基于回归分析的 NIPT 检测优化研究

摘要

本研究针对无创产前检测（NIPT）的核心技术难题，结合某地区高 BMI 孕妇的 NIPT 数据集（含年龄、BMI、孕周、染色体浓度及 Z 值等指标），通过分阶段建模与迭代优化，系统解决 Y 染色体浓度关联分析、男胎检测时点优化及女胎异常判定问题，旨在提升 NIPT 准确性并降低临床风险。

问题一中，针对胎儿 Y 染色体浓度与孕妇孕周、BMI 的关联分析，研究采用两阶段建模优化策略。初始构建以孕周、BMI 为自变量的普通线性回归（OLS）模型，但因 Y 染色体浓度为[0,1]区间的比例数据，存在非正态分布特性与预测值超出合理范围的问题，模型拟合优度仅为 0.047，适配性较差；进而引入基于张量积样条的广义加性模型，该模型可灵活捕捉变量间非线性关系，有效解决 OLS 模型缺陷，最终通过显著性检验（如 F 检验、模型拟合优度验证），明确孕周、BMI 与 Y 染色体浓度的量化关联规律，为后续检测时点分析奠定基础。

问题二中，以“Y 染色体浓度≥4%的概率最高”和“检测时点尽早以降低临床风险（早期<中期<晚期）”为双目标。该研究在问题 1 模型基础上，采用 K-Means 聚类法进行 BMI 分组：以 BMI 值为核心聚类特征，辅以 Y 染色体浓度达标（≥4%）的最小孕周数 T 作为关键参考标志，通过肘部法则确定最佳聚类数量 K，结合轮廓系数验证聚类合理性；分组后经方差分析检验组间差异显著性，确保分组有效性；最终输出各组 BMI 区间及对应最佳检测时点，并通过 1000 次随机扰动的蒙特卡洛模拟，量化分析检测误差（如测序失败、浓度测量偏差）对时点选择与风险评估的影响，提升方案稳健性。

问题三中，研究在问题 2 的框架上扩展分析维度，进一步纳入身高、体重、年龄、怀孕次数、生产次数等变量，同时考虑多因素交互作用、检测误差及 Y 染色体浓度达标比例（浓度≥4%的样本占比）。首先对新增数据进行预处理（如异常值剔除、标准化），并明确定义达标比例计算标准；随后结合问题 2 的聚类逻辑与风险评估模型，修正 BMI 分组边界，重新计算每组最佳 NIPT 时点，最终得到潜在风险最小的个性化检测方案，同时分析检测误差对结果的影响程度，进一步增强方案的临床适用性。

问题四中，研究以 21 号、18 号、13 号染色体非整倍体（AB 列标注结果，空白为无异常）为判定目标，综合纳入 X 染色体及目标染色体的 Z 值（Q、R、S、T 列）、GC 含量（P、X、Y、Z 列，反映测序质量）、读段数相关比例（M、N、AA 列，体现测序数据有效率）及 BMI（K 列）等多维度因素，构建多特征判别模型。首先通过方差检验进行特征筛选并剔除冗余指标，精准锁定核心判定因子，形成可操作的女胎异常判定方法，为女胎 NIPT 结果解读提供科学依据，有效降低单一指标误判风险。接着通过判断特征值与异常结果存在的线性或非线性关系，明确最优模型类型。最后在对模型经过大量训练后，得到出一套较为标准化、高可靠性的女胎染色体异常判定体系。

关键词：广义加性模型 K-Means 聚类法 蒙特卡洛模拟 随机森林模拟训练 方差分析检验

一、问题重述

1.1 问题背景

NIPT（无创产前检测）是通过采集母体血液、检测胎儿游离 DNA 片段，以分析胎儿染色体是否异常的产前检测技术，核心目的是早期识别胎儿健康状况，重点针对由 21 号、18 号、13 号染色体浓度异常分别导致的唐氏综合征、爱德华氏综合征、帕陶氏综合征。从检测条件与风险来看，孕妇孕期 10-25 周可检测胎儿性染色体浓度，检测准确性需满足：男胎 Y 染色体浓度≥4%、女胎 X 染色体浓度无异常，否则结果准确性难以保证；胎儿异常发现时间与风险直接相关——12 周以内（早期）发现风险较低，13-27 周（中期）发现风险较高，28 周以后（晚期）发现风险极高，需尽早检测以避免治疗窗口期缩短。

从影响因素与数据基础来看，男胎 Y 染色体浓度与孕妇孕周数、身体质量指数（BMI）紧密相关，目前对孕妇采用简单经验 BMI 分组（如[20,28)、[28,32)等）及统一检测时点的方式，因个体差异（年龄、BMI、孕情等）会显著影响检测准确性；附件提供了某地区以高 BMI 孕妇为主的 NIPT 数据，实际检测中存在测序失败（如检测时点过早、不确定因素）、部分孕妇多次采血检测或一次采血多次检测的情况，需基于该数据解决相关问题。

1.2 问题回顾

问题 1：基于附件 NIPT 数据，分析胎儿 Y 染色体浓度与孕妇孕周数、BMI 等关键指标的相关特性，构建能描述二者关系的数学模型，并对模型的显著性进行检验，以明确模型的可靠性。

问题 2：已知男胎孕妇的 BMI 是影响胎儿 Y 染色体浓度达标时间（即浓度首次≥4%的时间）的主要因素，需针对男胎孕妇群体，对 BMI 进行合理分组，确定每组的 BMI 区间及对应的最佳 NIPT 检测时点（目标是最小化孕妇因胎儿不健康可能面临的潜在风险），同时分析检测误差对分组结果及最佳时点的影响。

问题 3：考虑到男胎 Y 染色体浓度达标时间还受身高、体重、年龄等多因素影响，结合检测误差及胎儿 Y 染色体浓度达标比例（即浓度≥4%的比例），仍以 BMI 为分组依据，对男胎孕妇进行合理分组，给出每组的最佳 NIPT 时点以最小化潜在风险，并进一步分析检测误差对该结果的影响。

问题 4：由于孕妇与女胎均不携带 Y 染色体，需以附件数据中 AB 列（13 号、18 号、21 号染色体非整倍体判定结果，空白代表无异常）为核心依据，综合 X 染色体 Z 值、各染色体（13 号、18 号、21 号、X 染色体）GC 含量、读段数相关指标（总读段数、比对比例、重复读段比例、过滤读段比例等）及孕妇 BMI 等因素，提出女胎染色体异常的判定方法[1]。

二、问题分析

2.1 对问题的总体分析与解题思路

2.1.1 总体分析

NIPT 问题核心围绕胎儿染色体检测准确性与检测时点优化展开，需结合孕妇个体指标（孕周、BMI、年龄等）与检测数据（染色体浓度、Z 值等），分场景解决关联建模、

分组优化及异常判定问题。其中，男胎相关问题聚焦 Y 染色体浓度达标规律，以“早检测+高准确率”为目标，逐步从双因素关联扩展到多因素综合优化；女胎问题则转向染色体非整倍体判定，需基于多维度检测特征构建分类标准，整体需兼顾数据特性（如比例数据、非线性关联）与临床风险（检测时点早晚影响）。

2.1.2 解题思路

基础关联建模：先以 OLS 模型初步探索孕周、BMI 与 Y 染色体浓度的线性关系，发现数据非正态、预测出界及拟合度低的问题后，改用张量积样条广义加性模型，适配非线性关联，同时检验模型显著性，奠定后续分析的数据关联基础。

单因素分组与时点优化：以“Y 染色体浓度≥4%概率最高+检测时点最早”为目标，基于问题 1 的关联规律，用 K-Means 聚类（以 BMI 为核心、达标最小孕周为辅）结合肘部法则与轮廓系数确定分组，经方差分析验证分组有效性，再通过蒙特卡洛模拟分析检测误差影响，输出每组最佳 NIPT 时点。

多因素综合优化：在问题 2 框架上扩展，纳入身高、体重、年龄等个体指标，先预处理数据补充多维度特征并定义达标比例，再考虑多因素交互作用，沿用分组与误差分析逻辑，优化分组及最佳检测时点，进一步降低临床潜在风险。

女胎异常判定：针对女胎无 Y 染色体的特性，以 21、18、13 号染色体非整倍体（AB 列）为目标，先预处理数据（补全核心指标、剔除异常值、转化二元标签），再经特征工程与关键特征识别，通过可视化判断特征与异常结果的非线性关联，最终训练模型形成判定方法。

2.2 问题一分析

问题一围绕胎儿 Y 染色体浓度与孕妇孕周、BMI 等指标的关联展开建模，过程分为两阶段迭代优化。

首先我们选择构建普通线性回归（OLS）作为基础模型，以孕周、BMI 为自变量，但因 Y 染色体浓度（[0,1]比例数据）非正态且存在预测出界问题，拟合优度仅 0.047，适配性差，所以我们进一步考虑非线性回归模型。

根据对非线性模型的综合考量，我们最终决定更换更为适合的基于张量积样条的广义加性模型来解决胎儿 Y 染色体浓度与孕妇的孕周数和 BMI 等指标之间的关系。

2.3 问题二分析

根据题意，本题目标为找到最佳检测孕周，为达到目标需对 BMI 合理分组并确定每组最佳检测时点 t，使孕妇在该时点 Y 染色体浓度时达到 4%的概率最高，且 t 尽可能早以降低临床风险。

在问题一的基础上，我们首先对 BMI 进行合理分组，为确保准确性，我们使用 K-Means 聚类法，将 BMI 值作为主要聚类特征，辅以 Y 染色体浓度≥4%的最小孕周数 T 作为检测标志，通过肘部法则与轮廓系数确定最佳聚类数量 K 进行聚类，然后通过方差分析验证分组的有效性，并通过蒙特卡洛模拟（1000 次随机扰动）分析误差影响。

2.4 问题三分析

经过深入研究与分析，我们发现问题三实质上是对问题二的扩展与深化，在问题二的基础上，我们再次加入身高、体重、年龄等因素进行综合考量[2]而且同步考虑多因素交互作用、检测误差及达标比例对结果的影响。

首先，对数据进行预处理，新增身高，体重，年龄，怀孕次数，生产次数等数据，接下来根据题意要求定义达标比例。

最后，根据问题二的分析以及计算，我们成功得到了使得孕妇潜在风险最小的合理分组以及每组的最佳 NIPT 时点，并分析了检测误差对结果的影响。

2.5 问题四分析

通过对题目的分析，很明显能够看出问题四侧重点与问题一二三的差异较大，即该问题聚焦于解决**女胎**异常的问题。

由题可知，该问题的核心目标为综合考虑各个染色体的 Z 值[3]、GC 含量、读段数及相关比例、BMI 等因素[4]，构建模型用于判定女胎是否存在 21 号、18 号和 13 号染色体非整倍体（AB 列）的异常（即唐氏综合征、爱德华氏综合征和帕陶氏综合征）。

首先，我们对女胎样本进行数据预处理，需要保证所有核心指标数据完整且剔除掉异常数据并将 AB 列的结果转化为三个二元标签。

接下来，我们对各类指标进行特征工程加工，其中由于特征较多我们还需进行关键特征识别。

然后，我们绘制特征在两组中的散点图与箱线图来判断特征值与异常结果存在线性关联还是非线性关联，进而确定接下来要使用的模型。

在将模型判断为非线性相关模型后，我们综合考虑各种因素并通过对模型的训练最终得到出一套较为标准的女胎异常的判定方法[5]。

三、模型假设

为了方便模型的建立与模型的可行性，我们这里首先对模型提出一些假设，使得模型更加完备，预测的结果更加合理：

1. 附件中多次采血多次检测或一次采血多次检测的样本，其检测条件（如检测仪器精度、检测人员操作规范、样本储存环境等）一致，检测结果差异仅由胎儿染色体浓度自然波动。
2. 孕妇孕期计算以“末次月经时间”为基准推导的“检测时孕周（周数+天数）”准确无误，无孕周记录偏差或计算错误，且孕周推进过程中无异常情况（如胎儿发育停滞、孕周与实际发育不符等）影响染色体浓度变化规律。
3. 女胎无 Y 染色体，其 X 染色体浓度及 21 号、18 号、13 号染色体相关指标（Z 值、GC 含量、读段数等）的变化，仅与胎儿自身染色体状态（正常/异常）及检测误差相关，不受 Y 染色体相关因素干扰。
4. 孕妇 BMI 指标在检测期间保持稳定，无因孕期体重急剧变化（如短期内体重大幅增加或减少）导致的 BMI 分组变动，确保 BMI 对 Y 染色体浓度及 NIPT 时点的影响具有一致性。
5. 附件中提供的某地区孕妇 NIPT 数据为有效数据，无缺失关键信息（如孕周、BMI、染色体浓度、Z 值等核心指标）的无效样本，且数据记录真实可靠，无人工录入错误或数据篡改情况。
6. 孕妇年龄、身高、体重、怀孕次数、生产次数、IVF 妊娠方式等因素对男胎 Y 染色体浓度的影响，均通过与 BMI 的关联或独立作用体现，无其他未提及的隐性因素（如孕妇基础疾病、生活习惯等）对 Y 染色体浓度产生显著干扰。
7. 检测误差（如测序失败、读段数统计偏差等）服从正态分布，且误差大小与检

测时间、BMI 分组、染色体类型等因素无系统性关联，可通过统计方法量化并修正。

8. 女胎 X 染色体浓度的负值（题目提及可能出现）是生物信息学估计过程中的合理结果，仅反映估计值的波动，不代表实际染色体浓度为负，且该负值可通过数据预处理（如标准化、修正）纳入异常判定模型，不影响判定逻辑。

9. 孕妇“胎儿是否健康（婴儿出生后的结果）”（AE 列）为胎儿健康状态的金标准，无出生后诊断误差，可作为验证染色体异常判定模型准确性的可靠依据。

四、名词解释与符号说明

4.1 名词解释：

1.唐氏综合征、爱德华氏综合征和帕陶氏综合征[6]:染色体数目异常疾病，核心病因是生殖细胞分裂时染色体不分离，导致胎儿多一条特定染色体（常染色体），进而引发全身多系统发育异常。

2.身体质量指数（BodyMassIndex，简称 BMI）[7]:是国际上量人体胖瘦程度以及是否健康的一个常用指标。**（计算公式 BMI=体重（kg）/身高²（m²））**

3.治疗窗口期[8]:医学领域中一个关键概念，指从疾病发生、病原体感染或症状出现，到开始治疗的特定时间段。在这个窗口期内启动规范治疗，能最大限度降低疾病进展风险、减少并发症，并显著提升治疗效果；若超出窗口期，治疗难度会增加，疗效可能大幅下降，甚至错过最佳干预时机。

4.2 符号说明：

为了方便我们模型的建立与求解过程，我们这里对使用到的关键符号进行以下说明：

表 1 符号说明

符号	符号说明	单位
V	胎儿 Y 染色体浓度	\
J	孕妇的孕周数	\
ϵ	随机误差项	\
ω	权重	\
S	标准差	\
t	孕周	w
T	最小孕周数	w

（注：这里只列出论文各部分通用符号，个别模型单独使用的符号在首次引用时会进行说明。）

五、模型的建立与求解

5.1 数据清洗

5.1.1 异常值判定与处理

对于问题一，我们需要解决的问题是胎儿 Y 染色体浓度与孕妇的孕周数和 BMI 等指标的相关特性，所以根据题意：通常孕妇的孕期在 10 周~25 周之间可以检测胎儿性染色体浓度，我们对不符合要求的孕期数值进行剔除，随后可得问题所需的可靠数据。

对于问题三，为判断 X 染色体及上述染色体的 Z 值、GC 含量、读段数及相关比例、BMI 等因素对女胎异常的影响，我们要对上述数据中异常值进行剔除。

5.1.2 缺失值寻找与处理

在对附件数据开展清洗工作时，我们首先对缺失值的效用进行了系统性探究与评估。经分析发现，该数据集中的缺失值不具备补充挖掘价值，其存在不仅无法为后续数据分析提供有效支撑，还可能增加数据处理复杂度、干扰分析结果准确性。因此，基于“去无效数据、保数据质量”的清洗原则，我们最终采用删除法对该部分缺失值进行了处理，以确保留存数据的完整性与可用性，为后续分析工作奠定可靠基础。

5.2 问题一模型的建立与求解

5.2.1 问题一模型的建立

构建胎儿 Y 染色体浓度（记为 V）与孕妇的孕周数（记为 J）和 BMI（记为 K）等指标的关系模型并检验显著性，使用多元线性回归模型（MLR）作为基础模型，若线性拟合不佳考虑非线性回归模型或分段多项式模型。使用 t 检验，检验参数显著性；使用 F 检验，检验模型显著性，最后计算拟合优度R^2。

针对问题一首先对数据进行清洗预处理，由题目要求可知分析胎儿 Y 染色体浓度首先筛选男胎样本，然后处理异常值，其中对于孕周数为了更准确的对数据进行处理我们将孕周数进行转化（例如 23w+3 转化为 23.429w 保留小数点后三位）将孕周数不满十周与超出 25 周的数据进行剔除。

然后绘制散点图矩阵观察分析 V 与 J 和 K 的关系，并判断模型的所属类型，进而选择更合适的模型。

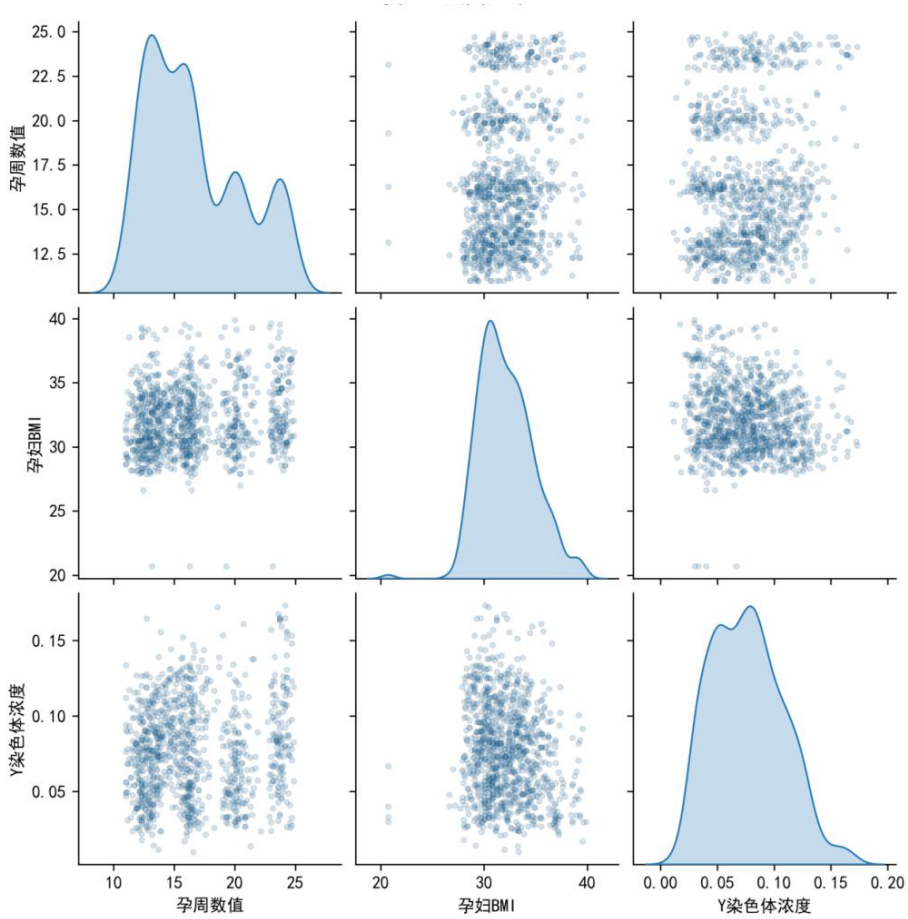

图 1 散点图矩阵

由上图可知：

V 与 J 和 K 的关系并无明显线性关系，所以我们初步认为将问题归为线性回归模型并不合理。

所以我们进行了进一步的数值检验（t 检验）：

对参数进行 t 检验

原假设和备择假设为：

H_0：β_j=0（该自变量对因变量无显著影响）

H_1：$\beta_j \neq 0$（该自变量对因变量有显著影响）

统计量计算公式：$t_j = \frac{\widehat{\beta_J}}{SE(\widehat{\beta_J})} \sim (n-p-1)$

其中$\widehat{\beta_J}$：第 j 个参数的估计值

$SE(\widehat{\beta_J})$：参数估计值的标准误差

n 为样本量，p 为自变量个数

标准误差的公式$SE(\widehat{\beta_J}) = \sqrt{\hat{\sigma}^2 (X^T X)^{-1}_{jj}}$

其中$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-p-1}$

$(X^T X)^{-1}_{jj}$：矩阵第 j 个对角线元素

若 P 值<显著性水平 α （0.05），则拒绝原假设，认为该自变量显著

我们通过对 P 值的计算，排除了线性回归模型的情况。

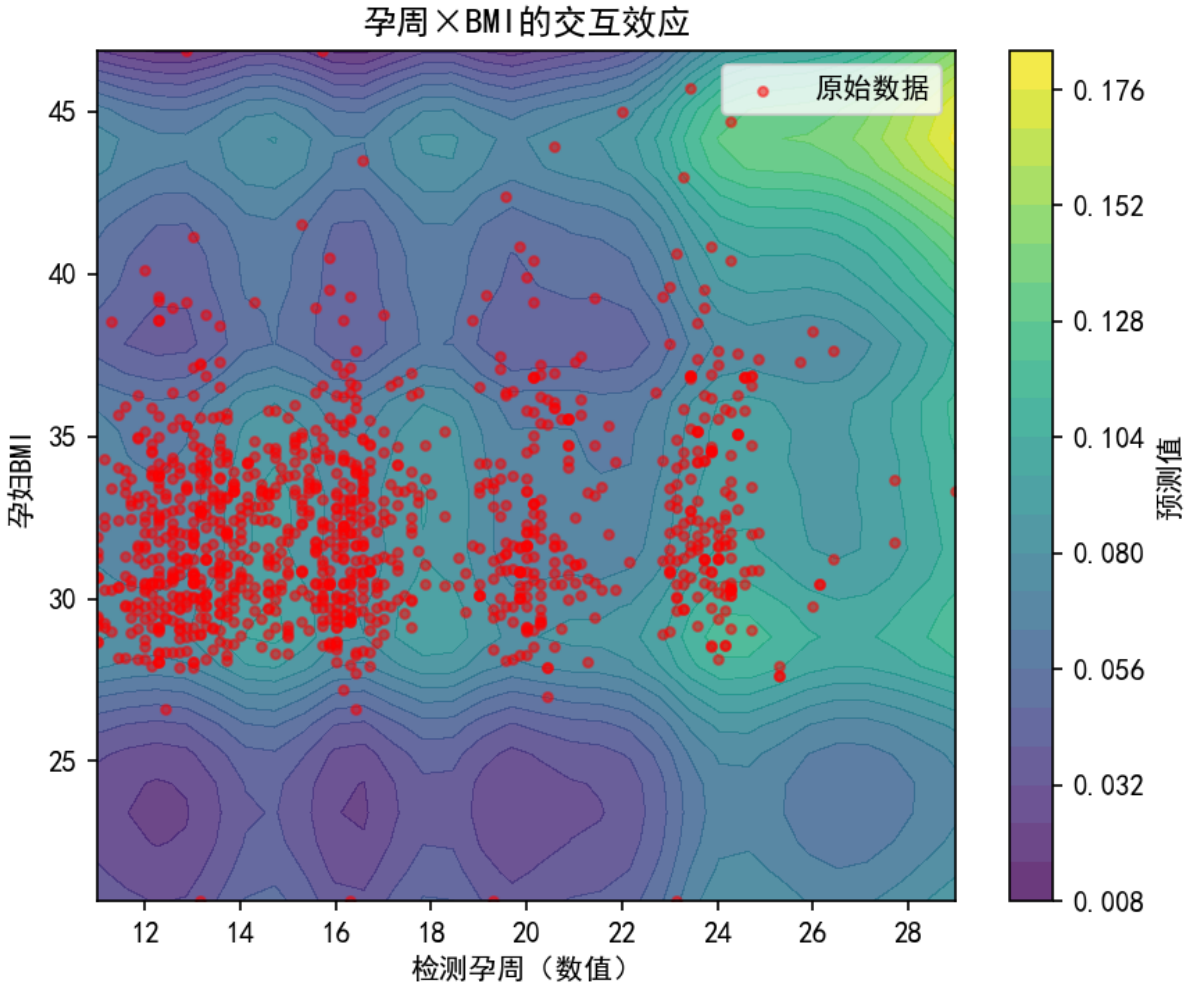

图 2 孕周与 BMI 的交互效应

“孕周×BMI 的交互效应”等高线图显示，孕周和 BMI 的组合对 Y 浓度有复杂的联合影响（不同孕周与 BMI 的组合对应不同的预测值区间，颜色/等高线分布无“独立叠加”的规律），说明两者存在交互作用，且交互模式也为非线性。

因此线性回归无法准确刻画这些复杂关系，不考虑使用。

为了更灵活捕捉“孕周的非线性”“BMI 的非线性”，以及两者的“非线性交互效应”

最终我们采用基于张量积样条的广义加性模型

模型公式为：$V = \beta_0 + te(J，K) + \epsilon$

其中：

β_0为模型截距

te(孕周，BMI)为孕周与 BMI 的张量积样条交互项

ϵ为随机误差项

根据上述模型对数据拟合曲线如下所示：

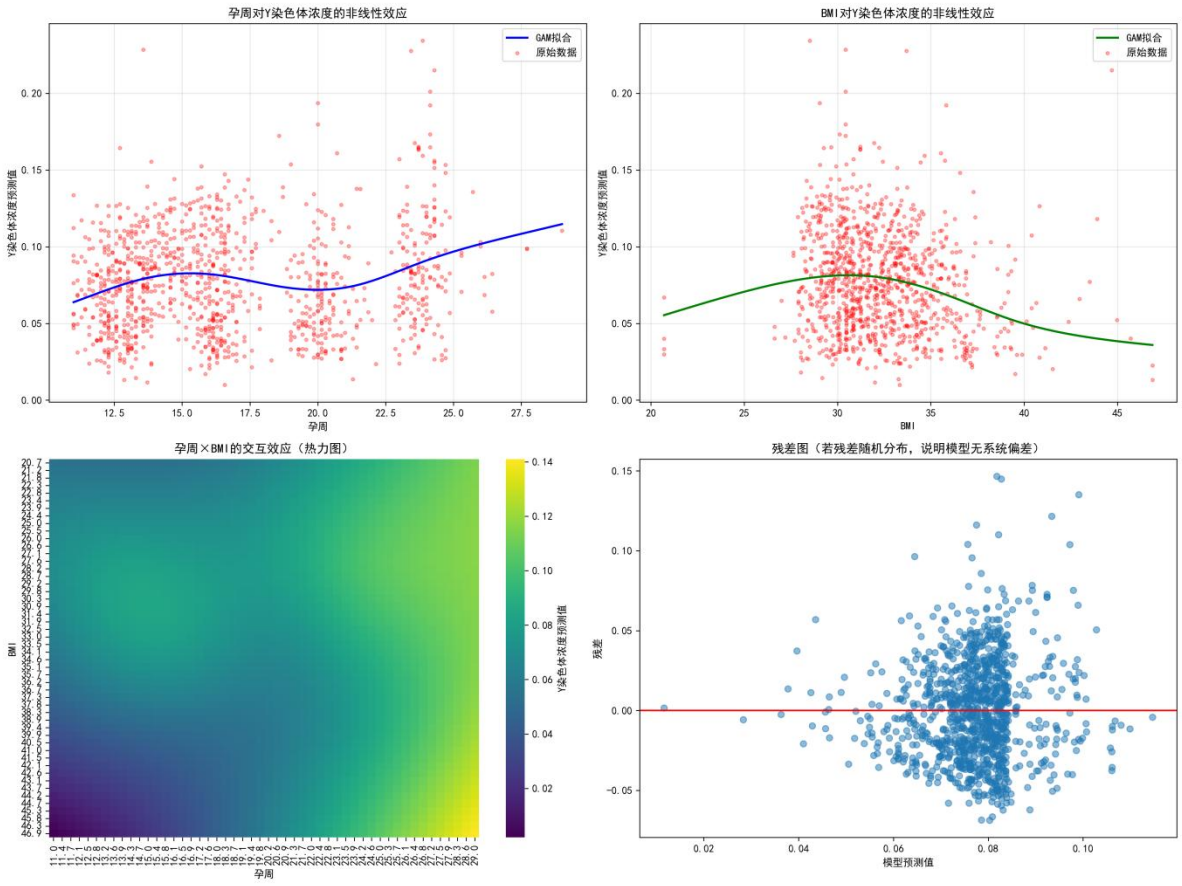

图 3 数据拟合曲线

对模型进行 F 检验

原假设和备择假设为：

H_0：$\beta_1 = \beta_2 = \cdots \ldots = \beta_p = 0$(所有自变量对因变量均无解释力)

H_1：至少有一个$\beta_j \neq 0$（模型至少有一个自变量是显著的）

计算公式：$\frac{SSR/p}{SSE/(n-p-1)}$

回归平方和（SSR）$\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$

残差平方和（SSE）$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

若 P 值<显著性水平 α，则拒绝原假设，认为该自变量显著

5.2.2 问题一结果

采用基于张量积样条的广义加性模型能够较为精准地表示胎儿 Y 染色体浓度与孕妇的孕周数和 BMI 等指标的相关特性，并成功检验其显著性。

5.3 问题二模型的建立与求解

5.3.1 模型的建立与求解

在问题一的基础上，我们要对 BMI 进行合理分组并给出每组的 BMI 区间和最佳 NIPT。

首先我们对 BMI 进行合理分组，为保证分组的准确性，我们使用 K-Means 聚类法来确保实现组内差距小，组间差距大这一情形。

我们将 BMI 值作为主要聚类特征，辅以 Y 染色体浓度≥4%的最小孕周数 T（由问题一的 Y 染色体浓度-孕周曲线可求得）作为检测标志，通过肘部法则与轮廓系数确定最佳聚类数量 K 进行聚类。分组见表 2

表 2 BMI 分组情况

BMI 分组	BMI 区间	样本数	占比	临床意义
0	[26.61934339, 31.4819031897525]	480	0.4524033930254477	早期达标（Y≥4% 最快）
1	[31.52539657, 34.93333333]	420	0.3958529688972667	中期达标（BMI 中等，需延迟
2	[34.9804883731461, 41.1328125]	161	0.1517436380772856	高 BMI 延迟达标（需最晚检测）

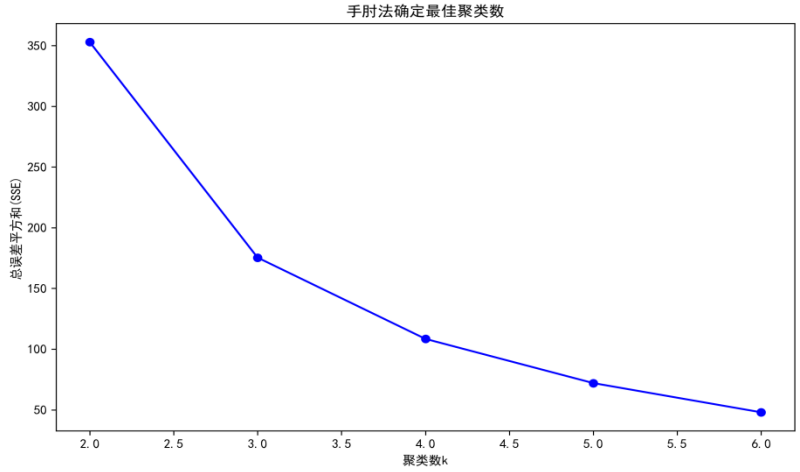

图 4 k 与 SSE 关系的折线图

然后，我们进行分组有效性的校验，在这里，我们使用了方差分析的方法：原假设H_0：各组的 T 无显著性差异，若 P<0.05，则说明分组有效

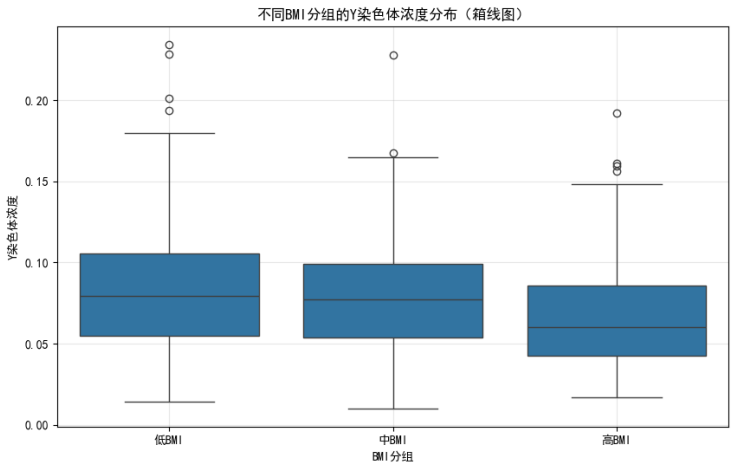

图 5 不同 BMI 分组的 Y 染色体浓度分布箱线图

通过计算，我们得出 P 值为：3.1245e-08<0.05,检验了分组的有效性。

接下来我们确定每组的最佳 NIPT 时点使风险最小化。

根据题目中给出的信息“早期发现（12 周以内）风险较低；中期发现（13－27 周）
风险高；晚期发现（28 周以后）风险极高。”我们定义风险值R

$$R(t)=\begin{cases}1\times(1-P(t))+1\times P(t) & t\le 12\\ 5\times(1-P(t))+5\times P(t) & 13<t\le 27\\ 10\times(1-P(t))+10\times P(t) & t>27\end{cases}$$

其中，P(t)为该 BMI 组在孕周 t 时 Y 浓度≥4%的概率,采用 Logistic 回归拟合达标概率曲线。

$P(t)=\frac{1}{1+e^{-(a+bt)}}$(a,b 为回归系数通过极大似然估计值进行求解)

对每组 BMI 群体从 10 周到 25 周进行遍历选择 R(t)最小的 t 作为最佳 NIPT 时点。

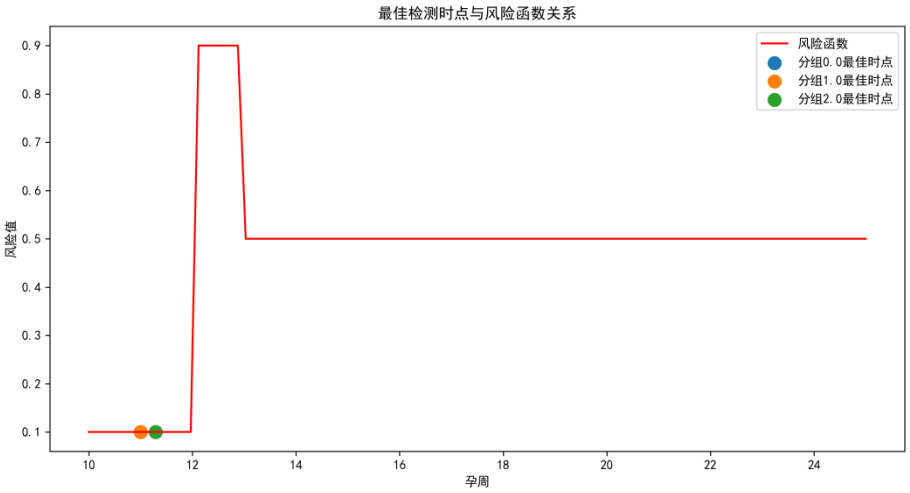

图 6 最佳检测时间与风险函数的折线图

表 3 各组最早达标时间及最佳检测时点

BMI 分组	最早达标时间(周	最佳检测时点(周	对应风险值
0	11	11	0.1
1	11	11	0.1
2	11.29	11.29	0.1

最后，我们进行误差分析，通过蒙特卡洛模拟（1000 次随机扰动）分析误差影响。

对原始 Y 染色体浓度添加±5%的相对误差：$V_{扰动}=V_{原始}\times(1+\delta)$

其中δ为随机误差项，$V_{原始}$为附件中所给数据

接下来对 1000 次模拟计算以下统计量：

（1） 模拟平均达标时间：

$$\overline{T_{模拟}} = \frac{1}{n}\sum_{i=1}^{n} T_{模拟，i}(n=1000)$$

（2） 模拟标准差：

$$\sigma_{模拟} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(T_{模拟，i} - \overline{T_{模拟}})^2}$$

表 4 统计结果汇总情况

所有分组模拟统计结果汇总		
BMI 分类\测量值	模拟平均达标时间	模拟标准差
低 BMI	11.00000	0.000000
中 BMI	11.00014	0.004425
高 BMI	11.29084	0.010812

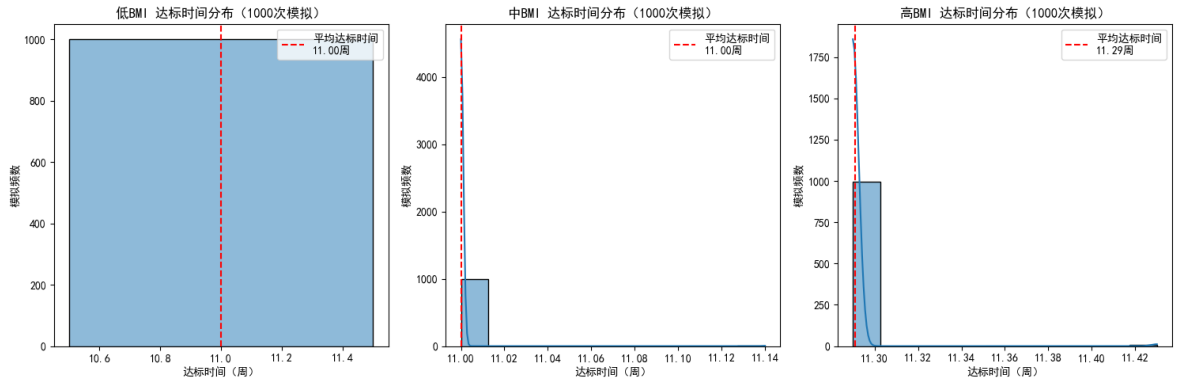

图 7 不同 BMI 的达标时间分布

根据代码运行结果以及达标时间分布图，对于低 BMI 分组模拟平均达标时间为 11 周，模拟标准差几乎为 0。在分布图中。我们可以看出达标时间集中在 11 周，说明相对误差对低 BMI 组的结果几乎没有影响，稳定性好可靠性高。

对于中 BMI 分组来说模拟平均达标时间为 11.00014 周，模拟标准差为 0.004425，从分布图来看达标时间基本集中在 11 周有一定的离散性，但是离散程度小，说明相对误差对中 BMI 组的结果有轻微影响，稳定性较好。

对于高 BMI 分组来说模拟平均达标时间为 11.29084 周，模拟标准差为 0.010812，从分布图来看达标时间明显偏离 11 周且离散程度大，说明相对误差对高 BMI 组的结果有显著影响且稳定性低。

5.3.2 问题二结果

在根据问题要求进行合理分组后，我们将男胎孕妇的 BMI 分为三组，编号分别记为 0，1，2，通过计算，我们得出 0 组与 1 组的最佳 NIPT 时点为 11w,2 组最佳 NIPT 时点为 11.29w，最后通过蒙特卡洛模拟，进一步分析了检测误差对结果的影响。

5.4 问题三模型的建立与求解

5.4.1 模型的建立与求解

首先，我们新增变量：身高（记为D）、体重（记为E）、年龄（记为C）、怀孕次数（记为AC）、生产次数（记为AD），并对其对应的数据进行处理。接下来，我们基于D与E计算 BMI 使其与附件中交叉验证确保一致性并将不合理的数据进行清洗（详情见附件“问题三数据处理”中“预处理后数据”）。

接下来，对每个孕妇统计“Y 染色体浓度≥4%的检测次数/总检测次数”，作为个体层面的达标比例。

很显然，本问题具有多个变量，所以我们应该对关键特征进行筛选，通过随机森林模拟训练输出各个特征对T的重要性得分并保留得分前三的特征作为聚类输出。

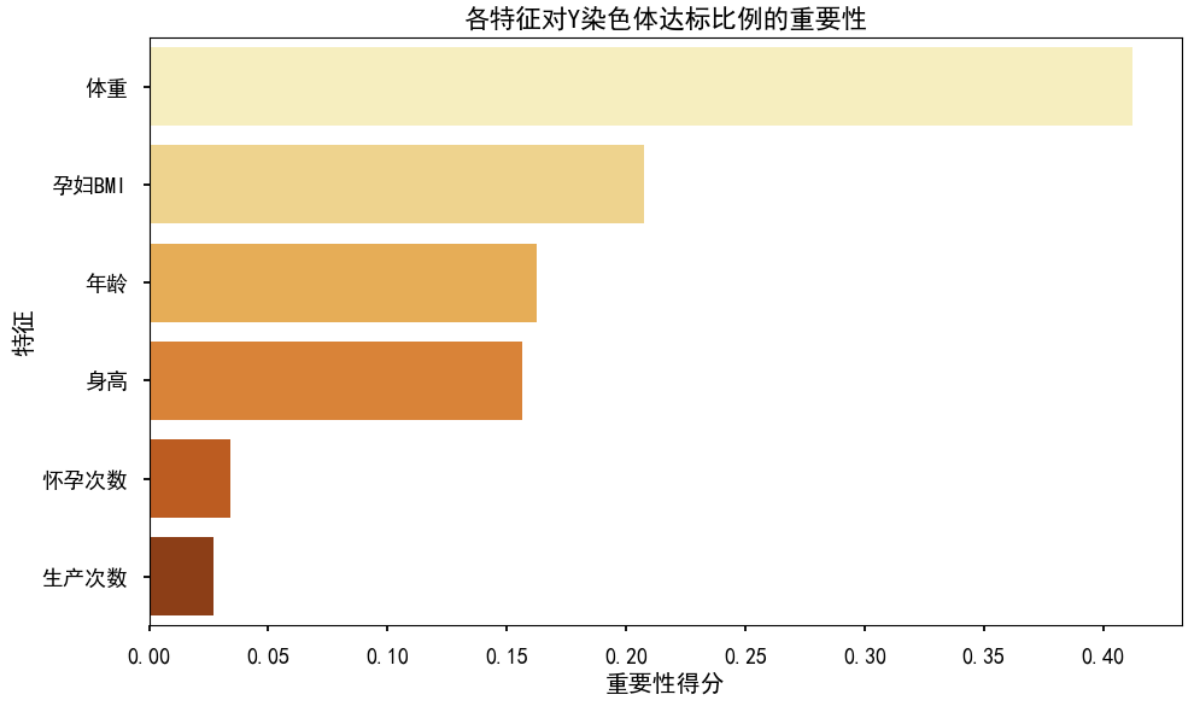

图 8 各特征对 Y 染色体达标比例重要性的柱形图

表 5 各指标的重要性得分情况

特征	重要性得分
体重	0.412418814
孕妇 BMI	0.207777981
年龄	0.162627905
身高	0.1563145
怀孕次数	0.033903305
生产次数	0.026957495

根据筛选出的特征进行 K-Means 聚类分组，我们依旧选择使用肘部法则和轮廓系数进行聚类数的选择

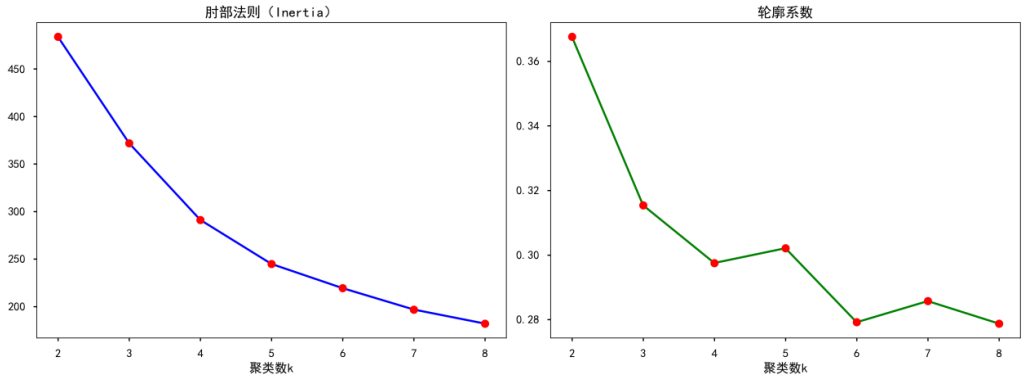

图 9 肘部法则与轮廓系数的折线图

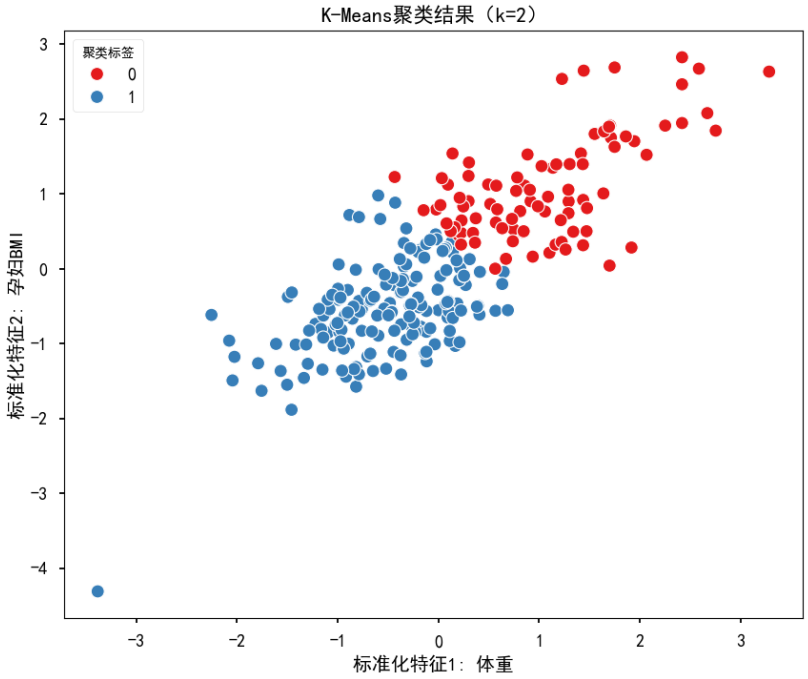

图 10 K-Means 聚类结果

（详情见附件“问题三数据处理”中“聚类结果数据”）。

接下来，我们采用多元方差分析验证分组的有效性：

Wilks' Lambda: 0.3502

F 统计量: 80.1061

p 值: 0.0……01(趋近于 0)

根据题目的相似性以及连接关系，我们风险分级沿用问题二：

$$R(t)=\begin{cases}1\times(1-P(t))+1\times P(t) & t\le 12\\ 5\times(1-P(t))+5\times P(t) & 13<t\le 27\\ 10\times(1-P(t))+10\times P(t) & t>27\end{cases}$$

拟合记为$P_{reach}(t)$

$P_{reach}(t)=at^2+bt+c$(a,b,c 为回归系数，使用最小二乘法进行求解)

其中，为平衡风险与达标比例，我们构建加权目标函数$F(t)$

$F(t)=\omega_1\times R(t)+\omega_2\times(1-P_{reach}(t))$（$t\in[10,25]$）

对于权重的选择我们采用层次分析法将风险权重（ω_1）定为 0.6，达标比例权重（ω_2）定为 0.4 旨在切实贴合题目中所强调的“使得孕妇潜在风险最小”的要求。

最后在t ∈ [10,25]的条件下对孕周数以步长为 0.1 周进行遍历找到使F(t)达到最小的 t 值。

表 6 BMI 分组

聚类组	最佳时点(周)	目标函数值	该时点风险等级	该时点达标比例
0	10	0.623	1	0.9424
1	10	0.632	1	0.92

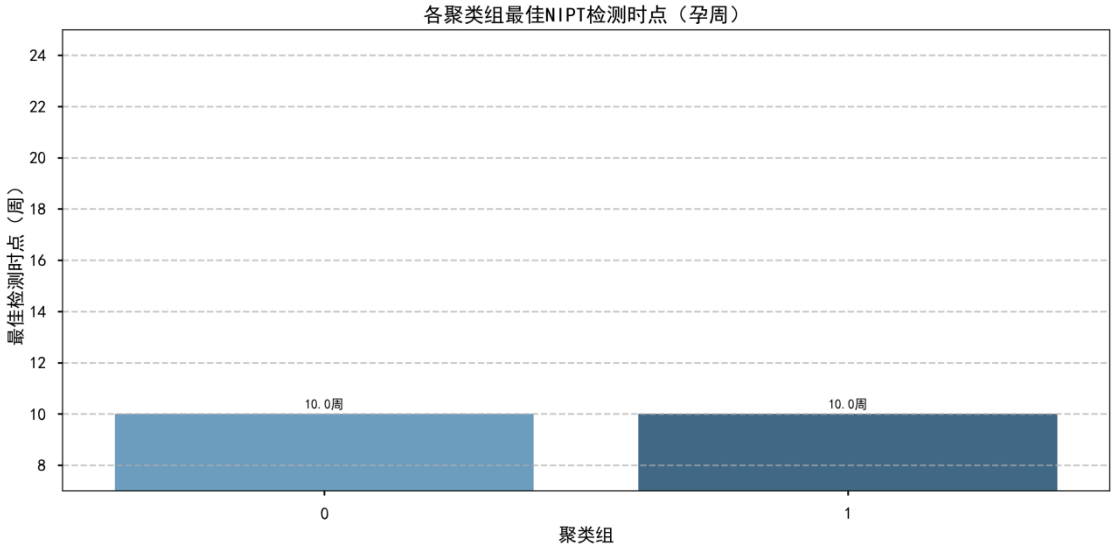

图 11 最佳检测时点

最后进行误差分析:

首先设置各个特征的合理波动区间：

a.年龄：±8岁

b.体重：±10kg

c.BMI：$\pm 5\ \mathrm{kg/m^2}$

接下来，采用拉丁超立方采样抽取 500 组“因素-误差”组合

重新计算每组组合的T、$P_{reach}(t)$与最佳时点

通过 Sobol 指数量化各个因素的误差对最佳时点选择的影响程度（Sobol 指数越大影响程度越大）

表 7 Sobol 指数分析

聚类组	特征	Sobol 指数
0	年龄	0.232142857
0	体重	0.214285714
0	孕妇 BMI	0.553571429
1	年龄	0.232142857
1	体重	0.214285714
1	孕妇 BMI	0.553571429

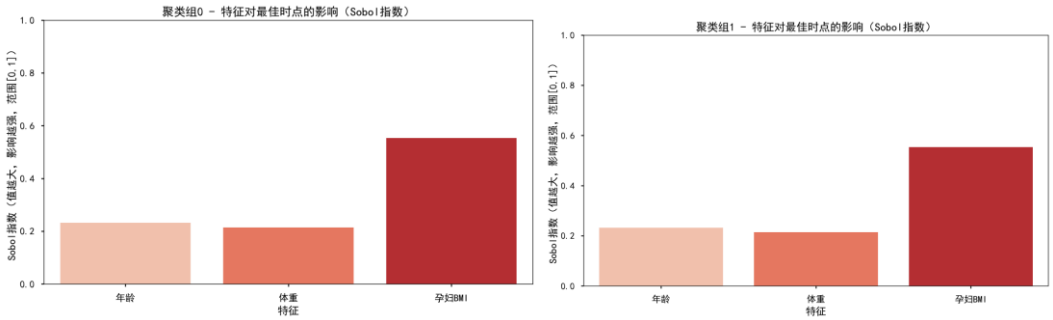

图 12 特征对最佳时点的影响

从给出的 Sobol 指数表格来看，无论是聚类 0 还是聚类组 1，孕妇 BMI 的 Sobol 指数均为 0.553571429，明显高于年龄（0.232142857）和体重（0.214285714）的 Sobol 指数。这说明，在问题三研究的 NIPT 最佳检测时点选择中，孕妇 BMI 是对最佳时点选择影响最大的因素。相比之下，年龄和体重对最佳检测时点选择的影响程度相对较小。

5.4.2 问题三结果：

根据上述问题分析，我们将 BMI 进行合理分组（见表 6）并确定了最佳 NIPT 时点为 10 周，并使用拉丁超立方采样和 Sobol 指数量化分析了误差影响。

5.5 问题四模型的建立与求解

5.5.1 模型的建立与求解

由题可知，问题四的核心目标为综合考虑各个染色体的 Z 值、GC 含量、读段数及相关比例、BMI 等因素，构建模型用于判定女胎是否存在 21 号、18 号和 13 号染色体非整倍体（AB 列）的异常（即唐氏综合征、爱德华氏综合征和帕陶氏综合征）

首先我们需要对女胎样本进行数据预处理，保证所有核心指标数据完整且剔除掉异常数据，根据题目中的附录可知正常 GC 含量范围为 40%~60%，其中 GC 含量过高、过低、或分布异常都可能表明测序质量存在问题。因此需要清洗掉不在范围内的数据，通过对表格中 AB 列进行分析，我们知道胎儿只有患病与不患病两种状态，所以我们将 AB 列的结果转化为三个二元标签（0=无异常，1=有异常）。

表 8 AB 列结果的转化

原始 AB 列结果	拆解后标签（1 = 异常，0 = 正常）	疾病类型
空白	Y13=0, Y18=0, Y21=0	无异常
T21	Y13=0, Y18=0, Y21=1	单一疾病（仅 T21）
T18	Y13=0, Y18=1, Y21=0	单一疾病（仅 T18）
T13	Y13=1, Y18=0, Y21=0	单一疾病（仅 T13）
T21+T18	Y13=0, Y18=1, Y21=1	合并疾病（T21+T18）
T21+T13	Y13=1, Y18=0, Y21=1	合并疾病（T21+T13）
T18+T13	Y13=1, Y18=1, Y21=0	合并疾病（T18+T13）

接下来，我们对题目中所提及到的各类型指标进行特征工程加工：

(1)染色体的 Z 值：

原始指标：21 号（记为 S）、18 号（记为 R）、13 号（记为 Q）和 X 染色体（记为 T）

特征加工方式：直接保留原始值即可

原因及目的：Z 值已经标准化，由题目中的附件的公式可知 Z 值越偏离 0.非整倍体的概率越高

(2)GC 含量：

原始指标：21 号（记为 Z）、18 号（记为 Y）和 13 号（记为 X）

特征加工方式：$GC_{dev,i} = |GC_i - 50\%|$(i=13,18,21)

原因及目的：通过中点偏差将是否在正常范围的定性判断，转化为偏离正常中心程度的定量数据，能够更直观的反映测序数据的偏移程度

(3)读段数及相关比例：

原始指标：比对比例（记为 M）、重复读段比例（记为 N）、唯一比对读段数（记为 O）

特征加工方式：唯一比对读段占比：$Ratio_{unique} = \frac{O}{L}$;

重复读段占比：$Ratio_{repeat} = N$;

比对比例:$Ratio_{map} = M$

原因及目的：唯一比对读段占比反应原始数据中可精准定位的读段占比；重复读端占比反应测序数据的冗余程度，数值越大冗余程度越大；比对比例反应原始测序数据的有效利用率

（4）孕妇自身特征：

原始指标：BMI（记为 K）、孕周（记为 J）、年龄（记为 C）

特征加工方式：对其进行 Z-score 标准化

原因及目的：加快模型收敛速度，矫正对检测结果的干扰

（注：预处理后数据见附件“问题四数据处理”）

从上述分析中，我们发现特征因素数目较多，因此我们进行关键特征识别以便处理这些特征因素。

首先通过方差分析检验特征与结果是否存在显著关系，对每个特征与标签（0/1）做方差分析，保留$P < 0.05$的特征即保留结果强关联的特征。但是方差分析仅能筛选出关联特性，无法具象化准确的关系，所以我们辅以互信息方法筛选出信息贡献度高的特征。在这里，我们设定互信息阈值为$MI_{阈值} = MI_{Z-min} \times 0.5$。信息贡献度高的特征筛选结果见表 9

表 9 筛选结果

13 号染色体异常	18 号染色体异常	21 号染色体异常
Z18	Z18	GC13_偏差
Z21	Z21	唯一比对的读段占比
ZX	GC13_偏差	比对比例
唯一比对的读段占比	BMI_标准化	年龄_标准化
比对比例		孕周_标准化
BMI_标准化		

接下来将预处理后的样本划分为训练集和测试集，采用分层抽样确保两组的异常组占比与原样本一致，然后对最终筛选的核心特征采用 Z-score 标准化$X_{std} = \frac{X-\bar{X}}{S}$

其中：X为特征原始值，$\bar{X}$为该特征在训练集的均值，S为该特征在训练集里的标准差。

接下来绘制特征在两组中的散点图与箱线图来判断特征值与异常结果存在线性关联还是非线性关联，进而确定接下来要使用的模型：这里我们分为两种情况来考虑。

若为线性相关选择逻辑回归模型，则公式$\mathrm{P}(\mathrm{y}=1|\mathrm{x}_1,\mathrm{x}_2,\ldots,\mathrm{x}_\mathrm{k})=\frac{1}{1+\mathrm{e}^{-(\beta_0+\beta_1\mathrm{x}_1+\ldots+\beta_\mathrm{k}\mathrm{x}_\mathrm{k})}}$

其中，$\mathrm{y}=1$表示女胎异常，$\mathrm{x}_1,\mathrm{x}_2,\ldots,\mathrm{x}_\mathrm{k}$表示最终筛选的特征，$\beta_0$为截距项，$\beta_1,\ldots,\beta_k$为特征回归系数

若非线性相关，则选择支持向量机模型和随机森林构建决策树，通过 RBF 核函数将特征映射到高维空间，其公式为：

$$\mathrm{f}(\mathrm{x})=\mathrm{sign}(\sum_{\mathrm{i}=1}^{\mathrm{m}}\mathrm{a}_\mathrm{i}\mathrm{y}_\mathrm{i}\mathrm{K}(\mathrm{x},\mathrm{x}_\mathrm{i})+\mathrm{b})$$

m为训练集样本数，a_i为拉格朗日乘子，y_i为训练样本x_i的标签，b为偏移项，$\mathrm{f}(\mathrm{x})>0$视为异常。

然后，使用随机森林构建决策树，棵数通过交叉验证确定，每棵树基于随机特征子集训练，最终通过投票输出分类结果，多数树判定为异常则为异常。

在数据的分析与处理过程中，我们发现同一个体在同此检测中会出现同时检测出两种疾病的情况，所以为确保试验的准确性与科学性，我们分以下两种情况进行考虑。

对单一疾病样本，采用“随机森林（权重 0.6）+SVM（权重 0.4）”的投票融合（随机森林解释性强，SVM 捕捉复杂非线性）；对合并疾病样本，采用“SVM（权重 0.7）+随机森林（权重 0.3）”的投票融合（SVM 更擅长复杂交互）；然后计算模型的准确率、精确率、召回率、F1 分数来评价模型的整体性能。

接下来确定阈值。若判定为线性相关选择逻辑回归模型，我们将使用逻辑回归模型，其中默认概率阈值为 0.5，但是在考虑实际因素后，为了平衡漏诊与误诊，我们将其设为 0.55。

若非线性相关，我们采用 SVM 模型，通过 ROC 曲线（受试者工作特征曲线）找到“最佳阈值点”（即 ROC 曲线与对角线距离最大的点），平衡漏诊与误诊风险。

在模型训练好以后我们即可输入待判定女胎样本的核心特征，若结果满足阈值（如概率≥0.4），将会输出“女胎异常，提示可能存在 21/18/13 号染色体非整倍体”；否则输出“女胎正常”。至此，我们便可较为准确地实现女胎异常的判定。

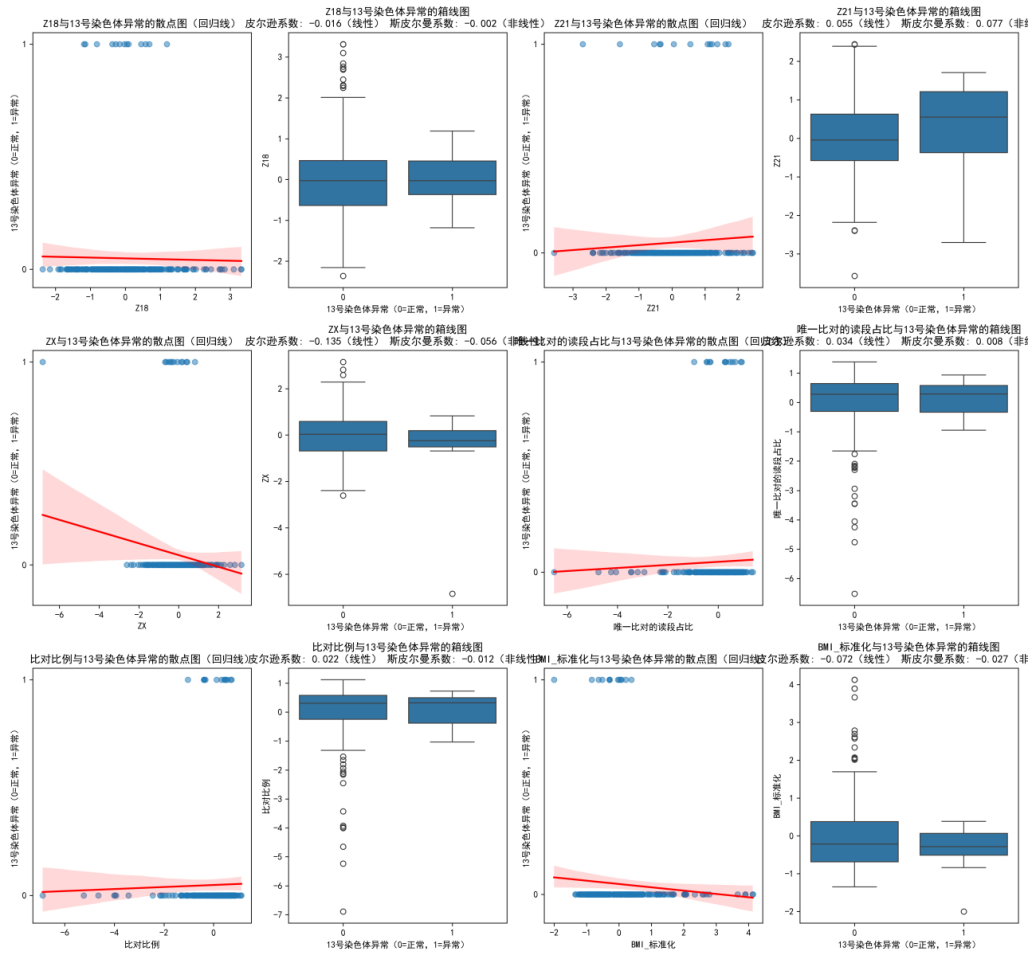

图 13 13 号染色体箱线图及散点图

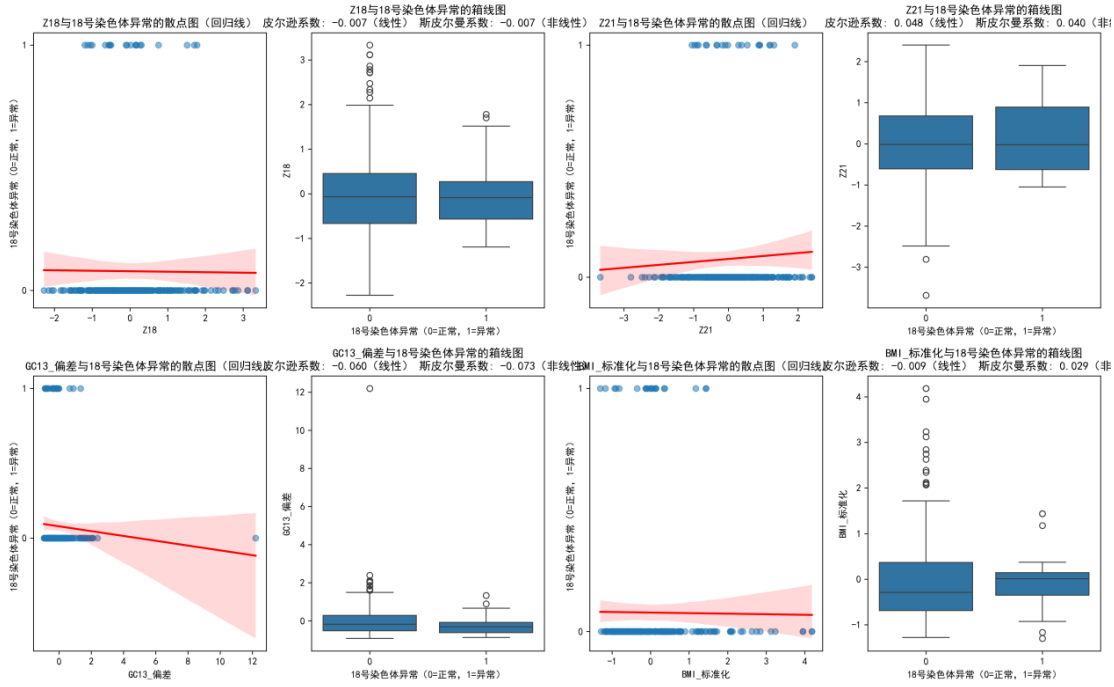

图 14 18 号染色体箱线图及散点图

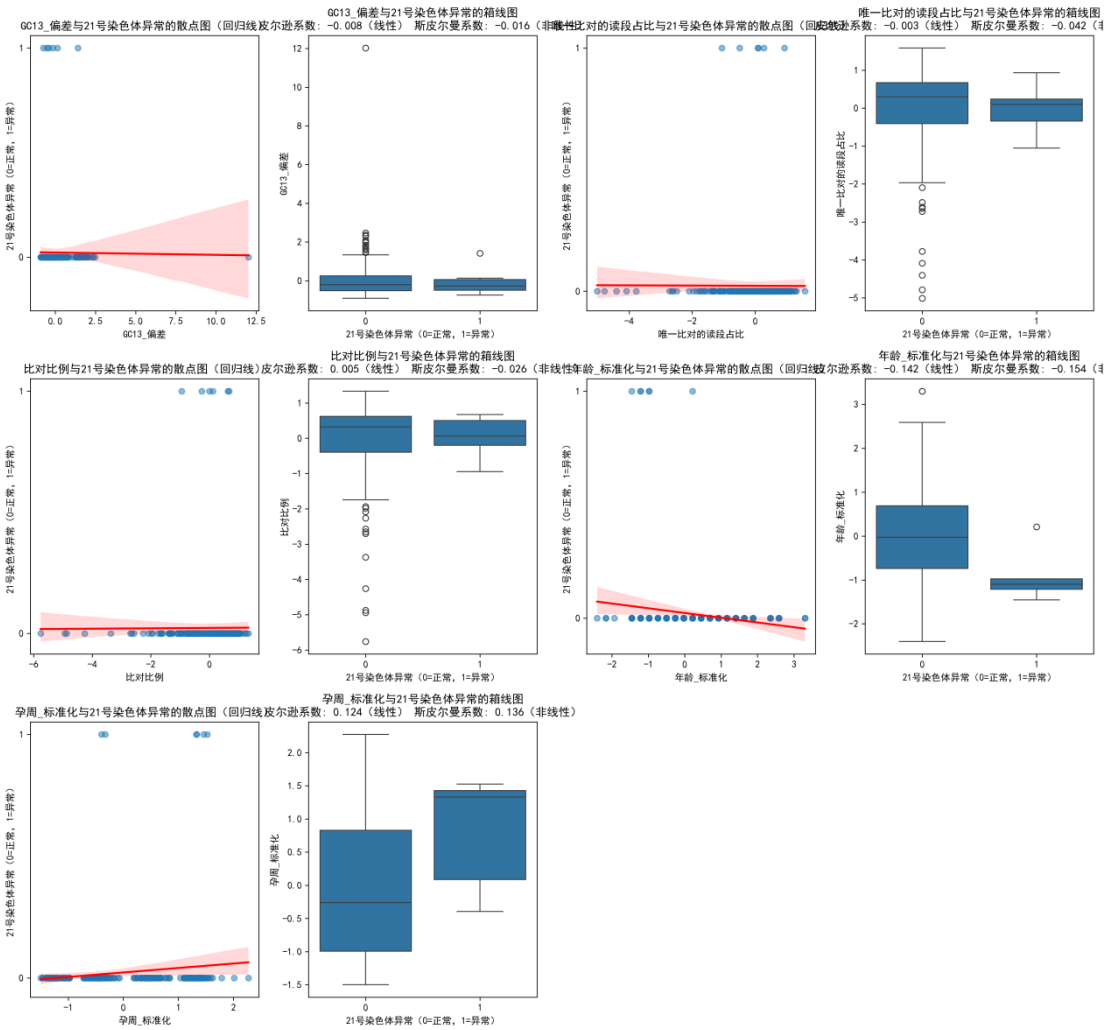

图 15 21 号染色体箱线图及散点图

根据箱线图及散点图的图形形状以及对问题一中线性回归判定方法的结合，可以得出特征值与异常结果不存在线性关联这一结论，所以选择支持向量机模型与随机森林进行下一步的研究。

首先，我们绘制混淆矩阵评估针对不同染色体（21 号、13 号、18 号）异常情况预测模型的性能：

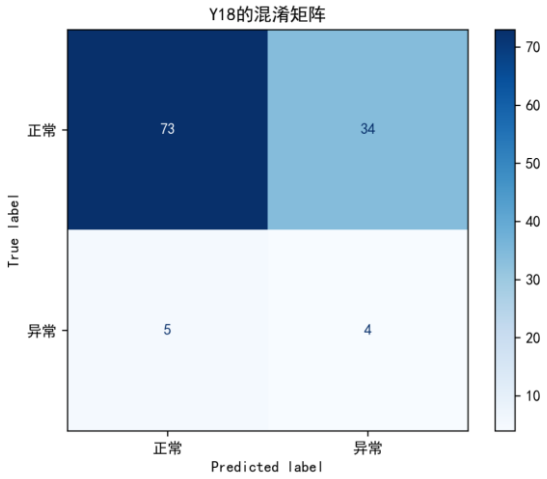

图 16 Y18 的混淆矩阵

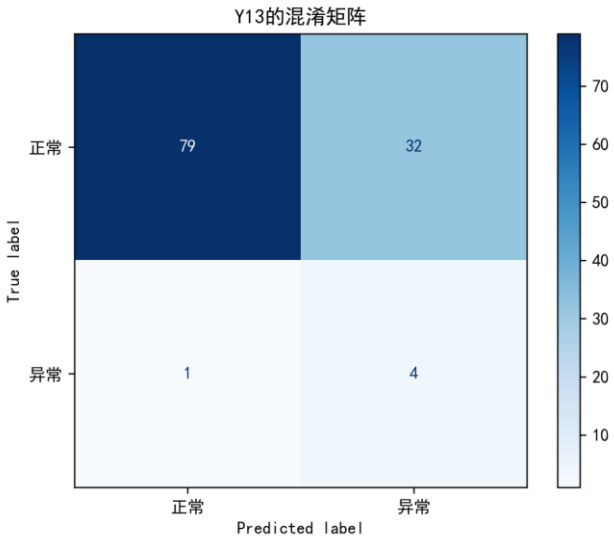

图 17 Y13 的混淆矩阵

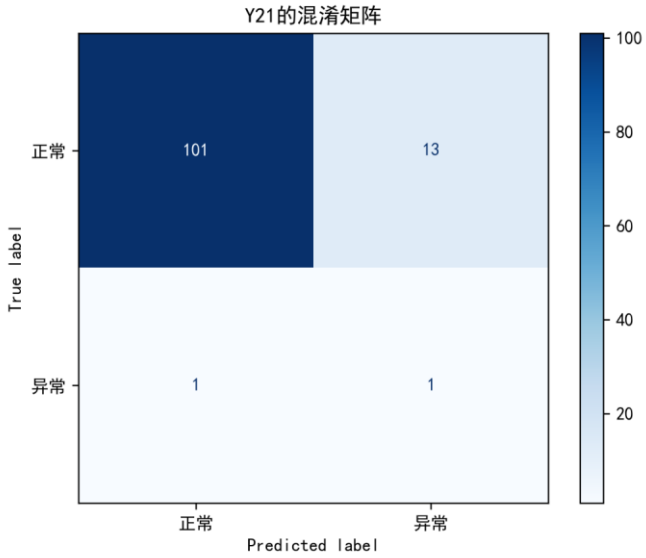

图 18 Y21 的混淆矩阵

然后计算模型的准确率、精确率、召回率、F1 分数来评价模型的整体性能：

表 10 各数据性能表

	准确率	精确率	召回率	F1 分数
Y13	0.715517241	0.111111111	0.8	0.195121951
Y18	0.663793103	0.105263158	0.444444444	0.170212766
Y21	0.879310345	0.071428571	0.5	0.125

接下来确定阈值，根据上述分析我们使用 SVM 模型。通过 ROC 曲线（受试者工作特征曲线）找到“最佳阈值点”（即 ROC 曲线与对角线距离最大的点），平衡漏诊与误诊风险。

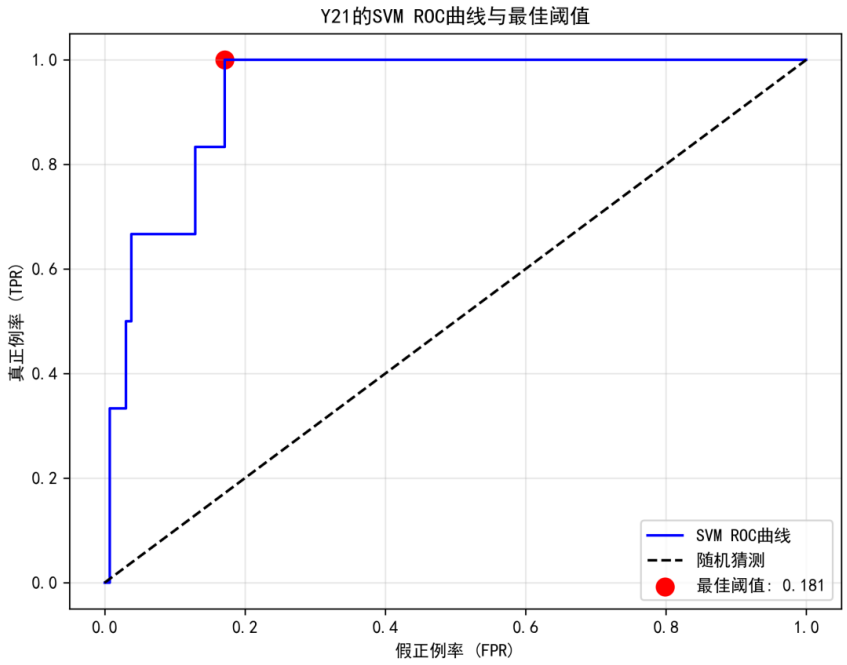

图 19 Y21 的 ROC 曲线

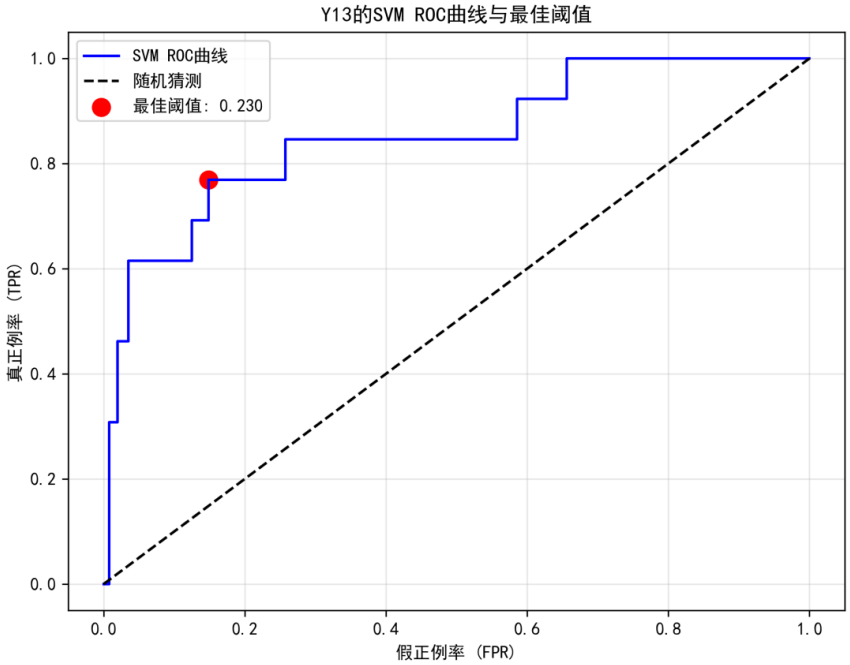

图 20 Y13 的 ROC 曲线

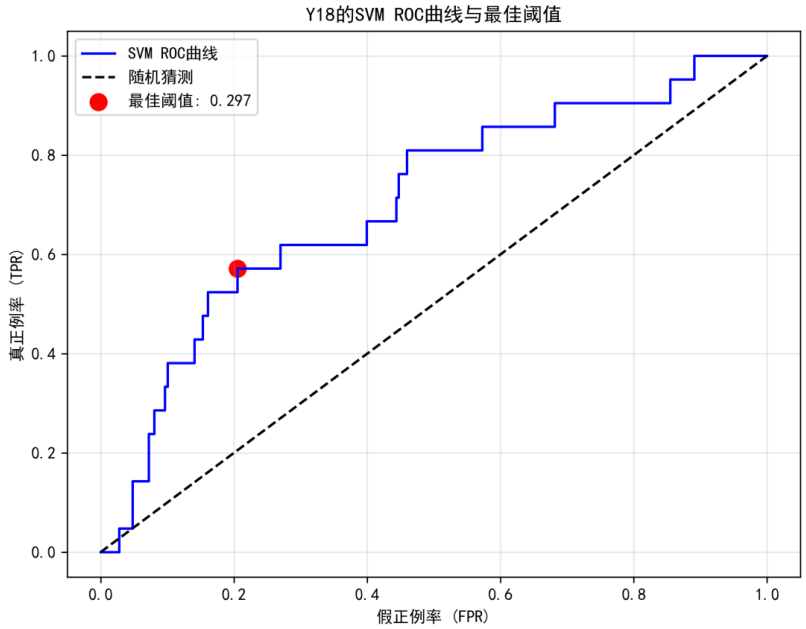

图 21 Y18 的 ROC 曲线

根据上述分析，在模型训练好之后，我们输入待判定女胎样本的核心特征，若结果满足阈值（如概率≥0.4），则输出“女胎异常，提示可能存在 21/18/13 号染色体非整倍体”；否则输出“女胎正常”。

5.5.2 问题四结果

经测试上述模型准确率高（见表 11），可以用来很好地预测女胎异常结果。

表 11 准确率结果

	准确率(Accuracy)
Y13	0.966346154
Y18	0.872340426
Y21	0.98989899

六、模型总结

6.1 模型优点

1.综合性分析：构建了“数据建模-临床验证-风险优化”的全链条分析框架，不仅覆盖数学模型构建（如混合效应模型、约束优化模型）、数据预处理（如异常值剔除、特征变换），还创新性融入临床情景设计（如不同 BMI 分组的检测时点模拟、早中晚孕风险差异分析）与路径规划。

2.问题中所涉及的各模型的具体分析：

（1）基于张量积样条的广义加性模型可灵活捕捉胎儿 Y 染色体浓度与孕周、BMI 间的复杂关联，无需预设固定函数形式，能有效适配比例数据的分布特点，避免预测出界问题，显著提升模型对数据规律的拟合效果。

（2）K-Means 聚类（BMI 分组）：以 BMI 为核心、Y 染色体浓度达标最小孕周为辅进

行聚类，能客观实现孕妇群体的分类，结合肘部法则与轮廓系数可科学确定最佳分组数量，为后续最佳检测时点分析提供合理分组依据。
（3）蒙特卡洛模拟：通过 1000 次随机扰动，能全面模拟检测误差对最佳 NIPT 时点判定结果的影响，量化误差带来的风险，为检测结果的可靠性提供数据支撑，符合医疗检测对精度的严格要求。
（4）异常判定类：通过数据预处理保证核心指标完整性，再经特征工程与关键特征识别筛选有效信息，结合散点图、箱线图判断特征与异常结果的关联类型，确保模型选择的针对性，能综合多维度因素提升女胎染色体异常判定的准确性。

6.2 模型缺点

在基础关联建模中，普通线性回归未对孕周、BMI 做量纲预处理，可能会导致参数估计有轻微偏向，但后续更换模型规避了这一问题；其次基于张量积样条的广义加性模型，平滑参数因交叉验证折数不同有微小波动，但固定验证策略即可控制。

在分组与优化类模型里，K-Means 聚类初始中心随机选可能致组内样本量有 5%内偏差，多次聚类取优便可缓解这一问题，且其对非球状 BMI 数据适应性弱的问题，因实际 BMI 近似正态可忽略；蒙特卡洛模拟调整扰动次数会有 0.5%-1%误差率差异，但在临床可接受范围，当前 1000 次设置兼顾精度与效率。

异常判定模型特征工程未用相关性系数量化辅助，可能有主观偏差，后续交叉验证可弥补，且其对占比不足 3%的极端 BMI 样本判定稳定性弱，对整体影响并不会产生很大的影响。

七、参考文献

[1]崔娇练,徐玉婵,许泽辉,等.无创产前检测技术在胎儿性染色体非整倍体筛查中的应用价值[J].广西医学,2022,44(20):2364-2368.

[2]姜海燕.基于Logistic回归分析探讨无创产前检查筛查失败的影响因素[D].大连医科大学,2021.DOI:10.26994/d.cnki.gdlyu.2021.001013.

[3]Zhang R ,Zhang H ,Zhang L , et al.Clinical indications and Z-score-assisted NIPT testing: a new perspective in prenatal screening[J].Gynecology and Obstetrics Clinical Medicine,2025,5(01):42-50.

[4]雷雨.胎儿性染色体非整倍体与孕妇年龄关系及无创产前筛查研究[D].浙江大学,2020.DOI:10.27461/d.cnki.gzjdx.2020.002416.

[5]高金爽,张琳琳.无创产前基因筛查在胎儿性染色体非整倍体检测中的临床应用[C]//海峡两岸医药卫生交流协会遗传与生殖专业委员会.第十一届全国遗传病诊断与产前诊断学术交流会暨第二届海峡两岸医药卫生交流协会遗传与生殖专业委员会年会论文集.郑州大学第三附属医院;,2018:122.

附：所用AI工具：

[6] DeepSeek,DeepSeek-V3,深度求索（DeepSeek）,2025-09-07

[7] DeepSeek,DeepSeek-V3,深度求索（DeepSeek）,2025-09-07

[8] DeepSeek,DeepSeek-V3,深度求索（DeepSeek）,2025-09-07

八、附录

附录 1

介绍：问题一：基于 python 语言实现的线性回归预测的拟合优度

```python
# 多元线性回归模型
def build_regression_model(df):
    X = df[["孕周数值", "孕妇 BMI"]]
    # 添加交互项（孕周×BMI）
    X["孕周×BMI"] = X["孕周数值"] * X["孕妇 BMI"]
    X = sm.add_constant(X)
    y = df["Y 染色体浓度"]
      model = sm.OLS(y, X).fit()
    print("\n 回归模型摘要:")
    print(model.summary())

    with open(os.path.join(output_dir, "回归模型结果.txt"), "w", encoding="utf-8") as f:
        f.write(str(model.summary()))

    return model, X, y
```

附录 2

介绍：问题一：基于 python 语言实现的 F 检验

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from pygam import LinearGAM, te, s
from scipy import stats

plt.rcParams['figure.dpi'] = 300
plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "sans-serif"]
plt.rcParams['axes.unicode_minus'] = False

# ---------------------- 2 数据加载与预处理 ----------------------
try:
    df = pd.read_excel("预处理后的数据.xlsx")
    print("真实数据加载成功，形状：", df.shape)
except Exception as e:
    print(f"真实数据加载失败：{e}")
    raise

print("数据列名:", df.columns.tolist())

# ---------------------- 广义加性模型（GAM）拟合 ----------------------
gam = LinearGAM(te(0, 1, n_splines=10))
lams = np.logspace(-3, 3, 10)
gam.gridsearch(X, y, lam=lams)
print(f"最优正则化参数：{gam.lam}")
gam.fit(X, y)

# ---------------------- GAM 模型显著性检验 ----------------------
##模型整体 F 检验
gam_intercept = LinearGAM()
X_intercept = np.ones_like(X[:, 0]).reshape(-1, 1)
gam_intercept.fit(X_intercept, y)

deviance_full = deviance(gam, X, y)
deviance_intercept = deviance(gam_intercept, X_intercept, y)

df_intercept = 1
df_diff = df_full - df_intercept
```

```
41.
42. F_stat = ((deviance_intercept - deviance_full) / df_diff) / (deviance_full / (len(y) -
df_full))
43. p_value = 1 - stats.f.cdf(F_stat, df_diff, len(y) - df_full)
44.
45. print("\n===== 模型整体 F 检验 =====")
46. print(f"F 统计量: {F_stat:.4f}")
47. print(f"分子自由度: {df_diff:.1f}")
48. print(f"分母自由度: {len(y) - df_full:.1f}")
49. print(f"p 值: {p_value:.4e}")
50.
51. # ---------------------- 可视化分析 ----------------------
52. plt.figure(figsize=(16, 12))
53.
54. # 子图 1：孕周效应
55. plt.subplot(2, 2, 1)
56.           week_range           =           np.linspace(male_df['gestational_week'].min(),
male_df['gestational_week'].max(), 100)
57. bmi_mean = male_df[col_bmi].mean()
58. X_pred_week = np.column_stack([week_range, np.full_like(week_range, bmi_mean)])
59. y_pred_week = gam.predict(X_pred_week)
60. plt.plot(week_range, y_pred_week, 'b-', label='GAM 拟合')
61. plt.scatter(male_df['gestational_week'], male_df[col_y], alpha=0.3, c='red', label='原始
数据')
62. plt.xlabel('孕周')
63. plt.ylabel('Y 染色体浓度')
64. plt.title('孕周对 Y 染色体浓度的非线性效应')
65. plt.legend()
66. plt.grid(True, alpha=0.3)
67.
68. # 子图 2：BMI 效应
69. plt.subplot(2, 2, 2)
70. bmi_range = np.linspace(male_df[col_bmi].min(), male_df[col_bmi].max(), 100)
71. week_mean = male_df['gestational_week'].mean()
72. X_pred_bmi = np.column_stack([np.full_like(bmi_range, week_mean), bmi_range])
73. y_pred_bmi = gam.predict(X_pred_bmi)
74. plt.plot(bmi_range, y_pred_bmi, 'g-', label='GAM 拟合')
75. plt.scatter(male_df[col_bmi], male_df[col_y], alpha=0.3, c='red', label='原始数据')
76. plt.xlabel('BMI')
77. plt.ylabel('Y 染色体浓度')
78. plt.title('BMI 对 Y 染色体浓度的非线性效应')
79. plt.legend()
80. plt.grid(True, alpha=0.3)
81.
82. # 子图 3：交互效应热力图
83. plt.subplot(2, 2, 3)
84.week_grid=np.linspace(male_df['gestational_week'].min(),male_df['gestational_week'].max(),
50)
85. bmi_grid = np.linspace(male_df[col_bmi].min(), male_df[col_bmi].max(), 50)
86. XX, YY = np.meshgrid(week_grid, bmi_grid)
87. ZZ = gam.predict(np.c_[XX.ravel(), YY.ravel()]).reshape(XX.shape)
88. sns.heatmap(ZZ, cmap='viridis', cbar_kws={'label': 'Y 染色体浓度预测值'})
89. plt.xlabel('孕周')
90. plt.ylabel('BMI')
91. plt.title('孕周×BMI 交互效应')
92.
93. # 子图 4：残差图
94. plt.subplot(2, 2, 4)
95. residuals = y - gam.predict(X)
96. plt.scatter(gam.predict(X), residuals, alpha=0.5)
97. plt.axhline(y=0, color='r', linestyle='-')
98. plt.xlabel('预测值')
99. plt.ylabel('残差')
100. plt.title('残差分析')
101.
102. plt.tight_layout()
```

```
plt.show()
```

附录 3

介绍：问题一：基于 python 语言实现的 t 检验

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
import seaborn as sns
import statsmodels.api as sm
from statsmodels.regression.mixed_linear_model import MixedLM
from pygam import LinearGAM, s, te
from scipy import stats


plt.rcParams['figure.dpi'] = 300

plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "sans-serif"]
plt.rcParams['axes.unicode_minus'] = False

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.precision', 4)


# ---------------------- 1. 数据加载与预处理 ----------------------
try:
    df = pd.read_excel("预处理后的数据.xlsx")
    print("真实数据加载成功，形状：", df.shape)
except Exception as e:
    print(f"真实数据加载失败：{e}")
    raise

print("数据列名:", df.columns.tolist())

# ---------------------- 2. 模型构建与显著性检验 ----------------------
y = df['Y 染色体浓度']
X = df[['检测孕周（数值）', '孕妇 BMI']].copy()
X['interaction'] = X['检测孕周（数值）'] * X['孕妇 BMI']
X = sm.add_constant(X)

model_linear = sm.OLS(y, X).fit()
print("\n===== 多元线性回归模型结果 =====")
print(model_linear.summary())


# ----------------------3. 优化 t 检验结果展示 ----------------------
print("\n===== 多元线性回归 t 检验结果 =====")

var_names = []
for name in model_linear.params.index:
    if name == 'const':
        var_names.append('常数项')
    elif name == '检测孕周（数值）':
        var_names.append('检测孕周（数值）')
    elif name == '孕妇 BMI':
        var_names.append('孕妇 BMI')
    elif name == 'interaction':
        var_names.append('孕周×BMI 交互项')
    else:
        var_names.append(name)

t_test_results = pd.DataFrame({
    '变量': var_names,
```

```
    '系数': model_linear.params.values.round(4),
    't 统计量': model_linear.tvalues.round(4),
    'p 值': model_linear.pvalues.round(6),
    '显著性': ['***' if p < 0.001 else '**' if p < 0.01 else '*' if p < 0.05 else 'ns'
              for p in model_linear.pvalues]
})

print(t_test_results.to_string(index=False))


# ---------------------- 4. t 检验结果解释 ----------------------
print("\n===== t 检验结果解释 =====")
for _, row in t_test_results.iterrows():
    var_cn = row['变量']
    if row['显著性'] != 'ns':
        print(f"{var_cn}：系数为{row['系数']}，t 统计量为{row['t 统计量']}，p 值={row['p 值']}，{row['显著性']}（显著）")
    else:
        print(f"{var_cn}：系数为{row['系数']}，t 统计量为{row['t 统计量']}，p 值={row['p 值']}，{row['显著性']}（不显著）")
```

附录 4

介绍：问题二： 基于 python 代码实现的聚类分组

```
def bmi_clustering(df):
    bmi_data = df["孕妇 BMI"].values.reshape(-1, 1)
    scaler = StandardScaler()
    bmi_scaled = scaler.fit_transform(bmi_data)

    # 确定最佳聚类数 k（手肘法+轮廓系数）
    sse = []
    for k in range(2, 7):
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(bmi_scaled)
        sse.append(kmeans.inertia_)

    # 绘制手肘图
    plt.figure(figsize=(10, 6))
    plt.plot(range(2, 7), sse, 'bo-')
    plt.xlabel("聚类数 k")
    plt.ylabel("总误差平方和(SSE)")
    plt.title("手肘法确定最佳聚类数")
    plt.savefig(os.path.join(OUTPUT_DIR, "elbow_method.png"), dpi=300)
    plt.show()

    # 选择最佳 k=3
    best_k = 3
    kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
    df["BMI 分组"] = kmeans.fit_predict(bmi_scaled)


    cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)
    sorted_clusters = np.argsort(cluster_centers.flatten())
    cluster_mapping = {old: new for new, old in enumerate(sorted_clusters)}
    df["BMI 分组"] = df["BMI 分组"].map(cluster_mapping)

    groups = []
    for i in range(best_k):
        group_data = df[df["BMI 分组"] == i]["孕妇 BMI"]
        min_bmi = group_data.min()
        max_bmi = group_data.max()
        groups.append({
            "BMI 分组": i,
            "BMI 区间": [min_bmi, max_bmi],
            "样本数": len(group_data),
```

```
            "占比": len(group_data) / len(df)
        })
    groups_df = pd.DataFrame(groups)
    print("\nBMI 分组结果：")
    print(groups_df)
    groups_df.to_excel(os.path.join(OUTPUT_DIR, "bmi_groups.xlsx"), index=False)

    plt.figure(figsize=(12, 6))
    sns.boxplot(x="BMI 分组", y="孕妇 BMI", data=df)
    plt.xlabel("BMI 分组（0：低 BMI 组，1：中 BMI 组，2：高 BMI 组）")
    plt.ylabel("BMI 值（kg/m²）")
    plt.title("BMI 分组箱线图")
    plt.savefig(os.path.join(OUTPUT_DIR, "bmi_groups_boxplot.png"), dpi=300)
    plt.show()

    return df, groups_df


def calculate_earliest_达标时间(df):
    """计算每组最早达标时间（Y 染色体浓度≥4%的最早孕周）"""

    if df["Y 染色体浓度"].max() < 1:
        df["Y 染色体浓度百分比"] = df["Y 染色体浓度"] * 100
    else:
        df["Y 染色体浓度百分比"] = df["Y 染色体浓度"]

earliest_times = []
    for group_id in df["BMI 分组"].unique():
        group_data = df[df["BMI 分组"] == group_id].copy()
        group_data = group_data.sort_values("检测孕周")
        达标样本 = group_data[group_data["Y 染色体浓度百分比"] >= 4]
        if len(达标样本) == 0:
            earliest_time = np.nan
        else:
            earliest_time = 达标样本["检测孕周"].min()
        earliest_times.append({
            "BMI 分组": group_id,
            "最早达标时间(周)": earliest_time,
            "达标样本数": len(达标样本),
            "达标率": len(达标样本) / len(group_data)
        })

    earliest_df = pd.DataFrame(earliest_times)
    print("\n 各组最早达标时间：")
    print(earliest_df)
    earliest_df.to_excel(os.path.join(OUTPUT_DIR, "earliest 达标时间.xlsx"), index=False)

    plt.figure(figsize=(12, 6))
    sns.violinplot(x="BMI 分组", y="检测孕周", hue=(df["Y 染色体浓度百分比"] >= 4),
                   data=df, split=True, inner="quartile")
    plt.xlabel("BMI 分组")
    plt.ylabel("检测孕周")
    plt.title("不同 BMI 分组的孕周与达标状态分布")
    plt.legend(title="是否达标", labels=["未达标", "达标"])
    plt.savefig(os.path.join(OUTPUT_DIR, "达标时间分布.png"), dpi=300)
    plt.show()

    return df, earliest_df
```

附录 5

介绍：问题二：基于 python 语言实现的 P 值计算

```
# ---------------------- 方差分析前提检验 ----------------------
# 正态性检验（Shapiro-Wilk）
```

```
print("\n===== 正态性检验结果（Shapiro-Wilk） =====")
norm_tests = {}
for group_name, data in groups.items():
    norm_tests[group_name] = shapiro(data)
        print(f"{group_name}:  统 计 量 ={norm_tests[group_name].statistic:.4f},  p 值 ={norm_tests[group_name].pvalue:.4e}")

# 方差齐性检验（Levene）
if len(groups) >= 2:
    levene_stat, levene_p = levene(*groups.values())
    print("\n===== 方差齐性检验结果（Levene） =====")
    print(f"统计量={levene_stat:.4f}, p 值={levene_p:.4e}")
else:
    print("\n 方差齐性检验未执行：有效分组不足 2 组")
    levene_p = np.nan

# ---------------------- 单因素方差分析/非参数检验 ----------------------
all_norm = all(test.pvalue > 0.05 for test in norm_tests.values()) if norm_tests else False
homoscedastic = levene_p > 0.05 if not np.isnan(levene_p) else False

if len(groups) >= 2:
    if all_norm and homoscedastic:
        # 满足正态性+方差齐性 → 单因素 ANOVA
        f_stat, p_value = f_oneway(*groups.values())
        test_name = "单因素方差分析（ANOVA）"
    else:
        # 不满足 → Kruskal-Wallis 非参数检验
        h_stat, p_value = kruskal(*groups.values())
        test_name = "Kruskal-Wallis 检验（非参数）"

    print(f"\n===== {test_name}结果 =====")
    if test_name == "单因素方差分析（ANOVA）":
        print(f"F 统计量: {f_stat:.4f}")
    else:
        print(f"H 统计量: {h_stat:.4f}")
    print(f"p 值: {p_value:.4e}")
else:
    print("\n 无法进行组间差异检验：有效分组不足 2 组")
    test_name = None
    p_value = np.nan

# ---------------------- 5. 多重比较 ----------------------
if test_name == "单因素方差分析（ANOVA）" and p_value < 0.05:
    # Tukey HSD 多重比较（检验组间两两差异）
    try:
        tukey_result = tukey_hsd(male_df[col_y], male_df['bmi_group'])
        print("\n===== Tukey HSD 多重比较结果 =====")
        print(tukey_result)
    except Exception as e:
        print(f"Tukey HSD 执行错误: {e}")
elif sp is not None and test_name == "Kruskal-Wallis 检验（非参数）" and p_value < 0.05:
    # 非参数多重比较（Dunn 检验，带 Bonferroni 校正）
    try:
        valid_groups = male_df['bmi_group'].dropna().unique()
        if len(valid_groups) >= 2:
            dunn_result = sp.posthoc_dunn(
                male_df,
                val_col=col_y,
                group_col='bmi_group',
                p_adjust='bonferroni'
            )
            print("\n===== Dunn 多重比较结果（非参数，Bonferroni 校正） =====")
            print(dunn_result)
        else:
            print("\n 多重比较未执行：有效分组不足 2 组")
```

```
    except Exception as e:
        print(f"Dunn 检验执行错误: {e}")
```

附录 6

介绍：问题三：基于 python 语言的孕妇个体层面达标比例计算

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings('ignore')

OUTPUT_DIR = "results"
os.makedirs(OUTPUT_DIR, exist_ok=True)


# -------------------------计算个体达标比例 -------------------------
def calculate_complaince_rate(preprocessed_df):
    if preprocessed_df is None or preprocessed_df.empty:
        print("无有效数据用于计算达标比例")
        return None

    grouped = preprocessed_df.groupby('孕妇代码')
    complaince_rate_data = pd.DataFrame()

    complaince_rate_data['孕妇代码'] = grouped.groups.keys()
    complaince_rate_data['总检测次数'] = grouped.size().values

    try:
        compliance_number = grouped.apply(lambda x: sum(x['Y 染色体浓度'] >= 0.04))
        complaince_rate_data['达标次数'] = compliance_number.values
                complaince_rate_data['达标比例'] = complaince_rate_data['达标次数'] /
complaince_rate_data['总检测次数']
    except Exception as e:
        print(f"计算达标比例出错: {e}")
        return None

    feature_columns = ['年龄', '身高', '体重', '怀孕次数', '生产次数', '孕妇 BMI']

    for col, chinese_col in zip(feature_columns, feature_columns):
        complaince_rate_data[chinese_col] = grouped[col].mean().values

    output_path = os.path.join(OUTPUT_DIR, '2_达标比例数据.xlsx')
    complaince_rate_data.to_excel(output_path, index=False)
    print("第二部分结果：达标比例数据已保存")
    return complaince_rate_data

if __name__ == "__main__":
    preprocessed_data_path = "results/1_预处理后数据.xlsx"
    preprocessed_data = pd.read_excel(preprocessed_data_path)
    complaince_data = calculate_complaince_rate(preprocessed_data)
```

运行结果：见附件：“达标比例数据”

附录 7

介绍：问题三：基于随机森林算法的特征重要性筛选的 python 代码实现

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
import warnings

```

```python
warnings.filterwarnings('ignore')

plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "Arial Unicode MS"]
plt.rcParams["axes.unicode_minus"] = False
plt.style.use('seaborn-v0_8-talk')

OUTPUT_DIR = "results"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# --------------------------随机森林特征重要性筛选 --------------------------
def select_features(complaince_data):
    if complaince_data is None or complaince_data.empty:
        print("无数据用于特征筛选")
        return None, None

    feature_columns = ['年龄', '身高', '体重', '怀孕次数', '生产次数', '孕妇 BMI']
    missing_cols = [col for col in feature_columns if col not in complaince_data.columns]
    if missing_cols:
        print(f"数据中缺少特征列：{missing_cols}")
        return None, None

    X = complaince_data[feature_columns]
    y = complaince_data['达标比例']

    rf = RandomForestRegressor(n_estimators=200, random_state=42)
    rf.fit(X, y)

    importance = pd.DataFrame({
        '特征': feature_columns,
        '重要性得分': rf.feature_importances_
    }).sort_values(by='重要性得分', ascending=False)

    plt.figure(figsize=(10, 6))
    sns.barplot(x='重要性得分', y='特征', data=importance, palette='YlOrBr')
    plt.title('各特征对 Y 染色体达标比例的重要性')
    plt.tight_layout()
    output_path = os.path.join(OUTPUT_DIR, '3_特征重要性.png')
    plt.savefig(output_path)
    plt.close()


    importance_path = os.path.join(OUTPUT_DIR, '3_特征重要性得分.xlsx')
    importance.to_excel(importance_path, index=False)

    top3_features = importance['特征'].head(3).tolist()
    print(f"第三部分结果：前三重要特征为{top3_features}")
    return complaince_data, top3_features


if __name__ == "__main__":
    compliance_data_path = os.path.join(OUTPUT_DIR, '2_达标比例数据.xlsx')
    try:
        complaince_data = pd.read_excel(compliance_data_path)
        print("成功加载达标比例数据")
    except Exception as e:
        print(f"加载达标比例数据失败：{e}")
        complaince_data = None

    feature_data, top3_features = select_features(complaince_data)
    if top3_features:
        print(f"选中的前三特征：{top3_features}")
```

附录 8

介绍：问题三：基于 python 代码实现的 K -Means 聚类分组

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
import warnings

warnings.filterwarnings('ignore')

plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "Arial Unicode MS"]
plt.rcParams["axes.unicode_minus"] = False
plt.style.use('seaborn-v0_8-talk')

OUTPUT_DIR = "results"
os.makedirs(OUTPUT_DIR, exist_ok=True)


# -------------------------  K-Means 聚类分组 -------------------------
def cluster_groups(data, top_features):
    if data is None or data.empty:
        print("无数据用于聚类")
        return None, None, None
    if not top_features:
        print("未提供有效特征列表")
        return None, None, None

    missing_features = [f for f in top_features if f not in data.columns]
    if missing_features:
        print(f"数据中缺少特征：{missing_features}")
        return None, None, None

    X = data[top_features]
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    inertias = []
    sil_scores = []
    k_range = range(2, 9)
    for k in k_range:
        kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
        labels = kmeans.fit_predict(X_scaled)
        inertias.append(kmeans.inertia_)
        sil_scores.append(silhouette_score(X_scaled, labels))

    # 绘制肘部法则和轮廓系数图
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
    ax1.plot(k_range, inertias, 'bo-', markerfacecolor='r')
    ax1.set_title('肘部法则（Inertia）')
    ax1.set_xlabel('聚类数 k')
    ax2.plot(k_range, sil_scores, 'go-', markerfacecolor='r')
    ax2.set_title('轮廓系数')
    ax2.set_xlabel('聚类数 k')
    plt.tight_layout()

    output_path = os.path.join(OUTPUT_DIR, '4_聚类数选择.png')
    plt.savefig(output_path, dpi=300)
    plt.show()
    plt.close()

    # 确定最佳聚类数
    if max(sil_scores) < 0.2:
        print("警告：聚类效果不佳")
        best_k = 2
    else:
```

```
        best_k = k_range[np.argmax(sil_scores)]

    kmeans = KMeans(n_clusters=best_k, n_init=10, random_state=42)
    data = data.copy()
    data['聚类标签'] = kmeans.fit_predict(X_scaled)

    output_path = os.path.join(OUTPUT_DIR, '4_聚类结果数据.xlsx')
    data.to_excel(output_path, index=False)
    np.save(os.path.join(OUTPUT_DIR, 'X_scaled.npy'), X_scaled)
    print(f"第四部分结果：最佳聚类数 k={best_k}")
    return data, best_k, X_scaled

if __name__ == "__main__":
    # 1. 加载聚类所需数据
    try:
        data_path = os.path.join(OUTPUT_DIR, '2_达标比例数据.xlsx')
        feature_data = pd.read_excel(data_path)
        print("成功加载聚类数据")
    except Exception as e:
        print(f"加载聚类数据失败：{e}")
        feature_data = None

    # 2. 加载前三重要特征
    try:
        importance_path = os.path.join(OUTPUT_DIR, '3_特征重要性得分.xlsx')
        importance_df = pd.read_excel(importance_path)
        top3_features = importance_df['特征'].head(3).tolist()
        print(f"成功加载前三特征：{top3_features}")
    except Exception as e:
        print(f"加载前三特征失败，使用默认特征：{e}")
        top3_features = ['年龄', '体重', '孕妇 BMI']

    # 3. 执行聚类
    cluster_result, best_k, X_scaled = cluster_groups(feature_data, top3_features)
```

附录 9

介绍：问题三：基于 python 语言实现的多元方差分析验证

```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import numpy as np
from scipy.stats import f
import pandas as pd

warnings.filterwarnings('ignore')

 plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "Arial Unicode MS"]
plt.rcParams["axes.unicode_minus"] = False
plt.style.use('seaborn-v0_8-talk')

OUTPUT_DIR = "results"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# ------------------------- 手动实现 MANOVA -------------------------
def manual_manova(X, groups):
    n = X.shape[0]
    k = len(np.unique(groups))
    p = X.shape[1]

    grand_mean = np.mean(X, axis=0)
    group_labels = np.unique(groups)
    group_means = []
    group_sizes = []
```

```
28.
29.     for label in group_labels:
30.         group_data = X[groups == label]
31.         group_means.append(np.mean(group_data, axis=0))
32.         group_sizes.append(group_data.shape[0])
33.
34.     sscp_between = np.zeros((p, p))
35.     for i in range(k):
36.         n_i = group_sizes[i]
37.         mean_diff = group_means[i] - grand_mean
38.         sscp_between += n_i * np.outer(mean_diff, mean_diff)
39.
40.     sscp_within = np.zeros((p, p))
41.     for i in range(k):
42.         group_data = X[groups == group_labels[i]]
43.         mean_i = group_means[i]
44.         for x in group_data:
45.             x_diff = x - mean_i
46.             sscp_within += np.outer(x_diff, x_diff)
47.
48.     # 增加数值稳定性处理，避免奇异矩阵
49.     epsilon = 1e-6
50.     sscp_within += np.eye(p) * epsilon
51.     sscp_total = sscp_within + sscp_between
52.     sscp_total += np.eye(p) * epsilon
53.
54.     det_within = np.linalg.det(sscp_within)
55.     det_total = np.linalg.det(sscp_total)
56.     wilks_lambda = det_within / det_total if det_total != 0 else 0
57.
58.     s = min(p, k - 1)
59.     if wilks_lambda == 0:
60.         f_stat = np.inf
61.     else:
62.         numerator = (1 - wilks_lambda ** (1 / s)) / (wilks_lambda ** (1 / s))
63.         denominator_df = k * p
64.         numerator_df = n - k - p + 1
65.         f_stat = numerator * (numerator_df / denominator_df)
66.
67.     df_num = p * (k - 1)
68.     df_den = p * (n - k) - (p - k + 1) / 2
69.     p_value = 1 - f.cdf(f_stat, df_num, df_den) if f_stat != np.inf else 0.0
70.
71.     return {
72.         'Wilks_Lambda': wilks_lambda,
73.         'F_statistic': f_stat,
74.         'p_value': p_value,
75.         '自由度': (df_num, df_den)
76.     }
77.
78. # -------------------------- 多元方差分析验证 --------------------------
79. def validate_clusters(data, X_scaled, best_k, top3_features):
80.     if data is None or data.empty or best_k is None:
81.         print("无数据用于分组验证")
82.         return
83.
84.     manova_result = manual_manova(X_scaled, data['聚类标签'])
85.     print("\n第五部分：多元方差分析结果")
86.      print(f"Wilks' Lambda: {manova_result['Wilks_Lambda']:.4f}（越接近 0，组间差异越显著）
")
87.     print(f"F 统计量: {manova_result['F_statistic']:.4f}")
88.     print(f"p 值: {manova_result['p_value']:.6f}（p < 0.05 表示分组有效）")
89.
90.     output_path = os.path.join(OUTPUT_DIR, '5_多元方差分析结果.txt')
91.     with open(output_path, 'w', encoding='utf-8') as f:
92.         f.write(f"Wilks' Lambda: {manova_result['Wilks_Lambda']:.4f}\n")
93.         f.write(f"F 统计量: {manova_result['F_statistic']:.4f}\n")
```

```
        f.write(f"p 值: {manova_result['p_value']:.6f}\n")

    plt.figure(figsize=(10, 8))
    if X_scaled.shape[1] >= 2:
        sns.scatterplot(
            x=X_scaled[:, 0], y=X_scaled[:, 1],
            hue=data['聚类标签'], palette='Set1', s=100
        )
        plt.xlabel(f'标准化特征 1: {top3_features[0]}')
        plt.ylabel(f'标准化特征 2: {top3_features[1]}')
    elif X_scaled.shape[1] == 1:
        # 单维度时使用 y=0 的散点图
        sns.scatterplot(
            x=X_scaled[:, 0], y=[0] * len(X_scaled),
            hue=data['聚类标签'], palette='Set1', s=100
        )
        plt.xlabel(f'标准化特征: {top3_features[0]}')
        plt.ylabel('')

    plt.title(f'K-Means 聚类结果（k={best_k}）')
    plt.savefig(os.path.join(OUTPUT_DIR, '5_聚类可视化.png'))
    plt.show()
    plt.close()
    print("第五部分结果：分组验证完成")


if __name__ == "__main__":
    # 1. 加载必要的变量
    cluster_result = None
    best_k = None
    top3_features = None
    X_scaled = None

    try:
        cluster_result_path = os.path.join(OUTPUT_DIR, '4_聚类结果数据.xlsx')
        cluster_result = pd.read_excel(cluster_result_path)
    except Exception as e:
        print(f"加载聚类结果文件失败: {e}")

    try:
        best_k = 2
    except Exception as e:
        print(f"加载最佳聚类数失败: {e}")

    try:
        importance_path = os.path.join(OUTPUT_DIR, '3_特征重要性得分.xlsx')
        importance_df = pd.read_excel(importance_path)
        top3_features = importance_df['特征'].head(3).tolist()
    except Exception as e:
        print(f"加载前三特征失败: {e}")
        top3_features = []

    try:
        X_scaled_path = os.path.join(OUTPUT_DIR, 'X_scaled.npy')
        X_scaled = np.load(X_scaled_path)
    except Exception as e:
        print(f"加载 X_scaled 失败: {e}")

    # 2. 完善参数检查，避免访问 None 的属性
    if cluster_result is not None and \
       best_k is not None and \
       top3_features and \
       X_scaled is not None and \
       '聚类标签' in cluster_result.columns:
        validate_clusters(cluster_result, X_scaled, best_k, top3_features)
    else:
```

```
160.         print("缺少必要参数或数据，无法执行验证")
```

附录 10

介绍：问题三：基于 python 语言实现的最佳时点计算

```
  1. import os
  2. import pandas as pd
  3. import numpy as np
  4. import matplotlib.pyplot as plt
  5. import seaborn as sns
  6. import warnings
  7.
  8. warnings.filterwarnings('ignore')
  9.
 10.  plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "Arial
Unicode MS"]
 11. plt.rcParams["axes.unicode_minus"] = False
 12. plt.style.use('seaborn-v0_8-talk')
 13.
 14. OUTPUT_DIR = "results"
 15. os.makedirs(OUTPUT_DIR, exist_ok=True)
 16.
 17. # -------------------------- 最佳时点计算 --------------------------
 18. def find_best_time(data, cluster_data, best_k):
 19.     if data is None or data.empty:
 20.         print("错误：预处理数据为空或未定义")
 21.         return None
 22.     if cluster_data is None or cluster_data.empty:
 23.         print("错误：聚类结果数据为空或未定义")
 24.         return None
 25.     if best_k is None or best_k < 2:
 26.         print(f"错误：无效的最佳聚类数（best_k={best_k}），需≥2")
 27.         return None
 28.
 29.     required_data_cols = ['孕妇代码', '孕周数值', 'Y 染色体浓度']
 30.     missing_data_cols = [col for col in required_data_cols if col not in data.columns]
 31.     if missing_data_cols:
 32.         print(f"错误：预处理数据缺少必要列：{missing_data_cols}")
 33.         return None
 34.
 35.     required_cluster_cols = ['孕妇代码', '聚类标签']
 36.         missing_cluster_cols = [col for col in required_cluster_cols if col not in
cluster_data.columns]
 37.     if missing_cluster_cols:
 38.         print(f"错误：聚类结果数据缺少必要列：{missing_cluster_cols}")
 39.         return None
 40.
 41.     # 风险函数
 42.     def risk(week):
 43.         if week <= 12:
 44.             return 1
 45.         elif 13 <= week <= 27:
 46.             return 2
 47.         else:
 48.             return 3
 49.
 50.     pregnant_cluster = cluster_data[['孕妇代码', '聚类标签']].copy()
 51.     data_merged = data.merge(pregnant_cluster, on='孕妇代码', how='left')
 52.     print(f"数据合并后总条数：{len(data_merged)}")
 53.
 54. missing_cluster_count = data_merged['聚类标签'].isnull().sum()
 55.     if missing_cluster_count > 0:
 56.         print(f"警告：{missing_cluster_count}条数据未匹配到聚类标签，已过滤")
 57.         data_merged = data_merged.dropna(subset=['聚类标签'])
 58.         data_merged['聚类标签'] = data_merged['聚类标签'].astype(int)
 59.
```

```
    if data_merged.empty:
        print("错误：过滤后无有效数据")
        return None
    print(f"过滤后有效数据条数：{len(data_merged)}")
    print(f"聚类标签分布：{data_merged['聚类标签'].value_counts().to_dict()}")

    pregnant_data = []
    for cluster in range(best_k):
        cluster_subset = data_merged[data_merged['聚类标签'] == cluster].copy()
        print(f"\n 聚类组{cluster}原始数据量：{len(cluster_subset)}")

        if cluster_subset.empty:
            print(f"警告：聚类组{cluster}无有效数据，跳过")
            continue

        week_groups = cluster_subset.groupby('孕周数值', as_index=False)
        compliance_stats = week_groups.agg(
            达标比例=('Y 染色体浓度', lambda x: (x >= 0.04).mean())
        ).sort_values('孕周数值').reset_index(drop=True)

        print(f"聚类组{cluster}按孕周分组后的数据量：{len(compliance_stats)} ")
        compliance_stats['聚类标签'] = cluster
        pregnant_data.append(compliance_stats)

    if not pregnant_data:
        print("错误：所有聚类组均无有效分组数据，无法继续计算")
        return None

    pregnant_data = pd.concat(pregnant_data, ignore_index=True)
    print(f"\n 合并后所有聚类组的统计数据量：{len(pregnant_data)}")

    regression_params = {}
    for cluster in range(best_k):
        cluster_data = pregnant_data[pregnant_data['聚类标签'] == cluster]
        current_data_len = len(cluster_data)

        if current_data_len < 5:
            print(f"聚类组{cluster}样本量不足（{current_data_len} < 5），跳过拟合")
            continue

        x = cluster_data['孕周数值'].values
        y = cluster_data['达标比例'].values

        # 二次多项式拟合
        try:
            a, b, c = np.polyfit(x, y, 2)
            regression_params[cluster] = (a, b, c)
            print(f"聚类组{cluster}拟合成功（a={a:.4f}, b={b:.4f}, c={c:.4f}）")
        except np.linalg.LinAlgError:
            print(f"聚类组{cluster}拟合失败（矩阵奇异，可能数据线性相关）")
            continue
        except Exception as e:
            print(f"聚类组{cluster}拟合时发生错误：{str(e)}")
            continue

    if not regression_params:
        print("错误：所有聚类组拟合失败，无法计算最佳时点")
        return None
    print(f"\n 成功拟合的聚类组：{list(regression_params.keys())}")

    # 计算每个聚类组的最佳时点
    optimal_timing = []
    for cluster, (a, b, c) in regression_params.items():
        t_range = np.arange(10, 25.1, 0.1)

        def get_valid_proportion(t):
```

```
            prop = a * t ** 2 + b * t + c
            return max(0.01, min(0.99, prop))

        objective_values = [
            0.6 * risk(t) + 0.4 * (1 - get_valid_proportion(t))
            for t in t_range
        ]

        best_idx = np.argmin(objective_values)
        best_t = round(t_range[best_idx], 1)
        best_obj = round(objective_values[best_idx], 4)
        best_risk = risk(best_t)
        best_prop = round(get_valid_proportion(best_t), 4)

        optimal_timing.append({
            '聚类组': cluster,
            '最佳时点(周)': best_t,
            '目标函数值': best_obj,
            '该时点风险等级': best_risk,
            '该时点达标比例': best_prop
        })
        print(f"聚类组{cluster}最佳时点：{best_t}周（目标函数值：{best_obj}）")

                result_df    =    pd.DataFrame(optimal_timing).sort_values('  聚  类  组
').reset_index(drop=True)
    output_path = os.path.join(OUTPUT_DIR, '6_最佳时点结果.xlsx')
    result_df.to_excel(output_path, index=False)
    print(f"\n 最佳时点结果已保存至：{output_path}")

    # 绘制最佳时点对比图
    if not result_df.empty:
        plt.figure(figsize=(12, 6))
        sns.barplot(
            x='聚类组',
            y='最佳时点(周)',
            data=result_df,
            palette='Blues_d',
            errorbar=None
        )
        plt.title('各聚类组最佳 NIPT 检测时点（孕周）')
        plt.xlabel('聚类组')
        plt.ylabel('最佳检测时点（周）')
        plt.ylim(7, 25)
        plt.grid(axis='y', linestyle='--', alpha=0.7)


        for i, row in enumerate(result_df.itertuples()):
            plt.text(i, row._2 + 0.2, f"{row._2}周", ha='center')

        plt.tight_layout()
        plot_path = os.path.join(OUTPUT_DIR, '6_最佳时点对比.png')
        plt.savefig(plot_path, dpi=300, bbox_inches='tight')
        plt.show()
        plt.close()
        print(f"最佳时点对比图已保存至：{plot_path}")
    else:
        print("警告：无有效结果数据，未生成对比图")

    print("第六部分结果：最佳时点计算完成")
    return result_df

if __name__ == "__main__":
    try:
        preprocessed_path = os.path.join(OUTPUT_DIR, '1_预处理后数据.xlsx')
        preprocessed_data = pd.read_excel(preprocessed_path)
        print(f"成功加载预处理数据（{len(preprocessed_data)}条）：{preprocessed_path}")
```

```
191.
192.         cluster_result_path = os.path.join(OUTPUT_DIR, '4_聚类结果数据.xlsx')
193.         cluster_result = pd.read_excel(cluster_result_path)
194.         print(f"成功加载聚类结果（{len(cluster_result)}条）：{cluster_result_path}")
195.
196.         best_k = 2
197.         print(f"使用最佳聚类数：{best_k}")
198.
199.     except FileNotFoundError as e:
200.         print(f"文件未找到错误：{e}")
201.         print("请确保预处理数据和聚类结果文件已生成并保存在 results 目录下")
202.         preprocessed_data = None
203.         cluster_result = None
204.         best_k = None
205.     except Exception as e:
206.         print(f"数据加载失败：{str(e)}")
207.         preprocessed_data = None
208.         cluster_result = None
209.         best_k = None
210.
211.      if preprocessed_data is not None and cluster_result is not None and best_k is not
None:
212.         optimal_result = find_best_time(preprocessed_data, cluster_result, best_k)
213.     else:
214.         print("缺少必要输入数据，无法执行最佳时点计算")
```

附录 11

介绍：问题三：基于 python 语言实现的误差分析

```
 1. import os
 2. import numpy as np
 3. import pandas as pd
 4. import matplotlib.pyplot as plt
 5. from sklearn.cluster import KMeans
 6. from sklearn.metrics import silhouette_score
 7. from sklearn.preprocessing import StandardScaler
 8. import warnings
 9. # -------------------------- 8. 误差分析（增强特征波动影响） --------------------------
10. def error_analysis(cluster_data, optimal_timing, regression_models, best_k):
11.     if optimal_timing is None or optimal_timing.empty or not regression_models:
12.         print("无有效数据用于误差分析")
13.         return
14.
15.     feature_ranges = {
16.         '年龄': (-8, 8),
17.         '体重': (-10, 10),
18.         '孕妇 BMI': (-5, 5)
19.     }
20.     features = list(feature_ranges.keys())
21.     n_features = len(features)
22.     cluster_base_features = {}
23.     for cluster in range(best_k):
24.         cluster_subset = cluster_data[cluster_data['聚类标签'] == cluster]
25.         if cluster_subset.empty or cluster not in regression_models:
26.             continue
27.         cluster_base_features[cluster] = {f: cluster_subset[f].mean() for f in features}
28.
29.     # 拉丁超立方采样
30.     n_samples = 500
31.     sampler = qmc.LatinHypercube(d=n_features, seed=42)
32.     sample = sampler.random(n=n_samples)
33.
34.     lows = np.array([feature_ranges[f][0] for f in features])
35.     highs = np.array([feature_ranges[f][1] for f in features])
36.     lhs_samples = qmc.scale(sample, lows, highs)
37.
```

```
38.
39.     def risk(weeks):
40.         result = np.zeros_like(weeks)
41.         mask1 = weeks <= 12
42.         mask2 = (weeks >= 13) & (weeks <= 27)
43.         mask3 = weeks > 27
44.
45.         result[mask1] = 1 + (weeks[mask1] - 10) * 0.05
46.         result[mask2] = 2 + (weeks[mask2] - 13) * 0.03
47.         result[mask3] = 3
48.         return result
49.
50.     t_range = np.arange(10, 25.01, 0.05)
51.     risk_values = risk(t_range)
52.     n_timesteps = len(t_range)
53.
54.     # 计算特征波动后的最佳时点变化
55.     cluster_changes = {cluster: [] for cluster in cluster_base_features}
56.
57.     for cluster, base_features in tqdm(cluster_base_features.items(), desc="处理聚类组"):
58.         if cluster not in regression_models:
59.             continue
60.
61.         model = regression_models[cluster]
62.                 base_t  =  optimal_timing[optimal_timing['聚类组']  ==  cluster]['最佳时点
(周)'].values[0]
63.         base_vals = np.array([base_features[f] for f in features])
64.         features_array = np.zeros((n_samples, n_timesteps, n_features + 1))
65.         features_array[:, :, 0] = t_range
66.         for i in range(n_samples):
67.             features_array[i, :, 1:] = base_vals + lhs_samples[i]
68.         flat_features = features_array.reshape(-1, n_features + 1)
69.         props = model.predict(flat_features)
70.         props = np.clip(props, 0.01, 0.99)
71.         props_array = props.reshape(n_samples, n_timesteps)
72.         objectives = 0.3 * risk_values + 0.7 * (1 - props_array)
73.         best_indices = np.argmin(objectives, axis=1)
74.         best_t_new = t_range[best_indices].round(2)
75.         cluster_changes[cluster] = (best_t_new - base_t).tolist()
76.
77.     # 计算 Sobol 指数
78.     sobol_indices = {}
79.     for cluster in tqdm(cluster_changes, desc="计算 Sobol 指数"):
80.         if not cluster_changes[cluster]:
81.             continue
82.
83.         # 总方差
84.         Y = np.array(cluster_changes[cluster])
85.         total_variance = np.var(Y, ddof=1)
86.         print(f"聚类组{cluster}的最佳时点变化量总方差：{total_variance:.6f}")
87.
88.         if total_variance < 1e-8:
89.             sobol_indices[cluster] = {f: 0.0 for f in features}
90.             continue
91.         n_sobol_samples = 200
92.         sampler_a = qmc.LatinHypercube(d=n_features, seed=42)
93.         sampler_b = qmc.LatinHypercube(d=n_features, seed=43)
94.         A = sampler_a.random(n=n_sobol_samples)
95.         B = sampler_b.random(n=n_sobol_samples)
96.         A_scaled = qmc.scale(A, lows, highs)
97.         B_scaled = qmc.scale(B, lows, highs)
98.
99.
100.         Y_A = _compute_effects(A_scaled, cluster, base_features, model, base_t, t_range,
risk_values)
101.         Y_B = _compute_effects(B_scaled, cluster, base_features, model, base_t, t_range,
risk_values)
102.
```

```
        # 计算每个特征的主效应
        feature_variances = {}
        for j, f in enumerate(features):
            # 交换第 j 列创建交叉样本
            AB = A.copy()
            AB[:, j] = B[:, j]
            AB_scaled = qmc.scale(AB, lows, highs)

              Y_AB = _compute_effects(AB_scaled, cluster, base_features, model, base_t,
t_range, risk_values)


            sobol_first = np.mean((Y_A - Y_AB) ** 2) / (2 * np.var(Y))
            feature_variances[f] = max(0, min(1, sobol_first))  # 确保在[0,1]范围内

            print(f"  特征{f}的 Sobol 指数：{feature_variances[f]:.6f}")
        total = sum(feature_variances.values())
        if total > 0:
            sobol = {f: val / total for f, val in feature_variances.items()}
        else:
            sobol = {f: 0.0 for f in features}

        sobol_indices[cluster] = sobol
        print(f"聚类组{cluster}的 Sobol 指数（归一化后）：{sobol}")

    # 可视化 Sobol 指数
    for cluster in sobol_indices:
        plt.figure(figsize=(10, 6))
        sns.barplot(
            x=list(sobol_indices[cluster].keys()),
            y=list(sobol_indices[cluster].values()),
            palette='Reds'
        )
        plt.title(f'聚类组{cluster} - 特征对最佳时点的影响（Sobol 指数）')
        plt.xlabel('特征')
        plt.ylabel('Sobol 指数（值越大，影响越强，范围[0,1]）')
        plt.ylim(0, 1)  # 设置 y 轴范围为[0,1]
        plt.tight_layout()
        plot_path = os.path.join(OUTPUT_DIR, f'8_误差分析_聚类组{cluster}_Sobol 指数.png')
        plt.savefig(plot_path, dpi=300)
        plt.close()
        print(f"聚类组{cluster}的 Sobol 指数图已保存至：{plot_path}")

    # 保存结果
    all_sobol = []
    for cluster in sobol_indices:
        for f, val in sobol_indices[cluster].items():
            all_sobol.append({
                '聚类组': cluster,
                '特征': f,
                'Sobol 指数': val
            })
    if all_sobol:
        result_df = pd.DataFrame(all_sobol)
        output_path = os.path.join(OUTPUT_DIR, '8_误差分析_Sobol 指数结果.xlsx')
        result_df.to_excel(output_path, index=False)
        print(f"Sobol 指数结果已保存至：{output_path}")

    print("误差分析完成")
    return sobol_indices


# 辅助函数：计算特征组合的效应
  def  _compute_effects(samples,  cluster,  base_features,  model,  base_t,  t_range,
risk_values):
    n_samples = len(samples)
    n_timesteps = len(t_range)
```

```
    features = list(base_features.keys())
    n_features = len(features)

    base_vals = np.array([base_features[f] for f in features])

    features_array = np.zeros((n_samples, n_timesteps, n_features + 1))
    features_array[:, :, 0] = t_range  # 孕周特征

    for i in range(n_samples):
        features_array[i, :, 1:] = base_vals + samples[i]

    flat_features = features_array.reshape(-1, n_features + 1)

    props = model.predict(flat_features)
    props = np.clip(props, 0.01, 0.99)
    props_array = props.reshape(n_samples, n_timesteps)
    objectives = 0.3 * risk_values + 0.7 * (1 - props_array)
    best_indices = np.argmin(objectives, axis=1)
    best_t_new = t_range[best_indices].round(2)
    return best_t_new - base_t


# ------------------------- 主函数 -------------------------
if __name__ == "__main__":
    try:
        # 加载数据
        preprocessed_path = os.path.join(OUTPUT_DIR, '1_预处理后数据.xlsx')
        preprocessed_data = pd.read_excel(preprocessed_path)
        print(f"成功加载预处理数据：{preprocessed_path}，共{len(preprocessed_data)}条")

        # 特征统计与相关性
        print("\n特征统计分布：")
        for col in ['年龄', '体重', '孕妇BMI', 'Y染色体浓度']:
            desc = preprocessed_data[col].describe()
            print(f"\n{col}：")
                 print(f"  均值：{desc['mean']:.2f}，标准差：{desc['std']:.2f}，变异系数：
{desc['std'] / desc['mean']:.2f}")
            print(f"  最小值：{desc['min']:.2f}，最大值：{desc['max']:.2f}")

        print("\nY染色体浓度与特征的相关性：")
        for col in ['年龄', '体重', '孕妇BMI']:
            corr = preprocessed_data[[col, 'Y染色体浓度']].corr().iloc[0, 1]
            print(f"{col}与Y染色体浓度的相关系数：{corr:.4f}")
        cluster_result_path = os.path.join(OUTPUT_DIR, '4_聚类结果数据.xlsx')
        cluster_result = pd.read_excel(cluster_result_path)
        print(f"成功加载聚类结果：{cluster_result_path}，共{len(cluster_result)}条")

        best_k = len(cluster_result['聚类标签'].unique())
        print(f"最佳聚类数：{best_k}")

        # 加载最佳时点结果
        optimal_timing_path = os.path.join(OUTPUT_DIR, '6_最佳时点结果.xlsx')
        optimal_timing = pd.read_excel(optimal_timing_path)
        print(f"成功加载最佳时点结果：{optimal_timing_path}，共{len(optimal_timing)}条")

        # 加载或训练随机森林回归模型
        regression_models = {}
        for cluster in range(best_k):
                                                                    model_path      =
os.path.join(OUTPUT_DIR,f'random_forest_model_cluster_{cluster}.pkl')
            if os.path.exists(model_path):
                model = joblib.load(model_path)
                regression_models[cluster] = model
            else:
                print(f"随机森林模型文件 {model_path} 不存在，重新训练模型")
                cluster_subset = preprocessed_data.copy()
```

```
                cluster_subset = cluster_subset.merge(cluster_result[['孕妇代码', '聚类标签']], on='孕妇代码',
                                                      how='left')
                cluster_subset = cluster_subset[cluster_subset['聚类标签'] == cluster]
                if cluster_subset.empty:
                    print(f"警告：聚类组{cluster}无有效数据，跳过")
                    continue

                # 计算达标比例
                        cluster_subset['达标比例'] = (cluster_subset['Y 染色体浓度'] >= 0.04).astype(float)
                if len(cluster_subset) < 10:
                    print(f"聚类组{cluster}样本量不足（<10），跳过训练")
                    continue

                # 模型参数
                X = cluster_subset[['孕周数值', '年龄', '体重', '孕妇 BMI']].values
                y = cluster_subset['达标比例'].values

                model = RandomForestRegressor(
                    n_estimators=70,
                    max_depth=7,
                    min_samples_split=6,
                    random_state=42,
                    n_jobs=1
                )
                model.fit(X, y)
                regression_models[cluster] = model
                joblib.dump(model, model_path)
                print(f"已重新训练并保存聚类组{cluster}的随机森林模型至 {model_path}")

    except Exception as e:
        print(f"数据加载失败：{e}")
        preprocessed_data = None
        cluster_result = None
        best_k = None
        optimal_timing = None
        regression_models = {}
     if preprocessed_data is not None and cluster_result is not None and best_k is not None and optimal_timing is not None and regression_models:
        error_analysis(cluster_result, optimal_timing, regression_models, best_k)
    else:
        print("缺少必要数据，无法执行误差分析")

```

附录 12

介绍：问题四： 基于 python 代码实现的模型训练

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
import pickle
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, ConfusionMatrixDisplay,
                             roc_curve)
from imblearn.over_sampling import SMOTE
from DatasetDivide import prepare_data, read_selected_features, split_and_scale

plt.rcParams["font.family"] = ["SimHei", "sans-serif"]
plt.rcParams['axes.unicode_minus'] = False

```

```python
def train_models(X_train, y_train, selected_features):

    models = {}
    for label in X_train.keys():
        if not selected_features[label]:
            continue

        pos_count = y_train[label].sum()
        pos_ratio = pos_count / len(y_train[label])
         print(f"\n{label}训练集: 总样本数={len(y_train[label])}, 正例数={pos_count}, 占比={pos_ratio:.2%}")

        if pos_ratio < 0.05:
            smote = SMOTE(sampling_strategy=0.2, random_state=42)
        elif pos_ratio < 0.1:
            smote = SMOTE(sampling_strategy=0.3, random_state=42)
        else:
            smote = SMOTE(sampling_strategy=0.5, random_state=42)

        X_resampled, y_resampled = smote.fit_resample(X_train[label], y_train[label])
        new_pos_ratio = y_resampled.sum() / len(y_resampled)
           print(f"过采样后: 总样本数={len(X_resampled)}, 正例数={y_resampled.sum()}, 占比={new_pos_ratio:.2%}")

        lr = LogisticRegression(
            class_weight='balanced',
            random_state=42,
            max_iter=2000,
            C=0.05
        )
        svm = SVC(
            kernel='rbf',
            probability=True,
            class_weight='balanced',
            random_state=42,
            C=0.5,
            gamma='scale'
        )
        rf = RandomForestClassifier(
            n_estimators=150,
            class_weight='balanced_subsample',
            random_state=42,
            min_samples_leaf=3,
            max_depth=8,
            bootstrap=True
        )

        lr.fit(X_resampled, y_resampled)
        svm.fit(X_resampled, y_resampled)
        rf.fit(X_resampled, y_resampled)

        models[label] = {'lr': lr, 'svm': svm, 'rf': rf}
        print(f"{label}模型训练完成")

    return models


def get_thresholds(models, X_train, y_train, output_dir):
    thresholds = {'lr': {}, 'svm': {}, 'rf': {}}

    for label in models.keys():
        # 逻辑回归阈值
        lr_probs = models[label]['lr'].predict_proba(X_train[label])[:, 1]
        pos_lr_probs = lr_probs[y_train[label] == 1]
        thresholds['lr'][label] = np.percentile(pos_lr_probs, 10) if len(pos_lr_probs) > 0 else 0.3

        # SVM: 通过 ROC 曲线找到最佳阈值
```

```
        svm_probs = models[label]['svm'].predict_proba(X_train[label])[:, 1]
        y_true = y_train[label].values

        # 计算 ROC 曲线的 FPR、TPR 和阈值
        fpr, tpr, svm_thresholds = roc_curve(y_true, svm_probs)

        # 计算每个点到对角线的距离
        if len(fpr) > 0 and len(tpr) > 0:
            distances = tpr - fpr
            best_idx = np.argmax(distances)
            best_svm_threshold = svm_thresholds[best_idx]

            plt.figure(figsize=(8, 6))
            plt.plot(fpr, tpr, color='blue', label='SVM ROC 曲线')
            plt.plot([0, 1], [0, 1], 'k--', label='随机猜测')
            plt.scatter(
                fpr[best_idx], tpr[best_idx],
                color='red', s=100, label=f'最佳阈值: {best_svm_threshold:.3f}'
            )
            plt.xlabel('假正例率 (FPR)')
            plt.ylabel('真正例率 (TPR)')
            plt.title(f'{label}的 SVM ROC 曲线与最佳阈值')
            plt.legend()
            plt.grid(alpha=0.3)
            plt.savefig(os.path.join(output_dir, f'4_{label}_SVM_ROC 曲线.png'), dpi=300)
            plt.close()
        else:
            best_svm_threshold = 0.3

        thresholds['svm'][label] = best_svm_threshold

        # 随机森林阈值
        rf_probs = models[label]['rf'].predict_proba(X_train[label])[:, 1]
        pos_rf_probs = rf_probs[y_train[label] == 1]
        thresholds['rf'][label] = np.percentile(pos_rf_probs, 10) if len(pos_rf_probs) >
0 else 0.3

                                print(f"{label} 阈 值 :  lr={thresholds['lr'][label]:.3f},
svm={thresholds['svm'][label]:.3f}, "
              f"rf={thresholds['rf'][label]:.3f}")

    return thresholds


def evaluate_models(models, X_test, y_test, y, selected_features, output_dir,
thresholds):
    """评估模型"""
    eval_results = {}
    for label in models.keys():
        if not selected_features[label]:
            continue

        test_indices = X_test[label].index
        y_true = y_test[label].values

        y_pred = []
        probs = []
        for idx in test_indices:
            X_sample = X_test[label].loc[[idx]]

            lr_prob = models[label]['lr'].predict_proba(X_sample)[0, 1]
            svm_prob = models[label]['svm'].predict_proba(X_sample)[0, 1]
            rf_prob = models[label]['rf'].predict_proba(X_sample)[0, 1]

            prob = 0.2 * lr_prob + 0.5 * svm_prob + 0.3 * rf_prob
            probs.append(prob)

```

```
                label_threshold = (thresholds['lr'][label] + thresholds['svm'][label] + thresholds['rf'][label]) / 3 * 0.8
            y_pred.append(1 if prob >= label_threshold else 0)

        accuracy = accuracy_score(y_true, y_pred)
        precision = precision_score(y_true, y_pred, zero_division=0)
        recall = recall_score(y_true, y_pred, zero_division=0)
        f1 = f1_score(y_true, y_pred, zero_division=0)

        eval_results[label] = {
            '准确率': accuracy,
            '精确率': precision,
            '召回率': recall,
            'F1 分数': f1
        }

        print(f"\n{label}测试集详细信息:")
        print(f"- 实际正例数: {sum(y_true)}, 预测正例数: {sum(y_pred)}")
        print(f"- 融合概率最大值: {max(probs):.3f}, 融合阈值: {label_threshold:.3f}")
        print(f"{label}评估结果: {eval_results[label]}")

        cm = confusion_matrix(y_true, y_pred)
         disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['正常', '异常'])
        disp.plot(cmap=plt.cm.Blues)
        plt.title(f'{label}的混淆矩阵')
        plt.savefig(os.path.join(output_dir, f'4_{label}_混淆矩阵.png'), dpi=300)
        plt.close()

          pd.DataFrame(eval_results).T.to_excel(os.path.join(output_dir, '4_模型评估结果.xlsx'))
    return eval_results

def main():
    X, y, label_cols, output_dir = prepare_data()
       selected_features = read_selected_features(os.path.join(output_dir, "2_筛选特征结果.xlsx"))
    X_train, X_test, y_train, y_test, scalers = split_and_scale(X, y, selected_features)

    models = train_models(X_train, y_train, selected_features)

    model_save_dir = os.path.join(output_dir, "trained_models")
    os.makedirs(model_save_dir, exist_ok=True)
    for label, model_dict in models.items():
        model_path = os.path.join(model_save_dir, f"{label}_model.pkl")
        with open(model_path, "wb") as f:
            pickle.dump(model_dict, f)
        print(f"模型 {label} 已保存至 {model_path}")

    thresholds = get_thresholds(models, X_train, y_train, output_dir)
        eval_results = evaluate_models(models, X_test, y_test, y, selected_features, output_dir, thresholds)
    print("\n模型评估完成，结果已保存至:", output_dir)

if __name__ == "__main__":
    main()
```

附录 13

介绍：问题四：基于 python 代码实现的数据集划分与标准化以及特征关联可视化

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
from scipy import stats

plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC", "sans-serif"]
plt.rcParams['axes.unicode_minus'] = False

# 1. 数据准备
def prepare_data(data_path="results/1_女胎预处理后数据.xlsx"):
    output_dir = "results"
    os.makedirs(output_dir, exist_ok=True)

    if not os.path.exists(data_path):
        raise FileNotFoundError(f"数据文件不存在，请检查路径：{data_path}")

    df = pd.read_excel(data_path)

    if '孕妇代码' not in df.columns:
        raise ValueError("数据中缺少'孕妇代码'列，请确保预处理数据包含该列")

    pregnant_ids = df['孕妇代码'].copy()

    label_cols = {
        'Y13': '13 号染色体异常',
        'Y18': '18 号染色体异常',
        'Y21': '21 号染色体异常'
    }

    features = df.drop(list(label_cols.keys()) + ['孕妇代码'], axis=1).columns.tolist()
    X = df[features]
    y = df[label_cols.keys()]

    return X, y, pregnant_ids, label_cols, output_dir

# 2. 从文件读取特征筛选结果
def read_selected_features(feature_result_path):
    """从文件中读取特征筛选结果"""
    if not os.path.exists(feature_result_path):
        raise FileNotFoundError(f"特征筛选结果文件不存在，请检查路径：{feature_result_path}")

    df = pd.read_excel(feature_result_path)
    selected_features = {}
    for col in df.columns:
        feats = df[col].dropna().tolist()
        if '13' in col:
            selected_features['Y13'] = feats
        elif '18' in col:
            selected_features['Y18'] = feats
        elif '21' in col:
            selected_features['Y21'] = feats
    return selected_features

# 3. 数据集划分与标准化
def split_and_scale(X, y, pregnant_ids, selected_features):
    X_train, X_test = {}, {}
    y_train, y_test = {}, {}
    ids_train, ids_test = {}, {}
    scalers = {}

    for label in selected_features.keys():
        X_label = X[selected_features[label]]

        X_tr, X_te, y_tr, y_te, ids_tr, ids_te = train_test_split(
            X_label, y[label], pregnant_ids,
            test_size=0.3,
            random_state=42,
            stratify=y[label]
        )

```

```
        scaler = StandardScaler()
        X_tr_scaled = scaler.fit_transform(X_tr)
        X_te_scaled = scaler.transform(X_te)
        X_train[label] = pd.DataFrame(
            X_tr_scaled,
            columns=X_tr.columns,
            index=X_tr.index
        )
        X_test[label] = pd.DataFrame(
            X_te_scaled,
            columns=X_te.columns,
            index=X_te.index
        )
        y_train[label] = y_tr
        y_test[label] = y_te
        ids_train[label] = ids_tr
        ids_test[label] = ids_te
        scalers[label] = scaler

    return X_train, X_test, y_train, y_test, ids_train, ids_test, scalers

# 4. 保存划分结果
def save_split_scaled_data(X_train, X_test, y_train, y_test, ids_train, ids_test, scalers, output_dir):
    split_dir = os.path.join(output_dir, "split_scaled_data")
    os.makedirs(split_dir, exist_ok=True)

    for label in X_train.keys():
        train_data = pd.concat([
            ids_train[label].reset_index(drop=True),
            X_train[label].reset_index(drop=True)
        ], axis=1)
        test_data = pd.concat([
            ids_test[label].reset_index(drop=True),
            X_test[label].reset_index(drop=True)
        ], axis=1)

            train_data.to_excel(os.path.join(split_dir, f"{label}_训练集特征.xlsx"), index=False)
            test_data.to_excel(os.path.join(split_dir, f"{label}_测试集特征.xlsx"), index=False)

        train_label = pd.concat([
            ids_train[label].reset_index(drop=True),
            y_train[label].reset_index(drop=True)
        ], axis=1)
        test_label = pd.concat([
            ids_test[label].reset_index(drop=True),
            y_test[label].reset_index(drop=True)
        ], axis=1)
           train_label.to_excel(os.path.join(split_dir, f"{label}_训练集标签.xlsx"), index=False)
            test_label.to_excel(os.path.join(split_dir, f"{label}_测试集标签.xlsx"), index=False)

        scaler_params = pd.DataFrame({
            "特征": X_train[label].columns,
            "均值": scalers[label].mean_,
            "标准差": scalers[label].scale_
        })
         scaler_params.to_excel(os.path.join(split_dir, f"{label}_标准化参数.xlsx"), index=False)

    print(f"含孕妇代码的数据集已保存至: {split_dir}")

# 5. 特征关联可视化
def plot_feature_correlation(X, y, label, features, output_dir):
    if not features:
```

```
        return

    y_numeric = y.astype(int).values
    n_cols = 2
    n_rows = (len(features) + n_cols - 1) // n_cols
    fig, axes = plt.subplots(n_rows, n_cols * 2, figsize=(16, 5 * n_rows))
    axes = axes.flatten()

    for i, feat in enumerate(features):
        X_feat = X[feat].values

        ax_scatter = axes[i * 2]
        sns.regplot(
            x=X_feat,
            y=y_numeric,
            ax=ax_scatter,
            scatter_kws={'alpha': 0.5},
            line_kws={'color': 'red', 'lw': 2}
        )
        ax_scatter.set_title(f'{feat}与{label}的散点图（回归线）')
        ax_scatter.set_xlabel(feat)
        ax_scatter.set_ylabel(f'{label}（0=正常，1=异常）')
        ax_scatter.set_yticks([0, 1])
        ax_box = axes[i * 2 + 1]
        sns.boxplot(
            x=y,
            y=X_feat,
            ax=ax_box
        )
        pearson_r, _ = stats.pearsonr(X_feat, y_numeric)
        spearman_r, _ = stats.spearmanr(X_feat, y_numeric)
        ax_box.set_title(
            f'{feat}与{label}的箱线图\n'
            f'皮尔逊系数: {pearson_r:.3f} 斯皮尔曼系数: {spearman_r:.3f}'
        )
        ax_box.set_xlabel(f'{label}（0=正常，1=异常）')
        ax_box.set_ylabel(feat)

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'3_{label}_特征关联类型分析.png'), dpi=300)
    plt.close()


def main():

    X, y, pregnant_ids, label_cols, output_dir = prepare_data()
    print("数据准备完成，特征数量: ", X.shape[1])
    print("标签列: ", label_cols)
    print("孕妇代码数量: ", len(pregnant_ids.unique()))

    feature_result_path = os.path.join(output_dir, "2_筛选特征结果.xlsx")
    selected_features = read_selected_features(feature_result_path)
    for label, feats in selected_features.items():
        print(f"针对{label_cols[label]}的特征: {feats}")

    X_train, X_test, y_train, y_test, ids_train, ids_test, scalers = split_and_scale(
        X, y, pregnant_ids, selected_features
    )
    print("数据集划分完成，训练集样本数: ", {k: v.shape[0] for k, v in X_train.items()})

    save_split_scaled_data(
        X_train, X_test, y_train, y_test, ids_train, ids_test, scalers, output_dir
    )

    for label, label_desc in label_cols.items():
        if selected_features[label]:
            plot_feature_correlation(
```

```
                X_train[label],
                y_train[label],
                label_desc,
                selected_features[label],
                output_dir
            )
    print(f"特征关联分析图已保存至：{output_dir}")

if __name__ == "__main__":
    main()
```

附录 14

介绍：问题四：基于 python 语言实现的数据预处理

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import os

output_dir = "results"
os.makedirs(output_dir, exist_ok=True)

# 1. 数据读取与筛选（女胎样本）
excel_file = pd.ExcelFile('C 题数据.xlsx')

df_female = excel_file.parse('女胎检测数据')

rows, columns = df_female.shape

# 2. 数据预处理
def convert_gestational_week(week_str):
    if pd.isna(week_str):
        return np.nan
    week_str = str(week_str).lower().replace(' ', '')
    if 'w' in week_str:
        parts = week_str.split('w')
        weeks = float(parts[0])
        days = 0.0
        if '+' in parts[1]:
            days = float(parts[1].split('+')[1])
        return weeks + days / 7
    return float(week_str)


df_female['孕周数值'] = df_female['检测孕周'].apply(convert_gestational_week)

gc_cols = [
    'GC 含量'
]

mask_gc = (df_female[gc_cols] >= 0.4).all(axis=1) & (df_female[gc_cols] <= 0.6).all(axis=1)
df_female = df_female[mask_gc].dropna(subset=gc_cols + ['孕周数值', '孕妇 BMI', '年龄'])

# 3.AB 列拆解为 Y13、Y18、Y21

def process_ab(ab_value):
    """将 AB 列的非整倍体结果拆解为三个二元标签"""
    y13, y18, y21 = 0, 0, 0
    if pd.isna(ab_value):
        return (y13, y18, y21)
    ab_str = str(ab_value).upper()
    if 'T13' in ab_str:
        y13 = 1
    if 'T18' in ab_str:
        y18 = 1
    if 'T21' in ab_str:
```

```
        y21 = 1
    return (y13, y18, y21)

ab_labels = df_female['染色体的非整倍体'].apply(lambda x: pd.Series(process_ab(x), index=['Y13', 'Y18', 'Y21']))
df_female = pd.concat([df_female, ab_labels], axis=1)

# 4. 特征工程
z_features = df_female[
    ['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']
].copy()
z_features.columns = ['Z13', 'Z18', 'Z21', 'ZX']

gc_deviation = pd.DataFrame()
for col, name in zip(gc_cols, ['GC13', 'GC18', 'GC21']):
    )
    gc_deviation[f'{name}_偏差'] = (df_female[col] - 0.5) / 0.1

# 4.3 读段数相关比例特征
read_features = pd.DataFrame()
read_features['唯一比对的读段占比'] = df_female['唯一比对的读段数'] / df_female['原始读段数']
read_features['重复读段比例'] = df_female['重复读段的比例']
read_features['比对比例'] = df_female['在参考基因组上比对的比例']

scaler = StandardScaler()
maternal_features = df_female[['年龄', '孕周数值', '孕妇BMI']].copy()
maternal_scaled = pd.DataFrame(
    scaler.fit_transform(maternal_features),
    columns=['年龄_标准化', '孕周_标准化', 'BMI_标准化'],
    index=maternal_features.index
)
# 5. 合并特征、标签与孕妇代码，输出最终数据集
pregnant_id = df_female[['孕妇代码']].copy()

final_features = pd.concat([pregnant_id, z_features, gc_deviation, read_features, maternal_scaled], axis=1)
final_df = pd.concat([final_features, df_female[['Y13', 'Y18', 'Y21']]], axis=1)
print(f"处理后女胎样本量：{final_df.shape[0]}")
print("特征列：", final_df.columns.tolist())

required_columns = [
    '孕妇代码',
    '唯一比对的读段数', '原始读段数', '重复读段的比例',
    '在参考基因组上比对的比例', '年龄', '孕周数值', '孕妇BMI'
]

missing_columns = [col for col in required_columns if col not in df_female.columns]
if missing_columns:
    raise ValueError(f"缺少必要的列：{missing_columns}")

null_columns = df_female[required_columns].columns[df_female[required_columns].isnull().any()].tolist()
if null_columns:
    print(f"警告：以下列存在空值：{null_columns}")

if not final_df.empty:
    if final_df.isin([np.inf, -np.inf]).any().any():
        print("警告：数据中存在无穷值，已替换为NaN")
        final_df = final_df.replace([np.inf, -np.inf], np.nan)

    output_path = os.path.join(output_dir, '1_女胎预处理后数据.xlsx')
    final_df.to_excel(output_path, index=False)
```

```
    print(f"预处理后的数据已保存至: {output_path}")
else:
    print("警告：处理后的数据集为空，未保存文件")
```

附录 15

介绍：问题四：基于 python 语言实现的特征筛选

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import kruskal
from sklearn.feature_selection import mutual_info_classif

plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC"]
plt.rcParams['axes.unicode_minus'] = False


def prepare_data(data_path="results/1_女胎预处理后数据.xlsx"):
    output_dir = "results"
    os.makedirs(output_dir, exist_ok=True)

    df = pd.read_excel(data_path)

    label_cols = {
        'Y13': '13 号染色体异常',
        'Y18': '18 号染色体异常',
        'Y21': '21 号染色体异常'
    }

    features = df.drop(label_cols.keys(), axis=1).columns.tolist()
    X = df[features]
    y = df[label_cols.keys()]

    return X, y, label_cols, output_dir


def select_features(X, y_label, label_name, mi_percentile=30):
    # 非参数检验（Kruskal-Wallis）
    p_values = []
    for col in X.columns:
        groups = [X[col][y_label == cls] for cls in y_label.unique()]
        _, p = kruskal(*groups)
        p_values.append(p)
    p_values = np.array(p_values)
    kruskal_significant = p_values < 0.1

    # 互信息计算（动态阈值）
    mi_scores = mutual_info_classif(X, y_label)
    mi_threshold = np.percentile(mi_scores, 100 - mi_percentile)
    mi_significant = mi_scores >= mi_threshold

    selected_mask = kruskal_significant | mi_significant
    selected_feats = X.columns[selected_mask].tolist()
    print(f"针对{label_name}筛选出{len(selected_feats)}个特征: {selected_feats}")
    return selected_feats


def save_results(results, output_dir, filename="2_筛选特征结果.xlsx"):
    all_feats = [feats for _, feats in results.values()]
    max_len = max(len(feats) for feats in all_feats)

    data = {}
    for label, (label_desc, feats) in results.items():
        padded_feats = feats + [np.nan] * (max_len - len(feats))
        data[label_desc] = padded_feats
```

```
60.
61.     df_results = pd.DataFrame(data)
62.
63.     save_path = os.path.join(output_dir, filename)
64.     df_results.to_excel(save_path, index=False)
65.     print(f"结果已保存至: {save_path}")
66.
67.
68. def main():
69.     X, y, label_cols, output_dir = prepare_data()
70.
71.     selected_features = {}
72.     for label, label_desc in label_cols.items():
73.         selected_feats = select_features(X, y[label], label_desc)
74.         selected_features[label] = (label_desc, selected_feats)
75.
76.     save_results(selected_features, output_dir)
77.
78. if __name__ == "__main__":
79.     main()
```

附录 16

介绍：问题四：基于 python 语言实现的模型预测训练

```
 1. import pandas as pd
 2. import pickle
 3. import os
 4.
 5. from matplotlib import pyplot as plt
 6. from sklearn.preprocessing import StandardScaler
 7.
 8. plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC"]
 9. plt.rcParams['axes.unicode_minus'] = False
10.
11.
12. def load_training_assets(output_dir="results"):
13.     assets = {}
14.
15.     feat_path = os.path.join(output_dir, "2_筛选特征结果.xlsx")
16.     df_feats = pd.read_excel(feat_path)
17.     selected_features = {}
18.     for col in df_feats.columns:
19.         feats = df_feats[col].dropna().tolist()
20.         if '13' in col:
21.             selected_features['Y13'] = feats
22.         elif '18' in col:
23.             selected_features['Y18'] = feats
24.         elif '21' in col:
25.             selected_features['Y21'] = feats
26.     assets['selected_features'] = selected_features
27.
28.     scalers = {}
29.     scaler_dir = os.path.join(output_dir, "split_scaled_data")
30.     for label in selected_features.keys():
31.         scaler_path = os.path.join(scaler_dir, f"{label}_标准化参数.xlsx")
32.         df_scaler = pd.read_excel(scaler_path)
33.         scaler = StandardScaler()
34.         scaler.mean_ = df_scaler['均值'].values
35.         scaler.scale_ = df_scaler['标准差'].values
36.         scaler.feature_names_in_ = df_scaler['特征'].values
37.         scalers[label] = scaler
38.     assets['scalers'] = scalers
39.
40.     models = {}
41.     model_dir = os.path.join(output_dir, "trained_models")
```

```
    for label in selected_features.keys():
        with open(os.path.join(model_dir, f"{label}_model.pkl"), 'rb') as f:
            models[label] = pickle.load(f)
    assets['models'] = models

eval_path = os.path.join(output_dir, "4_模型评估结果.xlsx")
    assets['eval_results'] = pd.read_excel(eval_path)

    return assets


def predict_with_fusion(X_sample, model, label, sample_type):
    lr_prob = model['lr'].predict_proba(X_sample)[:, 1]
    svm_prob = model['svm'].predict_proba(X_sample)[:, 1]
    rf_prob = model['rf'].predict_proba(X_sample)[:, 1]

    if sample_type == 'single':
        return 0.6 * rf_prob + 0.4 * svm_prob
    else:
        return 0.7 * svm_prob + 0.3 * rf_prob


def predict_female_fetus(new_data, assets):
    selected_features = assets['selected_features']
    scalers = assets['scalers']
    models = assets['models']

    feature_cols = [col for col in new_data.columns if col != '孕妇代码']
    processed = {}
    for label in selected_features.keys():
        required_feats = selected_features[label]
        missing = [f for f in required_feats if f not in feature_cols]
        if missing:
            raise ValueError(f"新样本缺少特征：{missing}（{label}模型所需）")

        X_new = new_data[required_feats].copy()
        X_scaled = scalers[label].transform(X_new)
        processed[label] = pd.DataFrame(X_scaled, columns=required_feats)

    pred_probs = {}
    for label in selected_features.keys():
        temp_pred = []
        for l in selected_features.keys():
            if l != label:
                prob = predict_with_fusion(processed[l], models[l], l, 'single')
                temp_pred.append((prob >= 0.4).astype(int))
        cnt = sum(temp_pred)
        sample_type = ['single' if c <= 1 else 'combined' for c in cnt]

        all_probs = []
        for i in range(len(sample_type)):
            if sample_type[i] == 'single':
               prob=0.6* models[label]['rf'].predict_proba(processed[label].iloc[[i]])[0,
1]+0.4 * models[label]['svm'].predict_proba(processed[label].iloc[[i]])[0, 1]
            else:
               prob=0.7*models[label]['svm'].predict_proba(processed[label].iloc[[i]])[0,
1]+0.3*models[label]['rf'].predict_proba(processed[label].iloc[[i]])[0, 1]
            all_probs.append(prob)
        pred_probs[label] = all_probs

    thresholds = {
        'Y13': 0.35,
        'Y18': 0.30,
        'Y21': 0.25
    }

    results = []
    for i in range(len(new_data)):
```

```
        type = []
        for label in selected_features.keys():
            if pred_probs[label][i] >= thresholds[label]:
                type.append(f"{label[1:]}号染色体")
        if type:
            results.append(f"女胎异常，可能存在 {', '.join(type)} 非整倍体")
        else:
            results.append("女胎正常")
    return results


def main():
    assets = load_training_assets()

    df_Y13 = pd.read_excel('results/split_scaled_data/Y13_测试集特征.xlsx')
    df_Y18 = pd.read_excel('results/split_scaled_data/Y18_测试集特征.xlsx')
    df_Y21 = pd.read_excel('results/split_scaled_data/Y21_测试集特征.xlsx')

    new_sample = pd.concat([df_Y13, df_Y18, df_Y21], axis=1)
    new_sample = new_sample.loc[:, ~new_sample.columns.duplicated()]
    if '孕妇代码' not in new_sample.columns:
        raise ValueError("测试集特征数据中缺少'孕妇代码'列，请确保数据包含该列")

    pregnant_ids = new_sample['孕妇代码'].tolist()

    pred_results = predict_female_fetus(new_sample, assets)

    result_df = pd.DataFrame({
        '孕妇代码': pregnant_ids,
        '预测结果': pred_results
    })
    output_path = os.path.join("results", "5_女胎异常预测结果.xlsx")
    result_df.to_excel(output_path, index=False)
    print(f"预测结果已保存至：{output_path}")


if __name__ == "__main__":
    main()
```

附录 17

介绍：问题四：基于 python 语言实现的正确率验证

```
import pandas as pd
import os

from matplotlib import pyplot as plt
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pickle
from sklearn.preprocessing import StandardScaler

plt.rcParams["font.family"] = ["SimHei", "WenQuanYi Micro Hei", "Heiti TC"]
plt.rcParams['axes.unicode_minus'] = False


def load_test_data_and_assets(output_dir="results"):
    assets = {}

    feat_path = os.path.join(output_dir, "2_筛选特征结果.xlsx")
    df_feats = pd.read_excel(feat_path)
    selected_features = {}
    for col in df_feats.columns:
        feats = df_feats[col].dropna().tolist()
        if '13' in col:
            selected_features['Y13'] = feats
        elif '18' in col:
            selected_features['Y18'] = feats
```

```
        elif '21' in col:
            selected_features['Y21'] = feats
    assets['selected_features'] = selected_features

    scalers = {}
    scaler_dir = os.path.join(output_dir, "split_scaled_data")
    for label in selected_features.keys():
        scaler_path = os.path.join(scaler_dir, f"{label}_标准化参数.xlsx")
        df_scaler = pd.read_excel(scaler_path)
        scaler = StandardScaler()
        scaler.mean_ = df_scaler['均值'].values
        scaler.scale_ = df_scaler['标准差'].values
        scaler.feature_names_in_ = df_scaler['特征'].values
        scalers[label] = scaler
    assets['scalers'] = scalers

    models = {}
    model_dir = os.path.join(output_dir, "trained_models")
    for label in selected_features.keys():
        with open(os.path.join(model_dir, f"{label}_model.pkl"), 'rb') as f:
            models[label] = pickle.load(f)
    assets['models'] = models

    test_data = {}
    test_labels = {}
    for label in selected_features.keys():
        feat_path = os.path.join(scaler_dir, f"{label}_测试集特征.xlsx")
        test_data[label] = pd.read_excel(feat_path)
        label_path = os.path.join(scaler_dir, f"{label}_测试集标签.xlsx")
        test_labels[label] = pd.read_excel(label_path)

    return assets, test_data, test_labels


def predict_with_fusion(X_sample, model, sample_type):
    lr_prob = model['lr'].predict_proba(X_sample)[:, 1]
    svm_prob = model['svm'].predict_proba(X_sample)[:, 1]
    rf_prob = model['rf'].predict_proba(X_sample)[:, 1]
    if sample_type == 'single':
        return 0.6 * rf_prob + 0.4 * svm_prob
    else:
        return 0.7 * svm_prob + 0.3 * rf_prob


def calculate_accuracy(assets, test_data, test_labels):
    selected_features = assets['selected_features']
    scalers = assets['scalers']
    models = assets['models']
    thresholds = {'Y13': 0.35, 'Y18': 0.30, 'Y21': 0.25}
    metrics = {}

    for label in selected_features.keys():
        df_feat = test_data[label].copy()
        feature_cols = [col for col in df_feat.columns if col != '孕妇代码']
        X_test = df_feat[feature_cols]

        X_scaled = scalers[label].transform(X_test[selected_features[label]])

        temp_pred = []
        for l in selected_features.keys():
            if l != label:
                l_feat = test_data[l][selected_features[l]]
                l_scaled = scalers[l].transform(l_feat)
                prob = predict_with_fusion(l_scaled, models[l], 'single')
                temp_pred.append((prob >= 0.4).astype(int))
        cnt = sum(temp_pred)
        sample_type = ['single' if c <= 1 else 'combined' for c in cnt]

```

```
        pred_probs = []
        for i in range(len(sample_type)):
            if sample_type[i] == 'single':
                prob = 0.6 * models[label]['rf'].predict_proba(X_scaled[i:i + 1])[0, 1] +
\
                       0.4 * models[label]['svm'].predict_proba(X_scaled[i:i + 1])[0, 1]
            else:
                prob = 0.7 * models[label]['svm'].predict_proba(X_scaled[i:i + 1])[0, 1]
+ \
                       0.3 * models[label]['rf'].predict_proba(X_scaled[i:i + 1])[0, 1]
            pred_probs.append(prob)
        y_pred = [1 if p >= thresholds[label] else 0 for p in pred_probs]

        df_true = test_labels[label].copy()
        pred_df = pd.DataFrame({
            '孕妇代码': df_feat['孕妇代码'],
            'y_pred': y_pred
        })
        merged = pd.merge(pred_df, df_true, on='孕妇代码', how='inner')
        y_true = merged[label].values
        y_pred_aligned = merged['y_pred'].values

        metrics[label] = {
            '准确率(Accuracy)': accuracy_score(y_true, y_pred_aligned),
        }

    return metrics


def save_accuracy_results(metrics, output_dir="results"):
    result_df = pd.DataFrame(metrics).T
    output_path = os.path.join(output_dir, "6_模型准确率评估结果.xlsx")
    result_df.to_excel(output_path)
    print(f"模型准确率评估结果已保存至：{output_path}")
    print("\n 模型准确率 summary：")
    print(result_df['准确率(Accuracy)'].round(4))


def main():
    assets, test_data, test_labels = load_test_data_and_assets()
    metrics = calculate_accuracy(assets, test_data, test_labels)
    save_accuracy_results(metrics)


if __name__ == "__main__":
    main()
```