

针对 NIPT 的时点选择与胎儿的异常判定的分析

摘要

本研究围绕 NIPT（无创产前检测）中胎儿 Y 染色体浓度与孕妇 BMI、孕周等因素的关系，基于 BMI 分组与动态优化提出检测时点选择模型，并解决多个相关问题。

针对问题一：研究了胎儿 Y 染色体浓度与孕妇 BMI、孕周的关系。采用多元回归模型，研究了 BMI、孕周及其多项式项（如平方、立方项）对 Y 染色体浓度的非线性影响，并扩展到多项式回归以捕捉这些非线性关系。引入了随机效应来反映个体差异，优化了回归模型的拟合度。

针对问题二：在问题一模型的基础上，进一步考虑孕妇 BMI 与孕周的差异性，采用 K-means 聚类方法对孕妇进行 BMI 分组，以捕捉不同 BMI 群体在 Y 染色体浓度上升速度上的差异。通过动态优化，结合过早检测与过晚发现的风险，确定了每个 BMI 组的最佳 NIPT 检测时点。模型实现了“过早检测”和“过晚发现”之间的平衡，从而最小化综合风险。

针对问题三：从胎儿 Y 染色体浓度随孕周变化动态性的角度出发，研究采用混合效应模型，结合 BMI、孕周和其他生理因素，探究了不同 BMI 人群的浓度变化轨迹。通过纵向数据建模，确定了每个群体的最佳检测时点，并使用动态规划方法进一步优化了各 BMI 组的检测时点选择。最终结果表明，BMI 的确对最佳检测时点有显著影响。

针对问题四：根据女胎缺乏 Y 染色体标志物的情况，研究构建了基于多维生物标志物的异常判定框架，分析了与女胎异常判定相关的 13 个候选指标（如染色体 Z 值、GC 含量、读段数、BMI 等）。通过 Spearman 等级相关系数分析了这些指标与异常判定结果的关系，并基于前期分析筛选出的关键特征，构建了机器学习模型。模型优化了数据中的类别不平衡问题，提高了女胎异常的判定准确性和召回率。

关键词：NIPT，BMI，最佳检测时点，动态优化，K-means 聚类，机器学习，假阴性，假阳性，女胎异常判定

一、问题重述

1.1 问题背景

随着医学检测技术的进步，在2010后，无创产前检测（NIPT, Non-Invasive Prenatal Testing）凭借高安全性和高准确率，逐渐成为我国孕期筛查的重要方法。它通过采集孕妇血浆中的胎儿游离DNA，对胎儿染色体数目异常进行检测，在避免传统有创产检带来流产风险的同时，显著提高了孕期检测的可接受度和普及性^[1]。

然而，NIPT的检测结果不是绝对稳定，而是受到多重因素影响。研究表明，孕妇体质指数（BMI）、孕周以及胎儿DNA浓度等变量均会影响检测准确性。其中，胎儿Y染色体浓度是判断检测可靠性的关键指标，但其浓度随孕周呈现复杂的非线性变化，并且不同孕妇间存在显著差异。对于BMI较高的孕妇，由于外周血中胎儿DNA相对浓度较低，往往需要更晚的孕周才能达到可靠检测的阈值，而BMI较低的孕妇则可能在较早孕周即可满足检测条件。如果太早进行NIPT检测，会导致胎儿DNA含量不够，造成检测假阴性的可能；如果太晚进行NIPT检测，容易错过发现问题的最佳时间点，造成临床的风险和社会成本上升。

此外，对于女胎而言，由于缺乏Y染色体这一显著标志物，判定其异常需要综合考虑染色体Z值、GC含量、读段数等多维生物学特征。这使得判定过程更为复杂，同时增加了假阳性和假阴性风险。因此，如何在不同孕周和BMI条件下建立合理的检测时点选择模型，并结合多维特征构建高效的异常判定机制，成为当前NIPT应用与推广的关键科学问题。在此基础上，有必要通过数学建模和数据分析的方法，对NIPT检测的多因素影响进行系统研究，从而为我国孕期产检服务提供更科学的决策依据。这不仅有助于提升检测准确性和降低漏诊率，并能够在更大范围内改善孕期健康管理水的同时，推动产前筛查在我国的规范化与精准化发展^[2]。

本题所提供的数据包含孕妇的BMI、孕周数、胎儿Y染色体浓度、胎儿性别以及多维生物学指标等。为便于对数据特征进行初步了解，附有两幅分布图：

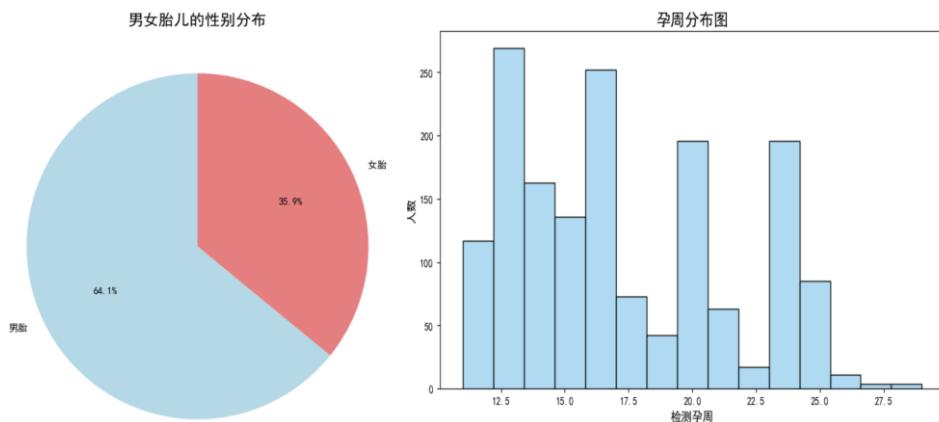


表 1.1 男女性别分布及孕周分布图

1.2 要解决的问题

附件所给出的是关于孕妇无创产前检测（NIPT）相关的临床检测数据，包括孕妇的体质指数（BMI）、孕周数、身高、体重、年龄等基本信息，以及胎儿Y染色体浓度、相关检测误差、各类染色体的Z值、GC含量和读段数等检测指标。为进一步科

学评估不同孕妇在不同检测条件下的最佳检测时点，降低临床风险，提高检测的准确性与可靠性，现需结合实际情况与附件所给信息建立数学模型，分析以下问题：

问题一：分析胎儿 Y 染色体浓度与孕妇的孕周数、BMI 等指标之间的关系，建立相应的数学模型，并进行显著性检验，验证各因素对 Y 染色体浓度的影响程度。

问题二：研究男胎孕妇的 BMI 对胎儿 Y 染色体浓度最早达标时间的影响，合理分组 BMI 并确定各组的最佳 NIPT 检测时点，优化检测时机，降低潜在风险，并分析检测误差的影响。

问题三：综合考虑身高、体重、年龄等因素以及 Y 染色体浓度达标比例，结合男胎孕妇 BMI，确定合理的分组方案和最佳 NIPT 检测时点，最小化孕妇潜在风险，并分析检测误差的影响。

问题四：针对女胎异常判定，结合 X 染色体、21 号、18 号和 13 号染色体的 Z 值、GC 含量、读段数、BMI 等因素，提出科学的女胎异常判定方法。

二、问题分析

本研究围绕胎儿 NIPT 检测时点优化模型的构建与验证，依次解决了 BMI 分组与最佳 NIPT 时点分析、多因素回归分析与最佳检测时点优化、检测误差对结果的影响分析、以及胎儿异常判定方法的模型构建等四个核心问题。

2.1 问题一的分析

本问题的核心是通过统计模型分析影响 Y 染色体浓度的各个因素（如孕妇 BMI、孕周及其高次项、交互项等），从而提出合理的最佳 NIPT 检测时点预测模型。主要任务是建立一个能够捕捉这些因素之间复杂关系的回归模型。

本问题的难点在于如何有效捕捉 BMI、孕周等因素之间的复杂交互作用，并通过引入非线性关系来增强模型的拟合能力。另一个难点是如何平衡随机效应和固定效应，在处理个体差异的同时保证模型的稳定性和预测能力。

我们使用了多元回归分析来建模，确定孕妇 BMI 和孕周与 Y 染色体浓度的直接关系，并进一步应用到多项式回归来捕捉这些因素之间的非线性关系。另外，模型还引入了随机效应和固定效应，以考虑个体差异对预测结果的影响，从而提高模型的拟合能力。

2.2 问题二的分析

本问题的核心是，在第一问基础模型的基础上，进一步考虑孕妇 BMI 与孕周之间的差异性，寻找合理的分组策略，并给出各组的最佳 NIPT 检测时点，从而最小化“过早检测”和“过晚发现”的综合风险。

本问题的难点在于如何兼顾“过早检测”和“过晚发现”。通过计算每个孕周点的风险，并对各风险进行加权求和，我们能够得出兼顾两类风险的最优检测时点。此外，BMI 分组的合理性及其对最佳检测时点的影响也是模型建立中的主要问题。

我们的思路是首先使用 K-means 聚类方法将孕妇按照 BMI 水平进行分组，捕捉不同 BMI 人群在 Y 染色体浓度变化上的差异。然后，在每个 BMI 组内，利用纵向数据拟合孕周与 Y 浓度的关系曲线，并结合检测噪声，计算在某一孕周时点检测成功的概率及累积达标比例。接着，通过设定过早与过晚的成本函数，在候选孕周范围内搜索期望成本最小的时点，即每个组的最佳检测孕周。

2.3 问题三的分析

本问题的核心是利用纵向建模的方法，分析 Y 染色体浓度随孕周的动态变化，并结合孕妇的 BMI 等特征，确定不同人群的最佳检测孕周 (t^*)。与前两个问题相比，本问题更强调在“过早检测”和“过晚检测”之间进行权衡。

本问题的难点在于如何同时解决纵向非线性、BMI 的分层差异以及成本最优化问题。通过引入随机效应来刻画个体差异，从而提高模型的稳健性和泛化能力。

我们采用了混合效应模型，控制年龄和 IVF 等因素的影响，进一步引入孕周和 BMI 的非线性效应及其交互作用来刻画不同 BMI 人群的浓度变化曲线。为了平衡“过早检测”和“过晚检测”之间的风险，我们构建了成本函数，并利用动态规划方法来确定不同 BMI 组的最优检测孕周。

2.4 问题四的分析

本问题的核心是当女胎缺乏 Y 染色体标志物的特殊情况，建立基于多维生物标志物的异常判定框架，评估 13 个候选指标对女胎异常判定的影响，并通过这些指标为女胎异常提供一个更加精确的判定方法。

我们首先采用完整案例分析策略，确保所有样本的 13 个候选指标无缺失值。通过 Spearman 等级相关系数，分析三个 BMI 分组下的候选指标与异常判定结果之间的关系。此外，我们基于前期分析结果，筛选出对异常判定最重要的特征，并构建机器学习分类模型来提高异常判定的准确性和召回率。

本研究的难点在于如何有效整合多维度特征、解决数据不平衡问题、控制假阴性风险，并优化模型以提升异常判定的准确性和召回率。

三、模型假设

3.1 胎儿 Y 染色体浓度与孕妇 BMI 和孕周的相关性假设：

假设胎儿的 Y 染色体浓度与孕妇 BMI 和孕周之间存在显著的线性或非线性关系。尤其是男胎的 Y 染色体浓度与孕妇 BMI 和孕周的关系较为密切。假设这些关系可以通过统计模型来描述，为不同 BMI 群体确定最佳的 NIPT 时点，以此达到优化检测的准确性和风险最小化。

3.2 数据分布与相关性分析假设：

假设所使用的数据（包括基因组特征、孕妇生理特征等）可能不符合正态分布，则不适用传统的 Pearson 相关系数，选用 Spearman 等级相关系数来衡量变量之间的相关性更合适。Spearman 系数比 Pearson 系数对非线性关系和离群值更为稳健，因此假设 Spearman 系数能够更准确地反映胎儿染色体浓度与其他变量之间的关系。

3.3 数据误差与类别不平衡假设：

假设数据中可能存在测序误差和其他影响因素（例如：检测时点过早或过晚等），这些误差可能影响 NIPT 结果的准确性。同时，异常样本（即检测到异常的胎儿）通常占比很小，因此假设数据集中会存在类别不平衡问题，正类（异常）的样本较少。为了应对这一问题，假设采用代价敏感学习策略、过采样或欠采样技术能有效减少类别不平衡的影响，特别是减少假阴性（漏诊）的风险。

3.4 模型评估与假设：

假设不同的机器学习模型（如逻辑回归、支持向量机、随机森林等）在女胎异常

判定中各有试用性，因此假设通过对多种模型进行比较，能够选择出最适合的模型，并根据评估指标（如准确率、精确率、召回率、F1 分数等）对模型进行调整和优化。假设通过调整模型的预测阈值，可以进一步优化精确率和召回率之间的平衡。

3.5 假阴性惩罚假设：

在模型训练中，假设漏诊（假阴性）的临床后果比误诊（假阳性）更为严重，因此在训练过程中会给予假阴性更高的惩罚权重。假设这一策略能有效减少漏诊的风险，尤其是在医学应用中，提高模型对异常样本的敏感性。

四、 符号说明

符号	意义
GW_{IJ}	第 <i>i</i> 个孕妇第 <i>j</i> 次检测时的孕周数
BMI_{ij}	第 <i>i</i> 个孕妇第 <i>j</i> 次检测时的体重指数
Y_{ij}	第 <i>i</i> 个孕妇第 <i>j</i> 次检测时的 Y 染色体浓度
μ, σ	正态分布的期望与标准差
$V(t)$	孕周 <i>t</i> 的 Y 染色体浓度
IVF_i	第 <i>i</i> 个孕妇的妊娠方式
Age_i	第 <i>i</i> 个孕妇的年龄
t^*	最佳检测时点
$\epsilon_{ij}, \varepsilon_{ij}$	测量误差或噪音项
u_i	孕妇个体的随机截距
$bs(week)_i$	<i>i</i> 维度下时间样条的固定效应
$bs(BMI)_i$	<i>i</i> 维度下 BMI 样条的固定效应
$bs(week)_i * BMI$	<i>i</i> 维度下时间和 BMI 交互样条的固定效应
X_i	第 <i>i</i> 个孕妇的多因素特征向量

五、 模型的建立与求解

5.1 针对问题一模型的建立与求解

5.1.1 数据预处理与特征选择

对于建立的回归模型，在数据清洗与预处理时，操作包括填补缺失值、处理异常值等。通过分析数据中的不同特征，选取了孕妇的 BMI、孕周作为自变量，Y 染色体浓度作为因变量。同时，我们还将自变量进行了多项式转换，以捕捉可能的非线性关系。

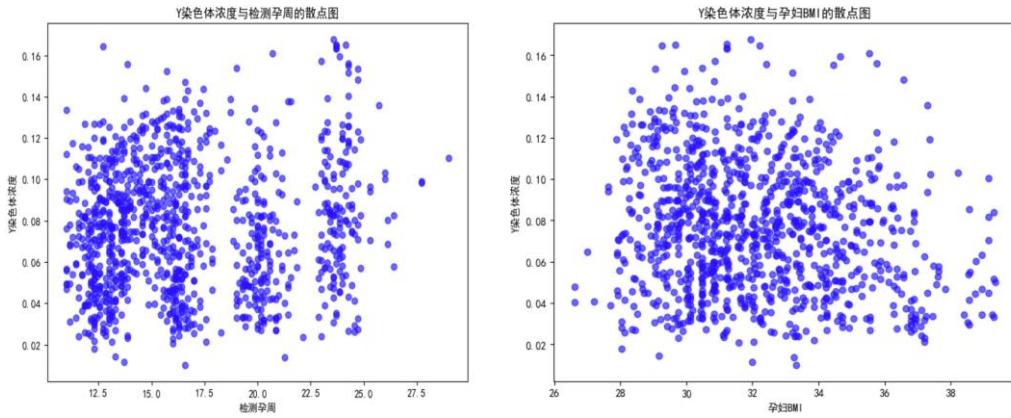


图 5.1.1 Y 染色体浓度与检测孕周、孕妇 BMI 之间的关系散点图

5.1.2 相关性分析

在回归模型建立之前，我们首先需要做相关性分析，用来探讨自变量与因变量之间的线性关系。具体步骤如下：

- 皮尔逊相关系数：衡量两个变量之间的线性关系，范围从-1 到 1，其中 1 表示完全正相关，-1 表示完全负相关，0 表示无线性关系。
- Spearman 等级相关系数：衡量两个变量之间的单调关系，不要求数据具有正态分布，适用于非线性关系。

通过相关性分析，我们可以了解孕周数、BMI 及其多项式项和交互项与 Y 染色体浓度之间的关系，从而帮助我们更好地理解这些因素之间的依赖性。

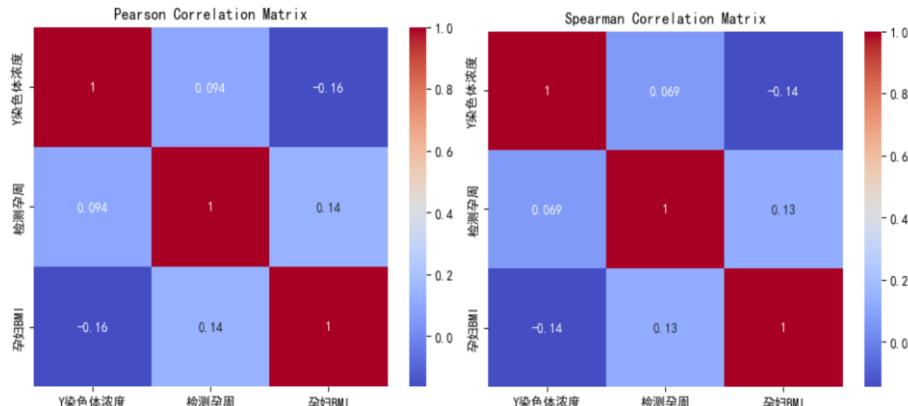


图 5.1.2 Y 染色体浓度与检测孕周、孕妇 BMI 的相关性分析图

总结：

变量	均值	标准差	最小值	最大值	Pearson 相关系数	Spearman 相关系数
Y 染色体浓度 (%)	7.66	3.10	1.00	16.76	1	1
检测孕周 (周)	16.75	4.04	11.00	29.00	0.094	0.069
孕妇 BMI (kg/m^2)	32.14	2.53	26.62	39.35	-0.16	-0.14

表 5.1.1 有关 Y 染色体浓度，检测孕周和孕妇 BMI 的数据特性及相关性分析系数

- 皮尔逊相关系数：Y 染色体浓度与孕周数和 BMI 的相关性分别是 0.094 和-0.16，这说明两者之间存在弱正相关和弱负相关。显然，皮尔逊相关系数表明这些变量之间没有强烈的线性关系。
- Spearman 等级相关系数：Y 染色体浓度与孕周数 和 BMI 的相关性分别为 0.069

和 0.13，均为较弱的正相关，这与皮尔逊的结果一致。

5.1.3 回归模型的建立

在本问题中，我们主要研究对象包括以下因素：

- 孕周数 (GW)：影响胎儿发育的重要因素。
- BMI (体质指数)：孕妇的体重指数，可能与胎儿发育情况相关。

运用多项式回归，我们可以进一步考虑孕周数和 BMI 的平方项，以捕捉这些变量与 Y 染色体浓度之间可能的非线性关系。

我们采用线性回归模型 (OLS)，随机森林模型与混合效应模型 (Mixed Effects Model) 来分析数据，探讨孕周和 BMI 等变量与胎儿 Y 染色体浓度之间的复杂关系。

线性回归模型：

$$Y = \beta_0 + \beta_1 GW + \beta_2 BMI + \epsilon_t$$

其中：

- Y 表示 Y 染色体浓度
- GW 表示检测孕周
- BMI 表示孕妇的 BMI
- ϵ_t 表示每次测量的误差项

随机森林模型：

$$\hat{Y} = f(X) = \frac{1}{T} \sum_{t=1}^T h_t(X)$$

其中 $X = (\text{检测孕周}, \text{孕妇} BMI)$ ， \hat{Y} 表示预测的胎儿 Y 染色体浓度。

非线性混合效应模型：

$$Y_{ij} = \beta_0 + \beta_1 GW_{ij} + \beta_2 BMI_{ij} + \beta_3 GW_{ij}^2 + \beta_4 GW_{ij}^3 + \beta_5 BMI_{ij}^2 \\ + \beta_6 (BMI_{ij} * GW_{ij}) + u_{oi} + \epsilon_{ij}$$

其中：

- Y_{ij} 表示第 i 个孕妇第 j 次检测时的 Y 染色体浓度
- GW_{ij} 表示第 i 个孕妇第 j 次检测时的孕周数
- BMI_{ij} 表示第 i 个孕妇第 j 次检测时的体重指数

5.1.4 线性回归模型和混合效应模型结果、比较与最终解释

变量	OLS 回归模型	随机森林模型	非线性混合效应模型
检测孕周	0.0009 *** (0.000)		0.066 *** (0.014)
孕妇 BMI	-0.0022 *** (0.000)		0.022 *** (0.008)
检测孕周_square			-0.004 *** (0.001)
检测孕周_cubic			0.000 ***

			(0.000)
孕妇 BMI_square			-0.000***
			(0.000)
BMI_检测孕周 _interaction			-0.000***
R_Squared	0.040	0.1587	(0.000)
Log-Likelihood	2168.1	-55.61	2219.3183
MSE	0.00093	0.000797	3.58e-06

表 5.1.2 各模型的变量系数, 标准差及显著性检验表 (标准差位于变量系数下方, ***表示 $p - value < 0.01$, **表示 $p - value < 0.05$, *表示 $p - value < 0.1$)

非线性混合效应模型在 Log-Likelihood 和 MSE 方面均表现最好, 说明该模型能够充分兼顾数据中的群体效应和非线性关系, 提供最优的拟合和预测能力。随机森林模型同样表现优秀, 特别是在数据的非线性关系处理上, 但其在模型复杂度和可解释性方面可能不如混合效应模型。OLS 回归模型的表现较差, 特别是在捕捉数据的非线性和交互效应方面, 其线性假设限制了模型的有效性。因此, 若数据存在复杂的非线性关系或群组效应, 建议使用非线性混合效应模型或随机森林等更复杂的模型。

综上所述, 胎儿 Y 染色体浓度与孕周期和 BMI 的关系既复杂的、又具有显著的非线性和很强烈的个体效应。Y 染色体浓度随孕周呈现一个非线性的三次函数关系, 其增长率随孕期变化而变化; 而孕妇 BMI 对其表现出非线性负向影响, BMI 越高, Y 染色体浓度会偏低。该模型的显著性也通过上述模型得到验证。

5.2 针对问题二模型的建立与求解

5.2.1 数据预处理与特征选择

在建模之前, 需要先完成数据预处理和特征选择。核心变量包括检测孕周、Y 染色体浓度以及孕妇 BMI。Y 浓度统一转换为比例形式, 如果原始数据以百分数表示, 则需换算为小数。在数据清理阶段, 仅保留孕周、浓度和 BMI 处于合理范围内的记录。对于同一孕妇在同一孕周的多次检测结果, 合并处理: Y 浓度取平均值, BMI 取中位数, 以避免重复计数带来的偏差。缺失关键字段的记录直接删除, 其余数值型变量则统一转换为数值格式, 以防文字与数字混杂影响后续计算。在特征选择方面, BMI 是最关键的分组依据。体型差异直接影响游离 DNA 浓度随孕周增长的上升速度, 因此以 BMI 划分人群是合理的。孕周与 Y 浓度之间可能存在非线性关系, 建模时可采用样条函数或多项式来进行拟合。年龄、身高、体重以及是否为 IVF 等变量作为辅助特征, 主要用于解释组内差异, 但不会改变以 BMI 为核心的分组主线。

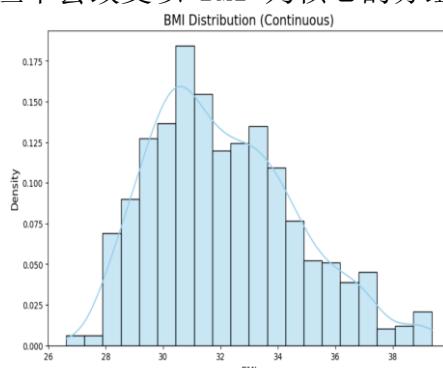


图 5.2.1 孕妇 BMI 分布图

5.2.2 模型的建立

在问题二中，需要对男胎孕妇的 BMI 进行合理分组，并在此基础上为每个 BMI 组确定最佳的 NIPT 检测时点，以最小化潜在风险（假阴性和假阳性）。为实现这一目标，研究结合了聚类分析与动态优化方法。

5.2.2.1 聚类分析

在本题中，首先利用聚类分析将男胎孕妇按照 BMI 分成三组，以刻画不同体重区间下胎儿 Y 染色体浓度达标时间的差异。通过聚类分析，可以提取出各区间的群体特征，从而为后续优化奠定分组基础。具体而言，采用 K-means 算法，步骤如下：

1. 初始化簇心：根据样本的 BMI 分布，选取 K 个初始簇心，并初步确定分组数。
2. 分配过程：计算每位孕妇的 BMI 与各簇心之间的欧氏距离，并将其划分到距离最近的簇中。簇心的选择直接影响分组效果，因此需要保证组间差异尽可能大，组内差异尽可能小。
3. 更新过程：计算每个簇内孕妇 BMI 的均值，并将其作为新的簇心。重复该过程，直到簇心收敛或达到最大迭代次数。

该方法的目标函数是最小化所有孕妇的 BMI 与其所属簇心之间的距离总和，从而实现有效分组。

目标函数：目标是最小化每个孕妇 BMI 与其所属簇心之间的距离总和，即：

$$J = \sum_{k=1}^K \sum_{x_j \in C_k} \|x_j - c_k\|^2$$

其中， x_j 表示第 j 个孕妇的 BMI 数据， c_k 是第 k 个簇的簇心。

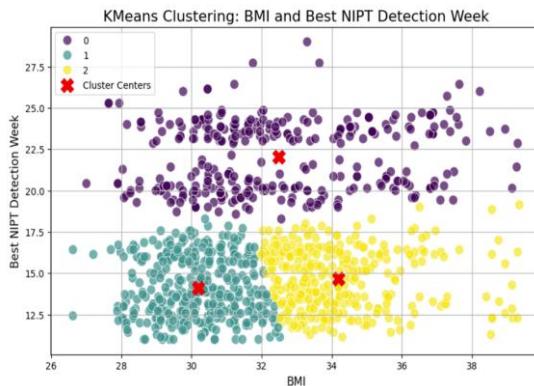


图 5.2.2 KMeans 聚类散点图

在本研究中，通过 K-means 聚类对孕妇 BMI 与最佳 NIPT 检测时点的关系进行分析，得到了以下分组结果：

簇	BMI 范围	聚类中心 BMI	最佳检测时点
紫色	[28, 32]	30	20
绿色	[32, 34]	33	18
黄色	[34, 38]	36	14

表 5.2.1 各个 BMI 范围内的最佳时点表

结论：研究结果显示，随着 BMI 升高，最佳 NIPT 检测时点越早。这一趋势表明，BMI 对最佳 NIPT 检测时点具有显著影响，BMI 较高的孕妇建议尽早进行 NIPT 检测。通过 K-means 聚类，我们发现 BMI 较低的孕妇（第一簇）主要集中在后期的检测时点，而 BMI 较高的孕妇（第三簇）则集中在更早的检测时点。这为个性化的 NIPT 检测策略提供了依据，并强调了 BMI 在孕期检测时点选择中的重要性。

风险分析：通过均方误差（MSE）来评估每组的潜在风险大小。MSE 代表了预测值与实际值之间的差异，MSE 越小，潜在风险越小。

BMI 组	MSE
[28, 31]	14.7898
[31, 34]	16.0691
[34, 37]	20.8291
[37, 40]	20.0779

表 5.2.2 各个 BMI 组的 MSE 表

从 MSE 的结果可以看出，[28, 31) 组的风险最小，而 [34, 37) 和 [37, 40) 组的风险较高。

5.2.2.2. 动态优化

在完成聚类分析之后，引入动态优化方法来求解每个 BMI 组的最佳 NIPT 检测时点。动态优化能够在给定检测误差条件下寻找最优检测时机，从而降低“过早检测”和“过晚发现”的风险。

在建模前，需要对孕妇进行合理分组。第一步是根据 BMI 将人群划分为若干区间。分组的目的在于让同一组内的个体在生理变化速度上尽量接近，使“最佳检测周”的差异主要体现于组间而不是组内。在实际求解中，BMI 最终被分为四类。这是样本量、解释性和稳定性之间的折中：分组过少难以反映人群差异，分组过多则会导致小样本不稳定，并可能引入过拟合风险。四个分组的边界并非人工指定，而是通过枚举候选边界并计算各区间的检测成本，再利用动态规划在所有切分方式中选择总成本最小的方案。这样得到的分组既保证了组内个体相似性，又实现了全局最优。最终的四类人群既有足够样本量支撑统计可靠性，也能在不同 BMI 水平上呈现差异化的推荐时点。

在每一类人群内部，需要将“第一次达到 4% 的孕周”看作一个随机变量。这是因为个体生理差异和检测噪声会导致达标时刻存在不确定性。具体做法是：对每位孕妇，在孕周序列中寻找首次达到或超过 4% 的时间点。如果检测仅在离散周次进行，则在相邻两次检测之间采用线性插值，从而更精确地确定首次达标的孕周。

$$T = t_m + \frac{0.04 - V(t_m)}{V(t_{m+1}) - V(t_m)} \times (t_{m+1} - t_m)$$

其中 $V(t)$ 表示在孕周 t 的 Y 染色体浓度，这样可以更精细确定首次达标时刻 T 。

把所有人的首次达标时间 T 汇总后可以得到该组的达标时间分布 $F(t)$ ，这相当于在任意周 t 之前有多少比例的人已经达到 4%。基于 $F(t)$ 可以计算在 t 周去检测时“仍未达标”的概率 $P(T > t)$ ，这就是早到风险的度量；

$$F(t) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}\{T_i \leq t\}, \quad S(t) = 1 - F(t) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}\{T_i > t\}$$

其中， $S(t)$ 表示在 t 周去检测时仍未达标的概率（早到风险）。同时也可以衡量“晚发现”的风险，常用的做法是在一个临床参考周 W_{late} 之后开始计入迟到影响， W_{late} 取 12 周，这样在 t 之前达标但晚于 12 周的人群比例 $F(t) - F(W_{late})$ 就反映了延后带来的潜在不利。

两类风险需要合并成一个可比较的指标，因此引入期望成本

$$J(t) = c_{early} \cdot S(t) + c_{late} \cdot [F(t) - F(W_{late})]$$

其中 c_{early} 与 c_{late} 分别表示“过早检测”和“过晚发现”的代价权重。

$J(t)$ 在一个合理的孕周网格上进行逐点计算，网格通常选在 11 到 16 周之间，步长可取 $\frac{1}{4}$ 周以兼顾精度与计算速度。对每个 BMI 组，沿着网格寻找使 $J(t)$ 取得最小值的周次，把该周次记作该组的最佳检测时点 t^* 。

组内最优检测时点为：

$$t^* = \operatorname{argmin}_t J(t)$$

通过这样的分配，得到的是在本组样本分布和设定权重下的最优权衡点，既控制了过早带来的重复检测，又避免了过晚带来的延误风险。

全局层面的求解依采用动态规划思想来求解。具体做法是先对所有候选 BMI 区间预先计算“若把人群恰好划在这个区间内时的最优 t 与对应的区间成本”，把这些成本组成一个区间成本矩阵。

设 BMI 轴上候选分割点为 $b_0 < b_1 < \dots < b_M$ 。任意区间 $(b_i, b_j]$ 的最优成本定义为：

$$C(i, j) = \min_t J_{(i, j]}(t)$$

其中 $J_{(i, j]}(t)$ 表示仅用区间 $(b_i, b_j]$ 内孕妇计算得到的期望成本

然后给定希望的组数，动态规划在所有可能的切分序列中找到总成本之和最小的那条路径，同时输出每个区间的最佳 t 。

设目标分组数为 K ，动态规划状态转移方程为：

$$DP[j, k] = \min_{0 \leq i < j} \{DP[i, k - 1] + C(i + 1, j)\}$$

边界条件为 $DP[0, 0] = 0$ ，其余 $DP[j, 0] = +\infty$

最优解为 $DP[M, K]$ ，并通过回溯得到分组边界及每组的最优检测时点 t^* 。

这种方法的优势在于：分组和时点不是分别单独决定，而是放在一个统一的目标函数里一并优化，最终得到的四组边界与各组 t^* 是相互配合、整体最优的。

数据结果：

BMI 组别	推荐检测周	早到概率	晚发现概率	期望样本
BMI $\in [26.81, 31.22]$	14.75	0.880	0.120	0.9682
BMI $\in [31.22, 32.91]$	14.00	0.904	0.096	0.9343

$BMI \in [32.91, 34.16]$	12.50	0.949	0.026	0.9530
$BMI \in [34.16, 39.30]$	13.00	0.925	0.057	0.9524

表 5.2.3 各个 BMI 组的最佳检测周及早到、晚发现概率表

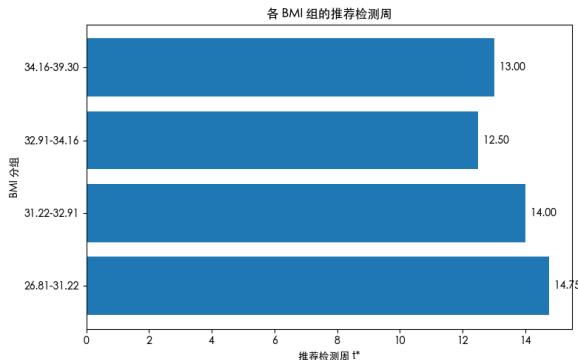


图 5.2.2 各 BMI 的推荐检测周表

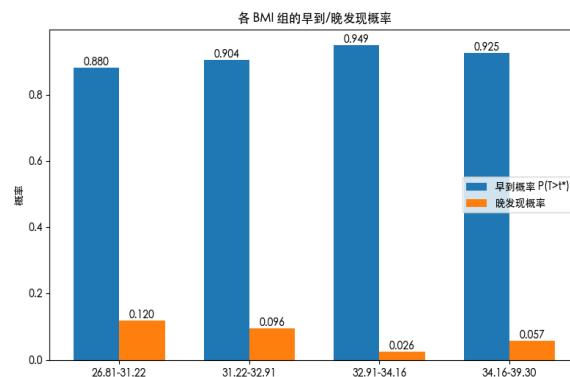


图 5.2.3 各 BMI 组的早到/晚发现概率图

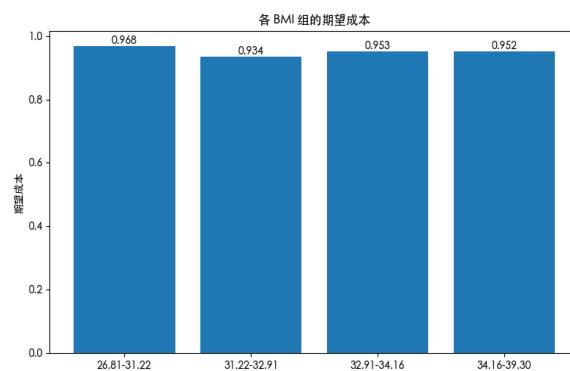


图 5.2.4 各 BMI 组的期望成本图

综合三幅图可以看到，BMI 的确影响最佳检测时点：BMI 越高的孕妇，其推荐检测周数相对更早（图 5.2.2）。这一结果符合生理直觉，即体型较大的人群在游离 DNA 浓度达到阈值时往往更慢，需要提前安排检测。进一步来看，各 BMI 组的早到概率均超过 88%，这表明推荐周数能够有效避免晚发现的风险，但同时也意味着部分孕妇可能需要重复检测（图 5.2.3）。在此基础上，不同分组之间的期望成本保持在接近水平，没有出现明显的不均衡（图 5.2.4），说明分组方案兼顾了准确性与公平性。

整体而言，本题采用的“先分组、再定时点”的方法能够把体型差异与检测时机纳入同一框架，并通过动态规划在保证风险可控的同时实现全局最优。该方案不仅在

医学上具有合理性，而且在实际应用中也具备可操作性：一方面可以为不同 BMI 人群提供差异化的检测建议，另一方面又避免了过度复杂化，保证了结论的清晰性和可推广性。

5.2.3 对于问题二的结论

我们结合聚类分析与动态优化方法，建立了一个基于 BMI 分组的 NIPT 检测时点优化模型。聚类分析用于对孕妇群体进行合理分组，保证组内个体差异较小；动态优化则在各组内确定最优检测时点，以平衡检测的准确性与潜在风险。该模型能够为不同 BMI 人群提供差异化的检测建议，从而提升 NIPT 的整体有效性和可操作性。

5.3 针对问题三模型的建立与求解

5.3.1 数据预处理与特征选择

本题的目标是在不同 BMI 人群中确定最佳检测孕周，使过早检测与延迟达标所带来的期望成本之和尽量降低。为此，首先需要对原始数据进行整理和转化，使其适合纵向建模的需求。我们统一了字段含义，把 Y 染色体浓度从百分比转成零到一之间的比例，便于在概率空间里解释结果。随后进行合理范围筛选，例如孕周限定在医学合理区间，BMI 也限制在常见范围，从而降低极端值对模型的拉扯。对于同一孕妇在同一孕周的多次检测，我们采用均值聚合，让纵向序列更平滑，减少偶然波动的干扰。例如，若同一位孕妇在第十四周做了三次检测，数值略有差异，那么取平均能够更好地代表她在这一时点的水平。

在特征设定方面，将孕周和 BMI 作为核心自变量，并引入年龄和 IVF 情况作为控制变量，以防止混杂效应。若不加控制，模型可能会将某些差异错误地归因于 BMI。之所以强调“控制变量”，可以用一个简单例子说明：如果年长孕妇恰好 BMI 也偏高，而我们不控制年龄，那么模型可能会把年龄带来的差异错认成 BMI 的效果。因此，加入控制变量是保证因果解释合理性的必要步骤。

5.3.2 混合效应纵向模型

相比问题二，这里的关键难点不再是对某个时间点的直接回归，而是要描述“浓度随孕周如何变化”，同时让这种变化能因 BMI 而不同，并考虑到个体差异。因此，选取混合效应纵向模型比较合适，它一方面用固定效应刻画总体规律，另一方面用随机效应吸收个体之间的系统性差异。在建模时，把 Y 浓度放在 logit 尺度上拟合，再映回概率刻度，这既能缓解比例数据在零和一端的偏态，又能自然衔接题目所给的阈值百分之四。想象把一条原本在零到一之间弯曲的曲线“拉直”到 logit 空间做回归，再把结果平滑地“放回去”，模型更稳，也更容易与“达标概率”对接。为了刻画“浓度随孕周的动态变化”，采用混合效应模型。在 logit 尺度下拟合浓度轨迹：

$$\text{logit}(Y_{it}) = f(\text{week}_{it}) + g(\text{BMI}_i) + h(\text{week}_{it}, \text{BMI}_i) + \gamma \cdot \text{Age}_i + \delta \cdot \text{IVF}_i + u_i + \varepsilon_{it}$$

其中：

- $f(\text{week})$: 用 B 样条刻画孕周的非线性趋势；
- $g(\text{BMI})$: BMI 的非线性主效应
- $h(\text{week}_{it}, \text{BMI}_i)$: BMI 与孕周的交互项（允许曲线斜率和拐点随 BMI 变化）；
- u_i : 孕妇个体的随机截距；
- ε_{it} : 残差噪音。

这样得到的 $\mu_i(t) = \text{logit}^{-1}(\cdot)$ 即为第 i 位孕妇在孕周 t 的期望浓度。

在固定效应部分，使用 B 样条来刻画孕周的非线性趋势。B 样条可以理解为“把时

间轴分成若干段，在每一段用低阶多项式逼近，并让段与段之间顺滑衔接”。如早孕期浓度上升更慢，中期上升更快，到了后期又趋于平缓，单条直线很难覆盖这种弯曲，而样条可以自然地“拐弯”。BMI 的作用被拆成两块：一块是 BMI 的非线性主效应，用来表达不同 BMI 人群在整体水平上的差异；另一块是与时间样条的交互，用来让曲线的形状随着 BMI 改变。这个交互尤其重要，它允许高 BMI 人群的曲线“爬坡速度”变慢，这正是达标时间变晚的机制所在。年龄和 IVF 作为协变量进入模型，起到“校正地基”的作用。

- BMI 的作用 = 主效应 $g(BMI_i)$ + 交互 $h(week_{it}, BMI_i)$
- $g(BMI_i)$: 允许不同 BMI 平台的整体水平不同（竖直方向差异）；
- $h(week_{it}, BMI_i)$: 允许曲线形状随 BMI 改变（不仅是平移，连达标“增长速度/拐点”都能随 BMI 变）。因为第三问的核心就是“多因素影响达标时间”。如果只有主效应，不同 BMI 只是在同一时间曲线周围上下平移，达标时间差不明显；有了交互，曲线的斜率/拐点都会变，从而不同 BMI 组的最佳 t^* 真正分开。

在随机效应的设定中，每位孕妇均包含一个随机截距；在样本信息充分时，也允许引入随机斜率，以在保留个体差异的同时避免固定效应出现过度波动。实际建模过程中，为了减少样条与交互项带来的多重共线问题，增加了方差过滤和列筛选步骤。当样本信息不足以支撑随机斜率时，模型会自动退化为仅包含随机截距，从而保证估计过程的可识别性与收敛性。

为了直观展示纵向模型的效果，绘制了不同 BMI 组在孕周 9 - 16 周区间的预测浓度曲线（见图 5.3.1）。结果显示，随着 BMI 的增加，浓度上升速度逐渐放缓，跨越 4% 阈值的时点也相应推迟。这一图形化结果验证了模型设定的合理性：BMI 不仅影响浓度的整体水平，还通过与孕周的交互作用改变了浓度轨迹的形态。

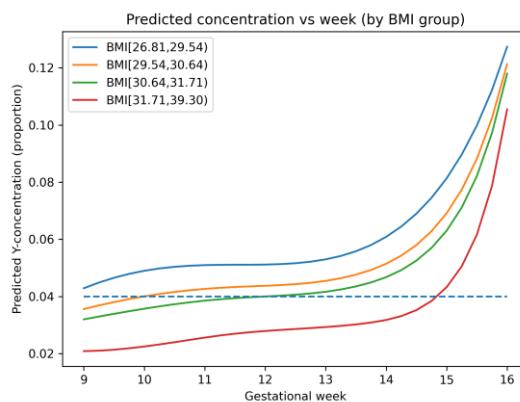


图 5.3.1 不同 BMI 组的 Y 染色体浓度预测曲线。虚线表示 4% 阈值。可以看出，高 BMI 人群的浓度曲线在更晚孕周才跨过阈值，说明她们的最佳检测时点较晚。

在模型拟合完成后，将 logit 预测结果映射回概率刻度，得到每位孕妇在任意孕周的期望浓度轨迹。为了将浓度水平与检测目标对应起来，需要把“数值大小”转化为“达标可能性”。为此，引入时点达标概率 $q(t)$ ，其含义是在孕周 t 时，浓度超过 4% 阈值的概率。

把浓度轨迹转为“达标概率”。定义在孕周 t 的达标概率：

$$q_i(t) = P(Y_i(t) \geq 0.04)$$

近似为：

$$q_i(t) = \Phi\left(\frac{\mu_i(t) - 0.04}{\sigma_y(t)}\right)$$

其中 $\Phi(\cdot)$ 为标准正态分布函数， $\sigma_y(t)$ 反映观测噪声或曲线不确定性。

进一步，定义首次达标时间 T_i ，其分布函数为：

$$F_i(t) = P(T_i \leq t)$$

直观地，若曲线在 t 时已经优较高概率超过阈值，则 $F_i(t)$ 也会较大。斜率越陡，时间上的不确定度越小， $F_i(t)$ 越集中。

用“曲线的斜率”估 $\sigma_y(t)$ （曲线越陡，时间越好辨、越“准”），构造一个“时间上的不确定度”，再得到在 t 之前已经达标过的概率 $F_i(t)$ 。这样最终要惩罚“太晚才达标”（错过窗口），需要一个时间维度上的概率分布，这一步把“生理量的轨迹”翻译成“事件概率的时间函数”，为成本函数服务。直觉例子可以帮助理解：假如某位孕妇在十四周的 $q(t)$ 是 90%，那么在这个时间点去检测，大概率已经达标；而十四周等于 70%，说明她在十四周之前有七成的可能性就已经达标过了。为了构造 $F(t)$ ，我们利用曲线的斜率估计时间不确定度，斜率越陡，说明“越过阈值”的时点更容易锁定；斜率越缓，时点就更“糊”，这会推高“过晚”的风险。

从临床应用的角度来看，关注的重点并非浓度曲线本身的形态，而是在特定孕周进行检测时能够有多大概率已经达到阈值。基于此，浓度预测结果被进一步转化为“到检即达标概率”曲线（见图 5.3.2）。该图展示了不同 BMI 组的达标概率随孕周的变化趋势，并在曲线上标注了各组对应的最优检测时点。结果表明，低 BMI 人群在较早孕周即可获得较高的达标概率，而高 BMI 人群则需要推迟检测才能保证检测的可靠性。

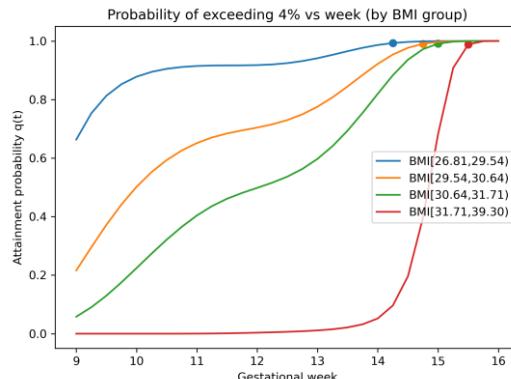


图 5.3.2 不同 BMI 组的达标概率曲线。横轴为孕周，纵轴为“到检即达标概率”。圆点为最优检测时点。结果表明，BMI 较高的人群达标曲线明显右移，意味着需要在更晚孕周安排检测。

在得到 $q(t)$ 与 $F(t)$ 之后，可以进一步定义综合成本。在某一 BMI 组内，如果选择孕周 t 作为检测时点，过早成本来源于“在该时点尚未达标”的概率，而过晚成本则来源于“超过业务窗口后才达标”的累积概率。两者加权相加即可得到总期望成本。不同的 BMI 组，其最优检测时点并不一致，因此需要在各组内分别进行优化。

实际求解时，在每个候选孕周上计算对应的综合成本，并选取使成本最小的 t 作为该组的最佳检测时点。为了避免凭经验随意划分 BMI 区间，首先利用分位点生成一系列候选边界，并对每个区间预先计算最小成本。随后，通过动态规划在给定组数的

条件下搜索最优切分方案，从而在全局意义上同时确定分组方式与检测时点。由此得到的结果不仅保证了组内与组间的合理性，也避免了局部随意性。在 BMI 组 g 内，综合成本函数定义为：

$$EC_g(t) = c_{early} \cdot [1 - q_g(t)] + c_{late} \cdot [F_g(t) - F_g(W_{late})]$$

其中：

“过早成本”由“此时仍未达标”的概率 $1 - q_g(t)$ 表示；

“过晚成本”由“超过业务窗口 W_{late} 后才达标”的比例 $F_g(t) - F_g(W_{late})$ 表示。

组内最优检测时点为：

$$t_g^* = \operatorname{argmin}_t EC_g(t)$$

动态规划选分组：先用 BMI 分位数生成一堆候选边界，预算算所有“区间的最小成本”，

$$C(i, j) = \min_t EC_{(i, j]}(t)$$

设目标组数为 K ，则动态规划递推为：

$$DP[j, k] = \min_{0 \leq i < j} \{DP[i, k - 1] + C(i + 1, j)\}, \quad DP[0, 0] = 0$$

最终最优解为 $DP[M, K]$ ，回溯得到分组边界与各组的 t_g^* 。

上述公式给出了期望成本的定义与最优检测时点的求解方式，但仅通过符号推导仍然欠缺直观性。为此，绘制了不同 BMI 组的期望成本曲线，并在曲线上标注了对应的最优检测时点（见图 5.3.3）。结果显示，每条曲线均呈现“先下降后上升”的趋势，其最低点即为该组的最佳检测周。低 BMI 组的最低点相对靠前，而高 BMI 组的最低点则明显右移，清晰地印证了“BMI 越高，最佳检测周越晚”的结论。

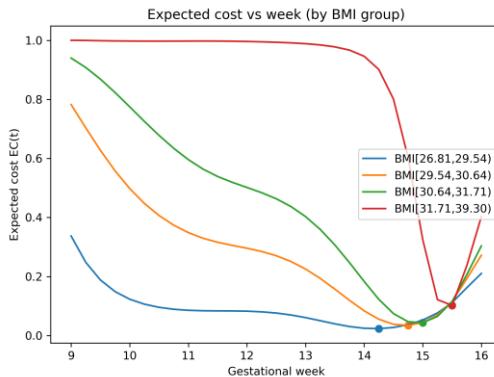


图 5.3.3 不同 BMI 组的期望成本曲线。横轴为孕周，纵轴为期望成本。曲线最低点即为该组的最优检测孕周 t_g^* 。可以看出，BMI 越高的组，最佳检测时点越晚。

相较于问题二，这里最大的不同在于问题的表达对象从“一个时间点的值”变成了“随时间演化的一条曲线”，而且这条曲线要允许 BMI 等因素“揉捏”出不同的形状。问题二更像是在问“谁先到达终点”，问题三则是在问“整段跑道上的速度曲线是什么样，在哪个位置踩线最合适”。为此，采用 logit 尺度的纵向建模，使浓度轨

迹在概率空间内具有可解释性，并通过 BMI 与孕周的交互项，揭示高 BMI 人群出现达标延迟的机制。在决策层面，纵向预测结果进一步转化为达标概率和达标时间分布，并结合成本函数与动态规划确定各组的最优检测时点。最终得到的不是单一的数值，而是一套可执行的分组化检测方案。

变量	混合效应纵向模型
$bs(week)[0]$	0.748 (1.270)
$bs(week)[1]$	-0.291 (0.899)
$bs(week)[2]$	-0.477 (1.592)
$bs(week)[3]$	1.118 (1.987)
$bs(week)[4]$	-0.858 (3.399)
$bs(BMI)[0]$	0.687 (0.402)*
$bs(BMI)[1]$	0.980 (0.344)***
$bs(BMI)[2]$	0.932 (0.532)*
$bs(week)[0] \times BMI$	-0.102 (0.011)***
$bs(week)[1] \times BMI$	-0.021 (0.040)
$bs(week)[2] \times BMI$	0.017 (0.029)
$bs(week)[3] \times BMI$	0.022 (0.050)
$bs(week)[4] \times BMI$	-0.014 (0.061)
年龄	0.072 (0.103)
IVF	-0.008 (0.008)
截距	-0.457 (0.312)

表 5.3.1 各模型的变量系数，标准差及显著性检验表（标准差位于变量系数下方，***表示 $p - value < 0.01$ ，**表示 $p - value < 0.05$ ，*表示 $p - value < 0.1$ ）

$bs(week)[0..4]$: 孕周的 5 个 B 样条基函数，不是分组，而是数学上的分段基底，用来拟合孕周随时间的非线性轨迹。

$bs(BMI)[0..2]$: BMI 的 3 个 B 样条基函数，用来捕捉不同 BMI 水平下的非线性差异。

交互项 $bs(week)[i] \times BMI$: 表示在孕周第 i 段，曲线形状是否会因 BMI 而不同。

BMI 组别	推荐检测周	早到概率	晚发现概率	期望成本
[26.81, 29.54)	15.00	0.021	0.065	0.0555
[29.54, 30.64)	15.50	0.032	0.123	0.0845
[30.64, 31.71)	15.50	0.011	0.115	0.0494
[31.71, 39.30)	15.75	0.062	0.222	0.1173

表 5.3.2 各个 BMI 组的最佳检测周及早到、晚发现概率表

5.3.3 对于问题三的结论

结果显示，孕周与 BMI 的交互作用在很大程度上决定了最佳检测时点。随着 BMI 升高，Y 染色体浓度的上升速度明显减缓，因此跨越 4% 阈值的时间被推迟，最佳检测孕周也相应后移。在回归结果中，BMI 的主效应系数为正，表明高 BMI 人群在早期可能略高；但其与孕周的交互项显著为负，说明浓度上升速度整体放缓，这是导致检测时点延后的关键机制。模型给出的各 BMI 组最优检测孕周清晰体现了这一趋势：低 BMI 人群的推荐检测时点约为 15 周，而高 BMI 人群则推迟至 15.75 周，以降低过早或过晚检测的风险。

进一步来看，年龄和 IVF 在模型中的作用并不显著，说明最佳时点的差异主要由 BMI 驱动。随机效应结果也提供了佐证：个体间的基线水平存在差异，但孕周斜率的个体差异几乎可以忽略。这表明“BMI × 孕周”的固定效应才是影响检测时点的核心因素。总体而言，模型结论与医学直觉一致，即 BMI 越高，检测时间应适度推迟。

5.4 针对问题四模型的建立与求解

5.4.1 问题背景与目标

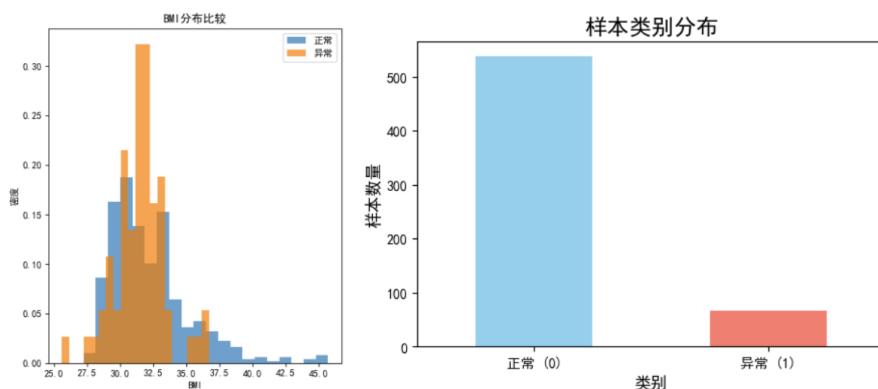


图 5.4.1 正常与异常女胎的孕妇 BMI 分布比较

研究针对女胎缺乏 Y 染色体标志物的特殊情况，建立基于多维生物标志物的异常判定框架。以 21 号、18 号和 13 号染色体非整倍体检测结果作为金标准真值，系统分析 13 个候选指标与异常判定结果的关联性。候选指标包括基因组特征（GC 含量）、染色体 Z 值（Z13、Z18、Z21、ZX）、测序质量参数（原始读段数、比对率、重复率）以及孕妇生理特征（检测孕周、体质量指数、年龄、身高、体重）。

5.4.2 数据预处理和相关性分析

数据预处理过程采用完整案例分析策略，仅保留所有 13 个指标均无缺失值的观测样本，保证后期相关性的检验可以实现同质比较。研究将体质量指数（BMI）划分为 3

个区间进行分层分析，区间跨度为 5 个体质量指数单位，从 25 开始至 40 的区间，这样可以较好的反映出在不同体重下相应的指标的变化。

考虑到医学数据可能不服从正态分布假设，研究采用 Spearman 等级相关系数作为主要关联性测度。对于二分类异常判定变量与连续型指标变量的关联分析，Spearman 相关系数的计算公式为：

$$r_s = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)}$$

其中 d_i 表示第 i 个观测在两变量等级序列中的等级差值， n 为样本量。同时计算 Pearson 相关系数作为对比，但优先采用 Spearman 系数的结果，因其对离群值和非线性关系具有更强的稳健性。

在两变量等级序列中的等级差值， n 为样本量。同时计算 Pearson 相关系数作为对比，但优先采用 Spearman 系数的结果，因其对离群值和非线性关系具有更强的稳健性。

变量	BMI 组 [25, 30]	BMI 组 [30, 35]	BMI 组 [35, 40]
年龄	0.05085 (0.55067)	-0.07831 (0.13008)	-0.02391 (0.83650)
身高	0.08006 (0.34704)	-0.09440 (0.06784)	-0.00531 (0.96342)
体重	-0.01019 (0.90485)	-0.10834 (0.03597)	-0.03430 (0.76712)
检测孕周	0.22541 (0.00741)	0.02007 (0.69847)	-0.09749 (0.39897)
孕妇 BMI	-0.15045 (0.07600)	-0.04487 (0.38623)	-0.07640 (0.50896)
原始读段数	-0.07521 (0.37713)	-0.02544 (0.62328)	0.21063 (0.06594)
在参考基因组上比对的比例	-0.05550 (0.53493)	0.02466 (0.63403)	-0.06319 (0.58508)
重复读段的比例	0.05287 (0.53493)	-0.03613 (0.48536)	-0.19747 (0.08516)
唯一比对的读段数	-0.08046 (0.34461)	-0.01632 (0.75272)	0.19747 (0.08516)
GC 含量	0.04237 (0.61916)	-0.02994 (0.56332)	0.10268 (0.37417)
13 号染色体的 Z 值	-0.04762 (0.57633)	-0.03514 (0.49750)	-0.13691 (0.23508)
18 号染色体的 Z 值	0.02726 (0.74918)	0.04119 (0.42632)	-0.11585 (0.31569)
21 号染色体的 Z 值	0.09491 (0.26462)	0.05502 (0.28785)	-0.08952 (0.43878)
X 染色体的 Z 值	-0.06732 (0.42928)	-0.06856 (0.18517)	0.03949 (0.73308)
X 染色体浓度	-0.29001	-0.28426	-0.37914

	(0.00051)	(2.115e-08)	(0.00067)
13 号染色体的 GC 含量	-0.12119 (0.15376)	0.08097 (0.11749)	0.20800 (0.06947)
18 号染色体的 GC 含量	-0.12645 (0.13655)	0.07591 (0.14231)	0.19747 (0.08516)
21 号染色体的 GC 含量	-0.17046 (0.04405)	0.02352 (0.64980)	0.20537 (0.07316)
被过滤掉读段数的比例	0.05419 (0.52482)	-0.05752 (0.26652)	0.12374 (0.28360)

表 5.4.1 不同 BMI 组别下各变量的相关性和统计显著性表

5.4.3 多维度指标评价体系的建立与分析

研究建立多维度指标评价框架，从相关性强度、统计显著性，高相关分组数三个角度评估各指标的重要性。设定相关性阈值为 0.15，显著性阈值为 0.05，基于这些标准计算三个关键指标：

平均相关性绝对值反映指标与异常判定结果的总体关联强度：

$$|\bar{r}|_i = \frac{1}{m} \sum_{j=1}^m |r_{ij}|$$

其中 r_{ij} 为第 i 个指标在第 j 个有效分组中的 Spearman 相关系数， m 为有效分组数。

显著相关分组数量化统计显著性：

$$N_{sig,i} = \sum_{j=1}^m I(p_{ij} < 0.05)$$

其中 $I(\cdot)$ 为指示函数， p_{ij} 为相应的 p 值。

高相关分组数评估一致性表现：

$$N_{high,i} = \sum_{j=1}^m I(|r_{ij}| \geq 0.15)$$

综合得分整合三个维度的信息：

$$S_i = |\bar{r}|_i + 0.3 \times \frac{N_{sig,i}}{m} + 0.2 \times \frac{N_{high,i}}{m}$$

其中权重系数体现相关性强度的主导地位，同时兼顾显著性和一致性的贡献。最终得出前九名依次为孕妇 BMI，体重，检测孕周，18 号染色体的 Z 值，唯一比对的读段，21 号染色体的 Z 值，GC 含量，13 号染色体的 Z 值，重复读段的比例。

5.4.4 基于机器学习下的评价模型准确率分析

基于前期相关性分析筛选出的前九名关键指标，建立了女胎异常判定的机器学习分类模型。这九个核心特征分别为孕妇 BMI，体重，检测孕周，18 号染色体的 Z 值，唯一比对的读段数，21 号染色体的 Z 值，GC 含量，13 号染色体的 Z 值，重复读段的比

例。它们在不同体质量指数组中均表现出与异常判定结果的强相关性和良好一致性。以 21 号、18 号和 13 号染色体非整倍体检测结果作为监督学习的标准答案，构建五维特征空间到二分类结果的映射关系。

研究系统比较了六种主流机器学习算法的性能表现。

逻辑回归模型基于广义线性模型框架，假设对数几率与特征呈线性关系：

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{i=1}^5 \beta_i x_i$$

其中 p 为异常概率， x_i 为第 i 个标准化特征。

支持向量机利用了径向基核函数，能够处理非线性分类边界：

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

通过核技巧将原始特征空间映射到高维空间，寻找最优分离超平面。

决策树算法依据信息增益或基尼不纯度进行特征选择和分割点确定，从而得到可解释的分类规则。随机森林作为集成方法，通过自助抽样和随机特征的选择方式构建多个决策树，最终预测结果通过投票机制确定：

$$\hat{y} = mode\{h_1(x), h_2(x), \dots, h_T(x)\}$$

其中 $h_t(x)$ 为第 t 棵决策树的预测结果， T 为树的总数。

朴素贝叶斯模型基于条件独立假设，通过贝叶斯定理计算后验概率：

$$P(y=1|x) = \frac{P(y=1) \prod_{i=1}^5 P(x_i|y=1)}{P(x)}$$

AdaBoost 集成学习将弱分类器经过迭代训练并调整样本权重，并通过调整样例的权重来进行最后的预测，最终预测结果为加权投票：

$$H(x) = sign\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$$

其中 α_t 为第 t 个弱分类器的权重。

建立全面的模型评估指标体系，包括准确率、精确率、召回率、F1 分数：

模型	准确率	精确率	召回率	F1 分数
逻辑回归	0.5604	0.1154	0.45	0.1837
支持向量机	0.5495	0.1026	0.4	0.1633
决策树	0.7143	0.1364	0.3	0.1875
随机森林	0.8901	0	0	0
朴素贝叶斯	0.8846	0	0	0
集成学习	0.8901	0	0	0

表 5.4.2 各机器学习模型下的评估数据表

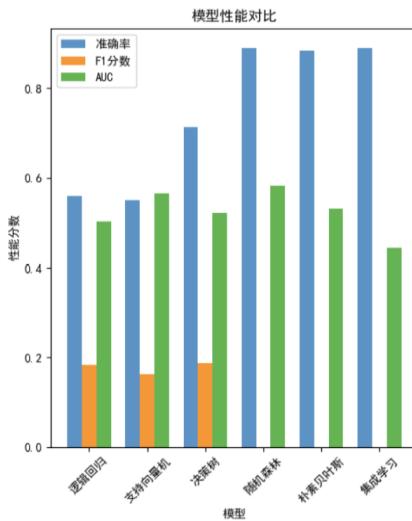


图 5.4.1 各机器学习模型性能对比图

较高准确率的模型（如随机森林、集成学习、朴素贝叶斯）在预测正类（1类）时几乎没有效果，这可能是数据中负类（0类）占据主导地位，导致模型偏向于预测负类，从而提高了准确率，但导致精确率、召回率和 F1 分数为 0。F1 分数较低的模型表明，模型没有成功平衡精确度和召回率。决策树的结果表现介于中间水平，但也没有很好地平衡精确率和召回率。

六、模型的评价与改进

6.1 模型的优点

通过建立胎儿 Y 染色体浓度与孕周、BMI 等关键变量之间的回归关系，模型能够系统刻画多因素对检测结果的影响规律，结论直观且具备较强的解释力，为临床提供了可量化的参考。

基于 BMI、身高、体重等多元指标实施分组或者聚类的方法可用于将人群进行分组，用于异质性较强的人群，选择适合该组别或者群体的时间点进行监测可以避免一部分不必要的临床风险。

引入检测误差到模型建立中可消除数据因波动或含有异常值带来的结论偏倚，使模型具有较好的实用性与鲁棒性。

基于非整倍体检出胎儿异常时，考虑 21、18、13 号染色体非整倍体，X 染色体 Z 值，GC 含量及测序读段数多种信息联合使用可提高异常筛查的灵敏度和特异性。

6.2 模型的缺点

各个指标（孕妇 BMI、孕周、年龄）之间可能存在复杂的交互作用，现有模型还不能较好地刻画其非线性和高维交互间的关系。

尽管引入了误差建模，但若数据中存在极端值，仍可能导致模型估计结果的偏离，影响判定的准确性。

在女胎异常判断时，多维指标阈值难以设定统一的或者科学合理标准，难免会造成误诊或者漏诊的风险。

6.3 模型的推广价值

Y 染色体浓度与孕周、BMI 等因素的关系模型不仅可以推广至其他浓度阈值判定场景，如药物血药浓度的个体化监测、肿瘤标志物的动态跟踪，其模型还具有拓展到公共卫生与人群管理中。使用聚类与分组的建模方法进行人群分层管理和慢性病随访研究，为个体化健康干预提供参考。同时，误差敏感性建模的思路具有跨领域价值，可迁移至环境监测、气象预报、金融风险评估等需处理不确定性和噪声的复杂系统。这样多指标综合判定的多维诊断框架可用于疾病的多因子危险因素风险的评估和诊断，也可以应用于心血管疾病的预测和遗传病的多指标筛查当中，对临床多维度诊断进行建模有一定借鉴意义。

七、参考文献

[1] 段红蕾, 王皖骏, 张颖, 等. 基于孕妇外周血胎儿游离 DNA 的无创产前筛查临床应用实践 [J]. 中华医学遗传学杂志, 2022, 39(3): 281–284.

[2] 赖允丽, 陈筠, 易升, 等. 无创产前检测的临床应用及随访研究 [J]. 重庆医学, 2016, 45(11): 1476–1479.

本参赛队未使用任何 AI 工具

八、附录

```
1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. %matplotlib inline
5. male_data = pd.read_excel(r'男胎.xlsx')
6. male_data.isnull().sum()
7. male_data['Y 染色体浓度'] = pd.to_numeric(male_data['Y 染色体浓度'],
     ], errors='coerce')
8. male_data['孕妇 BMI'] = pd.to_numeric(male_data['孕妇 BMI'], errors='coerce')
9.
10. # 使用 IQR (四分位距) 方法来检测和处理异常值
11. Q1 = male_data['Y 染色体浓度'].quantile(0.25)
12. Q3 = male_data['Y 染色体浓度'].quantile(0.75)
13. IQR = Q3 - Q1
14. lower_bound = Q1 - 1.5 * IQR
15. upper_bound = Q3 + 1.5 * IQR
16.
17. # 过滤异常值
18. male_data = male_data[(male_data['Y 染色体浓度']
   ] >= lower_bound) & (male_data['Y 染色体浓度'] <= upper_bound)]
19.
20. # 使用 IQR (四分位距) 方法来检测和处理异常值
21. Q1 = male_data['孕妇 BMI'].quantile(0.25)
22. Q3 = male_data['孕妇 BMI'].quantile(0.75)
23. IQR = Q3 - Q1
24. lower_bound = Q1 - 1.5 * IQR
25. upper_bound = Q3 + 1.5 * IQR
26.
27. # 过滤异常值
28. male_data = male_data[(male_data['孕妇 BMI'] >= lower_bound) & (male_data['孕
   妇 BMI'] <= upper_bound)]
29.
30. import re
31.
32. def extract_week(x):
33.
34.     if '+' in x:
35.
36.         week_number = int(re.findall(r'\d+', x)[0]) # 提取 'w' 前面的数字
37.         extra_days = int(re.findall(r'\d+', x.split('+')[1])[0]) # 提
   取 '+' 后面的数字
38.         return week_number + extra_days / 7
```

```

39.     else:
40.
41.         return int(re.findall(r'\d+', x)[0])
42.
43.
44. male_data['检测孕周'] = male_data['检测孕周'].apply(extract_week)
45.
46.
47. female_data = pd.read_excel(r'女胎.xlsx')
48.
49. female_data[female_data.duplicated()]
50.
51. female_data.isnull().sum()
52.
53. female_data['孕妇 BMI'] = pd.to_numeric(female_data['孕妇
   BMI'], errors='coerce')
54.
55. mean_value = female_data['孕妇 BMI'].mean() # 找到最常见的日期
56. female_data['孕妇 BMI'] = female_data['孕妇 BMI'].fillna(mean_value)
57. print(female_data['孕妇 BMI'].isnull().sum())
58.
59. female_data['孕妇 BMI'] = pd.to_numeric(female_data['孕妇
   BMI'], errors='coerce')
60. Q1 = female_data['孕妇 BMI'].quantile(0.25)
61. Q3 = female_data['孕妇 BMI'].quantile(0.75)
62. IQR = Q3 - Q1
63. lower_bound = Q1 - 1.5 * IQR
64. upper_bound = Q3 + 1.5 * IQR
65.
66. # 过滤异常值
67. female_data = female_data[(female_data['孕妇
   BMI'] >= lower_bound) & (female_data['孕妇 BMI'] <= upper_bound)]
68.
69. import re
70.
71. def extract_week(x):
72.
73.     if '+' in x:
74.
75.         week_number = int(re.findall(r'\d+', x)[0]) # 提取 'w' 前面的数字
76.         extra_days = int(re.findall(r'\d+', x.split('+')[1])[0]) # 提
           取 '+' 后面的数字
77.         return week_number + extra_days / 7
78.     else:

```

```
79.  
80.         return int(re.findall(r'\d+', x)[0])  
81.  
82.  
83. female_data['检测孕周'] = female_data['检测孕周'].apply(extract_week)
```

```
1. import pandas as pd  
2. import numpy as np  
3. import matplotlib.pyplot as plt  
4. %matplotlib inline  
5.  
6. male_data = pd.read_excel(r'男胎.xlsx')  
7. female_data = pd.read_excel(r'女胎.xlsx')  
8. combined_data = pd.concat([male_data, female_data], ignore_index=True)  
9.  
10. import matplotlib.pyplot as plt  
11.  
12. # 男胎和女胎的数量  
13. male_count = 1082  
14. female_count = 605  
15.  
16. labels = ['男胎', '女胎']  
17. sizes = [male_count, female_count]  
18. plt.rcParams['font.sans-serif'] = ['SimHei']  
19. plt.rcParams['axes.unicode_minus'] = False  
20. # 绘制饼状图  
21. plt.figure(figsize=(8, 8))  
22. plt.pie(sizes, labels=labels, autopct='%.1f%%', startangle=90, colors=[ 'lightblue', 'lightcoral'])  
23.  
24. # 设置标题  
25. plt.title('男女胎儿的性别分布', fontsize=16)  
26.  
27. # 显示图像  
28. plt.show()
```

```
1. import pandas as pd  
2. import numpy as np  
3. import matplotlib.pyplot as plt  
4. %matplotlib inline  
5.
```

```
6. cleaned_data = pd.read_excel(r'male_cleaned_data.xlsx')
7.
8. import matplotlib.pyplot as plt
9. x = cleaned_data[['检测孕周']]
10. y = cleaned_data['Y染色体浓度']
11. y_chromosome_concentration 和 detection_week
12. plt.figure(figsize=(8,6))
13. plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置字体为黑体
14. plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号
15. plt.scatter(x, y, color='blue', alpha=0.6)
16.
17. plt.title('Y染色体浓度与检测孕周的散点图')
18. plt.xlabel('检测孕周')
19. plt.ylabel('Y染色体浓度')
20.
21. plt.show()
22.
23. import matplotlib.pyplot as plt
24. x = cleaned_data[['孕妇 BMI']]
25. y = cleaned_data['Y染色体浓度']
26. y_chromosome_concentration 和 detection_week
27. plt.figure(figsize=(8,6))
28.
29. plt.scatter(x, y, color='blue', alpha=0.6)
30.
31. plt.title('Y染色体浓度与孕妇 BMI 的散点图')
32. plt.xlabel('孕妇 BMI')
33. plt.ylabel('Y染色体浓度')
34.
35. plt.show()
36.
37.
38. combined_data = pd.read_excel(r'combined_data.xlsx')
39. import seaborn as sns
40. plt.rcParams['font.sans-serif'] = ['SimHei']
41. plt.rcParams['axes.unicode_minus'] = False
42.
43. plt.figure(figsize=(10, 6))
44. sns.histplot(combined_data['检测孕周'],
   bins=15, kde=False, color='skyblue', stat='count')
45.
46.
47. plt.title('孕周分布图', fontsize=16)
48. plt.xlabel('检测孕周', fontsize=12)
```

```
49. plt.ylabel('人数', fontsize=12)
50.
51. plt.show()
52.
53. import seaborn as sns
54. pearson_corr = cleaned_data[['Y 染色体浓度', '检测孕周', '孕妇
   BMI']].corr(method='pearson')
55. print("Pearson correlation:\n", pearson_corr)
56.
57. spearman_corr = cleaned_data[['Y 染色体浓度', '检测孕周', '孕妇
   BMI']].corr(method='spearman')
58. print("Spearman correlation:\n", spearman_corr)
59.
60. plt.rcParams['font.sans-serif'] = ['SimHei']
61. plt.rcParams['axes.unicode_minus'] = False
62.
63. sns.heatmap(pearson_corr, annot=True, cmap='coolwarm')
64. plt.title("Pearson Correlation Matrix")
65. plt.show()
66.
67. sns.heatmap(spearman_corr, annot=True, cmap='coolwarm')
68. plt.title("Spearman Correlation Matrix")
69. plt.show()
70.
71. import statsmodels.api as sm
72. data_clean = cleaned_data.dropna(subset=['Y 染色体浓度', '检测孕周', '孕妇
   BMI'])
73.
74. X = data_clean[['检测孕周', '孕妇 BMI']] # 自变量
75. X = sm.add_constant(X) # 加入常数项（截距）
76. y = data_clean['Y 染色体浓度'] # 因变量
77.
78. model = sm.OLS(y, X).fit()
79.
80. print(model.summary())
81.
82. y_pred = model.predict(X)
83.
84. # 计算均方误差 (MSE)
85. mse = mean_squared_error(y, y_pred)
86. print("OLS 回归的 MSE:", mse)
87.
88. import numpy as np
89. import pandas as pd
```

```

90.
91. from sklearn.ensemble import RandomForestRegressor
92. from sklearn.model_selection import GroupKFold, train_test_split, RandomizedSearchCV
93. from sklearn.preprocessing import PolynomialFeatures
94. from sklearn.compose import ColumnTransformer
95. from sklearn.pipeline import Pipeline
96. from sklearn.metrics import r2_score, mean_squared_error
97. from scipy.special import expit, logit
98.
99. df = cleaned_data.copy().dropna(subset=['检测孕周', '孕妇 BMI', 'Y 染色体浓度', '孕妇代码']).reset_index(drop=True)
100.
101. X = df[['检测孕周', '孕妇 BMI']]
102. y = df['Y 染色体浓度'].values
103. groups = df['孕妇代码'].astype(str).values # 分组用孕妇 ID
104.
105. eps = 1e-6
106. y_clip = np.clip(y, eps, 1 - eps)
107. y_logit = logit(y_clip) # y' = log(y/(1-y))
108.
109.
110. unique_ids = np.unique(groups)
111. rng = np.random.RandomState(42)
112. test_ids = rng.choice(unique_ids, size=int(0.2*len(unique_ids)), replace=False)
113. is_test = np.isin(groups, test_ids)
114. X_train, X_test = X[~is_test], X[is_test]
115. y_train_logit, y_test_logit = y_logit[~is_test], y_logit[is_test]
116. y_test_true = y[is_test]
117. groups_train = groups[~is_test]
118.
119. poly = PolynomialFeatures(degree=2, include_bias=False) # 生成 GW, BMI, GW^2, BMI^2, GW*BMI
120. preprocess = ColumnTransformer([
121.     ('poly', poly, ['检测孕周', '孕妇 BMI']),
122. ], remainder='drop')
123.
124. rf = RandomForestRegressor(
125.     n_estimators=800,
126.     oob_score=True,
127.     random_state=42,
128.     n_jobs=-1
129. )

```

```

130.
131. pipe = Pipeline([
132.     ('prep', preprocess),
133.     ('rf', rf)
134. ])
135.
136. param_dist = {
137.     'rf__n_estimators': [600, 800, 1000, 1200],
138.     'rf__max_depth': [None, 6, 8, 10, 14, 18],
139.     'rf__min_samples_leaf': [5, 10, 15, 20, 30, 50],
140.     'rf__max_features': ['auto', 'sqrt', 0.5, 0.7],
141.     'rf__bootstrap': [True] # OOB 只在 bootstrap=True 时生效
142. }
143.
144. gkf = GroupKFold(n_splits=5)
145. search = RandomizedSearchCV(
146.     pipe,
147.     param_distributions=param_dist,
148.     n_iter=40,
149.     scoring='r2',
150.     cv=gkf.split(X_train, y_train_logit, groups_train),
151.     refit=True,
152.     random_state=42,
153.     n_jobs=-1,
154.     verbose=1
155. )
156.
157. search.fit(X_train, y_train_logit)
158.
159. best_model = search.best_estimator_
160. print("Best params:", search.best_params_)
161. print("CV best R2 (on logit target):", search.best_score_)
162.
163. y_pred_logit_test = best_model.predict(X_test)
164. y_pred_test = expit(y_pred_logit_test) # 反变换到 (0,1)
165.
166. r2_test = r2_score(y_test_true, y_pred_test)
167. mse_test = mean_squared_error(y_test_true, y_pred_test)
168. print(f"Test R^2: {r2_test:.4f}")
169. print(f"Test MSE: {mse_test:.6f}")
170.
171. y_pred_logit_tr = best_model.predict(X_train)
172. y_pred_tr = expit(y_pred_logit_tr)
173. y_tr_true = y[~is_test]

```

```

174.
175. r2_tr = r2_score(y_tr_true, y_pred_tr)
176. mse_tr = mean_squared_error(y_tr_true, y_pred_tr)
177. print(f"Train R2: {r2_tr:.4f}")
178. print(f"Train MSE: {mse_tr:.6f}")
179.
180. y_pred_prob = expit(y_pred_logit_test) # expit 是 sigmoid 函数
181.
182. # 计算 Log-Likelihood
183. log_likelihood = np.sum(y_test_true * np.log(y_pred_prob + 1e-10) + (1 - y_test_true) * np.log(1 - y_pred_prob + 1e-10))
184.
185. print(f"Log-Likelihood: {log_likelihood:.4f}")
186.
187. residuals = y_test - y_pred
188. n = len(y_test) # 样本数量
189. log_likelihood = -n / 2 * np.log(2 * np.pi * mse) - (1 / (2 * mse)) * np.sum(residuals ** 2)
190.
191. print("Log-Likelihood: ", log_likelihood)
192.
193. import pandas as pd
194. import statsmodels.api as sm
195. import statsmodels.formula.api as smf
196.
197. male_data['woman_id_time'] = male_data['孕妇代码'].astype(str) + " " + male_data['检测抽血次数'].astype(str)
198. male_data['孕妇 BMI_square'] = male_data['孕妇 BMI'] ** 2
199. male_data['BMI_检测孕周_interaction'] = male_data['孕妇 BMI'] * male_data['年龄']
200. male_data['检测孕周_square'] = male_data['检测孕周'] ** 2
201. male_data['检测孕周_cubic'] = male_data['检测孕周'] ** 3
202.
203. X = male_data[['检测孕周', '孕妇 BMI', '检测孕周_square', '检测孕周_cubic', '孕妇 BMI_square', 'BMI_检测孕周_interaction', 'woman_id_time']]
204. y = male_data['Y 染色体浓度']
205.
206. X = sm.add_constant(X)
207.
208. model = smf.mixedlm("Y 染色体浓度 ~ 检测孕周 + 孕妇 BMI + 检测孕周_square + 检测孕周_cubic + 孕妇 BMI_square + BMI_检测孕周_interaction", male_data, groups=male_data["woman_id_time"]).fit()
209.
210. print(model.summary())

```

```
211.  
212. from sklearn.metrics import mean_squared_error  
213. y_pred = model.fittedvalues  
214.  
215. y_actual = male_data['Y 染色体浓度']  
216.  
217. mse = mean_squared_error(y_actual, y_pred)  
218.  
219. print("Mixed Effects Model MSE:", mse)
```

```
1. from sklearn.cluster import KMeans  
2. import numpy as np  
3.  
4. X = cleaned_data[['孕妇 BMI', '检测孕周']].dropna() # 去除缺失值  
5.  
6. kmeans = KMeans(n_clusters=3, random_state=42)  
7. cleaned_data['cluster'] = kmeans.fit_predict(X)  
8.  
9. print(kmeans.cluster_centers_)  
10.  
11. import matplotlib.pyplot as plt  
12. import seaborn as sns  
13. from sklearn.cluster import KMeans  
14. import numpy as np  
15.  
16. X = cleaned_data[['孕妇 BMI', '检测孕周']].dropna() # 去除缺失值  
17.  
18. # 执行 KMeans 聚类分析  
19. kmeans = KMeans(n_clusters=3, random_state=42)  
20. cleaned_data['cluster'] = kmeans.fit_predict(X)  
21.  
22. # 获取聚类中心  
23. cluster_centers = kmeans.cluster_centers_  
24.  
25. # 可视化聚类结果  
26. plt.figure(figsize=(10, 6))  
27.  
28. sns.scatterplot(x='孕妇 BMI', y='检测孕周  
', hue='cluster', data=cleaned_data, palette='viridis', s=100, alpha=0.7)  
29.  
30. plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], s=200, c='red', mar  
ker='X', label='Cluster Centers')  
31.
```

```

32. plt.title('KMeans Clustering: BMI and Best NIPT Detection Week', fontsize=16)

33. plt.xlabel('BMI', fontsize=12)
34. plt.ylabel('Best NIPT Detection Week', fontsize=12)
35. plt.legend()
36.
37. plt.grid(True)
38. plt.show()
39.

40. for group in cleaned_data['BMI_group'].unique():
41.     group_data = cleaned_data[cleaned_data['BMI_group'] == group]
42.
43.     X = group_data[['孕妇 BMI']]
44.     y = group_data['检测孕周']
45.
46.     X = sm.add_constant(X)
47.
48.     model = sm.OLS(y, X).fit()
49.
50.     y_pred = model.predict(X)
51.     residuals = y - y_pred
52.
53.     mse = np.mean(residuals**2)
54.     print(f"Risk for BMI group {group}: MSE = {mse}")

```

```

1. from __future__ import annotations
2. import itertools
3. import math
4. from dataclasses import dataclass
5. from typing import Dict, List, Tuple, Optional
6.
7. import numpy as np
8. import pandas as pd
9. from sklearn.isotonic import IsotonicRegression
10.
11.
12. # CONFIG 区
13. CONFIG = {
14.     "file_path": "/Users/shiyidianqianyaoshuijiao/Desktop/数模国赛
15.                 /cleaned_data.xlsx",
16.     "id_col": "孕妇代码",      # 受试者 ID
17.     "week_col": "检测孕周",    # 孕周
18.     "y_col": "Y 染色体浓度",   # Y 浓度
19.     "bmi_col": "孕妇 BMI",    # BMI 列
20. }

```

```

19.     "is_excel": True,
20.     "excel_sheet": 0,
21. }
22.
23. # 数据清洗
24. def load_and_prepare_data(cfg: Dict) -> pd.DataFrame:
25.     if cfg.get("is_excel", True):
26.         df = pd.read_excel(cfg["file_path"], sheet_name=cfg.get("excel_sheet"
27. , 0))
28.     else:
29.         df = pd.read_csv(cfg["file_path"])
30.
31.     cols_map = {
32.         cfg["id_col"]: "id",
33.         cfg["week_col"]: "week",
34.         cfg["y_col"]: "y",
35.         cfg["bmi_col"]: "bmi",
36.     }
37.     try:
38.         df = df[list(cols_map.keys())].rename(columns=cols_map)
39.     except KeyError as e:
40.         missing = [k for k in cols_map.keys() if k not in df.columns]
41.         raise KeyError(f"数据中缺少这些列, 请在 CONFIG 里修正列名:
42. {missing}") from e
43.
44.     # 基本清洗
45.     df = df.dropna(subset=["id", "week", "y", "bmi"]).copy()
46.
47.     # 数值化 & 合理范围
48.     df["week"] = pd.to_numeric(df["week"], errors="coerce")
49.     df["y"] = pd.to_numeric(df["y"], errors="coerce")
50.     df["bmi"] = pd.to_numeric(df["bmi"], errors="coerce")
51.     df = df.dropna(subset=["week", "y", "bmi"]).copy()
52.
53.     # Y%
54.     if df["y"].median() > 1:
55.         df["y"] = df["y"] / 100.0
56.
57.     # 合理范围
58.     df = df[(df["week"] >= 0) & (df["week"] <= 45)].copy()
59.     df = df[(df["y"] >= 0) & (df["y"] <= 1)].copy()
60.     df = df[(df["bmi"] >= 10) & (df["bmi"] <= 60)].copy()
61.
62.     # 同一人同一孕周若有重复, 取均值 (Y 取均值, BMI 取中位)

```

```

61.     df = (df.groupby(["id", "week"], as_index=False)
62.             .agg({"y": "mean", "bmi": "median"}))
63.
64.     return df
65.
66. # 单人序列: 单调平滑 + 首次达标 T
67. def estimate_T_for_one(person_df: pd.DataFrame,
68.                         threshold: float = 0.04,
69.                         delta: float = 0.003,
70.                         enforce_monotonic: bool = True,
71.                         return_curve: bool = False) -> Tuple[Optional[float],
72.                                         Optional[np.ndarray], Optional[np.ndarray]]:
73.     """
74.     对同一受试者的序列 (列: week, y), 按孕周排序, 做平滑并求“首次达标周”T。
75.     返回:
76.         T (float 或 None) —— 若始终未达标, 返回 None (视作右删失)
77.         censored (bool) —— True 表示右删失 (至末次观测仍未 $\geq$ 阈值)
78.         weeks_sorted (np.ndarray) —— (可选) 排序后的周
79.         y_smooth (np.ndarray) —— (可选) 平滑/单调化后的 y
80.     """
81.     sdf = person_df.sort_values("week")
82.     w = sdf["week"].values
83.     y = sdf["y"].values
84.     if len(w) == 0:
85.         return None, True, None, None
86.
87.     # 单调平滑
88.     if enforce_monotonic and len(w) >= 2:
89.         iso = IsotonicRegression(increasing=True, out_of_bounds="clip")
90.         y_hat = iso.fit_transform(w, y)
91.     else:
92.         y_hat = y.copy()
93.
94.     thr = threshold + delta
95.
96.     # 若最后一点仍未达标, 视为右删失
97.     if y_hat[-1] < thr:
98.         return None, True, w, y_hat
99.
100.    # 找最早交叉点 (线性插值求精确周)
101.    for i in range(1, len(w)):
102.        if y_hat[i-1] < thr <= y_hat[i]:
103.            # 线性插值 t = w[i-1] + (thr - y1)*(w2-w1)/(y2-y1)

```

```

104.             y1, y2 = y_hat[i-1], y_hat[i]
105.             x1, x2 = w[i-1], w[i]
106.             if y2 == y1:
107.                 t_cross = x2
108.             else:
109.                 t_cross = x1 + (thr - y1) * (x2 - x1) / (y2 - y1)
110.             return float(t_cross), False, w, y_hat
111.
112.         return float(w[-1]), False, w, y_hat
113.
114.
115.     def build_T_dataset(df: pd.DataFrame,
116.                         threshold: float = 0.04,
117.                         delta: float = 0.003) -> pd.DataFrame:
118.         """为每个受试者计算 (T, censored, bmi)。bmi 取个体中位数。"""
119.         records = []
120.         for pid, g in df.groupby("id"):
121.             T, cens, _, _ = estimate_T_for_one(g[["week", "y"]], threshold=threshold, delta=delta)
122.             bmi_val = float(g["bmi"].median()) if len(g) else np.nan
123.             records.append({"id": pid, "T": T, "censored": cens, "bmi": bmi_val})
124.         out = pd.DataFrame(records)
125.         out = out.dropna(subset=["bmi"])
126.         return out
127.
128.     # Kaplan-Meier 生存 + 期望成本
129.     def km_survival(times: np.ndarray, events: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
130.         """
131.             简单 Kaplan-Meier:
132.             输入: times (事件或删失时间), events(1=事件发生; 0=右删失)
133.             输出: unique_event_times, S(t) 在这些事件时刻的取值 (阶梯函数右极限)
134.             备注: 用于估 P(T>t)。
135.         """
136.         # 仅在事件时刻更新 S
137.         df = pd.DataFrame({"t": times, "e": events}).sort_values(["t", "e"], ascending=[True, False])
138.         uniq_event_times = np.sort(df.loc[df["e"] == 1, "t"].unique())
139.         S = 1.0
140.         S_list = []
141.         at_risk_n = len(df)
142.         idx = 0
143.         # 迭代每个事件时刻, 计算当时的在险人数和事件数

```

```

144.     for u in uniq_event_times:
145.         # 在时刻 u 之前离开的（删失/事件 <u>）已在风险集中剔除；这里进行简化处理：逐时刻重算在险人数
146.         at_risk_n = ((df["t"] >= u)).sum()
147.         d_u = ((df["t"] == u) & (df["e"] == 1)).sum()
148.         if at_risk_n > 0:
149.             S *= (1.0 - d_u / at_risk_n)
150.         S_list.append(S)
151.     return uniq_event_times, np.array(S_list, dtype=float)
152.
153.
154. def S_of_t(t: float, event_times: np.ndarray, S_vals: np.ndarray) -> float
155.     """
156.     返回任意 t 的 S(t) (右连续阶梯)：取 t 所在的最后一个事件时刻的 S。
157.     if len(event_times) == 0:
158.         return 1.0
159.     idx = np.searchsorted(event_times, t, side="right") - 1
160.     if idx < 0:
161.         return 1.0
162.     return float(S_vals[idx])
163.
164. def expected_late_term(t: float, W_late: float,
165.                         event_times: np.ndarray,
166.                         S_vals: np.ndarray) -> float:
167.     """
168.     计算  $E[(t - T)_+ * \mathbb{1}_{\{T>W\_late\}}] \approx \sum_{u \in (W\_late, t]} (t - u) * dF(u)$ ，其中  $dF(u) \approx -\Delta S(u)$  在事件时刻的跳跃量（来自 KM 曲线）。
169.
170.     """
171.     if len(event_times) == 0 or t <= W_late:
172.         return 0.0
173.     # 计算跳跃量 dF(u) = -ΔS(u)
174.     S_prev = 1.0
175.     val = 0.0
176.     for u, S_u in zip(event_times, S_vals):
177.         if u <= W_late:
178.             S_prev = S_u
179.             continue
180.         if u > t:
181.             break
182.         dF = max(0.0, S_prev - S_u)
183.         val += (t - float(u)) * dF
184.     S_prev = S_u

```

```

185.     return float(val)
186.
187.
188. @dataclass
189. class CostParams:
190.     c_early: float = 1.0
191.     c_late: float = 1.0
192.     W_late: float = 12.0 # 晚发现开始计惩罚的阈值周（示例）
193.
194.
195. def compute_bin_cost(times: np.ndarray, events: np.ndarray,
196.                      t_grid: np.ndarray, params: CostParams) -> Tuple[floa
t, float, Dict]:
197.     """
198.         给定一个 BMI 分组样本(的 T, event), 在 t_grid 上找使 EC(t) 最小的 t*。
199.         返回: (最小成本, t*, 附加信息 dict)
200.         附加信息包含: P_early(t*), P_late(t*), S 曲线节点等。
201.     """
202.     # KM 生存曲线
203.     ev_times, S_vals = km_survival(times, events)
204.
205.     best_cost = math.inf
206.     best_t = None
207.     best_info = {}
208.
209.     for t in t_grid:
210.         P_early = S_of_t(t, ev_times, S_vals) # P(T>t)
211.         late_term = expected_late_term(t, params.W_late, ev_times, S_vals)
212.
213.         cost = params.c_early * P_early + params.c_late * late_term
214.         if cost < best_cost:
215.             # 粗略估计晚发现概率: P(W_late < T ≤ t) ≈ F(t) - F(W_late)
216.             S_t = S_of_t(t, ev_times, S_vals)
217.             S_w = S_of_t(params.W_late, ev_times, S_vals)
218.             P_late = max(0.0, (1 - S_t) - (1 - S_w))
219.             best_cost = cost
220.             best_t = float(t)
221.             best_info = {
222.                 "P_early": float(P_early),
223.                 "P_late": float(P_late),
224.                 "ev_times": ev_times,
225.                 "S_vals": S_vals,
226.             }
226.     return best_cost, best_t, best_info

```

```

227.
228.     # 监督分箱（动态规划切分）
229.     def precompute_bin_costs(dfT: pd.DataFrame,
230.                               candidate_edges: np.ndarray,
231.                               t_grid: np.ndarray,
232.                               params: CostParams,
233.                               min_bin_size: int = 30) -> Tuple[np.ndarray, np.n
234.                                         darray, List[List[Dict]]]:
235.         """
236.             对所有候选 BMI 区间 [edge[i], edge[j]) 预先计算：
237.             - cost[i,j]: 该区间的最小期望成本
238.             - best_t[i,j]: 对应的最优检测周
239.             - extras[i][j]: 字典，含 P_early, P_late 等
240.             不满足最小样本量的区间记为 +inf。
241.         """
242.         M = len(candidate_edges)
243.         cost = np.full((M, M), np.inf, dtype=float)
244.         best_t = np.full((M, M), np.nan, dtype=float)
245.         extras: List[List[Dict]] = [[{} for _ in range(M)] for __ in range(M)]
246.         for i in range(M - 1):
247.             for j in range(i + 1, M):
248.                 lo, hi = candidate_edges[i], candidate_edges[j]
249.                 sub = dfT[(dfT["bmi"] >= lo) & (dfT["bmi"] < hi)]
250.                 if len(sub) < min_bin_size:
251.                     continue
252.                 times = sub["T"].fillna(sub["T"].max() + 1e-6).values
253.                 events = (~sub["censored"]).astype(int).values
254.                 c, t_star, info = compute_bin_cost(times, events, t_grid, para
255.                                         ms)
256.                 cost[i, j] = c
257.                 best_t[i, j] = t_star
258.                 extras[i][j] = info | {"n": int(len(sub)), "bmi_range": (float
259.                                         (lo), float(hi))}
260.
261.     def optimal_binning_dp(candidate_edges: np.ndarray,
262.                             cost_mat: np.ndarray,
263.                             K: int) -> Tuple[List[Tuple[float, float]], List[T
264.                                         uple[int, int]]]:
265.         """
266.             动态规划：在候选边界上选 K 个分箱（K 个区间），使区间成本和最小。

```

```

266.     返回:
267.         bins_ranges: 每个箱的 (lo, hi)
268.         idx_pairs: 每个箱对应的 (i, j) 索引区间
269.         """
270.         M = len(candidate_edges)
271.         dp = np.full((K + 1, M), np.inf)
272.         prev = [[-1] * M for _ in range(K + 1)]
273.
274.         dp[0, 0] = 0.0
275.
276.         for k in range(1, K + 1):
277.             for j in range(1, M):
278.                 # 枚举上一个断点 i
279.                 best_val = np.inf
280.                 best_i = -1
281.                 for i in range(0, j):
282.                     c = cost_mat[i, j]
283.                     if not np.isfinite(c):
284.                         continue
285.                     if np.isfinite(dp[k - 1, i]):
286.                         v = dp[k - 1, i] + c
287.                         if v < best_val:
288.                             best_val = v
289.                             best_i = i
290.                         dp[k, j] = best_val
291.                         prev[k][j] = best_i
292.
293.             # 终点在 M-1 (用到最右边界)
294.             if not np.isfinite(dp[K, M - 1]):
295.                 raise RuntimeError("无法在给定候选边界与样本量约束下找到可行的 K 组切分; 请降低 K 或放宽 min_bin_size。")
296.
297.             # 回溯区间
298.             idx_pairs: List[Tuple[int, int]] = []
299.             j = M - 1
300.             for k in range(K, 0, -1):
301.                 i = prev[k][j]
302.                 if i < 0:
303.                     raise RuntimeError("回溯失败, 请检查 cost 矩阵是否连通。")
304.                 idx_pairs.append((i, j))
305.                 j = i
306.             idx_pairs.reverse()
307.

```

```

308.     bins_ranges = [(float(candidate_edges[i]), float(candidate_edges[j])))
309.     for (i, j) in idx_pairs]
310.
311. # 主流程 + 灵敏度分析
312. def run_pipeline(cfg: Dict,
313.                     threshold: float = 0.04,
314.                     delta: float = 0.003,
315.                     K: int = 4,
316.                     min_week: float = 9.0,
317.                     max_week: float = 16.0,
318.                     week_step: float = 0.25,
319.                     min_bin_size: int = 30,
320.                     cost_params: Optional[CostParams] = None,
321.                     candidate_mode: str = "quantile",
322.                     n_edges: int = 21,
323.                     verbose: bool = True) -> Dict:
324.     """
325.     运行完整管线:
326.     1) 读数-清洗; 2) 估计 T; 3) 预计算每个 BMI 区间成本; 4) DP 找最优分箱;
327.     5) 汇报结果。
328.     参数:
329.         - K: BMI 分组数量
330.         - delta: 阈值缓冲 (默认 0.003 = 0.3 个百分点)
331.         - candidate_mode: 候选边界生成方式 ('quantile' 或 'unique')
332.         - n_edges: 若为 quantile, 则在 [min,max] 均匀分位生成 n_edges 个边界点
333.     返回: 包含关键中间结果与最终方案的字典。
334.     if cost_params is None:
335.         cost_params = CostParams()
336.
337.     if verbose:
338.         print("[1/5] 读取与清洗数据...")
339.     df = load_and_prepare_data(cfg)
340.
341.     if verbose:
342.         print("[2/5] 估计各个体的首次达标周 T (带阈值缓冲) ...")
343.     dfT = build_T_dataset(df, threshold=threshold, delta=delta)
344.
345.     # 候选 BMI 边界
346.     if candidate_mode == "quantile":
347.         qs = np.linspace(0, 1, n_edges)
348.         edges = np.quantile(dfT["bmi"].values, qs)
349.         edges[0] = float(dfT["bmi"].min())

```

```

350.         edges[-1] = float(dfT["bmi"].max()) + 1e-9
351.         candidate_edges = np.unique(edges)
352.     else:
353.         vals = np.sort(dfT["bmi"].values)
354.         candidate_edges = np.r_[vals[:max(1, len(vals))//(n_edges-
1)], vals[-1] + 1e-9]
355.
356.     # t 的离散搜索网格
357.     t_grid = np.arange(min_week, max_week + 1e-9, week_step)
358.
359.     if verbose:
360.         print("[3/5] 预计算所有候选 BMI 区间的最优 t* 与成本...")
361.     cost_mat, best_t_mat, extras = precompute_bin_costs(
362.         dfT, candidate_edges, t_grid, cost_params, min_bin_size=min_bin_si-
ze
363.     )
364.
365.     if verbose:
366.         print("[4/5] 动态规划搜索 K 组的全局最优切分...")
367.     bin_ranges, idx_pairs = optimal_binning_dp(candidate_edges, cost_mat,
K)
368.
369.     # 汇总每组信息
370.     groups = []
371.     total_cost = 0.0
372.     for (i, j), (lo, hi) in zip(idx_pairs, bin_ranges):
373.         c = cost_mat[i, j]
374.         t_star = best_t_mat[i, j]
375.         info = extras[i][j]
376.         groups.append({
377.             "BMI_range": (lo, hi),
378.             "t_star": float(t_star),
379.             "expected_cost": float(c),
380.             "P_early": float(info.get("P_early", np.nan)),
381.             "P_late": float(info.get("P_late", np.nan)),
382.             "n": int(info.get("n", 0)),
383.         })
384.         total_cost += float(c)
385.
386.     result = {
387.         "config": cfg,
388.         "threshold": threshold,
389.         "delta": delta,
390.         "K": K,

```

```

391.         "min_week": min_week,
392.         "max_week": max_week,
393.         "week_step": week_step,
394.         "min_bin_size": min_bin_size,
395.         "cost_params": cost_params,
396.         "candidate_edges": candidate_edges,
397.         "t_grid": t_grid,
398.         "groups": groups,
399.         "total_expected_cost": total_cost,
400.         "dft": dft,
401.     }
402.
403.     if verbose:
404.         print("[5/5] 完成。")
405.         print("— 推荐方案 —")
406.         for g in groups:
407.             lo, hi = g["BMI_range"]
408.             print(f"BMI∈[{lo:.2f}, {hi:.2f}) → 推荐检测"
周 t* = {g['t_star']:.2f} (样本 n={g['n']}) \n"
409.             f"    早到概率≈P(T>t*)={g['P_early']:.3f}, 晚发现概率
        ={g['P_late']:.3f}, 期望成本={g['expected_cost']:.4f}")
410.             print(f"总期望成本 = {total_cost:.4f}")
411.
412.     return result
413.
414. # 敏感度分析工具
415.
416. def sensitivity_grid(cfg: Dict,
417.                         K_list: List[int] = (3, 4, 5),
418.                         delta_list: List[float] = (0.002, 0.003, 0.005),
419.                         c_early_list: List[float] = (1.0, 1.0, 1.0),
420.                         c_late_list: List[float] = (0.5, 1.0, 2.0),
421.                         W_late_list: List[float] = (11.0, 12.0, 13.0),
422.                         **kwargs) -> pd.DataFrame:
423.     """网格扫描关键参数, 比较总期望成本与方案稳定性。"""
424.     rows = []
425.     for K in K_list:
426.         for delta in delta_list:
427.             for ce in c_early_list:
428.                 for cl in c_late_list:
429.                     for Wl in W_late_list:
430.                         params = CostParams(c_early=ce, c_late=cl, W_late=
Wl)
431.                         try:

```

```

432.                     res = run_pipeline(cfg, K=K, delta=delta, cost
433.                         _params=params, verbose=False, **kwargs)
434.                     rows.append({
435.                         "K": K,
436.                         "delta": delta,
437.                         "c_early": ce,
438.                         "c_late": cl,
439.                         "W_late": Wl,
440.                         "total_cost": res["total_expected_cost"],
441.                     })
442.             except Exception as e:
443.                 rows.append({
444.                     "K": K, "delta": delta, "c_early": ce, "c_
445.                         late": cl, "W_late": Wl,
446.                         "total_cost": np.nan, "groups": str(e)
447.                     })
448.
449.     # 运行
450.     if __name__ == "__main__":
451.         # 1) 先把 CONFIG 里的列名改成你实际的数据列名, 然后再运行。
452.         # 2) 设定关键参数:
453.         cfg = CONFIG.copy()
454.
455.         params = CostParams(
456.             c_early=1.0,    # 过早(未达标/复检)的单位代价
457.             c_late=1.5,   # 过晚(超过 W_late 后)的单位代价
458.             W_late=12.0   # 晚发现起算周
459.         )
460.
461.         # 运行主流程(K=4 组; t 在 9~16 周、步长 0.25 周 里选最优)
462.         result = run_pipeline(
463.             cfg,
464.             threshold=0.04,
465.             delta=0.003,
466.             K=4,
467.             min_week=9.0,
468.             max_week=16.0,
469.             week_step=0.25,
470.             min_bin_size=30,
471.             cost_params=params,
472.             candidate_mode="quantile",

```

```

473.         n_edges=21,
474.         verbose=True,
475.     )
476.
477.
478. #结果生成图: 直观解释
479. import matplotlib
480. import matplotlib.pyplot as plt
481. import numpy as np
482. import pandas as pd
483.
484. plt.rcParams['font.sans-
    serif'] = ['PingFang SC','Heiti SC','Heiti TC','Arial Unicode MS','SimHei','N
    oto Sans CJK SC']
485. plt.rcParams['axes.unicode_minus'] = False
486.
487. USE_RESULT = False
488.
489. if USE_RESULT:
490.     # A) 直接用 run_pipeline 的返回值
491.     groups = []
492.     for g in result["groups"]:
493.         lo, hi = g["BMI_range"]
494.         groups.append({
495.             "BMI_range": (lo, hi),
496.             "t_star": g["t_star"],
497.             "P_early": g["P_early"],
498.             "P_late": g["P_late"],
499.             "expected_cost": g["expected_cost"],
500.         })
501. else:
502.     # B) 用贴出来的控制台结果手动构造
503.     groups = [
504.         {"BMI_range": (26.81, 31.22), "t_star": 14.75, "P_early": 0.880, "P_
    late": 0.120, "expected_cost": 0.9682},
505.         {"BMI_range": (31.22, 32.91), "t_star": 14.00, "P_early": 0.904, "P_
    late": 0.096, "expected_cost": 0.9343},
506.         {"BMI_range": (32.91, 34.16), "t_star": 12.50, "P_early": 0.949, "P_
    late": 0.026, "expected_cost": 0.9530},
507.         {"BMI_range": (34.16, 39.30), "t_star": 13.00, "P_early": 0.925, "P_
    late": 0.057, "expected_cost": 0.9524},
508.     ]
509.
510. df_groups = pd.DataFrame(groups)

```

```

511. labels = [f"{lo:.2f}-{hi:.2f}" for lo, hi in df_groups["BMI_range"]]
512.
513. # 图一: 各 BMI 组的推荐检测周 t*
514. plt.figure(figsize=(8, 5))
515. plt.barh(labels, df_groups["t_star"])
516. plt.xlabel("推荐检测周 t*")
517. plt.ylabel("BMI 分组")
518. plt.title("各 BMI 组的推荐检测周")
519. for y, v in enumerate(df_groups["t_star"]):
520.     plt.text(v, y, f" {v:.2f}", va="center") # 条形末尾标注
521. plt.tight_layout()
522. plt.show()
523.
524. # 图二: 各组早到概率 & 晚发现概率
525. x = np.arange(len(df_groups))
526. width = 0.35
527.
528. fig, ax = plt.subplots(figsize=(8, 5))
529. rects1 = ax.bar(x - width/2, df_groups["P_early"], width, label="早到概率 P(T>t*)")
530. rects2 = ax.bar(x + width/2, df_groups["P_late"], width, label="晚发现概率")
531.
532. ax.set_ylabel("概率")
533. ax.set_title("各 BMI 组的早到/晚发现概率")
534. ax.set_xticks(x)
535. ax.set_xticklabels(labels)
536. ax.legend()
537.
538. # 标注数值
539. for r in rects1:
540.     h = r.get_height()
541.     ax.text(r.get_x() + r.get_width()/2, h, f"{h:.3f}", ha="center", va="bottom")
542. for r in rects2:
543.     h = r.get_height()
544.     ax.text(r.get_x() + r.get_width()/2, h, f"{h:.3f}", ha="center", va="bottom")
545.
546. plt.tight_layout()
547. plt.show()
548.
549. # 图三: 各组期望成本
550. plt.figure(figsize=(8, 5))

```

```

551. plt.bar(labels, df_groups["expected_cost"])
552. plt.ylabel("期望成本")
553. plt.title("各 BMI 组的期望成本")
554. for i, v in enumerate(df_groups["expected_cost"]):
555.     plt.text(i, v, f"{v:.3f}", ha="center", va="bottom")
556. plt.tight_layout()
557. plt.show()

```

```

1. from __future__ import annotations
2. import math
3. from dataclasses import dataclass
4. from typing import Dict, List, Tuple, Optional
5.
6. import numpy as np
7. import pandas as pd
8. from patsy import dmatrix
9. import statsmodels.api as sm
10. from statsmodels.regression.mixed_linear_model import MixedLM
11.
12. CONFIG = {
13.     "file_path": "/Users/shiyidianqianyaoshuijiao/Desktop/数模国赛
14.                 /cleaned_data.xlsx",
15.     "id_col": "孕妇代码",
16.     "week_col": "检测孕周",
17.     "y_col": "Y 染色体浓度",
18.     "bmi_col": "孕妇 BMI",
19.     "age_col": "年龄",
20.     "height_col": "身高",
21.     "weight_col": "体重",
22.     "ivf_col": "IVF 妊娠",
23.     # Y 的建模刻度
24.     "q3_logit_y": True,
25.     "y_eps": 1e-3,
26. }
27.
28. # 全局模型超参
29. SPLINE_DF = 5          # bs(week) 自由度
30. BMI_SPLINE_DF = 3      # bs(bmi) 自由度
31. RANDOM_SLOPE = True    # 个体随机截距 + 随机斜率
32. THRESH = 0.04           # 判别阈值 (概率刻度)
33.
34. @dataclass

```

```

35. class CostParams:
36.     c_early: float = 1.0
37.     c_late: float = 1.0
38.     W_late: float = 12.0
39.
40. GRID_T_MIN = 9.0
41. GRID_T_MAX = 16.0
42. GRID_T_STEP = 0.25
43.
44. # 常用函数
45. def _logit(p: np.ndarray) -> np.ndarray:
46.     p = np.asarray(p, dtype=float)
47.     return np.log(p / (1.0 - p))
48.
49. def _inv_logit(z: np.ndarray) -> np.ndarray:
50.     z = np.asarray(z, dtype=float)
51.     return 1.0 / (1.0 + np.exp(-z))
52.
53. def _sigma_prob_from_logit(mu_logit: np.ndarray, sigma_logit: float) -> np.ndarray:
54.     p = _inv_logit(mu_logit)
55.     var_p = (p * (1.0 - p))**2 * (sigma_logit**2)
56.     return np.sqrt(np.maximum(var_p, 1e-12))
57.
58. def _make_interactions(A: np.ndarray, v: np.ndarray) -> np.ndarray:
59.     v = np.asarray(v, dtype=float).reshape(-1, 1)
60.     return A * v
61.
62. def _read_data(cfg: Dict) -> pd.DataFrame:
63.     df = pd.read_excel(cfg["file_path"], sheet_name=0)
64.     need = [cfg["id_col"], cfg["week_col"], cfg["y_col"], cfg["bmi_col"]]
65.     miss = [c for c in need if c not in df.columns]
66.     if miss:
67.         raise KeyError(f"数据缺少必要列: {miss}")
68.
69.     use_cols = need.copy()
70.     for k in ["age_col", "height_col", "weight_col", "ivf_col"]:
71.         col = cfg.get(k)
72.         if col and col in df.columns:
73.             use_cols.append(col)
74.
75.     df = df[use_cols].rename(columns={
76.         cfg["id_col"]: "id",
77.         cfg["week_col"]: "week",

```

```

78.         cfg["y_col"]: "y",
79.         cfg["bmi_col"]: "bmi",
80.     })
81.
82.     # 转换 Y 为比例
83.     df["y"] = pd.to_numeric(df["y"], errors="coerce")
84.     if df["y"].median() > 1:
85.         df["y"] = df["y"] / 100.0
86.
87.     # 数值化
88.     for col in ["week", "bmi"]:
89.         df[col] = pd.to_numeric(df[col], errors="coerce")
90.     for col in ["年龄", "身高", "体重"]:
91.         if col in df.columns:
92.             df[col] = pd.to_numeric(df[col], errors="coerce")
93.
94.     # IVF : 0/1
95.     if cfg.get("ivf_col") and cfg["ivf_col"] in df.columns:
96.         col = cfg["ivf_col"]
97.         df[col] = df[col].astype(str)
98.         df["ivf"] = df[col].str.contains("IVF", case=False).astype(int)
99.
100.    # 清洗
101.    df = df.dropna(subset=["id", "week", "y", "bmi"]).copy()
102.    df = df[(df["week"]>=0) & (df["week"]<=45) & (df["y"]>=0) & (df["y"]<=
1) & (df["bmi"]>=10) & (df["bmi"]<=60)]
103.
104.    # 同人同周聚合
105.    agg = {"y": "mean", "bmi": "median"}
106.    for c in ["年龄", "身高", "体重", "ivf"]:
107.        if c in df.columns:
108.            agg[c] = "median"
109.    df = df.groupby(["id", "week"], as_index=False).agg(agg)
110.
111.    # 再强制数值化
112.    for c in ["week", "y", "bmi", "年龄", "身高", "体重", "ivf"]:
113.        if c in df.columns:
114.            df[c] = pd.to_numeric(df[c], errors="coerce")
115.
116.    df = df.dropna(subset=["week", "y", "bmi"]).copy()
117.    return df
118.
119.    # 设计矩阵构造
120.

```

```

121. def _build_spline_week(weeks: np.ndarray, df_spl: int = SPLINE_DF) -> np.n
       darray:
122.     return dmatrix(f"bs(week, df={df_spl}, include_intercept=False)",
123.                      {"week": weeks}, return_type="dataframe").to_numpy()
124.
125. def _build_spline_bmi(bmi: np.ndarray, df_spl: int = BMI_SPLINE_DF) -> np.
       ndarray:
126.     return dmatrix(f"bs(bmi, df={df_spl}, include_intercept=False)",
127.                      {"bmi": bmi}, return_type="dataframe").to_numpy()
128.
129. # MixedLM (含列筛选返回)
130. def fit_mixed_longitudinal(df: pd.DataFrame,
131.                             df_week_spl: int = SPLINE_DF,
132.                             df_bmi_spl: int = BMI_SPLINE_DF,
133.                             random_slope: bool = RANDOM_SLOPE):
134.     # 1) 随机斜率可识别性
135.     grp_sizes = df.groupby("id")["week"].nunique()
136.     has_enough = (grp_sizes >= 2).sum()
137.     if random_slope and has_enough == 0:
138.         random_slope = False
139.     if random_slope:
140.         keep_ids = grp_sizes[grp_sizes >= 2].index
141.         df_use = df[df["id"].isin(keep_ids)].copy()
142.         if df_use["id"].nunique() < 10:
143.             random_slope = False
144.         df_use = df.copy()
145.     else:
146.         df_use = df.copy()
147.
148.     weeks = df_use["week"].values
149.     bmi_v = df_use["bmi"].values
150.
151.     # 2) FE: bs(week) + bs(bmi) + 交互 + (年龄, ivf)
152.     X_week = _build_spline_week(weeks, df_week_spl)
153.     fe_blocks = [X_week]
154.     fe_names = [f"bsW{i}" for i in range(X_week.shape[1])]
155.
156.     X_bmi = _build_spline_bmi(bmi_v, df_bmi_spl)
157.     fe_blocks.append(X_bmi)
158.     fe_names += [f"bsB{i}" for i in range(X_bmi.shape[1])]
159.
160.     X_wxb = _make_interactions(X_week, bmi_v)
161.     fe_blocks.append(X_wxb)
162.     fe_names += [f"bsW{i}:bmi" for i in range(X_week.shape[1])]
```

```

163.
164.     for col in ["年龄", "ivf"]:
165.         if col in df_use.columns:
166.             v = df_use[[col]].values.astype(float)
167.             fe_blocks.append(v)
168.             fe_names.append(col)
169.
170.     X = np.hstack(fe_blocks)
171.     X = sm.add_constant(X)
172.     fe_names = ["const"] + fe_names
173.
174.     # 3) 因变量
175.     y_raw = pd.to_numeric(df_use["y"], errors="coerce").astype(float).values
176.     if CONFIG.get("q3_logit_y", False):
177.         eps = float(CONFIG.get("y_eps", 1e-3))
178.         y_clip = np.clip(y_raw, eps, 1.0 - eps)
179.         y = _logit(y_clip)
180.     else:
181.         y = y_raw
182.
183.     # 4) 列筛选: 方差过滤 + QR 列主元
184.     X = np.asarray(X, dtype=float)
185.
186.     col_std = X.std(axis=0)
187.     keep1 = col_std > 1e-8           # 对 add_constant 后的列
188.     X1 = X[:, keep1]
189.     fe_names1 = [n for n, k in zip(fe_names, keep1) if k]
190.
191.     from scipy.linalg import qr
192.     Q, R, P = qr(X1, mode='economic', pivoting=True)
193.     tol = max(X1.shape) * (np.abs(R).max() if R.size else 0.0) * 1e-12
194.     diagR = np.abs(np.diag(R)) if R.size else np.array([])
195.     rank = int(np.sum(diagR > tol)) if diagR.size else X1.shape[1]
196.     indep_idx = np.sort(P[:rank])      # 在 X1 上的列索引
197.
198.     X_final = X1[:, indep_idx]
199.     fe_names_final = [fe_names1[i] for i in indep_idx]
200.
201.     # 5) 随机效应设计
202.     if random_slope:
203.         Z = np.column_stack([np.ones(len(df_use)), df_use["week"].values])
204.         re_names = ["re_intercept", "re_week"]

```

```

205.     else:
206.         Z = np.ones((len(df_use), 1))
207.         re_names = ["re_intercept"]
208.
209.         # 6) 掩码并拟合
210.         Z = np.asarray(Z, dtype=float)
211.         mask = np.isfinite(y) & np.isfinite(X_final).all(axis=1) & np.isfinite
212.             (Z).all(axis=1)
213.         y = y[mask]; X_final = X_final[mask, :]; Z = Z[mask, :]
214.         groups = df_use.loc[mask, "id"].astype("category")
215.
216.         model = MixedLM(endog=y, exog=X_final, groups=groups, exog_re=Z)
217.         try:
218.             result = model.fit(method="lbfgs", maxiter=300, disp=False) # REM
219.         L
220.         except np.linalg.LinAlgError:
221.             try:
222.                 result = model.fit(method="lbfgs", reml=False, maxiter=300, di
223.                     sp=False)
224.             except Exception:
225.                 Z = np.ones((len(y), 1))
226.                 model2 = MixedLM(endog=y, exog=X_final, groups=groups, exog_re
227.                     =Z)
228.                 result = model2.fit(method="lbfgs", reml=False, maxiter=300, d
229.                     isp=False)
230.                 re_names = ["re_intercept"]
231.
232.                 resid = y - result.fittedvalues
233.                 sigma = float(np.std(resid, ddof=X_final.shape[1]))
234.
235.                 design_reduction = {
236.                     "keep_var_mask": keep1.tolist(), # 针对 add_constant 后的列
237.                     "qr_indep_idx": indep_idx.tolist() # 针对 keep1 过滤后的列
238.                 }
239.
240.                 return result, sigma, (fe_names_final, re_names), design_reduction
241.
242.             # 概率与成本 (从个体到组)
243.             def _norm_cdf(z: np.ndarray) -> np.ndarray:
244.                 from math import erf
245.                 z = np.asarray(z, dtype=float)
246.                 return 0.5 * (1.0 + np.vectorize(erf)(z / np.sqrt(2.0)))
247.

```

```

243. def predict_mu_for_subject(result, df_row: pd.Series, t_grid: np.ndarray,
244.                               fe_names: List[str],
245.                               df_week_spl: int = SPLINE_DF,
246.                               df_bmi_spl: int = BMI_SPLINE_DF,
247.                               design_reduction: Optional[dict] = None) -> np.
248.                               ndarray:
249.                                 # bs(week) 到 bs(bmi) 到 交互 到 年龄 到 ivf 到 add_constant
250.                                 Xw = _build_spline_week(t_grid, df_week_spl)
251.                                 blocks = [Xw]
252.                                 bmi_val = float(df_row["bmi"]) if "bmi" in df_row.index else np.nan
253.                                 if np.isfinite(bmi_val):
254.                                   Xb = _build_spline_bmi(np.full_like(t_grid, bmi_val, dtype=float),
255.                                             df_bmi_spl)
256.                                   blocks.append(Xb)
257.                                   blocks.append(Xw * bmi_val)
258.                                 for col in ["年龄", "ivf"]:
259.                                   if col in df_row.index and np.isfinite(df_row[col]):
260.                                     blocks.append(np.full((len(t_grid), 1), float(df_row[col])))
261.
262.                                 X_fixed_full = np.hstack(blocks)
263.                                 X_fixed_full = sm.add_constant(X_fixed_full)
264.
265.                                 # 复用训练时的列筛选
266.                                 if design_reduction is not None:
267.                                   keep1 = np.array(design_reduction["keep_var_mask"], dtype=bool)
268.                                   indep_idx = np.array(design_reduction["qr_indep_idx"], dtype=int)
269.
270.                                   if X_fixed_full.shape[1] != keep1.size:
271.                                     raise ValueError(f"预测矩阵列数({X_fixed_full.shape[1]})与 keep_var_mask({keep1.size})不一致")
272.                                   X1 = X_fixed_full[:, keep1]
273.                                   X_fixed = X1[:, indep_idx]
274.                                 else:
275.                                   X_fixed = X_fixed_full
276.
277.                                 beta = np.asarray(getattr(result, "fe_params", result.params)).ravel()
278.
279.                                 if X_fixed.shape[1] != beta.size:
280.                                   raise ValueError(f"预测特征列({X_fixed.shape[1]})与系数维度 ({beta.size})不一致")

```

```

280.     mu = X_fixed @ beta
281.
282.     rid = df_row["id"] if "id" in df_row.index else None
283.     re = result.random_effects.get(rid)
284.     if re is not None:
285.         re = np.asarray(re, dtype=float).ravel()
286.         if re.size >= 2:
287.             mu += re[0] + re[1] * t_grid
288.         elif re.size == 1:
289.             mu += re[0]
290.     return mu
291.
292. def estimate_sigma_T(mu_prob_t: np.ndarray, t_grid: np.ndarray, sigma_y_pr
   ob: float) -> np.ndarray:
293.     dmu = np.gradient(mu_prob_t, t_grid)
294.     slope = np.clip(np.abs(dmu), 1e-4, None)
295.     sigma_T = sigma_y_prob / slope
296.     return np.clip(sigma_T, 1e-2, 10.0)
297.
298. def compute_q_and_F(mu_prob_t: np.ndarray,
299.                      sigma_y_prob: np.ndarray | float,
300.                      sigma_T: np.ndarray,
301.                      t_grid: np.ndarray,
302.                      thresh: float = THRESH):
303.     mu_prob_t = np.asarray(mu_prob_t, dtype=float)
304.     sigma_T = np.asarray(sigma_T, dtype=float)
305.     if np.isscalar(sigma_y_prob):
306.         sigma_y_arr = np.full_like(mu_prob_t, float(sigma_y_prob))
307.     else:
308.         sigma_y_arr = np.asarray(sigma_y_prob, dtype=float)
309.         if sigma_y_arr.shape != mu_prob_t.shape:
310.             raise ValueError("sigma_y_prob 向量长度必须与 t_grid 一致")
311.
312.     z_y = (mu_prob_t - thresh) / sigma_y_arr
313.     q = _norm_cdf(z_y)
314.
315.     z_T = (mu_prob_t - thresh) / sigma_T
316.     F = _norm_cdf(z_T)
317.     return q, np.clip(F, 0.0, 1.0)
318.
319. def integral_late_term(F: np.ndarray, t_grid: np.ndarray, W_late: float, t
   _star: float) -> float:
320.     if t_star <= W_late:
321.         return 0.0

```

```

322.     mask = (t_grid >= w_late) & (t_grid <= t_star)
323.     tg = t_grid[mask]
324.     Fg = F[mask]
325.     if len(tg) < 2:
326.         return 0.0
327.     dF = np.diff(Fg)
328.     centers = tg[1:]
329.     val = np.sum((t_star - centers) * np.maximum(dF, 0.0))
330.     return float(val)
331.
332. def compute_group_cost_for_bin(df_subjects: pd.DataFrame, result, sigma_y_
    logit: float, fe_names: List[str],
    t_grid: np.ndarray, params: CostParams,
    design_reduction: dict) -> Tuple[float, flo
    at, Dict]:
333.     if len(df_subjects) == 0:
334.         return math.inf, np.nan, {}
335.     Q_list, F_list = [], []
336.     for _, row in df_subjects.iterrows():
337.         mu_lin = predict_mu_for_subject(result, row, t_grid, fe_names,
338.                                         design_reduction=design_reduction)
339.
340.         if CONFIG.get("q3_logit_y", False):
341.             mu_prob = _inv_logit(mu_lin)
342.             sigma_y_prob_vec = _sigma_prob_from_logit(mu_lin, sigma_y_logi
    t)
343.             sigma_T_i = estimate_sigma_T(mu_prob, t_grid, float(np.mean(si
    gma_y_prob_vec)))
344.             q_i, F_i = compute_q_and_F(mu_prob, sigma_y_prob_vec, sigma_T_
    i, t_grid)
345.         else:
346.             mu_prob = mu_lin
347.             sigma_T_i = estimate_sigma_T(mu_prob, t_grid, sigma_y_logit)
348.             q_i, F_i = compute_q_and_F(mu_prob, float(sigma_y_logit), sigm
    a_T_i, t_grid)
349.             Q_list.append(q_i); F_list.append(F_i)
350.
351.             Q = np.vstack(Q_list)
352.             F = np.vstack(F_list)
353.
354.             best_cost, best_t, best_info = math.inf, None, {}
355.             for t in t_grid:

```

```

359.         k = int(round((t - t_grid[0]) / (t_grid[1] - t_grid[0])))
360.         k = np.clip(k, 0, len(t_grid)-1)
361.
362.         early = float(np.mean(1.0 - Q[:, k]))
363.         late_vals = [integral_late_term(F[i], t_grid, params.W_late, t) for
364.             i in range(F.shape[0])]
365.         late = float(np.mean(late_vals))
366.
367.         cost = params.c_early * early + params.c_late * late
368.
369.         if cost < best_cost:
370.             idx_w = int(round((params.W_late - t_grid[0]) / (t_grid[1] - t
371.                 _grid[0])))
372.             idx_w = np.clip(idx_w, 0, len(t_grid)-1)
373.             P_late = float(np.mean(np.maximum(0.0, (F[:, k] - F[:, idx_w])
374.                 )))
375.             best_cost, best_t = cost, float(t)
376.             best_info = {"P_early": float(early), "P_late": float(P_late),
377.                 "n": int(F.shape[0])}
378.
379.         return best_cost, best_t, best_info
380.
381.     # BMI 分箱 + DP
382.     def optimal_binning_by_dp(df_subjects: pd.DataFrame, result, sigma_y_logit
383.         : float, fe_names: List[str],
384.             K: int, params: CostParams, min_bin_size: int =
385.                 30,
386.                 n_edges: int = 21, design_reduction: Optional[di
387.                     ct] = None) -> Dict:
388.         edges = np.quantile(df_subjects["bmi"].values, np.linspace(0, 1, n_edg
389.             es))
390.         edges[0] = float(df_subjects["bmi"].min())
391.         edges[-1] = float(df_subjects["bmi"].max()) + 1e-9
392.         edges = np.unique(edges)
393.         M = len(edges)
394.         t_grid = np.arange(GRID_T_MIN, GRID_T_MAX + 1e-9, GRID_T_STEP)

```

```

395.             sub = df_subjects[(df_subjects["bmi"] >= lo) & (df_subjects["b
396.                 if len(sub) < min_bin_size:
397.                     continue
398.                 c, t_star, info = compute_group_cost_for_bin(sub, result, sigm
a_y_logit, fe_names,
399.                                                 t_grid, params, d
400.             cost[i, j] = c
401.             t_star_mat[i, j] = t_star
402.             extras[i][j] = info | {"bmi_range": (float(lo), float(hi))}

403.
404.             dp = np.full((K+1, M), np.inf)
405.             prev = [[-1]*M for _ in range(K+1)]
406.             dp[0, 0] = 0.0
407.
408.             for k in range(1, K+1):
409.                 for j in range(1, M):
410.                     best_val, best_i = np.inf, -1
411.                     for i in range(0, j):
412.                         if not np.isfinite(cost[i, j]):
413.                             continue
414.                         if np.isfinite(dp[k-1, i]):
415.                             v = dp[k-1, i] + cost[i, j]
416.                             if v < best_val:
417.                                 best_val, best_i = v, i
418.             dp[k, j] = best_val
419.             prev[k][j] = best_i
420.
421.             if not np.isfinite(dp[K, M-1]):
422.                 raise RuntimeError("无法在给定 K 和样本量阈值下找到可行切分; 请调
整 K 或 min_bin_size。")
423.
424.             idx_pairs = []
425.             j = M-1
426.             for k in range(K, 0, -1):
427.                 i = prev[k][j]
428.                 idx_pairs.append((i, j))
429.                 j = i
430.             idx_pairs.reverse()
431.
432.             groups = []
433.             total_cost = 0.0
434.             for (i, j) in idx_pairs:

```

```

435.         lo, hi = edges[i], edges[j]
436.         info = extras[i][j]
437.         c = cost[i, j]
438.         t_star = t_star_mat[i, j]
439.         groups.append({
440.             "BMI_range": (float(lo), float(hi)),
441.             "t_star": float(t_star),
442.             "expected_cost": float(c),
443.             "P_early": float(info.get("P_early", np.nan)),
444.             "P_late": float(info.get("P_late", np.nan)),
445.             "n": int(info.get("n", 0)),
446.         })
447.         total_cost += float(c)
448.
449.     return {"edges": edges, "groups": groups, "total_expected_cost": total
450.             _cost, "K": K, "t_grid": t_grid}
451. # 主流程
452.
453. def run_method_B(cfg: Dict = CONFIG, K: int = 4, params: Optional[CostPara
454. ms] = None, min_bin_size: int = 30) -> Dict:
455.     if params is None:
456.         params = CostParams()
457.     print("[1/4] 读取并整理纵向数据...")
458.     df = _read_data(cfg)
459.
460.     print("[2/4] MixedLM: bs(week) + bs(bmi) + bs(week)xbs(bmi) + 年
461.           龄 + ivf + 随机效应 ...")
462.     result, sigma_y_logit, (fe_names, re_names), design_reduction = fit_mi
463.     xed_longitudinal(
464.         df, SPLINE_DF, BMI_SPLINE_DF, RANDOM_SLOPE
465.     )
466.     print(result.summary())
467.     print(f"估计噪声 (logit 尺度) σ ≈ {sigma_y_logit:.4f}")
468.
469.     # 每个体一行
470.     agg = {"bmi": "median"}
471.     for c in ["年龄", "ivf"]:
472.         if c in df.columns:
473.             agg[c] = "median"
474.     rep = df.groupby("id", as_index=False).agg(agg)

475.     print("[3/4] 按 BMI 监督分箱 + 动态规划, 搜索最优分组与各组 t* ...")

```

```

475.         out = optimal_binning_by_dp(rep, result, sigma_y_logit, fe_names, K=K,
476.                                         params=params,
477.                                         min_bin_size=min_bin_size, design_reduction
478.                                         n=design_reduction)
479.
480.         print("[4/4] 完成。推荐方案: ")
481.         for g in out["groups"]:
482.             lo, hi = g["BMI_range"]
483.             print(f"BMI∈[{lo:.2f}, {hi:.2f}) → t* = {g['t_star']:.2f},
484.                   n={g['n']}, "
485.                   f"P_early={g['P_early']:.3f}, P_late≈{g['P_late']:.3f},
486.                   EC={g['expected_cost']:.4f}")
487.
488.     if __name__ == "__main__":
489.         params = CostParams(c_early=1.0, c_late=1.0, W_late=12.0)
490.         res = run_method_B(CONFIG, K=4, params=params, min_bin_size=30)
491.
492.
493.
494.     ###画图, 直观解释数据
495.     import pandas as pd
496.     import matplotlib.pyplot as plt
497.
498.     def _mode_int(series: pd.Series, default_val: int = 0) -> int:
499.         if series is None or len(series) == 0:
500.             return default_val
501.         try:
502.             return int(series.mode(dropna=True).iloc[0])
503.         except Exception:
504.             return default_val
505.
506.     def build_group_long_table(res_pack: dict,
507.                               cost_params: CostParams,
508.                               save_csv: str = "q3_outputs_long.csv") -> pd.Da
509.                               taFrame:
510.                               """
511.                               基于现有的 res (run_method_B 的返回), 为每个 BMI 组在 t_grid 上
512.                               生成长表: group, week, mu, q, F, EC, 并计算该组 t_star。

```

```

512.      """
513.
514.      result = res_pack["model_result"]
515.      sigma_y_logit = res_pack["sigma_y_logit"]
516.      fe_names = res_pack["fe_names"]
517.      dp_out = res_pack["dp_out"]
518.      design_reduction = res_pack["design_reduction"]
519.      t_grid = dp_out["t_grid"]
520.
521.      # 取用于代表组内的“典型特征”: 中位 BMI、年龄中位数、IVF 众数
522.      # 在这里用 DP 已经选出的 BMI 区间, 在训练用的“个体代表表”(dp 分箱时使用
      的 rep) 的集合上再取代表
523.      # 简化处理: 用 dp_out 的 groups 里的 BMI_range, 结合 run_method_B 内部
      的 rep 重新构造
524.      # 为了复用, 从 model 的原训练数据无法直接取到 df/rep, 这里采用 dp_out 边
      界 + result 的随机效应 id 列表做近似
525.
526.      df_rep = None
527.      try:
528.          pass
529.      except Exception:
530.          pass
531.
532.      # 默认代表值 (若无法取到原 df, 则采用)
533.      age_default = 30.0
534.      ivf_default = 0
535.
536.      groups_rows = []
537.      for g in dp_out["groups"]:
538.          lo, hi = g["BMI_range"]
539.          bmi_mid = 0.5 * (lo + hi)
540.          for w in t_grid:
541.              row = {
542.                  "group": f"BMI[{lo:.2f},{hi:.2f})",
543.                  "week": float(w),
544.                  "bmi": float(bmi_mid),
545.                  "年龄": age_default,
546.                  "ivF": ivf_default,
547.              }
548.              groups_rows.append(row)
549.      df_pred = pd.DataFrame(groups_rows)
550.
551.      # 逐行调用已有的“纵向预测”工具, mu (logit 到 prob)
552.      mu_list = []

```

```

553.     for _, r in df_pred.iterrows():
554.         mu_lin = predict_mu_for_subject(result, r, np.array([r["week"]]),
555.                                         fe_names,
556.                                         design_reduction=design_reduction)

556.         if CONFIG.get("q3_logit_y", False):
557.             mu_prob = _inv_logit(mu_lin)[0]
558.         else:
559.             mu_prob = float(mu_lin[0])
560.         mu_list.append(mu_prob)
561.     df_pred["mu"] = np.clip(np.array(mu_list, dtype=float), 0.0, 1.0)
562.
563.     # 组内构造 q(t), F(t), EC(t); 并找 t_star
564.     out_blocks = []
565.     for gname, gdf in df_pred.groupby("group"):
566.         gdf = gdf.sort_values("week").copy()
567.         # 估计每个点的概率尺度噪声 (由 logit 残差换算)
568.         # 用每点的 mu_lin 无法直接获得 (上面只算了一个单点); 这里用一个近似: 固定 sigma_y_prob 为 mu 对应的平均
569.         mu_lin_full = predict_mu_for_subject(
570.             result,
571.             pd.Series({"bmi": float(gdf["bmi"].iloc[0]), "年齡":
572.                         ": age_default, "ivf": ivf_default}),
573.             gdf["week"].values,
574.             fe_names,
575.             design_reduction=design_reduction
576.         )
577.         if CONFIG.get("q3_logit_y", False):
578.             mu_prob_full = _inv_logit(mu_lin_full)
579.             sigma_y_prob_vec = _sigma_prob_from_logit(mu_lin_full, res_pac
580.                 k["sigma_y_logit"])
581.         else:
582.             mu_prob_full = mu_lin_full
583.             sigma_y_prob_vec = np.full_like(mu_prob_full, res_pack["sigma_
584.                 y_logit"], dtype=float)
585.
586.             # q(t), F(t)
587.             q_vec, F_vec = compute_q_and_F(mu_prob_full, sigma_y_prob_vec, sig
588.                 ma_T_vec, gdf["week"].values, thresh=THRESH)

```

```

589.         # EC(t)
590.         # 这里使用与组成本一致的构造（瞬时形式）： EC(t)=c_early*(1-
      q(t)) + c_late*(F(t)-F(W_late))
591.         # F(W_late) 用该组在最接近 W_late 的 t 的 F 值
592.         idx_w = (np.abs(gdf["week"].values - cost_params.W_late)).argmin()

593.         F_wlate = F_vec[idx_w]
594.         EC_vec = cost_params.c_early * (1.0 - q_vec) + cost_params.c_late
      * (F_vec - F_wlate)
595.
596.         # 最优 t*
597.         j_star = int(np.argmin(EC_vec))
598.         t_star = float(gdf["week"].values[j_star])
599.
600.         blk = pd.DataFrame({
601.             "group": gname,
602.             "week": gdf["week"].values,
603.             "mu": mu_prob_full,
604.             "q": q_vec,
605.             "F": F_vec,
606.             "EC": EC_vec,
607.             "t_star": t_star
608.         })
609.         out_blocks.append(blk)
610.
611.         df_long = pd.concat(out_blocks, ignore_index=True)
612.         df_long.to_csv(save_csv, index=False)
613.         return df_long
614.
615.     #图一
616.     def plot_mu_by_group(df_long: pd.DataFrame, thresh: float = THRESH, savepa
      th: str = "fig_mu_by_group.png"):
617.         plt.figure()
618.         for g, gdf in df_long.sort_values(["group", "week"]).groupby("group"):

619.             plt.plot(gdf["week"], gdf["mu"], label=str(g))
620.             wmin, wmax = df_long["week"].min(), df_long["week"].max()
621.             plt.hlines(thresh, wmin, wmax, linestyles="dashed")
622.             plt.xlabel("Gestational week")
623.             plt.ylabel("Predicted Y-concentration (proportion)")
624.             plt.title("Predicted concentration vs week (by BMI group)")
625.             plt.legend()
626.             plt.tight_layout()
627.             plt.savefig(savepath, dpi=300)

```

```

628.     plt.close()
629.
630.     #图二
631.     def plot_q_by_group(df_long: pd.DataFrame, savepath: str = "fig_q_by_group
632.                           .png"):
632.         plt.figure()
633.         for g, gdf in df_long.sort_values(["group", "week"]).groupby("group"):

634.             plt.plot(gdf["week"], gdf["q"], label=str(g))
635.             # 标注 t*
636.             t_star = float(gdf["t_star"].iloc[0])
637.             row = gdf.iloc[(gdf["week"] - t_star).abs().values.argmin()]
638.             plt.scatter([row["week"]], [row["q"]])
639.             plt.xlabel("Gestational week")
640.             plt.ylabel("Attainment probability q(t)")
641.             plt.title("Probability of exceeding 4% vs week (by BMI group)")
642.             plt.legend()
643.             plt.tight_layout()
644.             plt.savefig(savepath, dpi=300)
645.             plt.close()
646.
647.     #图三
648.     def plot_EC_by_group(df_long: pd.DataFrame, savepath: str = "fig_EC_by_gro
649.                           up.png"):
649.         plt.figure()
650.         for g, gdf in df_long.sort_values(["group", "week"]).groupby("group"):

651.             plt.plot(gdf["week"], gdf["EC"], label=str(g))
652.             j = gdf["EC"].values.argmin()
653.             plt.scatter([gdf.iloc[j]["week"]], [gdf.iloc[j]["EC"]])
654.             plt.xlabel("Gestational week")
655.             plt.ylabel("Expected cost EC(t)")
656.             plt.title("Expected cost vs week (by BMI group)")
657.             plt.legend()
658.             plt.tight_layout()
659.             plt.savefig(savepath, dpi=300)
660.             plt.close()
661.
662.     def make_q3_figures(res_pack: dict, cost_params: CostParams):
663.         df_long = build_group_long_table(res_pack, cost_params)
664.         plot_mu_by_group(df_long)
665.         plot_q_by_group(df_long)
666.         plot_EC_by_group(df_long)

```

```
667.     print("☒ 已输出:  
fig_mu_by_group.png / fig_q_by_group.png / fig_EC_by_group.png 以  
及 q3_outputs_long.csv")
```

```
1. import pandas as pd  
2. import numpy as np  
3. import matplotlib.pyplot as plt  
4. %matplotlib inline  
5.  
6. female_data = pd.read_excel(r'女胎.xlsx')  
7. female_data.columns  
8.  
9. female_data['染色体的非整倍体'] = female_data['染色体的非整倍体  
'].apply(lambda x: 1 if pd.notna(x) and x != '' else 0)  
10.  
11. female_data['孕妇 BMI'] = pd.to_numeric(female_data['孕妇  
BMI'], errors='coerce')  
12.  
13. # 方法 1: 使用最频繁的日期填补  
14. mean_value = female_data['孕妇 BMI'].mean() # 找到最常见的日期  
15. female_data['孕妇 BMI'] = female_data['孕妇 BMI'].fillna(mean_value)  
16. print(female_data['孕妇 BMI'].isnull().sum())  
17.  
18. bins = [25, 30, 35, 40]  
19. labels = ['[25,30)', '[30,35)', '[35,40)'] # 标签  
20. female_data['BMI_group'] = pd.cut(female_data['孕妇  
BMI'], bins=bins, labels=labels, right=False)  
21.  
22. from scipy.stats import spearmanr  
23. for group in labels:  
24.     group_data = female_data[female_data['BMI_group'] == group] # 按照 BMI 组  
筛选数据  
25.     spearman_corr_group = group_data[['年龄', '身高', '体重',  
26.         '检测孕周', '孕妇 BMI', '原始读段数', '在参考基因组上比对的比例', '重复读段  
的比例', '唯一比对的读段数', 'GC 含量',  
27.         '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', 'X 染色体的 Z  
值', 'X 染色体浓度', '13 号染色体的 GC 含量', '18 号染色体的 GC 含量', '21 号染色体的  
GC 含量',  
28.         '被过滤掉读段数的比例']].apply(  
29.             lambda x: spearmanr(x, group_data['染色体的非整倍体  
'])][0], axis=0) # 计算相关性  
30.     print(f"\nBMI 组 {group} 的相关性: ")
```

```

31.     print(spearman_corr_group)
32.
33. feature_columns = ['年龄', '身高', '体重',
34.                     '检测孕周', '孕妇 BMI', '原始读段数', '在参考基因组上比对的比例', '重复读段
   的比例', '唯一比对的读段数', 'GC 含量',
35.                     '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', 'X 染色体的 Z
   值', 'X 染色体浓度', '13 号染色体的 GC 含量', '18 号染色体的 GC 含量', '21 号染色体的
   GC 含量',
36.                     '被过滤掉读段数的比例']
37. feature_columns_1 = ['身高', '体重',
38.                     '检测孕周', '孕妇 BMI', '原始读段数', '在参考基因组上比对的比例', '重复读段
   的比例', '唯一比对的读段数', 'GC 含量',
39.                     '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', 'X 染色体的 Z
   值', 'X 染色体浓度', '13 号染色体的 GC 含量', '18 号染色体的 GC 含量', '21 号染色体的
   GC 含量',
40.                     '被过滤掉读段数的比例', '染色体的非整倍体']
41.
42.
43. bmi_group_25_30 = female_data[(female_data['孕妇 BMI'] >= 25) & (female_data['
   孕妇 BMI'] < 30)]
44.
45. # 获取特征列和目标变量
46. X = bmi_group_25_30[['年龄', '身高', '体重',
47.                     '检测孕周', '孕妇 BMI', '原始读段数', '在参考基因组上比对的比例', '重复读段
   的比例', '唯一比对的读段数', 'GC 含量',
48.                     '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', 'X 染色体的 Z
   值', 'X 染色体浓度', '13 号染色体的 GC 含量', '18 号染色体的 GC 含量', '21 号染色体的
   GC 含量',
49.                     '被过滤掉读段数的比例']] # 特征列
50. Y = bmi_group_25_30['染色体的非整倍体'] # 目标变量, 替换为实际的目标列名
51.
52. # 计算 Spearman 相关性系数和 p-value
53. for feature in feature_columns:
54.     X = bmi_group_25_30[feature] # 单独选择特征列
55.     corr, p_val = spearmanr(X, Y) # 计算相关性和 p-value
56.     print(f"Feature: {feature}")
57.     print(f"Spearman Correlation: {corr}")
58.     print(f"p-value: {p_val}")
59.     print("\n")
60.
61. bmi_group_30_35 = female_data[(female_data['孕妇 BMI'] >= 30) & (female_data['
   孕妇 BMI'] < 35)]
62.
63. # 获取特征列和目标变量

```

```

64. X = bmi_group_30_35[['年龄', '身高', '体重',
65.      '检测孕周', '孕妇 BMI', '原始读段数', '在参考基因组上比对的比例', '重复读段
   的比例', '唯一比对的读段数', 'GC 含量',
66.      '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', 'X 染色体的 Z
   值', 'X 染色体浓度', '13 号染色体的 GC 含量', '18 号染色体的 GC 含量', '21 号染色体的
   GC 含量'],
67.      '被过滤掉读段数的比例']] # 特征列
68. Y = bmi_group_30_35['染色体的非整倍体'] # 目标变量, 替换为实际的目标列名
69.
70. # 计算 Spearman 相关性系数和 p-value
71. for feature in feature_columns:
72.     X = bmi_group_30_35[feature] # 单独选择特征列
73.     corr, p_val = spearmanr(X, Y) # 计算相关性和 p-value
74.     print(f"Feature: {feature}")
75.     print(f"Spearman Correlation: {corr}")
76.     print(f"p-value: {p_val}")
77.     print("\n")
78.
79. bmi_group_35_40 = female_data[(female_data['孕妇 BMI'] >= 35) & (female_data['
   孕妇 BMI'] < 40)]
80.
81. # 获取特征列和目标变量
82. X = bmi_group_35_40[['年龄', '身高', '体重',
83.      '检测孕周', '孕妇 BMI', '原始读段数', '在参考基因组上比对的比例', '重复读段
   的比例', '唯一比对的读段数', 'GC 含量',
84.      '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', 'X 染色体的 Z
   值', 'X 染色体浓度', '13 号染色体的 GC 含量', '18 号染色体的 GC 含量', '21 号染色体的
   GC 含量'],
85.      '被过滤掉读段数的比例']] # 特征列
86. Y = bmi_group_35_40['染色体的非整倍体'] # 目标变量, 替换为实际的目标列名
87.
88. # 计算 Spearman 相关性系数和 p-value
89. for feature in feature_columns:
90.     X = bmi_group_35_40[feature] # 单独选择特征列
91.     corr, p_val = spearmanr(X, Y) # 计算相关性和 p-value
92.     print(f"Feature: {feature}")
93.     print(f"Spearman Correlation: {corr}")
94.     print(f"p-value: {p_val}")
95.     print("\n")
96.
97. significant_feature = ['孕妇 BMI', '检测孕周', '体重', '重复读段的比例', 'GC 含量
   ', '唯一比对的读段数', '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值
   ']
98.

```

```

99. significant_feature_1 = ['孕妇 BMI', '检测孕周', '体重', '重复读段的比例', 'GC 含量', '唯一比对的读段数', '13 号染色体的 Z 值', '18 号染色体的 Z 值', '21 号染色体的 Z 值', '染色体的非整倍体']

100.

101. spearman_corr, p_values = spearmanr(female_data[significant_feature], female_data['染色体的非整倍体'])

102.

103. spearman_corr_with_target = spearman_corr[:, -1] # 获取与目标变量的 Spearman 相关系数

104. p_values_with_target = p_values[:, -1] # 获取与目标变量的 p 值

105.

106. corr_df = pd.DataFrame({
107.     'Feature': significant_feature_1,
108.     'Spearman Correlation': spearman_corr[0, :],
109.     'p-value': p_values[0, :].round(5)
110. })

111.

112. # 计算显著性分数 N_sigi, p 值小于 0.05 的特征
113. corr_df['N_sigi'] = (corr_df['p-value'] < 0.05).astype(int)

114.

115. # 计算高相关性分数 N_high, 相关系数的绝对值大于 0.15 的特征
116. corr_df['N_high'] = (np.abs(corr_df['Spearman Correlation']) >= 0.15).astype(int)

117.

118. # 综合得分 S_i = |r_i| + 0.3 * N_sigi + 0.2 * N_high
119. corr_df['S_i'] = np.abs(corr_df['Spearman Correlation']) + 0.3 * corr_df['N_sigi'] + 0.2 * corr_df['N_high']

120.

121. # 排序并显示特征及其得分
122. sorted_features = corr_df.sort_values(by='S_i', ascending=False)
123. print(sorted_features[['Feature', 'Spearman Correlation', 'p-value', 'N_sigi', 'N_high', 'S_i']])

```

```

1. import matplotlib.pyplot as plt
2. import pandas as pd
3.
4. # 假设 'y' 是目标变量
5. y = female_data['染色体的非整倍体']
6.
7. # 绘制柱状图
8. plt.figure(figsize=(6, 4))
9. y.value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
10. plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置字体为黑体

```

```

11. plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号
12. # 设置标题和标签
13. plt.title('样本类别分布', fontsize=16)
14. plt.xlabel('类别', fontsize=12)
15. plt.ylabel('样本数量', fontsize=12)
16. plt.xticks(ticks=[0, 1], labels=['正常 (0)', '异常 (1)'], rotation=0)
17.
18. # 显示图表
19. plt.show()
20.
21.     class_weights = compute_class_weight('balanced', classes=np.unique(y_train),
22.                                         n, y=y_train)
23.     class_weight_dict = {0: class_weights[0], 1: class_weights[1]}
24. #
25. models = {
26.     '逻辑回归':
27.         '支持向量机':
28.             '决策树':
29.                 '随机森林':
30.                     '朴素贝叶斯': GaussianNB(),
31.                     '集成学习': AdaBoostClassifier(random_state=42, n_estimators=100)
32.     }
33.
34. # 存储结果
35. model_results = {}
36. performance_metrics = np.zeros((len(models), 8)) # [准确率, 精确率, 召回
37.                               率, F1, 特异性, AUC, 平衡准确率, MCC]
38. print(f"\n==== 多种分类模型训练和评估 ===")
39.
40. for i, (model_name, model) in enumerate(models.items()):
41.     print(f"\n--- 模型{i + 1}: {model_name} ---")
42.
43.     try:
44.         # 训练模型

```

```

45.         model.fit(X_train, y_train)
46.
47.         # 预测
48.         y_pred = model.predict(X_test)
49.
50.         # 获取预测概率
51.         if hasattr(model, "predict_proba"):
52.             y_prob = model.predict_proba(X_test)[:, 1]
53.         elif hasattr(model, "decision_function"):
54.             y_prob = model.decision_function(X_test)
55.             # 将决策函数值转换为概率
56.             y_prob = 1 / (1 + np.exp(-y_prob))
57.         else:
58.             y_prob = y_pred.astype(float)
59.
60.         # 计算性能指标
61.         accuracy, precision, recall, f1, specificity, auc, balanced_acc,
62.         mcc = calculate_metrics(
63.             y_test, y_pred, y_prob)
64.         performance_metrics[i, :] = [accuracy, precision, recall, f1, spe
65.         cificity, auc, balanced_acc, mcc]
66.
67.         # 存储模型和结果
68.         model_results[model_name] = {
69.             'model': model,
70.             'predictions': y_pred,
71.             'probabilities': y_prob,
72.             'metrics': [accuracy, precision, recall, f1, specificity, auc
73.             , balanced_acc, mcc]
74.         }
75.
76.         print(f"准确率: {accuracy:.4f}")
77.         print(f"精确率: {precision:.4f}, 召回
78.             率: {recall:.4f}, F1: {f1:.4f}")
79.         print(f"特异性: {specificity:.4f}, AUC: {auc:.4f}")
80.         print(f"平衡准确率: {balanced_acc:.4f}, MCC: {mcc:.4f}")
81.
82.     except Exception as e:
83.         print(f"模型训练失败: {e}")
84.         performance_metrics[i, :] = np.nan
85.         print(f"\n==== 模型性能综合比较 ===")
86.         metric_names = ['准确率', '精确率', '召回率', 'F1 分数', '特异性', 'AUC',
87.         '平衡准确率', 'MCC']

```

```

84.     model_names = list(models.keys())
85.
86.     # 创建性能比较表
87.     performance_df = pd.DataFrame(performance_metrics, columns=metric_names,
88.                                     index=model_names)
89.
90.     print(performance_df.round(4))
91.
92.     # 选择最佳模型（综合考虑 AUC 和平衡准确率）
93.     valid_mask = ~np.isnan(performance_metrics[:, 5]) # AUC 不为 NaN 的模型
94.     if np.any(valid_mask):
95.         composite_score = 0.6 * performance_metrics[:, 5] + 0.4 * performance
96.             _metrics[:, 6] # 0.6*AUC + 0.4*平衡准确率
97.         composite_score[~valid_mask] = -1 # 无效模型得分设为-1
98.         best_model_idx = np.argmax(composite_score)
99.         best_model_name = model_names[best_model_idx]
100.        best_model_data = model_results[best_model_name]
101.
102.        print(f"\n最佳模型: {best_model_name} (综合得
103. 分: {composite_score[best_model_idx]:.4f})")
104.    else:
105.        print("所有模型训练失败")
106.        return
107.
108.    if len(valid_performance) > 0:
109.        metrics_to_plot = ['准确率', 'F1 分数', 'AUC']
110.        metric_indices = [0, 3, 5]
111.
112.        feature_to_plot = 0 # 绘制第一个特征
113.        normal_data = X_clean[y_clean == 0, feature_to_plot]
114.        abnormal_data = X_clean[y_clean == 1, feature_to_plot]
115.
116.        axes[4].hist(normal_data, alpha=0.7, label='正常
117. ', bins=20, density=True)
118.        axes[4].hist(abnormal_data, alpha=0.7, label='异常
119. ', bins=20, density=True)
120.        axes[4].set_xlabel(feature_names[feature_to_plot])
121.        axes[4].set_ylabel('密度')
122.        axes[4].set_title(f'{feature_names[feature_to_plot]}分布比较')
123.        axes[4].legend()

```

```
123.  
124.         for i, (metric_name, metric_idx) in enumerate(zip(metrics_to_plot,  
               metric_indices)):  
125.             axes[7].bar(x + i * width, valid_performance[:, metric_idx], w  
               idth, label=metric_name, alpha=0.8)  
126.  
127.             axes[7].set_xlabel('模型')  
128.             axes[7].set_ylabel('性能分数')  
129.             axes[7].set_title('模型性能对比')  
130.             axes[7].set_xticks(x + width)  
131.             axes[7].set_xticklabels(valid_names, rotation=45)  
132.             axes[7].legend()  
133.  
134.         plt.tight_layout()  
135.         plt.savefig('可视化结果.png', dpi=300, bbox_inches='tight')  
  
plt.show()
```