two\_stage\_R": two\_stage\_risk,

}

def multifactor\_grouping\_optimization(

self, df, max\_groups=5, min\_attain\_rate=0.92, measurement\_error=0.03

):

"""多因素分组优化（使用强制分离的分组策略）"""

print(

f"多因素分组优化 (最小达标率: {min\_attain\_rate:.1%}, 测量误差: {measurement\_error:.1%})..."

)

# 使用强制分离的分组策略

best\_groups = None

best\_total\_risk = np.inf

best\_strategy = ""

print("尝试强制BMI区间分离的分组策略...")

groups = self.\_group\_by\_forced\_separation(

df, max\_groups, min\_attain\_rate, measurement\_error

)

if groups is not None:

total\_risk = (

groups["expected\_risk"] \* groups["n\_patients"]

).sum() / groups["n\_patients"].sum()

best\_total\_risk = total\_risk

best\_groups = groups

best\_strategy = "强制区间分离分组"

self.optimal\_groups = best\_groups

print(f"最优分组策略: {best\_strategy}, 总体期望风险: {best\_total\_risk:.4f}")

return self.optimal\_groups

def \_group\_by\_forced\_separation(

self, df, n\_groups, min\_attain\_rate, measurement\_error

):

"""使用强制分离的分组方法，确保BMI区间完全不重叠"""

# 基于BMI排序

df\_sorted = df.sort\_values("K").reset\_index(drop=True)

# 计算理想的分组大小

total\_samples = len(df\_sorted)

target\_size = total\_samples // n\_groups

groups\_info = []

for i in range(n\_groups):

start\_idx = i \* target\_size

if i == n\_groups - 1: # 最后一组包含剩余所有样本

end\_idx = total\_samples

else:

end\_idx = (i + 1) \* target\_size

group\_df = df\_sorted.iloc[start\_idx:end\_idx].copy()

if len(group\_df) < 10:

continue

# 计算明确的BMI区间边界

bmi\_min = group\_df["K"].min()

bmi\_max = group\_df["K"].max()

# 为了确保完全分离，调整边界

if i < n\_groups - 1:

# 不是最后一组，需要确保与下一组分离

next\_start\_idx = (i + 1) \* target\_size

if next\_start\_idx < total\_samples:

next\_min\_bmi = df\_sorted.iloc[next\_start\_idx]["K"]

# 设置当前组的最大值为当前组最后一个样本的BMI值

bmi\_max = group\_df["K"].iloc[-1]

# 确保与下一组有明确分界

if bmi\_max >= next\_min\_bmi:

# 如果仍有重叠，使用中点作为分界

bmi\_max = (bmi\_max + next\_min\_bmi) / 2 - 0.01

# 添加cluster标签用于后续模型拟合

group\_df = group\_df.copy()

group\_df["cluster"] = i

# 优化该组的检测时间

opt\_results = self.optimize\_group\_testing\_time(

group\_df,

min\_attain\_rate=min\_attain\_rate,

measurement\_error=measurement\_error,

)

group\_stats = self.calculate\_group\_statistics(group\_df)

# 提取单阶段结果用于兼容性

single\_stage = opt\_results["single\_stage"]

two\_stage\_r = opt\_results["two\_stage\_R"]

groups\_info.append(

{

"group\_id": i + 1,

"bmi\_min": bmi\_min,

"bmi\_max": bmi\_max,

"bmi\_range": f"[{bmi\_min:.1f}, {bmi\_max:.1f}{')'if i < n\_groups - 1 else ']'}",

# 单阶段结果（保持兼容性）

"optimal\_week": single\_stage["optimal\_week"],

"expected\_risk": single\_stage["expected\_risk"],

"attainment\_rate": single\_stage["attainment\_rate"],

"effective\_attainment\_rate": single\_stage[

"effective\_attainment\_rate"

],

# 两阶段结果 - Risk策略

"two\_stage\_R\_t0": two\_stage\_r["t0"] if two\_stage\_r else None,

"two\_stage\_R\_t1": two\_stage\_r["t1"] if two\_stage\_r else None,

"two\_stage\_R\_F0": two\_stage\_r["F0"] if two\_stage\_r else None,

"two\_stage\_R\_F1": two\_stage\_r["F1"] if two\_stage\_r else None,

"two\_stage\_R\_retest\_rate": (

two\_stage\_r["retest\_rate"] if two\_stage\_r else None

),

"two\_stage\_R\_value": two\_stage\_r["val"] if two\_stage\_r else None,

\*\*group\_stats,

}

)

# 为整个数据集创建cluster列用于事件时间模型

df\_with\_clusters = df.copy()

df\_with\_clusters["cluster"] = -1

for i, group\_info in enumerate(groups\_info):

bmi\_min = group\_info["bmi\_min"]

bmi\_max = group\_info["bmi\_max"]

if i < len(groups\_info) - 1: # 不是最后一组

mask = (df\_with\_clusters["K"] >= bmi\_min) & (

df\_with\_clusters["K"] < bmi\_max

)

else: # 最后一组

mask = (df\_with\_clusters["K"] >= bmi\_min) & (

df\_with\_clusters["K"] <= bmi\_max

)

df\_with\_clusters.loc[mask, "cluster"] = i

# 为每个cluster拟合组级事件时间模型

self.\_fit\_group\_event\_models(

df\_with\_clusters, id\_col="B", time\_col="J\_week", value\_col="V", thr=0.04

)

return pd.DataFrame(groups\_info) if groups\_info else None

def calculate\_group\_statistics(self, group\_df):

"""计算群体统计特征 - 修正：使用唯一孕妇ID计算人数"""

# 计算唯一孕妇数量，而不是记录数量

unique\_patients = group\_df["B"].nunique() # 使用孕妇代码计算唯一人数

stats = {

"avg\_age": group\_df["C"].mean(),

"avg\_height": group\_df["D"].mean(),

"avg\_weight": group\_df["E"].mean(),

"avg\_bmi": group\_df["K"].mean(),

"n\_patients": unique\_patients, # 修正：使用唯一孕妇数量

}

return stats

def build\_intervals\_from\_longitudinal(

self,

df: pd.DataFrame,

id\_col: str = "A",

time\_col: str = "J\_week",

value\_col: str = "V",

thr: float = 0.04,

) -> pd.DataFrame:

"""

将同一孕妇的多次测量转换为首次达标的区间删失 (L, R]：

- 若从未达标：右删失 (C, +inf)，此处用 L=C, R=inf, censor='right'

- 若第一次测到达标为首测：左删失 (-inf, R]

- 否则：区间删失 (L, R]

返回列：id, L, R, censor\_type

"""

dfc = df[[id\_col, time\_col, value\_col]].dropna(subset=[id\_col, time\_col]).copy()

dfc = dfc.sort\_values([id\_col, time\_col])

rows = []

for pid, g in dfc.groupby(id\_col):

weeks = g[time\_col].to\_numpy(dtype=float)

vals = g[value\_col].to\_numpy(dtype=float)

hit\_idx = np.where(vals >= thr)[0]

if hit\_idx.size == 0:

# 右删失：用最后一次检测周作为C (L=C, R=+inf)

if weeks.size == 0:

continue

L = weeks[-1]

R = np.inf

censor = "right"

else:

r = int(hit\_idx[0])

R = weeks[r]

if r == 0:

L = -np.inf

censor = "left"

else:

L = weeks[r - 1]

censor = "interval"

rows.append(

{"id": pid, "L": float(L), "R": float(R), "censor\_type": censor}

)

return pd.DataFrame(rows)

# ---------------- 新增：Turnbull NPMLE（简化、数值稳定） ----------------

def turnbull\_npmle(

self, iv\_df: pd.DataFrame, max\_iter: int = 3000, tol: float = 1e-10

):

"""

输入 iv\_df 包含 L, R, censor\_type，返回 dict {support, mass, S, F}

简化实现：支持点取所有有限端点并用 EM 更新质量

"""

g = iv\_df.copy()

L = g["L"].to\_numpy(dtype=float)

R = g["R"].to\_numpy(dtype=float)

finite\_L = L[np.isfinite(L)]

finite\_R = R[np.isfinite(R)]

if finite\_L.size + finite\_R.size == 0:

return {

"support": np.array([]),

"mass": np.array([]),

"S": np.array([]),

"F": np.array([]),

}

support = np.unique(np.concatenate([finite\_L, finite\_R]))

K = len(support)

n = len(g)

# 构建 J\_i 集合

J = []

for Li, Ri in zip(L, R):

if np.isneginf(Li) and np.isfinite(Ri):

idx = np.where(support <= Ri)[0]

elif np.isposinf(Ri) and np.isfinite(Li):

idx = np.where(support > Li)[0]

else:

idx = np.where((support > Li) & (support <= Ri))[0]

if idx.size == 0:

near = np.searchsorted(support, np.nan\_to\_num(Ri, nan=Li))

near = min(max(0, near - 1), K - 1)

idx = np.array([near], dtype=int)

J.append(idx)

# 初始化均匀分布

p = np.full(K, 1.0 / K, dtype=float)

for it in range(max\_iter):

counts = np.zeros(K, dtype=float)

for idx in J:

den = p[idx].sum()

if den <= 0:

counts[idx] += 1.0 / len(idx)

else:

counts[idx] += p[idx] / den

p\_new = counts / n

# 数值修正

p\_new[p\_new < 1e-12] = 0.0

s = p\_new.sum()

if s <= 0:

p\_new = np.full(K, 1.0 / K, dtype=float)

else:

p\_new = p\_new / s

if np.linalg.norm(p\_new - p, ord=1) < tol:

p = p\_new

break

p = p\_new

F = np.cumsum(p)

S = 1.0 - F

return {"support": support, "mass": p, "S": S, "F": F}

# ---------------- 新增：阶梯生存函数在任意 t 上求值 ----------------

def eval\_step\_S(

self, t: np.ndarray, support: np.ndarray, S\_support: np.ndarray

) -> np.ndarray:

"""

阶梯函数评估：对于 t < support[0] -> S=1；对于 support[j-1] < t <= support[j] -> S=S\_support[j]

"""

if support.size == 0:

return np.ones\_like(t, dtype=float)

idx = np.searchsorted(support, t, side="right") - 1

idx = np.clip(idx, -1, len(support) - 1)

out = np.empty\_like(t, dtype=float)

mask\_before = idx < 0

out[mask\_before] = 1.0

mask\_other = ~mask\_before

out[mask\_other] = S\_support[idx[mask\_other]]

return out

# ---------------- 新增：对每个聚类组拟合事件时间模型 ----------------

def \_fit\_group\_event\_models(

self, df\_with\_cluster, id\_col="A", time\_col="J\_week", value\_col="V", thr=0.04

):

"""

对含有 cluster 列的数据，构建组内区间删失并为每组计算 NPMLE，结果保存在 self.group\_surv（dict）

"""

if "cluster" not in df\_with\_cluster.columns:

self.group\_surv = {}

return

iv\_all = self.build\_intervals\_from\_longitudinal(

df\_with\_cluster,

id\_col=id\_col,

time\_col=time\_col,

value\_col=value\_col,

thr=thr,

)

group\_surv = {}

for g in sorted(df\_with\_cluster["cluster"].unique()):

ids = df\_with\_cluster[df\_with\_cluster["cluster"] == g][id\_col].unique()

iv\_g = iv\_all[iv\_all["id"].isin(ids)].reset\_index(drop=True)

if iv\_g.shape[0] == 0:

group\_surv[g] = {

"support": np.array([]),

"mass": np.array([]),

"S": np.array([]),

}

continue

res = self.turnbull\_npmle(iv\_g)

group\_surv[g] = res

self.group\_surv = group\_surv

def \_group\_by\_clustering(self, df, n\_groups, min\_attain\_rate, measurement\_error):

"""基于多因素的聚类分组 - 改进版本减少重叠"""

from sklearn.cluster import KMeans

from sklearn.tree import DecisionTreeRegressor

# 使用改进的聚类方法，重点解决重叠问题

# 准备聚类特征，给BMI更高的权重

cluster\_features = df[["K", "C", "D", "E"]].copy()

# 标准化，但保持BMI的相对重要性

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(cluster\_features)

# 给BMI列（第0列）增加权重

X\_scaled[:, 0] \*= 3.0 # BMI权重增加3倍

# 尝试多次聚类，选择BMI分离度最好的结果

best\_labels = None

best\_separation\_score = -np.inf

for random\_state in range(42, 52): # 尝试10个不同的随机种子

kmeans = KMeans(n\_clusters=n\_groups, random\_state=random\_state, n\_init=10)

labels = kmeans.fit\_predict(X\_scaled)

# 计算BMI分离度分数

separation\_score = self.\_calculate\_bmi\_separation\_score(

df, labels, n\_groups

)

if separation\_score > best\_separation\_score:

best\_separation\_score = separation\_score

best\_labels = labels.copy()

df\_temp = df.copy()

# 进一步优化：使用决策树来优化边界

best\_labels = self.\_optimize\_boundaries\_with\_tree(

df\_temp, best\_labels, n\_groups

)

df\_temp["cluster"] = best\_labels

# 为每个cluster拟合组级事件时间模型

self.\_fit\_group\_event\_models(

df\_temp, id\_col="B", time\_col="J\_week", value\_col="V", thr=0.04

)

groups\_info = []

for i in range(n\_groups):

group\_df = df\_temp[df\_temp["cluster"] == i]

if len(group\_df) < 10:

continue

# 优化该组的检测时间

opt\_results = self.optimize\_group\_testing\_time(

group\_df,

min\_attain\_rate=min\_attain\_rate,

measurement\_error=measurement\_error,

)

group\_stats = self.calculate\_group\_statistics(group\_df)

# 提取单阶段结果用于兼容性

single\_stage = opt\_results["single\_stage"]

groups\_info.append(

{

"group\_id": i + 1,

"bmi\_min": group\_df["K"].min(),

"bmi\_max": group\_df["K"].max(),

"bmi\_range": f"[{group\_df['K'].min():.1f}, {group\_df['K'].max():.1f}]",

"optimal\_week": single\_stage["optimal\_week"],

"expected\_risk": single\_stage["expected\_risk"],

"attainment\_rate": single\_stage["attainment\_rate"],

"effective\_attainment\_rate": single\_stage[

"effective\_attainment\_rate"

],

\*\*group\_stats,

}

)

return pd.DataFrame(groups\_info) if groups\_info else None

def \_calculate\_bmi\_separation\_score(self, df, labels, n\_groups):

"""计算BMI分组的分离度分数，分数越高表示重叠越少"""

group\_ranges = []

for i in range(n\_groups):

mask = labels == i

if mask.sum() > 0:

bmi\_min = df[mask]["K"].min()

bmi\_max = df[mask]["K"].max()

group\_ranges.append((bmi\_min, bmi\_max))

if len(group\_ranges) < 2:

return 0

# 按最小BMI排序

group\_ranges.sort(key=lambda x: x[0])

# 计算分离度：相邻组之间的间隙 vs 组内范围

separation\_score = 0

total\_overlap = 0

for i in range(len(group\_ranges) - 1):

current\_max = group\_ranges[i][1]

next\_min = group\_ranges[i + 1][0]

if current\_max > next\_min: # 有重叠

overlap = current\_max - next\_min

total\_overlap += overlap

else: # 有间隙，这是好的

gap = next\_min - current\_max

separation\_score += gap

# 分离度分数 = 间隙总和 - 重叠惩罚

return separation\_score - total\_overlap \* 2

def \_optimize\_boundaries\_with\_tree(self, df, initial\_labels, n\_groups):

"""使用决策树优化分组边界以减少重叠"""

from sklearn.tree import DecisionTreeClassifier

# 按BMI对分组重新排序

group\_medians = []

for i in range(n\_groups):

mask = initial\_labels == i

if mask.sum() > 0:

median\_bmi = df[mask]["K"].median()

group\_medians.append((i, median\_bmi))

group\_medians.sort(key=lambda x: x[1])

# 重新映射标签

label\_mapping = {

old\_id: new\_id for new\_id, (old\_id, \_) in enumerate(group\_medians)

}

ordered\_labels = np.array([label\_mapping[label] for label in initial\_labels])

# 使用决策树来学习更好的分组边界

features = df[["K", "C", "D", "E"]].values

try:

dt = DecisionTreeClassifier(

max\_depth=3, min\_samples\_split=20, min\_samples\_leaf=10, random\_state=42

)

dt.fit(features, ordered\_labels)

# 使用决策树预测新的标签

new\_labels = dt.predict(features)

# 验证新标签是否减少了重叠

old\_score = self.\_calculate\_bmi\_separation\_score(

df, ordered\_labels, n\_groups

)

new\_score = self.\_calculate\_bmi\_separation\_score(df, new\_labels, n\_groups)

if new\_score > old\_score:

return new\_labels

else:

return ordered\_labels

except Exception as e:

print(f"决策树优化失败，使用原始聚类结果: {e}")

return ordered\_labels

def \_adjust\_clusters\_for\_bmi\_separation(self, df, cluster\_labels, n\_groups):

"""调整聚类结果以减少BMI重叠"""

df\_temp = df.copy()

df\_temp["original\_cluster"] = cluster\_labels

# 计算每个聚类的BMI中位数

cluster\_medians = []

for i in range(n\_groups):

mask = cluster\_labels == i

if mask.sum() > 0:

median\_bmi = df\_temp[mask]["K"].median()

cluster\_medians.append((i, median\_bmi))

# 按BMI中位数排序聚类

cluster\_medians.sort(key=lambda x: x[1])

# 重新分配聚类标签

new\_labels = np.zeros\_like(cluster\_labels)

for new\_id, (old\_id, \_) in enumerate(cluster\_medians):

mask = cluster\_labels == old\_id

new\_labels[mask] = new\_id

# 进一步调整边界以减少重叠

adjusted\_labels = new\_labels.copy()

for i in range(len(cluster\_medians) - 1):

# 找到相邻两组的BMI重叠区域

group\_i\_mask = new\_labels == i

group\_j\_mask = new\_labels == i + 1

if group\_i\_mask.sum() == 0 or group\_j\_mask.sum() == 0:

continue

group\_i\_max = df\_temp[group\_i\_mask]["K"].max()

group\_j\_min = df\_temp[group\_j\_mask]["K"].min()

# 如果有重叠，找到合适的分割点

if group\_i\_max > group\_j\_min:

# 使用两组BMI的加权平均作为分割点

group\_i\_median = df\_temp[group\_i\_mask]["K"].median()

group\_j\_median = df\_temp[group\_j\_mask]["K"].median()

split\_point = (group\_i\_median + group\_j\_median) / 2

# 重新分配重叠区域的点

overlap\_mask = (df\_temp["K"] >= group\_j\_min) & (

df\_temp["K"] <= group\_i\_max

)

overlap\_indices = df\_temp[overlap\_mask].index

for idx in overlap\_indices:

if df\_temp.loc[idx, "K"] <= split\_point:

adjusted\_labels[df\_temp.index.get\_loc(idx)] = i

else:

adjusted\_labels[df\_temp.index.get\_loc(idx)] = i + 1

return adjusted\_labels

def sensitivity\_analysis(self, df):

"""敏感性分析"""

print("进行敏感性分析...")

# 不同参数组合

sensitivity\_params = [

{"min\_attain\_rate": 0.88, "measurement\_error": 0.02, "name": "宽松约束"},

{"min\_attain\_rate": 0.92, "measurement\_error": 0.03, "name": "标准约束"},

{"min\_attain\_rate": 0.96, "measurement\_error": 0.05, "name": "严格约束"},

{"min\_attain\_rate": 0.90, "measurement\_error": 0.01, "name": "高精度测量"},

{"min\_attain\_rate": 0.94, "measurement\_error": 0.07, "name": "保守策略"},

]

sensitivity\_results = []

for params in sensitivity\_params:

print(f"测试{params['name']}...")

# 重新优化分组

temp\_groups = self.multifactor\_grouping\_optimization(

df,

min\_attain\_rate=params["min\_attain\_rate"],

measurement\_error=params["measurement\_error"],

)

if temp\_groups is not None and len(temp\_groups) > 0:

sensitivity\_results.append(

{

"scenario": params["name"],

"min\_attain\_rate": params["min\_attain\_rate"],

"measurement\_error": params["measurement\_error"],

"n\_groups": len(temp\_groups),

"avg\_optimal\_week": temp\_groups["optimal\_week"].mean(),

"avg\_attainment\_rate": temp\_groups["attainment\_rate"].mean(),

"total\_risk": (

temp\_groups["expected\_risk"] \* temp\_groups["n\_patients"]

).sum()

/ temp\_groups["n\_patients"].sum(),

}

)

return pd.DataFrame(sensitivity\_results)

def analyze\_factor\_importance(self):

"""分析因素重要性"""

if self.prob\_model is None:

print("请先拟合概率模型")

return None

# 使用随机森林的特征重要性

importance\_df = pd.DataFrame(

{

"feature": self.feature\_names,

"importance": self.prob\_model.feature\_importances\_,

}

).sort\_values("importance", ascending=False)

print("多因素重要性排序:")

print(importance\_df.to\_string(index=False))

return importance\_df

def plot\_results(self, df):

"""绘制分析结果"""

fig, axes = plt.subplots(3, 3, figsize=(20, 18))

fig.suptitle("多因素BMI分组与两阶段策略优化分析结果", fontsize=16)

# 1. BMI分组可视化

axes[0, 0].hist(df["K"], bins=30, alpha=0.7, edgecolor="black")

if self.optimal\_groups is not None:

for \_, group in self.optimal\_groups.iterrows():

axes[0, 0].axvline(

group["bmi\_min"], color="red", linestyle="--", alpha=0.7

)

axes[0, 0].set\_xlabel("BMI")

axes[0, 0].set\_ylabel("频次")

axes[0, 0].set\_title("BMI分布及最优分组")

# 2. 两阶段策略时点对比

if self.optimal\_groups is not None:

group\_ids = range(len(self.optimal\_groups))

# 绘制t0和t1

has\_two\_stage = self.optimal\_groups["two\_stage\_R\_t0"].notna().any()

if has\_two\_stage:

t0\_values = self.optimal\_groups["two\_stage\_R\_t0"].fillna(0)

t1\_values = self.optimal\_groups["two\_stage\_R\_t1"].fillna(0)

axes[0, 1].scatter(

group\_ids,

t0\_values,

color="green",

s=100,

label="首次检测时间(t0)",

marker="o",

alpha=0.8,

)

axes[0, 1].scatter(

group\_ids,

t1\_values,

color="red",

s=100,

label="保底检测时间(t1)",

marker="^",

alpha=0.8,

)

# 连线显示时间窗口

for i, (t0, t1) in enumerate(zip(t0\_values, t1\_values)):

if pd.notna(t0) and pd.notna(t1):

axes[0, 1].plot([i, i], [t0, t1], "k-", alpha=0.3, linewidth=2)

axes[0, 1].text(

i,

(t0 + t1) / 2,

f"{t1-t0:.1f}周",

ha="center",

fontsize=8,

bbox=dict(

boxstyle="round,pad=0.3", facecolor="white", alpha=0.7

),

)

axes[0, 1].set\_ylabel("检测孕周")

axes[0, 1].set\_title("两阶段策略时点分布")

axes[0, 1].legend()

else:

axes[0, 1].plot(

self.optimal\_groups["avg\_bmi"],

self.optimal\_groups["optimal\_week"],

"ro-",

linewidth=2,

markersize=8,

)

axes[0, 1].set\_xlabel("平均BMI")

axes[0, 1].set\_ylabel("最佳检测孕周")

axes[0, 1].set\_title("最佳检测时点vs BMI")

axes[0, 1].set\_xlabel("分组ID")

axes[0, 1].set\_xticks(group\_ids)

axes[0, 1].set\_xticklabels([f"组{i+1}" for i in group\_ids])

axes[0, 1].grid(True, alpha=0.3)

# 3. 两阶段策略效果对比

if self.optimal\_groups is not None:

x\_pos = range(len(self.optimal\_groups))

# 检查是否有两阶段数据

has\_two\_stage = self.optimal\_groups["two\_stage\_R\_F0"].notna().any()

if has\_two\_stage:

# 显示早期达标率和最终达标率

early\_rates = self.optimal\_groups["two\_stage\_R\_F0"].fillna(0)

final\_rates = self.optimal\_groups["two\_stage\_R\_F1"].fillna(0)

bars1 = axes[0, 2].bar(

[x - 0.2 for x in x\_pos],

early\_rates,

width=0.4,

label="早期达标率(t0)",

alpha=0.7,

color="lightgreen",

)

bars2 = axes[0, 2].bar(

[x + 0.2 for x in x\_pos],

final\_rates,

width=0.4,

label="最终达标率(t1)",

alpha=0.7,

color="lightcoral",

)

else:

# 回退到原有显示

bars1 = axes[0, 2].bar(

[x - 0.2 for x in x\_pos],

self.optimal\_groups["attainment\_rate"],

width=0.4,

label="实际达标率",

alpha=0.7,

)

bars2 = axes[0, 2].bar(

[x + 0.2 for x in x\_pos],

self.optimal\_groups["effective\_attainment\_rate"],

width=0.4,

label="有效达标率",

alpha=0.7,

)

axes[0, 2].axhline(y=0.9, color="red", linestyle="--", label="90%目标线")

axes[0, 2].set\_xlabel("分组")

axes[0, 2].set\_ylabel("达标率")

axes[0, 2].set\_title(

"两阶段策略达标率对比" if has\_two\_stage else "各组达标率对比"

)

axes[0, 2].set\_xticks(x\_pos)

axes[0, 2].set\_xticklabels([f"组{i + 1}" for i in x\_pos])

axes[0, 2].legend()

# 4. 多因素相关性热图

factors = ["K", "C", "D", "E", "J\_week", "V"]

available\_factors = [f for f in factors if f in df.columns]

if len(available\_factors) > 2:

corr\_matrix = df[available\_factors].corr()

sns.heatmap(

corr\_matrix, annot=True, cmap="coolwarm", center=0, ax=axes[1, 0]

)

axes[1, 0].set\_title("多因素相关性热图")

# 5. 特征重要性

if self.prob\_model is not None:

importance = self.prob\_model.feature\_importances\_

top\_indices = np.argsort(importance)[-10:]

axes[1, 1].barh(range(len(top\_indices)), importance[top\_indices])

axes[1, 1].set\_yticks(range(len(top\_indices)))

axes[1, 1].set\_yticklabels([self.feature\_names[i] for i in top\_indices])

axes[1, 1].set\_xlabel("重要性")

axes[1, 1].set\_title("特征重要性 (Top 10)")

# 6. 年龄vs达标概率分布

if len(df) > 50: # 确保有足够数据

age\_bins = pd.cut(df["C"], bins=5)

age\_attain\_rates = df.groupby(age\_bins)["V"].apply(

lambda x: (x >= 0.04).mean()

)

age\_centers = [

interval.mid

for interval in age\_attain\_rates.index

if pd.notna(interval.mid)

]

valid\_rates = [rate for rate in age\_attain\_rates.values if pd.notna(rate)]

if len(age\_centers) > 0 and len(valid\_rates) > 0:

axes[1, 2].plot(age\_centers, valid\_rates, "o-")

axes[1, 2].set\_xlabel("年龄")

axes[1, 2].set\_ylabel("达标率")

axes[1, 2].set\_title("年龄vs达标率")

# 7. 风险分布

if self.optimal\_groups is not None:

risks = self.optimal\_groups["expected\_risk"]

group\_names = [f"组{i + 1}" for i in range(len(risks))]

bars = axes[2, 0].bar(group\_names, risks, alpha=0.7, color="orange")

axes[2, 0].set\_ylabel("期望风险")

axes[2, 0].set\_title("各组期望风险")

# 标注数值

for bar, risk in zip(bars, risks):

axes[2, 0].text(

bar.get\_x() + bar.get\_width() / 2,

bar.get\_height() + 0.1,

f"{risk:.2f}",

ha="center",

va="bottom",

)

# 8. 两阶段策略可视化 (类似notebook中的策略图)

if (

self.optimal\_groups is not None

and self.optimal\_groups["two\_stage\_R\_t0"].notna().any()

):

# 使用生存函数数据创建类似notebook的策略图

axes[2, 1].set\_title("两阶段策略示意图(前3组)")

# 为前3组绘制假想的F\_g(t)曲线和策略线

colors = ["blue", "green", "orange"]

for idx in range(min(3, len(self.optimal\_groups))):

group = self.optimal\_groups.iloc[idx]

if pd.notna(group["two\_stage\_R\_t0"]) and pd.notna(

group["two\_stage\_R\_t1"]

):

t0 = group["two\_stage\_R\_t0"]

t1 = group["two\_stage\_R\_t1"]

F0 = group["two\_stage\_R\_F0"]

F1 = group["two\_stage\_R\_F1"]

# 创建示意性的累积达标曲线

t\_range = np.linspace(10, 25, 100)

# 使用sigmoidal函数模拟达标曲线

F\_curve = F1 / (1 + np.exp(-0.5 \* (t\_range - (t0 + t1) / 2)))

# 绘制达标曲线

axes[2, 1].plot(

t\_range,

F\_curve,

color=colors[idx],

linewidth=2,

label=f"组{idx+1} F\_g(t)",

)

# 标记关键时点

axes[2, 1].axvline(t0, color=colors[idx], linestyle="--", alpha=0.7)

axes[2, 1].axvline(t1, color="red", linestyle="--", alpha=0.7)

axes[2, 1].scatter([t0], [F0], color=colors[idx], s=80, zorder=5)

# 添加标注

axes[2, 1].text(

t0,

F0 + 0.05,

f"F={F0:.2f}",

ha="center",

fontsize=8,

bbox=dict(

boxstyle="round,pad=0.2", facecolor="white", alpha=0.8

),

)

# 添加95%基准线

axes[2, 1].axhline(

0.95, color="orange", linestyle=":", alpha=0.7, label="95%目标"

)

axes[2, 1].set\_xlabel("孕周(周)")

axes[2, 1].set\_ylabel("累积达标率 F\_g(t)")

axes[2, 1].legend(fontsize=8)

axes[2, 1].set\_ylim(0, 1.05)

axes[2, 1].grid(True, alpha=0.3)

else:

# 原有的身高体重分布图（作为后备）

if len(df) > 0:

达标\_mask = df["V"] >= 0.04

scatter1 = axes[2, 1].scatter(

df[达标\_mask]["D"],

df[达标\_mask]["E"],

alpha=0.6,

label="达标",

c="green",

s=30,

)

scatter2 = axes[2, 1].scatter(

df[~达标\_mask]["D"],

df[~达标\_mask]["E"],

alpha=0.6,

label="未达标",

c="red",

s=30,

)

axes[2, 1].set\_xlabel("身高")

axes[2, 1].set\_ylabel("体重")

axes[2, 1].set\_title("身高体重分布(按达标状态着色)")

axes[2, 1].legend()

# 9. 两阶段策略结果汇总表

if self.optimal\_groups is not None:

axes[2, 2].axis("off")

table\_data = []

has\_two\_stage = self.optimal\_groups["two\_stage\_R\_t0"].notna().any()

for \_, row in self.optimal\_groups.iterrows():

if has\_two\_stage and pd.notna(row.get("two\_stage\_R\_t0")):

# 包含两阶段信息的表格

table\_data.append(

[

f"组{row['group\_id']}",

row["bmi\_range"],

f"{row['two\_stage\_R\_t0']:.1f}/{row['two\_stage\_R\_t1']:.1f}",

f"{row['two\_stage\_R\_F0']:.2f}",

f"{row['two\_stage\_R\_F1']:.2f}",

f"{row['n\_patients']}",

]

)

else:

# 原有的单阶段表格

table\_data.append(

[

f"组{row['group\_id']}",

row["bmi\_range"],

f"{row['optimal\_week']:.1f}",

f"{row['attainment\_rate']:.2f}",

f"{row['effective\_attainment\_rate']:.2f}",

f"{row['n\_patients']}",

]

)

if has\_two\_stage:

col\_labels = [

"分组",

"BMI范围",

"t0/t1(周)",

"早期达标率",

"最终达标率",

"样本数",

]

title = "两阶段策略详细结果"

else:

col\_labels = [

"分组",

"BMI范围",

"最佳时点",

"实际达标率",

"有效达标率",

"样本数",

]

title = "分组详细结果"

table = axes[2, 2].table(

cellText=table\_data,

colLabels=col\_labels,

cellLoc="center",

loc="center",

)

table.auto\_set\_font\_size(False)

table.set\_fontsize(8)

table.scale(1, 1.5)

axes[2, 2].set\_title(title)

plt.tight\_layout()

return fig

def plot\_two\_stage\_strategy(self, optimal\_groups):

"""绘制两阶段策略可视化图（类似notebook中的策略图）"""

if optimal\_groups is None or not optimal\_groups["two\_stage\_R\_t0"].notna().any():

return None

groups = optimal\_groups[optimal\_groups["two\_stage\_R\_t0"].notna()]

G = len(groups)

if G == 0:

return None

from math import ceil

ncols = 3 if G > 6 else 2

nrows = ceil(G / ncols)

fig, axes = plt.subplots(

nrows, ncols, figsize=(5 \* ncols, 4 \* nrows), sharex=True, sharey=True

)

axes = np.atleast\_1d(axes).ravel()

fig.suptitle("两阶段NIPT检测策略", fontsize=16, fontweight="bold")

for i, (\_, group) in enumerate(groups.iterrows()):

if i >= len(axes):

break

ax = axes[i]

# 获取该组数据

t0 = group["two\_stage\_R\_t0"]

t\_star = group["two\_stage\_R\_t1"]

F\_t0 = group["two\_stage\_R\_F0"]

F\_t1 = group["two\_stage\_R\_F1"]

# 创建时间网格

t\_grid = np.linspace(10, 25, 100)

# 使用分段函数模拟F\_g(t)曲线

# 在t0之前缓慢增长，t0处达到F\_t0，t\_star处达到F\_t1

Fg = np.zeros\_like(t\_grid)

for j, t in enumerate(t\_grid):

if t <= t0:

# t0之前的线性增长

Fg[j] = F\_t0 \* (t - 10) / (t0 - 10) if t0 > 10 else F\_t0

elif t <= t\_star:

# t0到t\_star之间的线性增长

Fg[j] = F\_t0 + (F\_t1 - F\_t0) \* (t - t0) / (t\_star - t0)

else:

# t\_star之后保持不变

Fg[j] = F\_t1

# 限制在[0,1]范围内

Fg = np.clip(Fg, 0, 1)

# 绘制F\_g(t)曲线（阶梯函数风格）

ax.step(t\_grid, Fg, where="post", linewidth=3, label="F\_g(t)", color="blue")

# 标记关键时点

# t0时点

ax.axvline(

t0, color="green", linestyle="--", linewidth=2, label=f"t0={t0:.1f}周"

)

ax.plot([t0], [F\_t0], "go", markersize=8)

ax.text(

t0,

min(1.0, F\_t0 + 0.06),

f"F={F\_t0:.2f}",

ha="center",

color="green",

fontweight="bold",

bbox=dict(boxstyle="round,pad=0.3", facecolor="lightgreen", alpha=0.7),

)

# t\*\_g时点

ax.axvline(

t\_star,

color="red",

linestyle="--",

linewidth=2,

label=f"t\*={t\_star:.1f}周",

)

ax.plot([t\_star], [F\_t1], "rs", markersize=8)

# 95%基准线

ax.axhline(

0.95,

color="orange",

linestyle=":",

alpha=0.8,

linewidth=2,

label="95%目标",

)

# 设置标题和标签

bmi\_range = (

group["bmi\_range"]

if "bmi\_range" in group

else f"组{int(group['group\_id'])}"

)

ax.set\_title(

f'组{int(group["group\_id"])}: {bmi\_range}\n两阶段策略',

fontsize=12,

fontweight="bold",

)

if i >= (nrows - 1) \* ncols: # 最底行

ax.set\_xlabel("孕周(周)", fontsize=10)

if i % ncols == 0: # 最左列

ax.set\_ylabel("累积达标率 F\_g(t)", fontsize=10)

ax.grid(True, alpha=0.3)

ax.set\_ylim(0, 1.05)

ax.set\_xlim(10, 25)

# 添加图例（只在第一个图中显示）

if i == 0:

ax.legend(loc="lower right", fontsize=9, framealpha=0.9)

# 添加策略效果文本

advance\_weeks = t\_star - t0

ax.text(

0.02,

0.98,

f"提前: {advance\_weeks:.1f}周\n早期成功率: {F\_t0:.1%}",

transform=ax.transAxes,

fontsize=9,

verticalalignment="top",

bbox=dict(boxstyle="round,pad=0.3", facecolor="lightblue", alpha=0.8),

)

# 关闭多余子图

for k in range(G, len(axes)):

axes[k].axis("off")

plt.tight\_layout()

return fig

def run\_problem3():

"""运行问题3分析"""

print("=" \* 60)

print("问题3：多因素BMI分组与达标比例约束分析")

print("=" \* 60)

# 初始化数据处理器

processor = NIPTDataProcessor()

# 加载数据

if not processor.load\_data(

r"D:\HP\OneDrive\Desktop\学校\竞赛\数模国赛\CUMCM2025Problems\C题\prenatal-testing\Q3\男胎检测数据\_预处理后.csv"

):

print("数据加载失败，请检查文件'男胎检测数据\_预处理后.csv'是否存在")

return None

male\_df = processor.preprocess\_male\_data()

if male\_df is None or len(male\_df) == 0:

print("男胎数据预处理失败")

return None

# --- 插入：构造区间删失并诊断事件时间信息量 ---

# 去除同一ID同一孕周的重复：保留最大V（或最近一次）

male\_df = (

male\_df.sort\_values(["B", "J\_week", "V"])

.groupby(["B", "J\_week"], as\_index=False)

.last()

)

# 统计具有多次随访的ID数量

id\_counts = male\_df["B"].value\_counts() # 使用孕妇代码而不是序号

n\_multi = int((id\_counts > 1).sum())

print(f"具有多次测量的ID数量: {n\_multi}")

# 构建区间删失表并展示前几行

iv\_df = Problem3Solver(processor).build\_intervals\_from\_longitudinal(

male\_df, id\_col="B", time\_col="J\_week", value\_col="V", thr=0.04 # 使用孕妇代码

)

print("区间删失样例（最多10条）：")

print(iv\_df.head(10).to\_string(index=False))

# 若信息足够，拟合全体 NPMLE 并在若干时间点评估 F(t)

if iv\_df.shape[0] >= 30 and n\_multi >= 30:

solver\_tmp = Problem3Solver(processor)

npmle\_all = solver\_tmp.turnbull\_npmle(iv\_df)

print(

"NPMLE support\_len:",

len(npmle\_all["support"]),

" mass\_sum:",

float(np.sum(npmle\_all["mass"])),

)

for t in [12, 16, 20, 24]:

St = solver\_tmp.eval\_step\_S(

np.array([t]), npmle\_all["support"], npmle\_all["S"]

)[0]

print(f"F({t}) = {1.0 - St:.3f}")

else:

print(

"警告：纵向信息不足，NPMLE 估计可能不可靠（建议至少若干十个有重复测量的ID）"

)

# -----------------------------------------------------

# 创建求解器

solver = Problem3Solver(processor)

# 步骤1：拟合多因素概率模型

print("\n步骤1：拟合多因素达标概率模型")

prob\_model = solver.fit\_probability\_model(male\_df)

# 步骤2：多因素分组优化

print("\n步骤2：多因素分组优化")

optimal\_groups = solver.multifactor\_grouping\_optimization(

male\_df, min\_attain\_rate=0.92, measurement\_error=0.03

)

# 步骤3：敏感性分析

print("\n步骤3：敏感性分析")

sensitivity\_results = solver.sensitivity\_analysis(male\_df)

# 步骤4：分析因素重要性

print("\n步骤4：分析多因素重要性")

importance\_df = solver.analyze\_factor\_importance()

# 步骤5：绘制结果

print("\n步骤5：生成可视化结果")

fig = solver.plot\_results(male\_df)

plt.savefig("results/problem3\_analysis.png", dpi=300, bbox\_inches="tight")

plt.show()

# 生成两阶段策略专门图像

if optimal\_groups is not None and optimal\_groups["two\_stage\_R\_t0"].notna().any():

print("生成两阶段策略可视化图...")

fig\_strategy = solver.plot\_two\_stage\_strategy(optimal\_groups)

if fig\_strategy is not None:

plt.savefig(

"results/problem3\_two\_stage\_strategy.png", dpi=300, bbox\_inches="tight"

)

plt.show()

# 步骤6：保存结果

print("\n步骤6：保存分析结果")

if optimal\_groups is not None and len(optimal\_groups) > 0:

# 保存最优分组结果

optimal\_groups.to\_csv(

"results/problem3\_optimal\_groups.csv", index=False, encoding="utf-8-sig"

)

# 保存敏感性分析结果

if len(sensitivity\_results) > 0:

sensitivity\_results.to\_csv(

"results/problem3\_sensitivity.csv", index=False, encoding="utf-8-sig"

)

# 保存特征重要性

if importance\_df is not None:

importance\_df.to\_csv(

"results/problem3\_feature\_importance.csv",

index=False,

encoding="utf-8-sig",

)

# 保存多因素分析详细数据

multifactor\_analysis = []

for \_, group in optimal\_groups.iterrows():

multifactor\_analysis.append(

{

"group\_id": group["group\_id"],

"bmi\_range": group["bmi\_range"],

"optimal\_week": group["optimal\_week"],

"attainment\_rate": group["attainment\_rate"],

"effective\_attainment\_rate": group["effective\_attainment\_rate"],

"expected\_risk": group["expected\_risk"],

"avg\_age": group["avg\_age"],

"avg\_height": group["avg\_height"],

"avg\_weight": group["avg\_weight"],

"avg\_bmi": group["avg\_bmi"],

"n\_patients": group["n\_patients"],

}

)

multifactor\_df = pd.DataFrame(multifactor\_analysis)

multifactor\_df.to\_csv(

"results/problem3\_multifactor\_analysis.csv",

index=False,

encoding="utf-8-sig",

)

print("=" \* 60)

print("问题3分析完成！")

print("生成的文件：")

print("- results/problem3\_analysis.png (综合分析结果)")

print("- results/problem3\_two\_stage\_strategy.png (两阶段策略可视化)")

print("- results/problem3\_optimal\_groups.csv (最优分组结果)")

print("- results/problem3\_sensitivity.csv (敏感性分析)")

print("- results/problem3\_feature\_importance.csv (特征重要性)")

print("- results/problem3\_multifactor\_analysis.csv (多因素详细分析)")

print("- results/problem3\_two\_stage\_analysis.csv (两阶段策略分析)")

print("- results/problem3\_personalized\_recommendations.csv (个性化检测建议)")

print("=" \* 60)

# 输出关键结果摘要

# 生成两阶段策略分析报告

two\_stage\_summary = []

for \_, group in optimal\_groups.iterrows():

if group.get("two\_stage\_R\_t0") is not None:

# 计算提前周数

advance\_weeks = group["two\_stage\_R\_t1"] - group["two\_stage\_R\_t0"]

# 计算期望检测时间

expected\_time = group["two\_stage\_R\_t0"] \* group[

"two\_stage\_R\_F0"

] + group["two\_stage\_R\_t1"] \* (1 - group["two\_stage\_R\_F0"])

# 风险评级

if group["two\_stage\_R\_F0"] >= 0.8:

risk\_level = "推荐"

advice = "早期检测成功率高"

elif group["two\_stage\_R\_F0"] >= 0.6:

risk\_level = "建议"

advice = "早期检测中等成功率，需关注复测"

else:

risk\_level = "谨慎"

advice = "早期检测成功率较低，强化复测方案"

two\_stage\_summary.append(

{

"组号": int(group["group\_id"]),

"BMI区间": group["bmi\_range"],

"样本量": int(group["n\_patients"]),

"最优t0(周)": round(float(group["two\_stage\_R\_t0"]), 1),

"保底t1(周)": round(float(group["two\_stage\_R\_t1"]), 1),

"早期达标率": f"{float(group['two\_stage\_R\_F0']):.1%}",

"期望检测时间(周)": round(float(expected\_time), 2),

"风险评级": risk\_level,

"临床建议": advice,

"提前周数": round(float(advance\_weeks), 1),

}

)

if two\_stage\_summary:

print("\n=== 两阶段联合优化结果 ===")

two\_stage\_df = pd.DataFrame(two\_stage\_summary)

print(two\_stage\_df.to\_string(index=False))

# 保存两阶段分析结果

two\_stage\_df.to\_csv(

"results/problem3\_two\_stage\_analysis.csv",

index=False,

encoding="utf-8-sig",

)

print("\n=== 关键结果摘要 ===")

print(f"总样本数: {len(male\_df)}")

print(f"最优分组数: {len(optimal\_groups)}")

print("\n最优多因素分组方案:")

for \_, group in optimal\_groups.iterrows():

print(

f"组{group['group\_id']}: BMI {group['bmi\_range']}, "

f"最佳检测时点 {group['optimal\_week']:.1f}周"

)

print(

f" 实际达标率 {group['attainment\_rate']:.2f}, "

f"有效达标率 {group['effective\_attainment\_rate']:.2f}"

)

print(

f" 平均年龄 {group['avg\_age']:.1f}岁, "

f"平均身高 {group['avg\_height']:.1f}cm, "

f"平均体重 {group['avg\_weight']:.1f}kg"

)

# 添加两阶段结果展示（仅Risk策略）

if group.get("two\_stage\_R\_t0") is not None:

print(

f" 两阶段策略: t0={group['two\_stage\_R\_t0']:.1f}周, t1={group['two\_stage\_R\_t1']:.1f}周"

)

print(

f" 首测达标率: {group['two\_stage\_R\_F0']:.3f}, 复测达标率: {group['two\_stage\_R\_F1']:.3f}"

)

print(

f" 复测率: {group['two\_stage\_R\_retest\_rate']:.3f}, Risk值: {group['two\_stage\_R\_value']:.3f}"

)

print()

print(f"平均最佳检测时点: {optimal\_groups['optimal\_week'].mean():.1f}周")

print(f"整体达标率: {optimal\_groups['attainment\_rate'].mean():.3f}")

print(

f"总体期望风险: {(optimal\_groups['expected\_risk'] \* optimal\_groups['n\_patients']).sum() / optimal\_groups['n\_patients'].sum():.4f}"

)

if len(sensitivity\_results) > 0:

print("\n敏感性分析结果:")

print(sensitivity\_results.to\_string(index=False))

if importance\_df is not None and len(importance\_df) > 0:

print(f"\n最重要的影响因素:")

for i, (\_, row) in enumerate(importance\_df.head(5).iterrows()):

print(f"{i + 1}. {row['feature']}: {row['importance']:.4f}")

# 生成个性化检测方案建议（基于两阶段结果）

if two\_stage\_summary:

print("\n=== 个性化检测方案建议 ===")

recommendations = []

for item in two\_stage\_summary:

# 时间窗口建议

t0 = item["最优t0(周)"]

if t0 < 12:

timing\_advice = f"在{t0:.1f}周进行首次检测(早期窗口)"

elif t0 < 16:

timing\_advice = f"在{t0:.1f}周进行首次检测(标准窗口)"

else:

timing\_advice = f"在{t0:.1f}周进行首次检测(延后窗口)"

full\_advice = f"{timing\_advice}，{item['临床建议']}"

recommendations.append(

{

"组别": f"组{item['组号']}",

"BMI范围": item["BMI区间"],

"样本特征": f"N={item['样本量']}",

"推荐等级": item["风险评级"],

"首次检测时间": f"{item['最优t0(周)']}周",

"保底检测时间": f"{item['保底t1(周)']}周",

"早期成功率": item["早期达标率"],

"临床建议": full\_advice,

"备注": f"预期节省{item['提前周数']}周检测时间",

}

)

recommendations\_df = pd.DataFrame(recommendations)

print(recommendations\_df.to\_string(index=False))

# 保存个性化建议

recommendations\_df.to\_csv(

"results/problem3\_personalized\_recommendations.csv",

index=False,

encoding="utf-8-sig",

)

# 生成临床实施建议

print("\n=== 临床实施建议 ===")

avg\_early\_rate = sum(

[

float(item["早期达标率"].rstrip("%")) / 100

for item in two\_stage\_summary

]

) / len(two\_stage\_summary)

avg\_advance\_weeks = sum(

[item["提前周数"] for item in two\_stage\_summary]

) / len(two\_stage\_summary)

clinical\_advice = f"""

1. 个性化检测策略：

- 根据孕妇BMI分组，采用不同的两阶段检测方案

- 平均早期检测成功率: {avg\_early\_rate:.1%}

- 平均节省检测时间: {avg\_advance\_weeks:.1f}周

2. 质量控制要点：

- 确保早期检测的技术可靠性

- 建立完善的复测跟踪机制

- 设置风险阈值预警系统

3. 成本效益考量：

- 早期检测可显著减少整体检测时间

- 需平衡检测成本与临床效果

- 建议建立动态成本调节机制

4. 实施监控：

- 定期评估各组实际达标率

- 监控成本控制效果

- 根据实际情况调整参数

5. 分组策略特点：

- 低BMI组: 早期检测时点可适当提前，成功率通常较高

- 高BMI组: 需要更谨慎的检测时点选择，加强保底检测

- 建议定期根据实际数据调整分组阈值

"""

print(clinical\_advice)

else:

print("警告：未能生成有效的分组结果")

return solver

if \_\_name\_\_ == "\_\_main\_\_":

run\_problem3()

## 问题四代码

### Q4\_final.py

# -\*- coding: utf-8 -\*-

"""

问题四 · 女胎异常判定（完整可运行版，按“评分踩点清单”改造）

改动要点：

1) 维持原有孕周解析逻辑；

2) 训练集内：Pipeline+K折交叉验证；

3) 四种采样策略并行（None/ROS/RUS/SMOTE），与两类模型（L1-LogReg, RandomForest）组合对比；

4) 只在训练集上CV与调参，外部测试集只评一次；

5) 报告并导出 AUROC 与 AUPRC（数值与曲线），并给出阳性率基线；

6) 可选稳健处理：winsorize 截尾；

7) 输出：模型对比表、最佳模型测试集指标、阈值优化、校准曲线点、特征重要性等。

保持与原版的一致性：保留“女胎异常定义=是否存在13/18/21任一异常(AB)”。

"""

import os

import warnings

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, GridSearchCV

from sklearn.preprocessing import StandardScaler, RobustScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (

    classification\_report,

    confusion\_matrix,

    roc\_auc\_score,

    roc\_curve,

    precision\_recall\_curve,

    average\_precision\_score,

    brier\_score\_loss,

)

from sklearn.calibration import calibration\_curve

# 采样与Pipeline（带兜底）

IMBLEARN\_OK = True

try:

    from imblearn.pipeline import Pipeline as ImbPipeline

    from imblearn.over\_sampling import RandomOverSampler, SMOTE

    from imblearn.under\_sampling import RandomUnderSampler

except Exception:

    IMBLEARN\_OK = False

    print(

        "警告：未安装 imbalanced-learn，将退化为 class\_weight 策略（无采样Pipeline）。"

    )

warnings.filterwarnings("ignore")

plt.rcParams["font.sans-serif"] = ["SimHei", "Arial Unicode MS", "DejaVu Sans"]

plt.rcParams["axes.unicode\_minus"] = False

# 结果目录

os.makedirs("results", exist\_ok=True)

class NIPTDataProcessor:

    """数据加载/预处理（保留原孕周解析规则）"""

    def \_\_init\_\_(self):

        self.male\_data = None

        self.female\_data = None

    def load\_data(self, file\_path: str) -> bool:

        try:

            self.male\_data = pd.read\_excel(file\_path, sheet\_name="男胎检测数据")

            self.female\_data = pd.read\_excel(file\_path, sheet\_name="女胎检测数据")

            columns = [

                "A",

                "B",

                "C",

                "D",

                "E",

                "F",

                "G",

                "H",

                "I",

                "J",

                "K",

                "L",

                "M",

                "N",

                "O",

                "P",

                "Q",

                "R",

                "S",

                "T",

                "U",

                "V",

                "W",

                "X",

                "Y",

                "Z",

                "AA",

                "AB",

                "AC",

                "AD",

                "AE",

            ]

            self.male\_data.columns = columns[: len(self.male\_data.columns)]

            self.female\_data.columns = columns[: len(self.female\_data.columns)]

            print(f"成功加载：男胎{len(self.male\_data)}，女胎{len(self.female\_data)}")

            return True

        except Exception as e:

            print(f"数据加载失败：{e}")

            return False

    @staticmethod

    def parse\_gestational\_week(week\_str):

        """解析孕周：如 '11w+6' -> 11 + 6/7 = 11.857（保留原逻辑）"""

        if pd.isna(week\_str) or week\_str == "":

            return np.nan

        try:

            s = str(week\_str).strip()

            if "w" in s:

                w, rest = s.split("w", 1)

                weeks = float(w)

                if "+" in rest:

                    days = float(rest.replace("+", ""))

                    return weeks + days / 7.0

                return weeks

            return float(s)

        except Exception:

            return np.nan

    def preprocess\_female\_data(self) -> pd.DataFrame:

        if self.female\_data is None:

            print("请先加载数据")

            return pd.DataFrame()

        df = self.female\_data.copy()

        # 孕周

        df["J\_week"] = df["J"].apply(self.parse\_gestational\_week)

        # 基础QC（与原版一致）

        before = len(df)

        df = df[(df["P"] >= 0.35) & (df["P"] <= 0.65)]

        df = df[df["L"] >= 1\_000\_000]

        df = df[df["AA"] <= 0.5]

        # 标签：AB 非空为异常

        df["abnormal"] = (~df["AB"].isna()).astype(int)

        # 关键字段齐备

        df = df.dropna(subset=["K", "J\_week", "Q", "R", "S", "T"])

        print(

            f"女胎预处理：{before} -> {len(df)}；异常 {df['abnormal'].sum()} (占 {df['abnormal'].mean():.1%})"

        )

        return df.reset\_index(drop=True)

class Problem4Solver:

    """问题四：按“评分踩点清单”重构的训练/评估流水线"""

    def \_\_init\_\_(self, processor: NIPTDataProcessor, winsorize=False):

        self.p = processor

        self.winsorize = winsorize

        self.best\_pipeline\_name = None

        self.best\_pipeline = None

        self.feature\_names = None

        self.optimal\_threshold = 0.5

    # ========= 特征工程 =========

    def prepare\_female\_features(self, df: pd.DataFrame) -> pd.DataFrame:

        X = pd.DataFrame()

        # Z 值

        X["Z13"] = df["Q"]

        X["Z18"] = df["R"]

        X["Z21"] = df["S"]

        X["ZX"] = df["T"]

        # X 浓度

        X["X\_conc"] = df.get("W", pd.Series(0, index=df.index)).fillna(0)

        # GC

        X["GC\_13"] = df["X"]

        X["GC\_18"] = df["Y"]

        X["GC\_21"] = df["Z"]

        X["GC\_all"] = df["P"]

        # 读段/质量

        X["total\_reads"] = np.log10(df["L"].clip(lower=1))

        X["mapped\_ratio"] = df["M"]

        X["dup\_ratio"] = df["N"]

        X["unique\_reads"] = np.log10(df["O"].clip(lower=1))

        X["filtered\_ratio"] = df["AA"]

        # 临床

        X["BMI"] = df["K"]

        X["gw"] = df["J\_week"]

        X["age"] = df["C"]

        X["height"] = df["D"]

        X["weight"] = df["E"]

        # 派生

        X["Z\_max"] = X[["Z13", "Z18", "Z21"]].abs().max(axis=1)

        X["Z\_sum"] = X[["Z13", "Z18", "Z21"]].abs().sum(axis=1)

        X["GC\_var"] = X[["GC\_13", "GC\_18", "GC\_21"]].var(axis=1)

        X["GC\_mean"] = X[["GC\_13", "GC\_18", "GC\_21"]].mean(axis=1)

        X["BMI\_age"] = X["BMI"] \* X["age"]

        X["ZxBMI"] = X["Z\_max"] \* X["BMI"]

        X["ZxGW"] = X["Z\_max"] \* X["gw"]

        # 质量评分

        X["q\_score"] = (

            X["mapped\_ratio"] \* 0.3

            + (1 - X["dup\_ratio"]) \* 0.3

            + (1 - X["filtered\_ratio"]) \* 0.4

        )

        # 传统规则指示

        X["z\_any\_ge\_3"] = (X[["Z13", "Z18", "Z21"]].abs().max(axis=1) >= 3.0).astype(

            int

        )

        X["z\_any\_ge\_3p5"] = (X[["Z13", "Z18", "Z21"]].abs().max(axis=1) >= 3.5).astype(

            int

        )

        X["z\_any\_2p5\_3p5"] = ((X["Z\_max"] >= 2.5) & (X["Z\_max"] < 3.5)).astype(int)

        # GC异常

        X["gc\_abn"] = ((X["GC\_all"] < 0.4) | (X["GC\_all"] > 0.6)).astype(int)

        # 质量偏低

        X["low\_q"] = (X["q\_score"] < 0.7).astype(int)

        if self.winsorize:

            # 对常见长尾列做截尾（1%/99%）

            for col in [

                "Z13",

                "Z18",

                "Z21",

                "ZX",

                "mapped\_ratio",

                "dup\_ratio",

                "filtered\_ratio",

                "q\_score",

            ]:

                a = X[col].to\_numpy()

                lo, hi = np.nanpercentile(a, 1), np.nanpercentile(a, 99)

                X[col] = X[col].clip(lo, hi)

        return X

    # ========= 采样×模型：构建管道 =========

    def \_sampler\_space(self):

        if not IMBLEARN\_OK:

            # 无imblearn时，仅返回“无采样”的假项

            return {"none": None}

        # SMOTE 邻居数会在 fit 时自动根据阳性样本数兜底

        return {

            "none": None,

            "ros": RandomOverSampler(random\_state=42),

            "rus": RandomUnderSampler(random\_state=42),

            "smote": SMOTE(random\_state=42, k\_neighbors=3),

        }

    def \_model\_space(self):

        # 两个强健基线

        return {

            "logreg\_l1": LogisticRegression(

                penalty="l1", solver="liblinear", max\_iter=2000, class\_weight="balanced"

            ),

            "rf": RandomForestClassifier(

                n\_estimators=300,

                max\_depth=None,

                min\_samples\_leaf=2,

                class\_weight="balanced",

                random\_state=42,

            ),

        }

    def \_build\_pipeline(self, sampler, model):

        if IMBLEARN\_OK:

            steps = [("scaler", RobustScaler())]

            if sampler is not None:

                steps.append(("sampler", sampler))

            steps.append(("clf", model))

            return ImbPipeline(steps)

        else:

            # 无imblearn时退化：仅缩放+模型（模型自身使用class\_weight）

            from sklearn.pipeline import Pipeline

            return Pipeline([("scaler", RobustScaler()), ("clf", model)])

    # ========= 训练/选择 =========

    def train(self, df: pd.DataFrame, random\_state=42):

        X = self.prepare\_female\_features(df)

        y = df["abnormal"].astype(int).to\_numpy()

        self.feature\_names = X.columns.tolist()

        # 外部测试集留出（分层）

        strat = y if y.sum() > 0 and (len(y) - y.sum()) > 0 else None

        X\_tr, X\_te, y\_tr, y\_te = train\_test\_split(

            X, y, test\_size=0.2, random\_state=random\_state, stratify=strat

        )

        # 仅在训练集上做K折CV

        skf = StratifiedKFold(

            n\_splits=min(5, max(2, int(y\_tr.sum()) if y\_tr.sum() > 1 else 2)),

            shuffle=True,

            random\_state=random\_state,

        )

        results = []

        best\_key, best\_auprc, best\_auroc, best\_pipe = None, -1.0, -1.0, None

        for s\_name, sampler in self.\_sampler\_space().items():

            for m\_name, model in self.\_model\_space().items():

                pipe = self.\_build\_pipeline(sampler, model)

                # 轻量网格，仅对关键超参

                if m\_name == "logreg\_l1":

                    param\_grid = {"clf\_\_C": [0.05, 0.1, 0.5, 1.0, 2.0]}

                else:  # rf

                    param\_grid = {"clf\_\_max\_depth": [None, 8, 12, 16]}

                grid = GridSearchCV(

                    pipe,

                    param\_grid=param\_grid,

                    scoring="average\_precision",  # 以AUPRC为主目标

                    cv=skf,

                    n\_jobs=-1,

                    refit=True,

                    verbose=0,

                )

                grid.fit(X\_tr, y\_tr)

                # 记录CV表现

                cv\_ap = grid.best\_score\_

                # 同时估计CV AUROC

                # 重新用同样cv手动计算一次（尽量节约时间，直接用best\_estimator\_预测折外）

                oof\_proba = np.zeros\_like(y\_tr, dtype=float)

                for fold, (a, b) in enumerate(skf.split(X\_tr, y\_tr)):

                    fold\_clf = self.\_build\_pipeline(sampler, model)

                    # 使用最佳参数

                    fold\_clf.set\_params(\*\*{k: v for k, v in grid.best\_params\_.items()})

                    fold\_clf.fit(X\_tr.iloc[a], y\_tr[a])

                    try:

                        oof\_proba[b] = fold\_clf.predict\_proba(X\_tr.iloc[b])[:, 1]

                    except Exception:

                        # 若模型不支持 predict\_proba

                        oof\_proba[b] = fold\_clf.decision\_function(X\_tr.iloc[b])

                cv\_auc = (

                    roc\_auc\_score(y\_tr, oof\_proba) if len(np.unique(y\_tr)) > 1 else 0.5

                )

                key = f"{s\_name}|{m\_name}"

                results.append(

                    {

                        "pipeline": key,

                        "best\_params": grid.best\_params\_,

                        "cv\_auprc": cv\_ap,

                        "cv\_auroc": cv\_auc,

                    }

                )

                # 选择规则：先看AUPRC，其次AUROC

                if cv\_ap > best\_auprc or (cv\_ap == best\_auprc and cv\_auc > best\_auroc):

                    best\_key = key

                    best\_auprc = cv\_ap

                    best\_auroc = cv\_auc

                    best\_pipe = grid.best\_estimator\_

        cv\_df = pd.DataFrame(results).sort\_values(

            ["cv\_auprc", "cv\_auroc"], ascending=False

        )

        cv\_df.to\_csv(

            "results/problem4\_cv\_summary.csv", index=False, encoding="utf-8-sig"

        )

        self.best\_pipeline\_name = best\_key

        self.best\_pipeline = best\_pipe

        # 在训练集上再次拟合最佳模型，然后用于测试集评估

        self.best\_pipeline.fit(X\_tr, y\_tr)

        # ========== 外部测试集仅一次评估 ==========

        y\_proba = self.\_predict\_proba(self.best\_pipeline, X\_te)

        y\_pred\_default = (y\_proba >= 0.5).astype(int)

        test\_metrics = self.\_evaluate\_on\_test(y\_te, y\_proba, y\_pred\_default)

        test\_metrics.update({"best\_pipeline": best\_key})

        pd.DataFrame([test\_metrics]).to\_csv(

            "results/problem4\_test\_summary.csv", index=False, encoding="utf-8-sig"

        )

        # 阈值优化（成本敏感：假阴性代价高）

        self.optimal\_threshold, thresh\_df = self.\_optimize\_threshold(

            y\_te, y\_proba, cost\_ratio=10

        )

        thresh\_df.to\_csv(

            "results/problem4\_threshold\_optimization.csv",

            index=False,

            encoding="utf-8-sig",

        )

        # 以最优阈值生成混淆矩阵等图表

        self.\_plot\_all(X, df["abnormal"].to\_numpy(), X\_te, y\_te, y\_proba)

        # 导出校准曲线点

        try:

            frac\_pos, mean\_pred = calibration\_curve(

                y\_te, y\_proba, n\_bins=min(10, max(3, len(y\_te) // 3))

            )

            pd.DataFrame({"mean\_pred": mean\_pred, "frac\_pos": frac\_pos}).to\_csv(

                "results/problem4\_calibration\_points.csv",

                index=False,

                encoding="utf-8-sig",

            )

        except Exception:

            pass

        # 导出特征重要性（如RF）

        self.\_export\_feature\_importance()

        print("\n=== 最佳Pipeline ===")

        print(best\_key)

        print(

            "测试集：AUPRC={:.3f}, AUROC={:.3f}, 阳性率(基线)={:.3f}".format(

                test\_metrics["auprc"], test\_metrics["auroc"], test\_metrics["pos\_rate"]

            )

        )

        print("最优阈值：{:.3f}".format(self.optimal\_threshold))

        return X\_te, y\_te

    # ========= 工具函数 =========

    @staticmethod

    def \_predict\_proba(model, X):

        try:

            return model.predict\_proba(X)[:, 1]

        except Exception:

            return model.decision\_function(X)

    @staticmethod

    def \_evaluate\_on\_test(y\_true, y\_proba, y\_pred):

        res = {}

        if len(np.unique(y\_true)) > 1:

            res["auroc"] = roc\_auc\_score(y\_true, y\_proba)

            res["auprc"] = average\_precision\_score(y\_true, y\_proba)

        else:

            res["auroc"] = 0.5

            res["auprc"] = np.mean(y\_true)  # 类先验

        res["pos\_rate"] = float(np.mean(y\_true))

        res["accuracy@0.5"] = float(np.mean(y\_pred == y\_true))

        try:

            res["brier"] = brier\_score\_loss(y\_true, y\_proba)

        except Exception:

            res["brier"] = np.nan

        # 分类报告（按0.5阈值）

        try:

            rep = classification\_report(y\_true, y\_pred, output\_dict=True)

            for k in ["0", "1", "macro avg", "weighted avg"]:

                if k in rep:

                    for m in ["precision", "recall", "f1-score", "support"]:

                        res[f"{k}\_{m}"] = rep[k].get(m, np.nan)

        except Exception:

            pass

        return res

    def \_optimize\_threshold(self, y\_true, y\_proba, cost\_ratio=10):

        thresholds = np.linspace(0.05, 0.95, 19)

        rows = []

        best\_score, best\_t = -1e9, 0.5

        for t in thresholds:

            y\_pred = (y\_proba >= t).astype(int)

            try:

                tn, fp, fn, tp = confusion\_matrix(y\_true, y\_pred).ravel()

            except Exception:

                # 类别不全时跳过

                continue

            sens = tp / (tp + fn) if (tp + fn) else 0

            spec = tn / (tn + fp) if (tn + fp) else 0

            prec = tp / (tp + fp) if (tp + fp) else 0

            # 成本敏感：假阴性代价高

            score = sens \* 0.7 + spec \* 0.3 - cost\_ratio \* fn / len(y\_true)

            rows.append(

                {

                    "threshold": t,

                    "sensitivity": sens,

                    "specificity": spec,

                    "precision": prec,

                    "tp": tp,

                    "fp": fp,

                    "fn": fn,

                    "tn": tn,

                    "cost\_score": score,

                }

            )

            if score > best\_score:

                best\_score, best\_t = score, t

        return best\_t, pd.DataFrame(rows)

    def \_plot\_all(self, X\_all, y\_all, X\_test, y\_test, y\_proba):

        # 画图与导出

        fig, axes = plt.subplots(2, 3, figsize=(18, 10))

        fig.suptitle("问题四：女胎异常判定（模型对比与测试集表现）", fontsize=16)

        # ROC

        if len(np.unique(y\_test)) > 1:

            fpr, tpr, \_ = roc\_curve(y\_test, y\_proba)

            auc = roc\_auc\_score(y\_test, y\_proba)

            axes[0, 0].plot(fpr, tpr, label=f"Best (AUC={auc:.3f})")

            axes[0, 0].plot([0, 1], [0, 1], "k--", alpha=0.5)

            axes[0, 0].set\_title("ROC 曲线")

            axes[0, 0].set\_xlabel("FPR")

            axes[0, 0].set\_ylabel("TPR")

            axes[0, 0].legend()

        else:

            axes[0, 0].text(0.5, 0.5, "单一类别，无法绘制ROC", ha="center", va="center")

        # PR

        if len(np.unique(y\_test)) > 1:

            prec, recall, \_ = precision\_recall\_curve(y\_test, y\_proba)

            ap = average\_precision\_score(y\_test, y\_proba)

            axes[0, 1].plot(recall, prec, label=f"Best (AP={ap:.3f})")

            axes[0, 1].axhline(

                np.mean(y\_test),

                ls="--",

                alpha=0.5,

                label=f"阳性率基线={np.mean(y\_test):.3f}",

            )

            axes[0, 1].set\_title("Precision-Recall 曲线")

            axes[0, 1].set\_xlabel("Recall")

            axes[0, 1].set\_ylabel("Precision")

            axes[0, 1].legend()

        else:

            axes[0, 1].text(0.5, 0.5, "单一类别，无法绘制PR", ha="center", va="center")

        # 校准

        try:

            frac\_pos, mean\_pred = calibration\_curve(

                y\_test, y\_proba, n\_bins=min(10, max(3, len(y\_test) // 3))

            )

            axes[0, 2].plot([0, 1], [0, 1], "k:", label="完全校准")

            axes[0, 2].plot(mean\_pred, frac\_pos, "s-", label="Best")

            axes[0, 2].set\_title("校准曲线")

            axes[0, 2].set\_xlabel("平均预测概率")

            axes[0, 2].set\_ylabel("实际阳性率")

            axes[0, 2].legend()

        except Exception:

            axes[0, 2].text(0.5, 0.5, "无法绘制校准曲线", ha="center", va="center")

        # 特征：Z值分布概览

        for col, ax in zip(["Z13", "Z18", "Z21"], [axes[1, 0], axes[1, 1]]):

            if col in X\_all.columns:

                ax.hist(X\_all[col], bins=20, alpha=0.8)

                ax.axvline(3.0, color="r", ls="--", alpha=0.7)

                ax.axvline(-3.0, color="r", ls="--", alpha=0.7)

                ax.set\_title(f"{col} 分布")

        # 混淆矩阵（最优阈值）

        y\_pred\_opt = (y\_proba >= self.optimal\_threshold).astype(int)

        if len(np.unique(y\_test)) > 1 and len(np.unique(y\_pred\_opt)) > 1:

            cm = confusion\_matrix(y\_test, y\_pred\_opt)

            sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=axes[1, 2])

            axes[1, 2].set\_title(f"混淆矩阵 (阈值={self.optimal\_threshold:.2f})")

            axes[1, 2].set\_xlabel("预测")

            axes[1, 2].set\_ylabel("实际")

        else:

            axes[1, 2].text(0.5, 0.5, "无法绘制混淆矩阵", ha="center", va="center")

        plt.tight\_layout()

        plt.savefig("results/problem4\_analysis.png", dpi=300, bbox\_inches="tight")

        plt.close(fig)

    def \_export\_feature\_importance(self):

        # 仅对 RF 导出特征重要性

        try:

            if self.best\_pipeline is None:

                return

            # 从 pipeline 中取出 rf

            clf = self.best\_pipeline.named\_steps.get("clf", None)

            if isinstance(clf, RandomForestClassifier):

                imp = pd.DataFrame(

                    {

                        "feature": self.feature\_names,

                        "importance": clf.feature\_importances\_,

                    }

                ).sort\_values("importance", ascending=False)

                imp.to\_csv(

                    "results/problem4\_feature\_importance\_rf.csv",

                    index=False,

                    encoding="utf-8-sig",

                )

        except Exception:

            pass

# ============== 运行入口 ==============

def run\_problem4(input\_excel\_path: str):

    print("=" \* 60)

    print("问题4：女胎异常判定（Pipeline+四采样×两模型）")

    print("=" \* 60)

    proc = NIPTDataProcessor()

    if not proc.load\_data(input\_excel\_path):

        print("数据加载失败，请检查路径/工作表名称。")

        return None

    female = proc.preprocess\_female\_data()

    if female.empty:

        print("女胎数据为空，退出。")

        return None

    solver = Problem4Solver(proc, winsorize=False)

    X\_test, y\_test = solver.train(female)

    print("\n生成的主要文件：")

    print("- results/problem4\_cv\_summary.csv        （采样×模型CV对比：AUPRC/AUROC）")

    print("- results/problem4\_test\_summary.csv      （最佳Pipeline外部测试集指标）")

    print("- results/problem4\_threshold\_optimization.csv （阈值-性能曲线）")

    print("- results/problem4\_calibration\_points.csv （校准曲线点集）")

    print("- results/problem4\_feature\_importance\_rf.csv （若最佳为RF）")

    print("- results/problem4\_analysis.png           （ROC/PR/校准/混淆矩阵等图）")

    return solver

if \_\_name\_\_ == "\_\_main\_\_":

    # TODO: 请把下面路径改成你的附件.xlsx 的实际路径

    excel\_path = (

        r"D:\HP\OneDrive\Desktop\学校\竞赛\数模国赛\CUMCM2025Problems\C题\附件.xlsx"

    )

    run\_problem4(excel\_path)