出血性脑卒中临床智能诊疗优化模型

摘　要

在现代医学临床技术中，通过分析临床数据，我们可以对血性脑卒中患者的血肿扩张风险、血肿周围水肿的发生和演进规律进行评估，并结合临床和影像信息来预测患者的临床预后。本研究采用了逻辑回归模型，并结合决策树和多项式回归的方法，建立了随机森林模型，通过考虑临床实际情况对模型进行优化，并提出相应的优化方案，最终得到了适用于一般情况的普适性模型。

针对问题一，我们首先根据提供的数据判断患者发病后 48 小时内是否发生了血肿扩张事件，并同时记录血肿扩张的发生情况。我们根据发病到首次影像检查的时间间隔和后续影像检查的时间间隔来确定是否在 48 小时内发生了血肿扩张。考虑到数据的连续性和稳定性，我们**采用贪心算法的思想来判断是否发生了血肿扩张，并通过建立逻辑回归模型来预测所有患者发生血肿扩张的概率**。逻辑回归模型通过使用 Sigmoid 函数将线性预测转换为概率值。

针对问题二，为了计算真实值与拟合曲线之间的残差，我们根据患者的水肿体积和发病到各个随访时间点的时间间隔构建了进展曲线，以探索患者水肿体积随时间变化的个体差异，并建立不同人群的进展曲线。根据构建的水肿体积随时间进展曲线，我们计算患者的真实值与曲线之间的残差。为了分析水肿体积进展模式的影响，我们**采用了多项式回归和随机森林模型**。考虑到患者个体差异，患者脑补血肿体积随时间进展可能不符合简单的多项式关系，因此我们最终使用随机森林模型进行曲线拟合，并作为最终预测结果的模型。

针对问题三，为了平衡个人史、疾病史和发病相关因素以及首次影像结果之间的关系，我们分别使用**逻辑回归模型、随机森林算法和 K 临近算法对患者的 mRS 评分进行预测。**根据已有的临床和治疗结果，我们预测了所有含有随访影像检查的患者在 90 天时的 mRS 评分。通过逻辑回归模型和分析预测结果，我们考虑到随机森林模型和 K 临近算法之间的精度，建议在对预后相关结果和因素进行准确预测时，可以适当采用随机森林模型以提高预测精度。

关键词：随机森林模型 逻辑回归模型 平均相对误差 多项式回归

目 录

摘　要.....2

一、问题重述.....4

1.1 问题背景.....4

1.2 问题提出.....4

二、模型假设.....6

2.1 模型的基本.....6

三、模型的符号说明.....7

四、问题一的模型建立与求解.....8

4.1 任务 a 的分析与处理.....8

4.1.1 任务 a 问题分析.....8

4.1.2 任务 a 特征提取及数据分析.....8

4.1.3 算法设计.....9

4.2 任务 b 的分析与处理.....9

4.2.1 任务 b 问题分析.....9

4.2.3 任务 b 模型的建立.....14

4.2.4 任务 b 模型结果.....14

五、问题二的模型建立与求解.....15

5.1 任务 a 的分析与处理.....15

5.2 任务 b 的分析与处理.....17

5.3 任务 c 的分析与处理.....19

5.4 任务 d 的分析与处理.....20

六、问题三的模型建立和求解.....21

6.1 问题 a 的分析与处理.....21

七、参考文献.....23

附录（代码）.....24

一、问题重述

1.1 问题背景

出血性脑卒中是指脑部血管破裂导致的脑内出血。它与缺血性脑卒中（脑血管阻塞）不同，属于一种较为严重的疾病。出血性脑卒中的常见原因包括高血压、脑动脉瘤、脑血管畸形和出血性疾病等。当血管破裂时，血液会进入脑组织，压迫周围健康脑组织并引起炎症反应，造成神经功能障碍。出血性脑卒中的症状可能会突然出现，如剧烈头痛、意识丧失、肢体无力、言语困难、视力改变等。如果怀疑发生了出血性脑卒中，应立即就医。治疗出血性脑卒中的方法包括控制高血压、手术修复破裂的血管、减轻颅内压力以及进行康复治疗等。同时，预防出血性脑卒中的关键是保持健康的生活方式，如保持血压正常、戒烟限酒、均衡饮食、适量运动、定期体检等。

出血性脑卒中指非外伤性脑实质内血管破裂引起的脑出血，占全部脑卒中发病率的 10-15%。其病因复杂，通常因脑动脉瘤破裂、脑动脉异常等因素，导致血液从破裂的血管涌入脑组织，从而造成脑部机械性损伤，并引发一系列复杂的生理病理反应。出血性脑卒中起病急、进展快，预后较差，急性期内病死率高达 45-50%，约 80%的患者会遗留较严重的神经功能障碍，为社会及患者家庭带来沉重的健康和经济负担。因此，发掘出血性脑卒中的发病风险，整合影像学特征、患者临床信息及临床诊疗方案，精准预测患者预后，并据此优化临床决策具有重要的临床意义。

出血性脑卒中后，血肿范围扩大是预后不良的重要危险因素之一。在出血发生后的短时间内，血肿范围可能因脑组织受损、炎症反应等因素逐渐扩大，导致颅内压迅速增加，从而引发神经功能进一步恶化，甚至危及患者生命。因此，监测和控制血肿的扩张是临床关注的重点之一。此外，血肿周围的水肿作为脑出血后继发性损伤的标志，在近年来引起了临床广泛关注。血肿周围的水肿可能导致脑组织受压，进而影响神经元功能，使脑组织进一步受损，进而加重患者神经功能损伤。综上所述，针对出血性脑卒中后的两个重要关键事件，即血肿扩张和血肿周围水肿的发生及发展，进行早期识别和预测对于改善患者预后、提升其生活质量具有重要意义。

事实上，出血性脑卒中出现的危险因素很多，我们可以联合患者个人信息、治疗方案和预后等数据，构建智能诊疗模型，明确因素，实现精准预测。相关研究成果及科学依据将能够进一步应用于临床实践，为改善出血性脑卒中患者预后作出贡献。

1.2 问题提出

本文以上述分析为背景，主要考虑以下几个问题

（1） 血脉扩张风险相关因素，根据血肿体积前后变化，后续检查比首次检查绝对体积增加或者相对体积增加判断是否发生血肿扩张。结合发病到首次影像

时间间隔和后续影像检查时间间隔判断当前影像检查是否在发病 48 小时内。并以此为目标变量，结合相关信息预测患者发生血肿扩张的概率。

（2） 结合水肿体积和重复检查时间点，构建一条全体患者水肿体积随时间进展曲线，计算真实值和所拟合曲线之间存在的残差。探索患者之间的个体差异，构建水肿体积随时间进展曲线，计算残差。观察残差的分布情况，采用方根误差或平均绝对误差评估拟合曲线的精度，较小的残差表示模型拟合效果较好。

（3） 通过拟合曲线分析不同的治疗方法对水肿体积进展模式的影响，我们可以通过逻辑回归模型分析血肿体积、水肿体积以及治疗方法三者之间的关系进一步探索关联关系。

（4） 基于前述模型以及算法分析，可以预测 mRS 结果，结合所有已知临床、治疗等的影像结果，预测所有含随访影像检查患者的 mRS 评分，构建出预测模型，并分析患者预后的影像特征等关联关系。

二、模型假设

2.1 模型的基本

在本文的研究中心，采用均匀先验理论所需要进行的问题假设：
（1）假设所有患者均发生血肿扩张，即所有患者的后续检查比首次检查绝对体积增加大于等于 6mL 或相对体积增加大于等于 33%。
（2）假设患者水肿体积随时间进展模式的差异均相同。
（3）假设所有含随访影像检查的患者 90 天 mRS 评分分布符合正态分布

三、模型的符号说明

表 3-1 模型符号说明

序号	符号	说明
1	$g, grad$	图像平均梯度
2	$c, cons$	图像对比度
3	μ	图像均值
4	ε	拟合残差
5	$\hbar$	时间
6	σ	图像均方差
7	Reg	图像子窗格
8	s	信号强度特征
9	y	实际观测值
10	$\hat{y}$	预测值

四、问题一的模型建立与求解

4.1 任务 a 的分析与处理

4.1.1 任务 a 问题分析

任务 a 要求判断患者是否在 48 小时内发生血肿扩张事件，若发生，则记为 1，同时记录从首次随访时间到发生血肿的时间；否则记为 0，同时发生血肿的时间记为 0 小时。

考虑到需要表 2 给出的数据种类众多，因而先提取数据。除此之外，由于需要记录发生血肿扩张的时间，因此需要从最近的随访时间开始遍历，故使用贪心算法的思想进行求解。

4.1.2 任务 a 特征提取及数据分析

首先需要对表 1 的数据进行特征提取，得到两列原始数据，注意到后续需要对数据时间进行操作，因此将表中数据转为时间戳，如表 4-1 所示。

表 4-1 表 1 特征提取后的前 5 行内容

入院首次影像检查流水号	发病到首次影像检查时间间隔
020161212002136	0 days 02:30:00
120160406002131	0 days 03:00:00
220160413000006	0 days 02:00:00
320161215001667	0 days 01:00:00
420161222000978	0 days 05:00:00

接着判断表 1 空数据的基本情况如表 4-2 所示

表 4-2 表 1 空数据的基本情况

入院首次影像检查流水号	0
发病到首次影像检查时间间隔	0
dtype:	int64
Null columns:	Index([],dtype='object')

通过上表可知表 1 特征提取后不包含空数据，结合表 1 两列特征的实际意义，也可得出该结论。

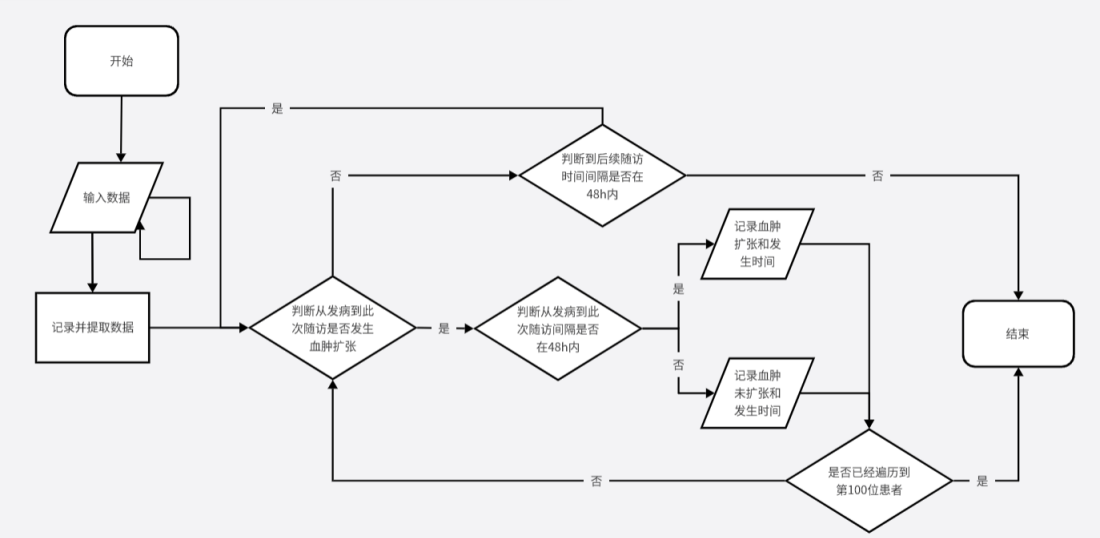
接着对表 2 的数据进行特征提取，得到问题所需的特征，提取结果如表 4-3 所示，不难发现，提取后的文件中包含大量空数据，考虑到空数据的实际意义和任务需求，故对空数据不做处理。

判断血肿发生的时间是否在 48h 内，需要查阅附表 1。在实现上，通过表 2 的流水号定位附表 1 对应的时间即可，因此问题 a 的数据处理已完成。

4.1.3 算法设计

任务 a 不仅需要判断是否发生血肿，还要记录血肿发生时间，因此考虑使用**贪心算法思想：对于每位患者，优先从最近的随访时间开始判断是否发生血肿，如果发生血肿，再判断时间是否在 48 小时内。**如果本次发生血肿的时间已经超过 48h，说明后续的随访必然会超过 48h。如果本次的随访时间在 48h 内，说明本次随访时间一定是最先发生随访的时间，即需要记录的时间，具体过程如图 4-1 所示。

图 4-1 任务 a 算法流程图



4.2 任务 b 的分析与处理

4.2.1 任务 b 问题分析

从问题 b 的描述中可知，我们要以是否发生血肿扩张事件为目标变量，基于患者的个人情况构建模型，来预测患者发生血肿扩张的概率。在这里我们通过逻辑回归的方法来预测概率。

4.2.2 任务 b 数据分析

首先对表 1 给出的原始数据进行数据分析，如表 4-4 所示：

表 4-3 表 1 原始数据的基本信息

#	列名	非空数据个数	数据类型
1	90 天 mRS	100	float64
2	数据集划分	160	object
3	入院首次影像检查流水号	160	int64
4	年龄	160	int64
5	性别	160	object

6	脑出血前 mRS 评分	160	int64
7	高血压病史	160	int64
8	卒中病史	160	int64
9	糖尿病史	160	int64
10	房颤史	160	int64
11	冠心病史	160	int64
12	吸烟史	160	int64
13	饮酒史	160	int64
14	发病到首次影像检查时间间隔	160	float64
15	血压	160	object
16	脑室引流	160	int64
17	止血治疗	160	int64
18	降颅压治疗	160	int64
19	降压治疗	160	int64
20	镇静、镇痛治疗	160	int64
21	止吐护胃	160	int64
22	营养神经	160	int64

根据题意，前三行数据不需要作为模型的输入，因此需要处理的列为“性别”和“血压”。将“性别”替换为布尔型，“血压”扩充为“高压”和“低压”。接下来对处理后的表 1 进行数据分析，先对统计各字段数据的类别如下图 4-2。

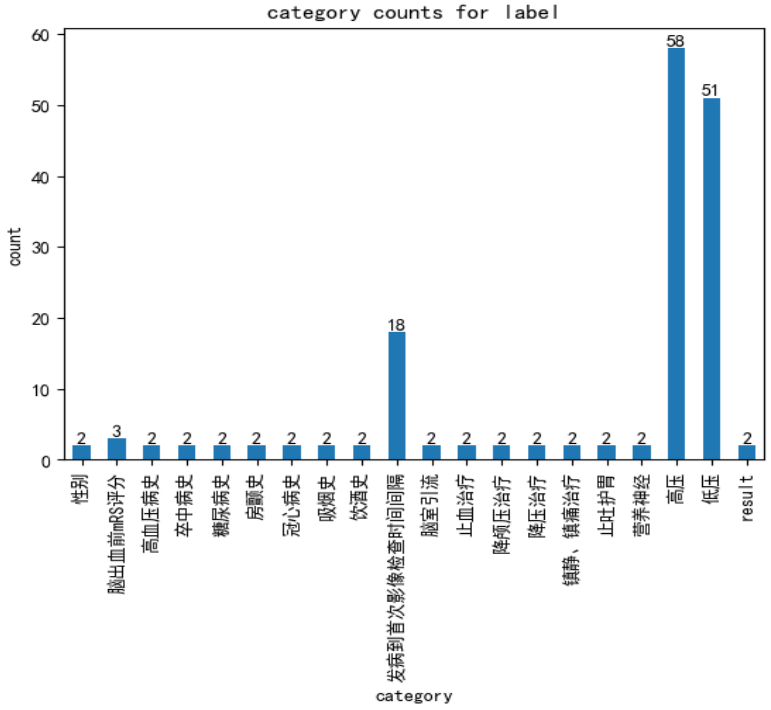


图 4-2 表 1 各字段类别统计图

“表 1”给出了较多的患者信息，通过上表得到不同字段的的个数，下图 4-3 各字段的分布情况。

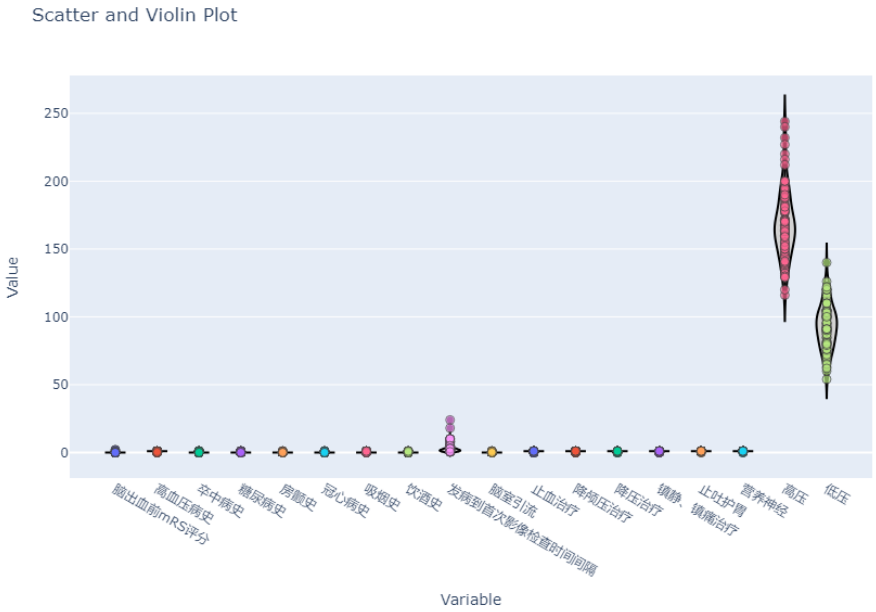


图 4-3 表 1 各字段数据分布图

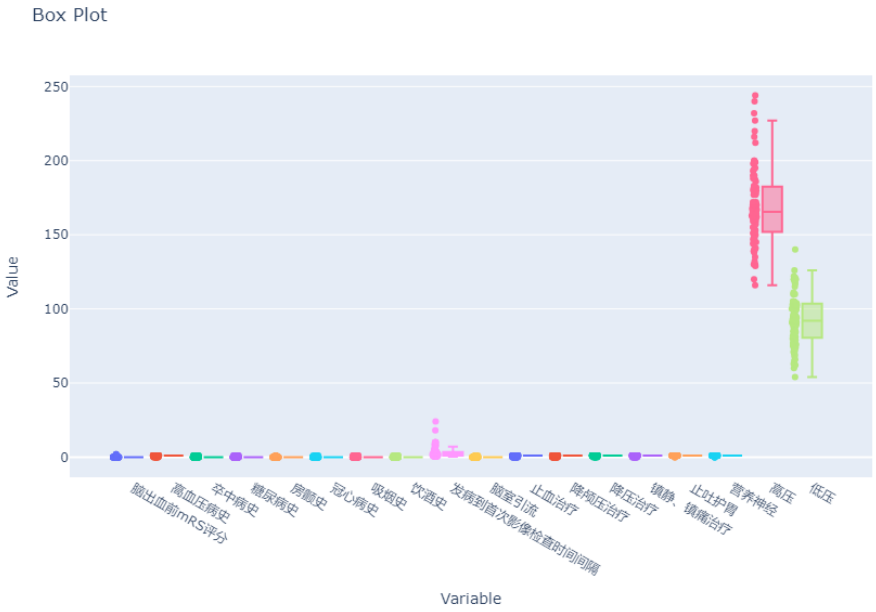


图 4-4 表 1 各字段盒图

根据图 4-4，可以轻易得知患者高压大多在 140-190 之间，低压大多在 60-110 之间。由于大多字段都为布尔型，因此需要进行进一步探索各个变量间的关系，如下图 4-5 所示。

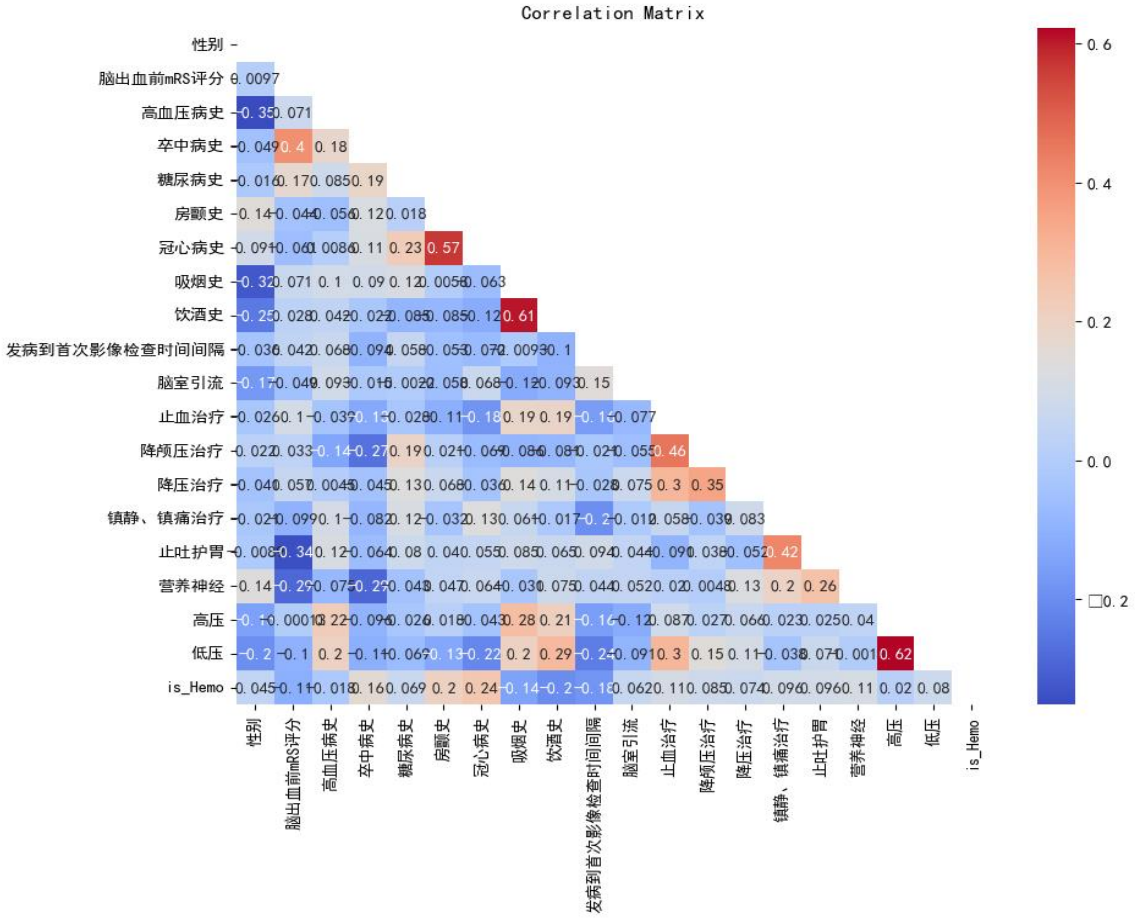


图 4-5 各影响因子对 Hemo 的相关性

根据上面的热力图，可以得到如下的相关性

表 4-5 与发生 Hemo 相关性表

影响因子	相关系数（绝对值）	正相关/负相关
脑出血前 mRs 评分	0.11	-
高血压病史	0.018	-
卒中病史	0.16	+
糖尿病史	0.069	+
房颤史	0.2	+
冠心病史	0.24	+
吸烟史	0.14	-
饮酒史	0.2	-
降颅压治疗	0.085	+
止血治疗	0.11	+

考虑到年龄对疾病的影响因素很大，因此将年龄作为横轴，进一步探索各变量和年龄关系图，如下图 4-5：

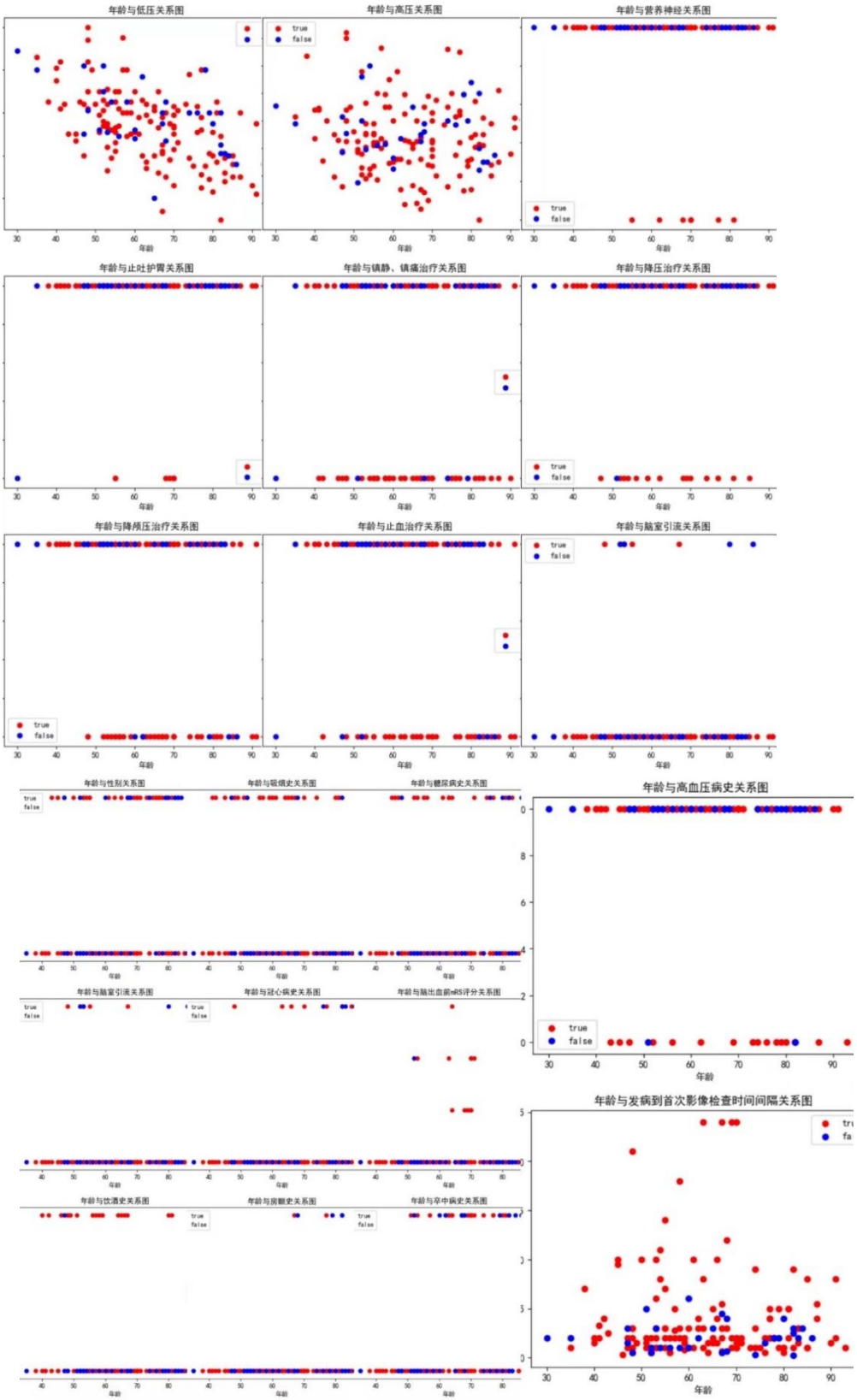


图 4-6 各指标与是否发生血肿对比

从图中不难看出，性别、营养神经与是否发生血肿扩张的关联性并不是那么密切，因此不将这两个影响因素作为模型的输入。

4.2.3 任务 b 模型的建立

任务 b 需要将 160 位患者分为数据集和测试集，这是一个小体量的监督分类问题，因此选择逻辑回归模型。在模型处理过程中，首先去掉输入中的“性别”和“营养神经”两个影响因子，作为模型一，再用所有字段作为逻辑回归模型的输入，记作模型二。

（数学表达式）逻辑回归模型通过使用 sigmoid 函数将线性预测转换为概率值，其数学公式如下：

$$P(Y=1|X)=\frac{1}{\left(1+\exp\left(-(\beta^0+\beta^1X^1+\beta^2X^2+\ldots+\beta_nX_n)\right)\right)}\quad(4-1)$$

其中，(P(Y=1|X)) 表示在给定输入特征 (X) 的情况下，目标变量 (Y) 为类别 1 的概率。(β_0, β_1, β_2, …, β_n) 表示模型的系数（参数），($X_1, X_2, \ldots, X_n$) 表示输入特征。

逻辑回归模型在训练阶段通过最大似然估计或正则化方法来学习参数值，从而得到最佳模型。在代码中，使用 GridSearchCV 进行网格搜索调优，并使用训练集数据 train_data 和 train_label 来训练模型并找到最佳的超参数组合 best_params_ 和最佳模型 best_model。

接下来，使用训练集数据 train_data 获取最佳模型的输出概率，通过调用 predict_proba 方法，得到对于每个样本属于各个类别的概率值。在代码中，train_prob 和 test_prob 分别存储了训练集和测试集的输出概率。

最后，使用列表解析和格式化字符串的方式，将输出概率值保留四位小数，并将其存储在 prob_issue1 列表中。

4.2.4 任务 b 模型结果

通过对两个模型的训练，得到如下的残差结果如表 4-6：

表 4-6 两类模型的总残差

模型名	总残差
模型一	1.2484214129536988
模型二	1.248439554901102

不难看出“性别”和“营养神经”会对模型的精度产影响，但是影响不大这与图 4-6，图 4-6 所分析的结果一致。

五、问题二的模型建立与求解

5.1 任务 a 的分析与处理

在预测患者脑部血肿体积随时间的进展时，患者脑部血肿体积的变化可能不符合简单的线性关系，而是会受到多种复杂因素的影响，因此**我们倾向于建立多项式回归和随机森林模型来进行预测，使用最小二乘法来拟合数据来寻找最佳的拟合曲线**。通过计算残差来评估模型的拟合效果和预测准确性，并绘制观测值与预测值之间的散点图，观察预测效果。

由于患者之间个体的差异，我们把患者分为 4 个亚组，对各个组进行**多项式回归模型**进行预测，并计算真实值和拟合曲线的残差。

5.2.1 任务 a 模型

基于上一小节所进行的数据处理，可以使用不同的方法建立回归模型，以期得到最优结果，包括**多项式回归和随机森林模型**。

考虑到患者个体差异因此患者脑补血肿体积随时间进展可能不符合简单的线性关系。首先使用多项式回归模型进行拟合曲线。

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \ldots + \beta_n X^n \qquad (5\text{-}1)$$

在这个公式中，Y 是目标变量，X 是输入变量。β_0、β_1、β_2、...、β_n 是模型的参数，分别对应截距和输入变量的权重。X^2、X^3、...、X^n 表示输入变量的多项式项。$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \ldots + \beta_n X^n$

在这个公式中，Y 是目标变量，X 是输入变量。β_0、β_1、β_2、...、β_n是模型的参数，分别对应截距和输入变量的权重。X^2、X^3、...、X^n 表示输入变量的多项式项。在本例中 X 对应时间间隔，Y 对应水肿体积。

通过对上一小节中对数据处理和分析知道，时间大于 500h 后，其数据不在变化，因此数据不具有参考行。考虑数据筛选，将时间间隔超过 500h 的数据过滤掉得到如下肿块随时间变化图（图 5-1）

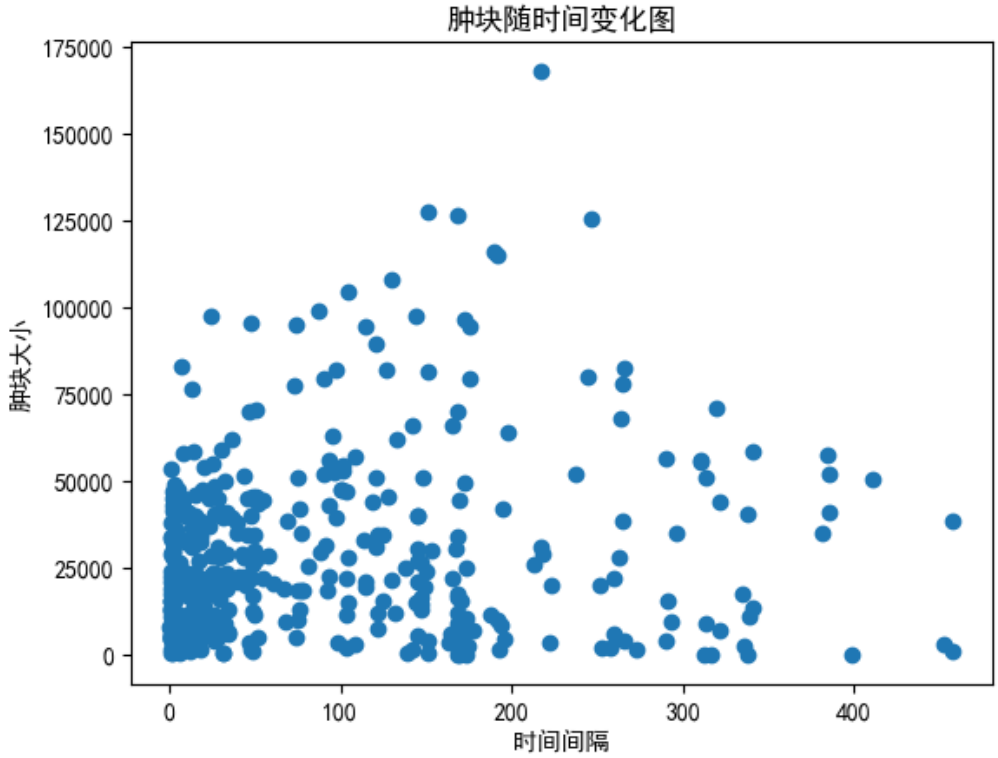


图 5-1 肿块随时间变化散点图

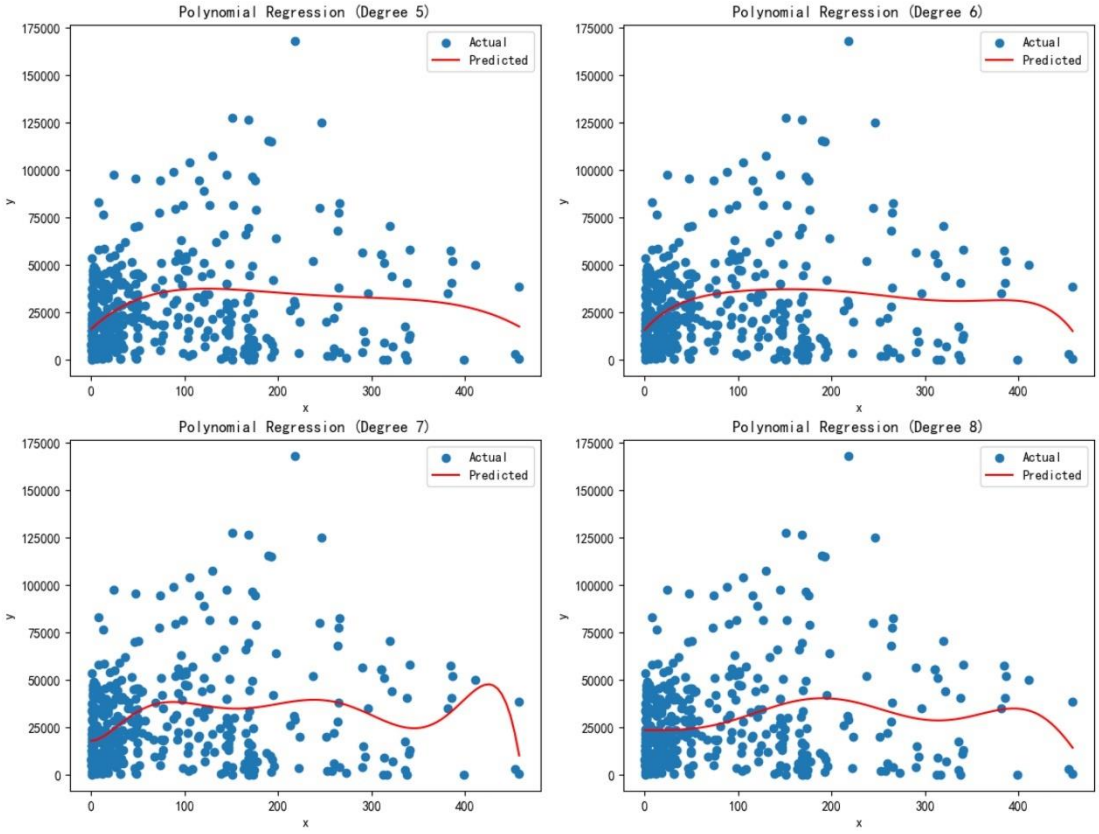


图 5-2 多项式阶数分别为 5,6,7,8 时拟合曲线和散点对比图

在尝试不同的多项式阶数后，通过模型输出结果得出在多项式阶数达到 7 阶时真实值和拟合值之间残差最小拟合效果最好。

接下来，使用随机森林模型进行拟合曲线。随机森林是一种集成学习方法，它通过构建多个决策树来进行预测和分类。决策树用于回归任务时，其预测公式如下：

$$y_{pred} = Mean(y_{leaf}) \qquad (5-2)$$

决策树将每个样本分配到一个叶节点，并将该叶节点中样本的平均值作为预测结果。通过组合多个决策树的预测结果来进行最终的预测。对于回归任务，预测公式如下：

$$y_{pred} = Mean(y_{tree}) \qquad (5-3)$$

随机森林将每个决策树的预测结果取平均，得到最终的预测结果拟合出如下图曲线。

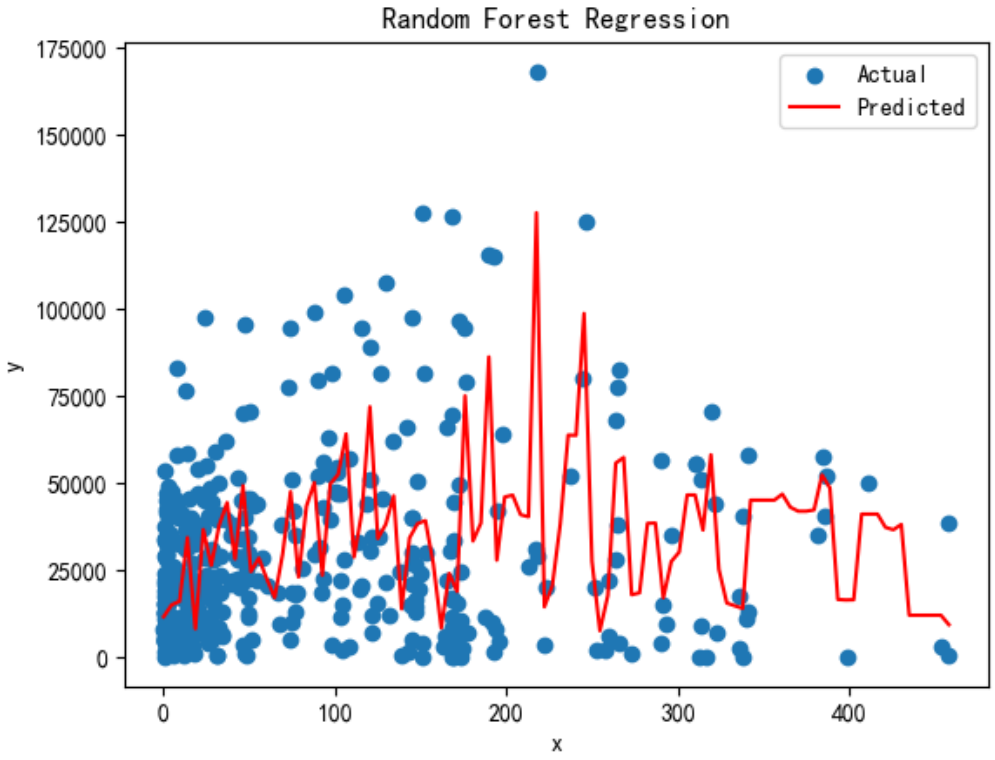


图 5-3 随机森林法构建拟合曲线

5.2 任务 b 的分析与处理

根据题意，我们将患者分为以下 4 个亚组，Class1：既抽烟又喝酒，Class2：只抽烟不喝酒，Class3：只喝酒不抽烟，Class4：不抽烟也不喝酒。绘制如下图所示的不同亚组水肿体积随时间变化散点图。

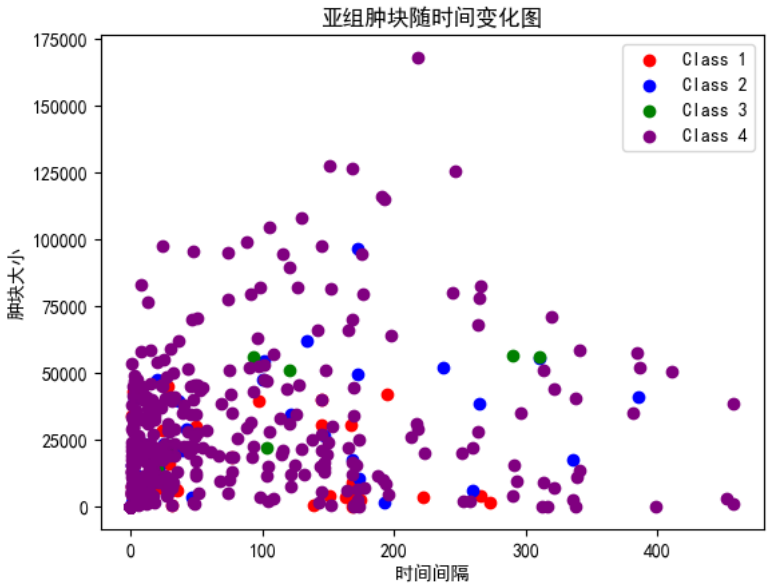


图 5-4 水肿体积随时间变化图

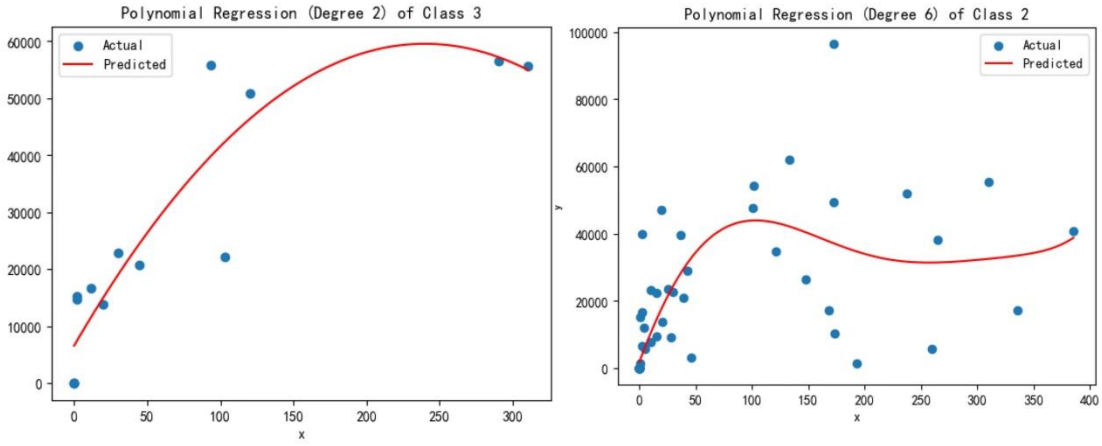


图 5-5 Class3 与 Class2 对比

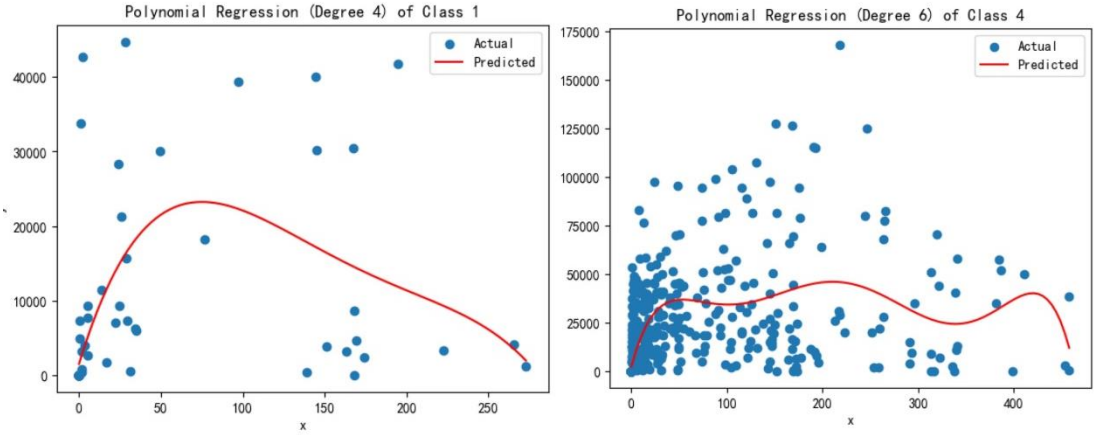


图 5-6 Class1 与 Class4 对比

通过图（5-5）和（5-6）不难看出抽烟喝酒对水肿体积影响较大。

5.3 任务 c 的分析与处理

通过结合“表 1”和“表 2”绘制不同治疗方法和水肿体积相关性热力图如下（图 5-7）

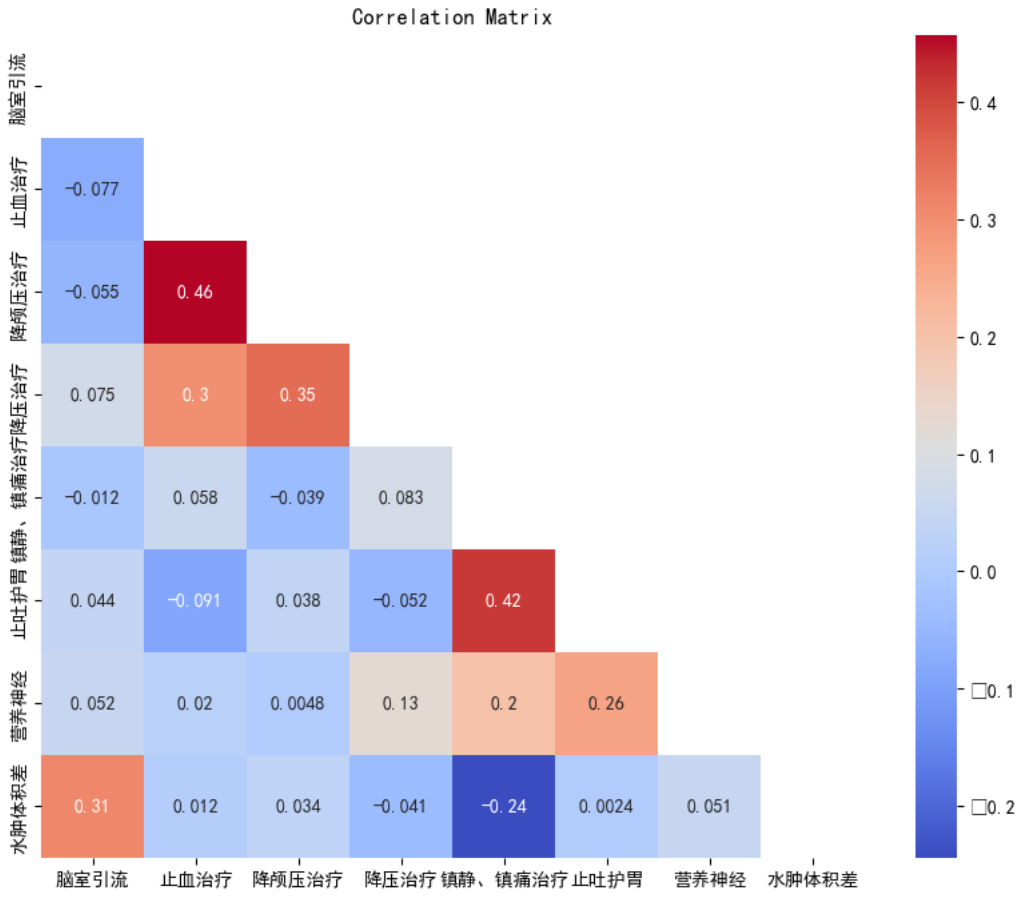


图 5-7 不同治疗方法和水肿体积相关性图

通过图 5-7 可以绘制出下表（表 5-1）

表 5-1 不同治疗方法和水肿体积相关性表

影响因子	相关系数（绝对值）	正负
脑室引流	0.31	+
止血治疗	0.012	+
降颅压治疗	0.034	+
降压治疗	0.041	-
镇静镇痛治疗	0.24	-
止吐护胃	0.0024	+
营养神经	0.051	+

通过表 5-1 可以看出**降压治疗和镇静镇痛治疗对患者水肿体积存在副作用，而其他治疗手段均会对治疗起正向作用，有利于改善患者症状，其中脑室引流效果最好。**

5.4 任务 d 的分析与处理

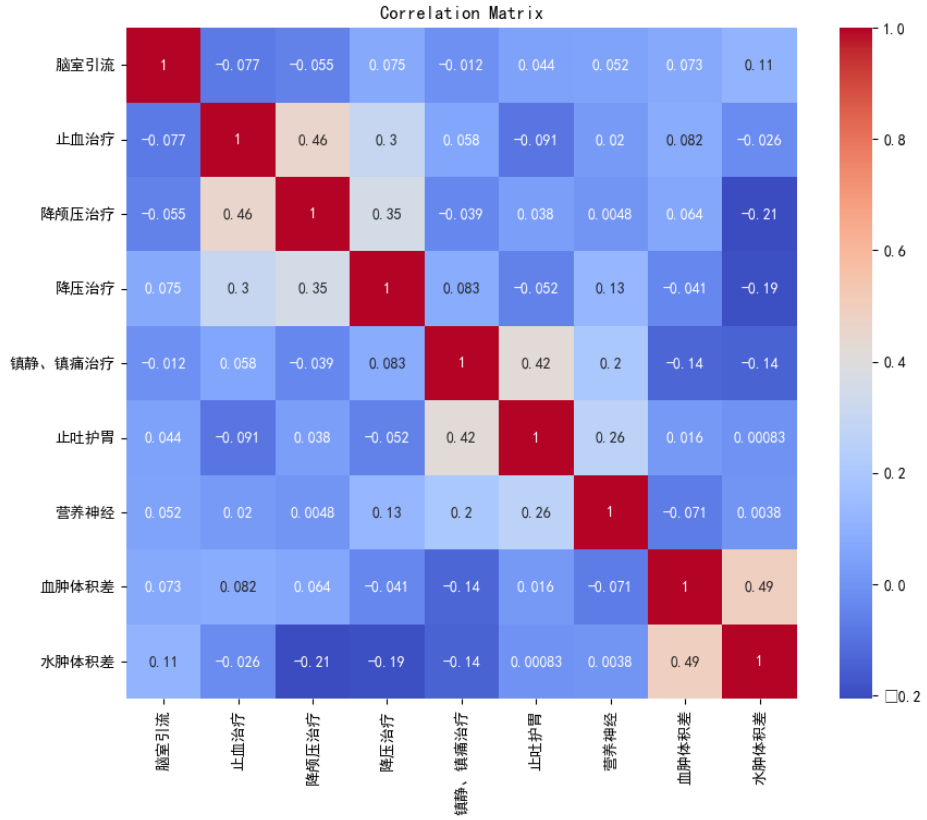


图 5-8 治疗手段与血肿水肿体积相关性

通过图 5-8 可以看出**血肿体积和水肿体积成正相关，而且降压治疗和镇静、镇痛治疗对血肿和水肿体积起反作用，因此这两种治疗方案不适合对患者使用，推荐使用脑室引流行治疗。**

六、问题三的模型建立和求解

6.1 问题 a 的分析与处理

由于患者个体差异，患者的发病历史和 mRs 评分可能不符合简单的线性关系，因此在预测 mRs 评分时选择以下三种回归模型并选择精度最好的模型进行预测。

1）逻辑回归模型

2）随机森林模型

3）K 临近算法模型

K 近邻（K-Nearest Neighbors，简称 KNN）算法是一种基于距离度量的分类和回归算法。下面是 KNN 算法的公式表示：

对于两个样本 x_i和 x_j，常用的距离度量包括欧氏距离、曼哈顿距离等，表示为 $d(x_i, x_j)$；对于一个新样本 x_{new}，KNN 算法通过以下公式进行分类预测：

$$y_{pred} = MajorityVote(y_{neighbors}) \qquad (6-1)$$

其中$y_{neighbors}$是x_{new}的 K 个最近邻居的类别标签。

KNN 算法通过计算样本之间的距离，并利用最近邻居的信息进行分类预测，采用多数表决的方式确定预测结果。

逻辑回归模型和随机森林模型的公式在上一小节有过介绍，在这里不再赘述。

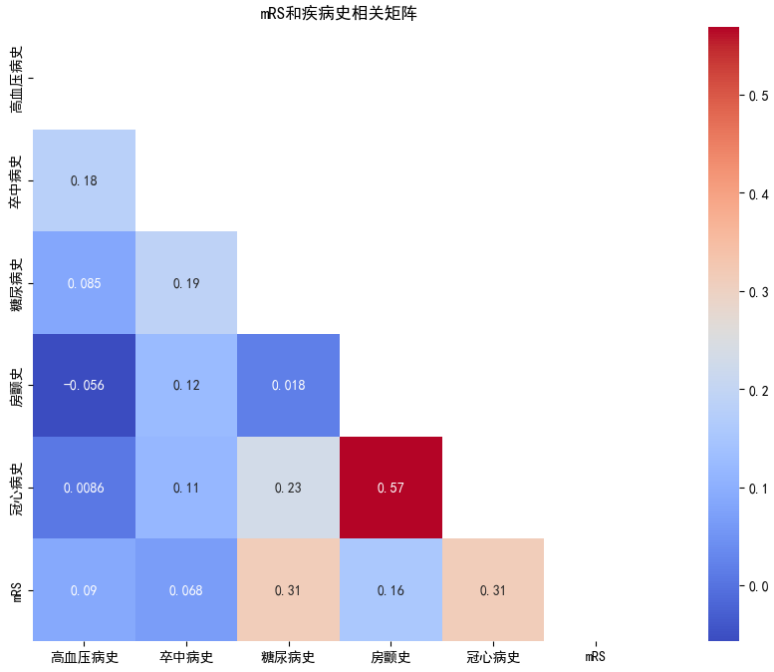


图 6-1 mRS 和疾病史相关性

从图 6-1 可以看出患者有相关病史会促使 mRS 评分升高。

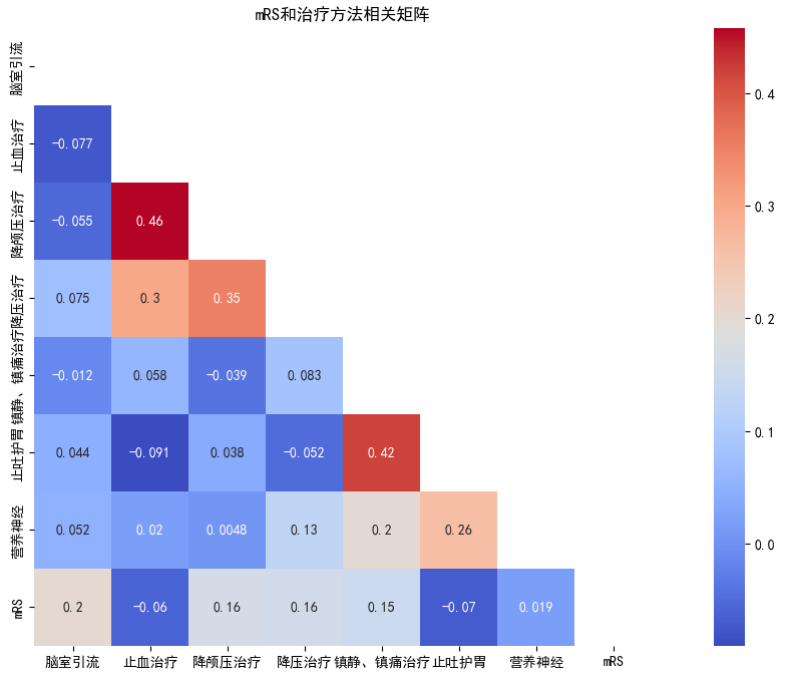


图 6-2 mRS 和治疗方法相关性

从图 6-2 可以看出患者进行相关治疗手段之后会使 mRs 评分降低或者减慢 mRS 评分上涨的速度。

七、参考文献

[1]Yu J, Zheng J, Xu R, et al. Expansion of Intracerebral Hemorrhage: Incidence and Predictors. World Neurosurg. 2017;104:22-30.

[2] Demchuk AM, Dowlatshahi D, Rodriguez-Luna D, et al. Prediction of Hematoma Growth and Outcome in Patients With Intracerebral Hemorrhage Using the CT-Angiography Spot Sign (PREDICT): A Prospective Observational Study. Lancet Neurol. 2012;11(4):307-14.

[3] 刘娟，王治国，胡志坚. 脑出血后血肿扩张的危险因素和预测方法[J]. 临床神经病学杂志，2018，31(6): 507-511.

[4] 余骏，李恭膺. 脑出血中的“Spot Sign”及扩散性血肿与病死率的关系[J]. 实用神经疾病杂志，2019，22(11): 26-29.

[5] Guo D, Wilkinson DA, Thompson BG, et al. Hematoma expansion predictors in intracerebral hemorrhage: Clinical and molecular perspectives. Neurology. 2016;86(11): 949-56.

[6] van Asch CJ, Luitse MJ, Rinkel GJ, et al. Incidence, case fatality, and functional outcome of intracerebral haemorrhage over time, according to age, sex, and ethnic origin: a systematic review and meta-analysis. Lancet Neurol. 2010;9(2):167-76.

[7] 杨秀芳，楼晓琳，吴荣华. 出血性脑卒中的临床特点及转归分析[J]. 医学综述，2019，25(14): 2778-2781.

[8] 陈萌，彭红，周锐. 出血性脑卒中患者的康复治疗研究进展[J]. 中国康复医学杂志，2020，35(2): 230-233.

[9] 张海滨，赵智民，李鹏. 出血性脑卒中的急性处理与护理进展[J]. 护理学杂志，2017，32(16): 54-57.

[10] 李宜春，胡晓江，范丹. 出血性脑卒中超早期治疗进展[J]. 中东国际医学杂志，2020，20(3): 282-287.

[11] Huynh TJ, Aviv RI, Dowlatshahi D, et al. Validation of the 9-point and 24-point hematoma expansion prediction scores and derivation of the PREDICT A/B scores. Stroke. 2015;46(11):3105-3110. doi:10.1161/STROKEAHA.115.010274

[12] 王利娟，陈晓通，邓伟新. 出血性脑卒中的影像学诊断和评估新进展[J]. 中华放射学杂志，2018，52(11): 878-882.

附录（代码）

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
file_path = '../E 题/数据/表 1-患者列表及临床信息.xlsx'
table1 = pd.read_excel(file_path)
table1.info()
# 从 table_1 选择两列
table_1_issue1 = table_1[['入院首次影像检查流水号', '发病到首次影像检查时间间隔']]

# 将时间间隔转换为时间间隔类型
table_1_issue1['发病到首次影像检查时间间隔'] = pd.to_timedelta(table_1_issue1['发病到首次影像检查时间间隔'], unit='h')
table_1_issue1.head()
# 统计空值的个数
null_count = table_1_issue1.isnull().sum()

# 打印空值的个数
print('Null count:')
print(null_count)

# 找到包含空值的列
null_columns = null_count[null_count > 0].index

# 打印包含空值的列
print('Null columns:')
print(null_columns)
file_path = '../E 题/数据/表 2-患者影像信息血肿及水肿的体积及位置.xlsx'
table_2 = pd.read_excel(file_path)
table_2.head()
schedule_1 = pd.read_excel('../E 题/数据/附表 1-检索表格-流水号 vs 时间.xlsx')

schedule_1.head()
# 筛选 table2 的列
```

```
index = ['首次检查流水号', '随访 1 流水号', '随访 2 流水号', '随访 3 流水号', '随访 4 流水号', '随访 5 流水号']
index_list = []
for i in index:
    index_list.append(table_2.columns.get_loc(i))
    index_list.append(table_2.columns.get_loc(i) + 1)
table_2_issue1 = table_2.iloc[: , index_list]
table_2_issue1.head()
from datetime import timedelta
# 判断血肿体积变化
result, time_list = [0 for _ in range(160)], ['0 小时' for _ in range(160)]
latest_index = 0
for index, row in table_2_issue1.iterrows():
    for num in range(1, 6):
        latest_index = 2 * num + 1
        # 先判断是否发生血肿
        if row[latest_index] - row[1] >= 6000 or (row[latest_index] - row[1]) / row[1] > 0.33:
            # 再判断发生血肿是否在 48h 内
            # 计算率先达到血肿条件的时间戳索引，判断这段时间是否超出 48h
            row_index, col_index = np.where(schedule_1.values == row[latest_index-1])
            # 如果血肿在 48h 内
            if schedule_1.iloc[row_index, col_index-1].values[0] != None and pd.Timedelta(days=2) >= schedule_1.iloc[row_index, col_index-1].values[0][0] + table_1_issue1.iloc[index]['发病到首次影像检查时间间隔'] - schedule_1.iloc[row_index, 2].values[0]:
                interval_time = schedule_1.iloc[row_index, col_index-1].values[0][0] - schedule_1.iloc[row_index, 2].values[0]
                interval_time = timedelta(microseconds=int(interval_time / np.timedelta64(1, 'us')))
                result[index] = 1
                time_list[index] = f'{interval_time.total_seconds() / 3600:.4f}小时'
                # print(index, num, result[index], time_list[index])
                break
print(result)
print(time_list)
```

```
from openpyxl import load_workbook

# 打开答案文件.xlsx
workbook = load_workbook('../E 题/数据/表 4-答案文件.xlsx')

# 选择活动的工作表
worksheet = workbook.active

# 获取起始单元格的列号和行号
result_start_column = 'C'
start_row = 4
time_start_column = 'D'

# 逐行写入 result 列表的元素
for i in range(len(result)):
    # 定位
    result_current_column = result_start_column
    time_current_column = time_start_column
    result_cell_coordinate = f"{result_current_column}{start_row + i}"
    time_cell_coordinate = f"{time_current_column}{start_row + i}"
    # 写入
    worksheet[result_cell_coordinate] = result[i]
    worksheet[time_cell_coordinate] = time_list[i]

# 保存工作簿到文件
workbook.save('../E 题/数据/表 4-答案文件.xlsx')
# 处理血压
table_1[["高压", "低压"]] = table_1["血压"].str.split("/", expand=True)
table_1[["高压", "低压"]] = table_1[["高压", "低压"]].astype(int)
table_1 = table_1.drop("血压", axis=1)
table_1.head()
# 处理性别 0-男 1-女
table_1['性别'] = table_1['性别'].replace({"男": 0, "女": 1})
table_1.head()
# 将 result 拼接过来
table_1 = pd.concat([table_1, pd.Series(result).rename('result')], axis=1)
# 统计空值的个数
null_count = table_1.isnull().sum()
```

```
102.
103.# 打印空值的个数
104.print('Null count:')
105.print(null_count)
106.
107.# 找到包含空值的列
108.null_columns = null_count[null_count > 0].index
109.
110.# 打印包含空值的列
111.print('Null columns:')
112.print(null_columns)
113.data = table_1.iloc[:, 2:]
114.train_data, test_data = data.iloc[:100, :], data.iloc[100:, :]
115.train_label, test_label = result[:100], result[100:]
116.train_data.shape, test_data.shape, len(train_label), len(test_la
   bel)
117.plt.rcParams['font.sans-serif'] = ['SimHei']
118.category_counts = train_data.nunique()
119.
120.plt.figure()
121.category_counts.plot(kind='bar')
122.plt.title(f'category counts for label')
123.plt.xlabel('category')
124.plt.ylabel('count')
125.# 显示数字
126.for i, v in enumerate(category_counts):
127.    plt.text(i, v, str(v), ha='center', va='bottom')
128.
129.plt.show()
130.train_data.info()
131.table_1_a = table_1.iloc[:, :12]
132.table_1_b = table_1.iloc[:, 12:]
133.import plotly.graph_objects as go
134.import pandas as pd
135.# 创建数据
136.data = train_data.iloc[:, 1:-1]
137.
138.# 创建图表布局
139.fig = go.Figure()
140.
141.# 添加每个变量的散点图
142.for column in data.columns:
```

```
    fig.add_trace(go.Scatter(
        x=[column] * len(data),
        y=data[column],
        mode='markers',
        name=column,
        marker=dict(
            size=8,
            line=dict(width=1),
            opacity=0.7
        )
    ))

# 设置图表布局
fig.update_layout(
    title='Multiple Indicator Strip Plot',
    xaxis=dict(title='Variable'),
    yaxis=dict(title='Value'),
    showlegend=False,
    width=900,
    height=600
)

# 显示图表
fig.show()
# 创建数据
data = train_data.iloc[:, 1:-1]

# 创建图表布局
fig = go.Figure()

# 添加每个变量的盒图
for column in data.columns:
    fig.add_trace(go.Box(
        y=data[column],
        name=column,
        boxpoints='all',
        jitter=0.3,
        pointpos=-1.8
    ))

# 设置图表布局
fig.update_layout(
```

```
    title='Box Plot',
    xaxis=dict(title='Variable'),
    yaxis=dict(title='Value'),
    showlegend=False,
    width=900,
    height=600
)

# 显示图表
fig.show()
# 创建数据
data = train_data.iloc[:, 1:-1]

# 创建图表布局
fig = go.Figure()

# 添加每个变量的散点图
for column in data.columns:
    fig.add_trace(go.Scatter(
        x=[column] * len(data),
        y=data[column],
        mode='markers',
        name=column,
        marker=dict(
            size=8,
            line=dict(width=1),
            opacity=0.7
        )
    ))

# 添加每个变量的小提琴图
for column in data.columns:
    fig.add_trace(go.Violin(
        y=data[column],
        name=column,
        box_visible=True,
        meanline_visible=True,
        fillcolor='lightgray',
        line_color='black'
    ))

# 设置图表布局
```

```
fig.update_layout(
    title='Scatter and Violin Plot',
    xaxis=dict(title='Variable'),
    yaxis=dict(title='Value'),
    showlegend=False,
    width=900,
    height=600
)

# 显示图表
fig.show()
train_data.shape
# 数据分析
# for i in range(2, len(table_1.columns)-1):
#     subset1 = table_1[table_1['result'] == 0]
#     subset2 = table_1[table_1['result'] == 1]

#     plt.scatter(subset1['年龄
'], subset1[table_1.columns[i]], color='red', label='true')
#     plt.scatter(subset2['年龄
'], subset2[table_1.columns[i]], color='blue', label='false')

#     plt.xlabel('年龄')
#     plt.ylabel(table_1.columns[i])
#     plt.title(f'年龄与{table_1.columns[i]}关系图')
#     plt.legend()
#     plt.show()
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# 创建数据
data = train_data.iloc[:, :-1].assign(is_Hemo=train_label)  # 假设train_data的标签位于第一列

# 计算相关系数矩阵
corr_matrix = data.corr()

# 创建遮盖矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

```

```
# 绘制相关矩阵
plt.figure(figsize=(16, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
plt.title('Correlation Matrix')
plt.show()
table_3 = pd.read_excel('../E 题/数据/表 3-患者影像信息血肿及水肿的形状及灰度分布.xlsx')
table_3
# 创建数据
data = table_2.iloc[:100, 2:24].assign(is_Hemo=train_label)  # 假设 train_data 的标签位于第一列

# 计算相关系数矩阵
corr_matrix = data.corr()

# 创建遮盖矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# 绘制相关矩阵
plt.figure(figsize=(16, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
plt.title('Correlation Matrix')
plt.show()
# 创建数据
data = table_3.iloc[:100, :].assign(is_Hemo=train_label)  # 假设 train_data 的标签位于第一列

# 计算相关系数矩阵
corr_matrix = data.corr()

# 创建遮盖矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# 绘制相关矩阵
plt.figure(figsize=(16, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
plt.title('Correlation Matrix')
plt.show()
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV

# 创建逻辑回归模型对象
model = LogisticRegression()

# 定义超参数的取值范围
param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

# 创建 GridSearchCV 对象
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)

# 使用训练集进行网格搜索调优
grid_search.fit(train_data, train_label)

# 输出最佳超参数组合
print(grid_search.best_params_)

# 输出最佳模型
best_model = grid_search.best_estimator_

# 使用训练集获取模型的输出概率
train_prob = best_model.predict_proba(train_data)
test_prob = best_model.predict_proba(test_data)
prob_issue1 = [f"{x:.4f}" for x in train_prob[:, 1:].reshape(-1)] + [f"{x:.4f}" for x in test_prob[:, 1:].reshape(-1)]
print(prob_issue1)
sum_resid = sum(np.abs(test_label-test_prob[:, 1:].reshape(-1)))

sum_resid
# data = table_1.iloc[:, 2:]
# data1 = data.drop(columns=['性别', '营养神经'])
# data1.head()
# train_data, test_data = data1.iloc[:100, :], data1.iloc[100:, :]
# # 创建逻辑回归模型对象
# model1 = LogisticRegression()

```

```
# # 定义超参数的取值范围
# param_grid = {
#     'C': [0.1, 1, 10],
#     'penalty': ['l1', 'l2'],
#     'solver': ['liblinear']
# }

# # 创建 GridSearchCV 对象
# grid_search = GridSearchCV(estimator=model1, param_grid=param_grid, cv=5)

# # 使用训练集进行网格搜索调优
# grid_search.fit(train_data, train_label)

# # 输出最佳超参数组合
# print(grid_search.best_params_)

# # 输出最佳模型
# best_model = grid_search.best_estimator_

# # 使用训练集获取模型的输出概率
# train_prob = best_model.predict_proba(train_data)
# test_prob = best_model.predict_proba(test_data)
# prob_issue1 = [f"{x:.4f}" for x in train_prob[:, 1:].reshape(-1)] + [f"{x:.4f}" for x in test_prob[:, 1:].reshape(-1)]
# print(prob_issue1)
# sum_resid = sum(np.abs(test_label-test_prob[:, 1:].reshape(-1)))

# sum_resid
from openpyxl import load_workbook

# 打开答案文件.xlsx
workbook = load_workbook('../E 题/数据/表 4-答案文件.xlsx')

# 选择活动的工作表
worksheet = workbook.active

# 获取起始单元格的列号和行号
prob_start_column = 'E'
start_row = 4

```

```
# 逐行写入 result 列表的元素
for i in range(len(prob_issue1)):
    # 定位
    prob_current_column = prob_start_column
    prob_cell_coordinate = f"{prob_current_column}{start_row + i}"
    # 写入
    worksheet[prob_cell_coordinate] = prob_issue1[i]

# 保存工作簿到文件
workbook.save('../E 题/数据/表 4-答案文件.xlsx')
# 筛选 table2 的列
index = [f'随访{i}流水号' for i in range(1, 9)]
index.insert(0, '首次检查流水号')
table_indices, schedule_indices = [], [2*i for i in range(1, len(index)+1)]
for i in index:
    table_indices.append(table_2.columns.get_loc(i) + 12)

Y_train = table_2.iloc[:100 , table_indices]
schedule_1_issue2 = schedule_1.iloc[:100, schedule_indices]
Y_train.head()
# 创建空的 x_train DataFrame，与 schedule_1_issue2 有相同的结构
X_train = pd.DataFrame(columns=schedule_1_issue2.columns)

# 遍历 schedule_1_issue2 中的每一行
for i in range(len(schedule_1_issue2)):
    # 获取当前行的时间戳
    current_row = schedule_1_issue2.iloc[i]

    # 创建空的 diff 列表，用于存储时间差值
    diff = []

    # 遍历当前行的每个单元格
    for j in range(len(current_row)):
        # 判断当前单元格是否为空
        if pd.isnull(current_row[j]):
            # 如果当前单元格为空，则不进行处理，将时间差值设置为 NaN
            diff.append(float('nan'))
        else:
            # 获取前一个单元格的时间戳
            if j == 0:
```

```
                previous_cell = pd.to_datetime(current_row[j])
            else:
                previous_cell = schedule_1_issue2.iloc[i, j - 1]

            # 将当前单元格时间戳转换为 datetime 类型
            current_cell = pd.to_datetime(current_row[j])

            # 计算时间差，并将结果添加到 diff 列表
            time_diff = current_cell - previous_cell + table_1_issue1.iloc[i]['发病到首次影像检查时间间隔']
            diff.append(round(time_diff.total_seconds() / 3600, 2))

    # 将 diff 列表添加到 x_train
    X_train.loc[i] = diff

# 打印处理后的 x_train
X_train.head()
import numpy as np
import matplotlib.pyplot as plt

# 获取对应位置的 x 和 y 值
x_values = X_train.values.flatten().tolist()
y_values = Y_train.values.flatten().tolist()

# 创建筛选后的数据列表
filtered_x = []
filtered_y = []

# 根据条件筛选数据
for x, y in zip(x_values, y_values):
    if x <= 500:
        filtered_x.append(x)
        filtered_y.append(y)

# 绘制散点图
plt.scatter(filtered_x, filtered_y)

# 设置横纵坐标标签
plt.xlabel('时间间隔')
plt.ylabel('肿块大小')
plt.title('肿块随时间变化图')
```

```
460.
461.# 显示图像
462.plt.show()
463.# 找到空数据所在的索引
464.nan_indices = np.isnan(filtered_x) | np.isnan(filtered_y)
465.
466.# 获取空数据所在的索引
467.indices = np.where(nan_indices)[0]
468.
469.# 删除空数据所在的索引对应的元素
470.x = np.delete(filtered_x, indices)
471.y = np.delete(filtered_y, indices)
472.len(x), len(y)
473.import numpy as np
474.from sklearn.preprocessing import PolynomialFeatures
475.from sklearn.linear_model import LinearRegression
476.
477.# 将 train_data 转换为二维数组
478.x_train = np.array(x).reshape(-1, 1)
479.
480.# 将 train_label 转换为二维数组
481.y_train = np.array(y).reshape(-1, 1)
482.
483.best_degree = None
484.best_residual_sum = float('inf')
485.
486.# 尝试不同的多项式阶数
487.for degree in range(1, 10):
488.    # 使用多项式特征转换
489.    poly_features = PolynomialFeatures(degree=degree)
490.    x_train_poly = poly_features.fit_transform(x_train)
491.
492.    # 创建多项式回归模型并进行训练
493.    model = LinearRegression()
494.    model.fit(x_train_poly, y_train)
495.
496.    # 使用模型对训练数据进行预测
497.    y_train_pred = model.predict(x_train_poly)
498.
499.    # 计算残差和
500.    residual_sum = np.sum(np.abs(y_train - y_train_pred))
501.
```

```
    # 更新最小残差和和对应的阶数
    if residual_sum < best_residual_sum:
        best_residual_sum = residual_sum
        best_degree = degree

# 使用最佳阶数重新训练模型
poly_features = PolynomialFeatures(degree=best_degree)
x_train_poly = poly_features.fit_transform(x_train)
model = LinearRegression()
model.fit(x_train_poly, y_train)

# 生成测试数据
x_test = np.linspace(np.min(x_train), np.max(x_train), 100).reshape(-1, 1)
x_test_poly = poly_features.transform(x_test)

# 使用模型进行预测
y_pred = model.predict(x_test_poly)

# 绘制真实值和预测值的图像
plt.scatter(x_train, y_train, label='Actual')
plt.plot(x_test, y_pred, color='red', label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (Degree {})'.format(best_degree))
plt.legend()
plt.show()

# 使用模型对训练数据进行预测
y_train_pred = model.predict(x_train_poly)

# 计算最终的残差和
residual_sum = np.sum(np.abs(y_train - y_train_pred))

# 打印最终的残差和
print(residual_sum, best_degree)
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# 将 train_data 转换为二维数组
x_train = np.array(x).reshape(-1, 1)
```

```
542.
543.# 将 train_label 转换为一维数组
544.y_train = np.array(y)
545.
546.# 创建随机森林回归模型并进行训练
547.model = RandomForestRegressor(n_estimators=100)
548.model.fit(x_train, y_train)
549.
550.# 生成测试数据
551.x_test = np.linspace(np.min(x_train), np.max(x_train), 100).resh
   ape(-1, 1)
552.
553.# 使用模型进行预测
554.y_pred = model.predict(x_test)
555.
556.# 绘制真实值和预测值的图像
557.plt.scatter(x_train, y_train, label='Actual')
558.plt.plot(x_test, y_pred, color='red', label='Predicted')
559.plt.xlabel('x')
560.plt.ylabel('y')
561.plt.title('Random Forest Regression')
562.plt.legend()
563.plt.show()
564.# 使用模型对训练数据进行预测
565.y_train_pred = model.predict(x_train)
566.
567.# 计算最终的残差和
568.residual_sum = np.sum(np.abs(y_train - y_train_pred))
569.
570.# 打印最终的残差和
571.print(residual_sum)
572.x_values_without_nan = np.nan_to_num(x_values)
573.y_values_without_nan = np.nan_to_num(y_values)
574.y_values_pred = model.predict(x_values_without_nan.reshape(-
   1, 1))
575.# 获取 x_values_without_nan 中数值为 0 的索引位置
576.zero_indices = np.where(x_values_without_nan == 0)
577.
578.# 将 y_values_pred 中对应索引位置的值替换为 0
579.y_values_pred[zero_indices] = 0
580.# 计算 y_values_pred 和 y_values_without_nan 之间的绝对值差
581.abs_diff = y_values_pred - y_values_without_nan
```

```
582.
583.# 每 9 个数据求和并存储在 resid_list 中
584.resid_list = []
585.for i in range(0, len(abs_diff), 9):
586.    resid_sum = np.sum(abs_diff[i:i+5])
587.    resid_list.append(resid_sum / 9)
588.
589.# resid_list
590.# resid = y_train.reshape(-1) - y_train_pred
591.# 打开答案文件.xlsx
592.workbook = load_workbook('../E 题/数据/表 4-答案文件.xlsx')
593.
594.# 选择活动的工作表
595.worksheet = workbook.active
596.
597.# 获取起始单元格的列号和行号
598.start_column = 'F'
599.start_row = 4
600.
601.# 逐行写入 result 列表的元素
602.for i in range(len(resid_list)):
603.    # 定位
604.    current_column = start_column
605.    current_column = start_column
606.    cell_coordinate = f"{current_column}{start_row + i}"
607.    # 写入
608.    worksheet[cell_coordinate] = resid_list[i]
609.
610.# 保存工作簿到文件
611.workbook.save('../E 题/数据/表 4-答案文件.xlsx')
612.train_table_1 = table_1.iloc[:100, :]
613.idx_1 = train_table_1[(train_table_1["吸烟史
   "] == 1) & (train_table_1["饮酒史"] == 1)].index.tolist()
614.idx_2 = train_table_1[(train_table_1["吸烟史
   "] == 1) & (train_table_1["饮酒史"] == 0)].index.tolist()
615.idx_3 = train_table_1[(train_table_1["吸烟史
   "] == 0) & (train_table_1["饮酒史"] == 1)].index.tolist()
616.idx_4 = train_table_1[(train_table_1["吸烟史
   "] == 0) & (train_table_1["饮酒史"] == 0)].index.tolist()
617.len(idx_1) + len(idx_2) + len(idx_3) + len(idx_4)
618.# 根据索引分类 x_train 的行
619.x_train_1 = X_train.loc[idx_1]
```

```
x_train_2 = X_train.loc[idx_2]
x_train_3 = X_train.loc[idx_3]
x_train_4 = X_train.loc[idx_4]

x_train_1.shape, x_train_2.shape, x_train_3.shape, x_train_4.shape
# 根据索引分类x_train 的行
y_train_1 = Y_train.loc[idx_1]
y_train_2 = Y_train.loc[idx_2]
y_train_3 = Y_train.loc[idx_3]
y_train_4 = Y_train.loc[idx_4]

y_train_1.shape, y_train_2.shape, y_train_3.shape, y_train_4.shape
# 填充缺失值为均值
x_train_1 = x_train_1.fillna(0)
x_train_2 = x_train_2.fillna(0)
x_train_3 = x_train_3.fillna(0)
x_train_4 = x_train_4.fillna(0)

y_train_1 = y_train_1.fillna(0)
y_train_2 = y_train_2.fillna(0)
y_train_3 = y_train_3.fillna(0)
y_train_4 = y_train_4.fillna(0)
import numpy as np
import matplotlib.pyplot as plt

# 获取对应位置的 x 和 y 值
x_values_1 = x_train_1.values.flatten().tolist()
y_values_1 = y_train_1.values.flatten().tolist()
x_values_2 = x_train_2.values.flatten().tolist()
y_values_2 = y_train_2.values.flatten().tolist()
x_values_3 = x_train_3.values.flatten().tolist()
y_values_3 = y_train_3.values.flatten().tolist()
x_values_4 = x_train_4.values.flatten().tolist()
y_values_4 = y_train_4.values.flatten().tolist()

# 创建筛选后的数据列表
filtered_x1 = []
filtered_y1 = []
filtered_x2 = []
filtered_y2 = []
```

```
filtered_x3 = []
filtered_y3 = []
filtered_x4 = []
filtered_y4 = []

# 根据条件筛选数据
for x, y in zip(x_values_1, y_values_1):
    if x <= 500:
        filtered_x1.append(x)
        filtered_y1.append(y)

for x, y in zip(x_values_2, y_values_2):
    if x <= 500:
        filtered_x2.append(x)
        filtered_y2.append(y)

for x, y in zip(x_values_3, y_values_3):
    if x <= 500:
        filtered_x3.append(x)
        filtered_y3.append(y)

for x, y in zip(x_values_4, y_values_4):
    if x <= 500:
        filtered_x4.append(x)
        filtered_y4.append(y)

# 绘制散点图
plt.scatter(filtered_x1, filtered_y1, c='red', label='Class 1')
plt.scatter(filtered_x2, filtered_y2, c='blue', label='Class 2')
plt.scatter(filtered_x3, filtered_y3, c='green', label='Class 3'
)
plt.scatter(filtered_x4, filtered_y4, c='purple', label='Class 4
')

# 设置横纵坐标标签
plt.xlabel('时间间隔')
plt.ylabel('肿块大小')
plt.title('亚组肿块随时间变化图')

# 添加图例
plt.legend()

```

```
# 显示图像
plt.show()
# 找到空数据所在的索引
nan_indices = np.isnan(filtered_x1) | np.isnan(filtered_y1)

# 获取空数据所在的索引
indices = np.where(nan_indices)[0]

# 删除空数据所在的索引对应的元素
x1 = np.delete(filtered_x1, indices)
y1 = np.delete(filtered_y1, indices)
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 将 train_data 转换为二维数组
x_train = np.array(x1).reshape(-1, 1)

# 将 train_label 转换为二维数组
y_train = np.array(y1).reshape(-1, 1)

best_degree = None
best_residual_sum = float('inf')

# 尝试不同的多项式阶数
for degree in range(1, 10):
    # 使用多项式特征转换
    poly_features = PolynomialFeatures(degree=degree)
    x_train_poly = poly_features.fit_transform(x_train)

    # 创建多项式回归模型并进行训练
    model = LinearRegression()
    model.fit(x_train_poly, y_train)

    # 使用模型对训练数据进行预测
    y_train_pred = model.predict(x_train_poly)

    # 计算残差和
    residual_sum = np.sum(np.abs(y_train - y_train_pred))

    # 更新最小残差和和对应的阶数
    if residual_sum < best_residual_sum:
```

```
        best_residual_sum = residual_sum
        best_degree = degree

# 使用最佳阶数重新训练模型
poly_features = PolynomialFeatures(degree=best_degree)
x_train_poly = poly_features.fit_transform(x_train)
model = LinearRegression()
model.fit(x_train_poly, y_train)

# 生成测试数据
x_test = np.linspace(np.min(x_train), np.max(x_train), 100).reshape(-1, 1)
x_test_poly = poly_features.transform(x_test)

# 使用模型进行预测
y_pred = model.predict(x_test_poly)

# 绘制真实值和预测值的图像
plt.scatter(x_train, y_train, label='Actual')
plt.plot(x_test, y_pred, color='red', label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (Degree {}) of Class 1'.format(best_degree))
plt.legend()
plt.show()

# 使用模型对训练数据进行预测
y_train_pred = model.predict(x_train_poly)

# 计算最终的残差和
residual_sum = np.sum(np.abs(y_train - y_train_pred))

# 打印最终的残差和
print(residual_sum, best_degree)
resid_list2 = [0 for _ in range(100)]
i = 0
for idx in idx_1:
    resid = np.mean((y_train - y_train_pred)[i:i+8])
    resid_list2[idx] = resid
    i += 1
# 找到空数据所在的索引
```

```
nan_indices = np.isnan(filtered_x2) | np.isnan(filtered_y2)

# 获取空数据所在的索引
indices = np.where(nan_indices)[0]

# 删除空数据所在的索引对应的元素
x2 = np.delete(filtered_x2, indices)
y2 = np.delete(filtered_y2, indices)
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 将 train_data 转换为二维数组
x_train = np.array(x2).reshape(-1, 1)

# 将 train_label 转换为二维数组
y_train = np.array(y2).reshape(-1, 1)

best_degree = None
best_residual_sum = float('inf')

# 尝试不同的多项式阶数
for degree in range(1, 10):
    # 使用多项式特征转换
    poly_features = PolynomialFeatures(degree=degree)
    x_train_poly = poly_features.fit_transform(x_train)

    # 创建多项式回归模型并进行训练
    model = LinearRegression()
    model.fit(x_train_poly, y_train)

    # 使用模型对训练数据进行预测
    y_train_pred = model.predict(x_train_poly)

    # 计算残差和
    residual_sum = np.sum(np.abs(y_train - y_train_pred))

    # 更新最小残差和和对应的阶数
    if residual_sum < best_residual_sum:
        best_residual_sum = residual_sum
        best_degree = degree

```

```
# 使用最佳阶数重新训练模型
poly_features = PolynomialFeatures(degree=best_degree)
x_train_poly = poly_features.fit_transform(x_train)
model = LinearRegression()
model.fit(x_train_poly, y_train)

# 生成测试数据
x_test = np.linspace(np.min(x_train), np.max(x_train), 100).reshape(-1, 1)
x_test_poly = poly_features.transform(x_test)

# 使用模型进行预测
y_pred = model.predict(x_test_poly)

# 绘制真实值和预测值的图像
plt.scatter(x_train, y_train, label='Actual')
plt.plot(x_test, y_pred, color='red', label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (Degree {}) of Class 2'.format(best_degree))
plt.legend()
plt.show()

# 使用模型对训练数据进行预测
y_train_pred = model.predict(x_train_poly)

# 计算最终的残差和
residual_sum = np.sum(np.abs(y_train - y_train_pred))

# 打印最终的残差和
print(residual_sum, best_degree)
i = 0
for idx in idx_2:
    resid = np.mean((y_train - y_train_pred)[i:i+9])
    resid_list2[idx] = resid
    i += 1
# 找到空数据所在的索引
nan_indices = np.isnan(filtered_x3) | np.isnan(filtered_y3)

# 获取空数据所在的索引
indices = np.where(nan_indices)[0]
```

```
# 删除空数据所在的索引对应的元素
x3 = np.delete(filtered_x3, indices)
y3 = np.delete(filtered_y3, indices)
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 将 train_data 转换为二维数组
x_train = np.array(x3).reshape(-1, 1)

# 将 train_label 转换为二维数组
y_train = np.array(y3).reshape(-1, 1)

best_degree = None
best_residual_sum = float('inf')

# 尝试不同的多项式阶数
for degree in range(1, 4):
    # 使用多项式特征转换
    poly_features = PolynomialFeatures(degree=degree)
    x_train_poly = poly_features.fit_transform(x_train)

    # 创建多项式回归模型并进行训练
    model = LinearRegression()
    model.fit(x_train_poly, y_train)

    # 使用模型对训练数据进行预测
    y_train_pred = model.predict(x_train_poly)

    # 计算残差和
    residual_sum = np.sum(np.abs(y_train - y_train_pred))

    # 更新最小残差和和对应的阶数
    if residual_sum < best_residual_sum:
        best_residual_sum = residual_sum
        best_degree = degree

# 使用最佳阶数重新训练模型
poly_features = PolynomialFeatures(degree=best_degree)
x_train_poly = poly_features.fit_transform(x_train)
model = LinearRegression()
```

```
model.fit(x_train_poly, y_train)

# 生成测试数据
x_test = np.linspace(np.min(x_train), np.max(x_train), 100).reshape(-1, 1)
x_test_poly = poly_features.transform(x_test)

# 使用模型进行预测
y_pred = model.predict(x_test_poly)

# 绘制真实值和预测值的图像
plt.scatter(x_train, y_train, label='Actual')
plt.plot(x_test, y_pred, color='red', label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (Degree {}) of Class 3'.format(best_degree))
plt.legend()
plt.show()

# 使用模型对训练数据进行预测
y_train_pred = model.predict(x_train_poly)

# 计算最终的残差和
residual_sum = np.sum(np.abs(y_train - y_train_pred))

# 打印最终的残差和
print(residual_sum, best_degree)

i = 0
for idx in idx_3:
    resid = np.mean((y_train - y_train_pred)[i:i+7])
    resid_list2[idx] = resid
    i += 1
# 找到空数据所在的索引
nan_indices = np.isnan(filtered_x4) | np.isnan(filtered_y4)

# 获取空数据所在的索引
indices = np.where(nan_indices)[0]

# 删除空数据所在的索引对应的元素
x4 = np.delete(filtered_x4, indices)
```

```
y4 = np.delete(filtered_y4, indices)
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# 将 train_data 转换为二维数组
x_train = np.array(x4).reshape(-1, 1)

# 将 train_label 转换为二维数组
y_train = np.array(y4).reshape(-1, 1)

best_degree = None
best_residual_sum = float('inf')

# 尝试不同的多项式阶数
for degree in range(1, 10):
    # 使用多项式特征转换
    poly_features = PolynomialFeatures(degree=degree)
    x_train_poly = poly_features.fit_transform(x_train)

    # 创建多项式回归模型并进行训练
    model = LinearRegression()
    model.fit(x_train_poly, y_train)

    # 使用模型对训练数据进行预测
    y_train_pred = model.predict(x_train_poly)

    # 计算残差和
    residual_sum = np.sum(np.abs(y_train - y_train_pred))

    # 更新最小残差和和对应的阶数
    if residual_sum < best_residual_sum:
        best_residual_sum = residual_sum
        best_degree = degree

# 使用最佳阶数重新训练模型
poly_features = PolynomialFeatures(degree=best_degree)
x_train_poly = poly_features.fit_transform(x_train)
model = LinearRegression()
model.fit(x_train_poly, y_train)

# 生成测试数据
```

```
x_test = np.linspace(np.min(x_train), np.max(x_train), 100).reshape(-1, 1)
x_test_poly = poly_features.transform(x_test)

# 使用模型进行预测
y_pred = model.predict(x_test_poly)

# 绘制真实值和预测值的图像
plt.scatter(x_train, y_train, label='Actual')
plt.plot(x_test, y_pred, color='red', label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Polynomial Regression (Degree {}) of Class 4'.format(best_degree))
    plt.legend()
    plt.show()

    # 使用模型对训练数据进行预测
    y_train_pred = model.predict(x_train_poly)

    # 计算最终的残差和
    residual_sum = np.sum(np.abs(y_train - y_train_pred))

    # 打印最终的残差和
    print(residual_sum, best_degree)
    i = 0
    for idx in idx_4:
        resid = np.mean((y_train - y_train_pred)[i:i+9])
        resid_list2[idx] = resid
        i += 1
    from openpyxl import load_workbook

    # 打开答案文件.xlsx
    workbook = load_workbook('../E 题/数据/表 4-答案文件.xlsx')

    # 选择活动的工作表
    worksheet = workbook.active

    # 获取起始单元格的列号和行号
    prob_start_column = 'G'
    start_row = 4

```

```
    # 逐行写入 result 列表的元素
    for i in range(len(resid_list2)):
        # 定位
        prob_current_column = prob_start_column
        prob_cell_coordinate = f"{prob_current_column}{start_row + i}"
        # 写入
        worksheet[prob_cell_coordinate] = resid_list2[i]

    # 写入内容 1 到指定单元格
    for i, value in enumerate(idx_1):
        # 计算目标单元格的行和列索引
        row = 'H'
        column = value + 4

        # 将内容 1 写入目标单元格
        worksheet[f"{row}{column}"] = 1

    for i, value in enumerate(idx_2):
        # 计算目标单元格的行和列索引
        row = 'H'
        column = value + 4

        # 将内容 1 写入目标单元格
        worksheet[f"{row}{column}"] = 2

    for i, value in enumerate(idx_3):
        # 计算目标单元格的行和列索引
        row = 'H'
        column = value + 4

        # 将内容 1 写入目标单元格
        worksheet[f"{row}{column}"] = 3

    for i, value in enumerate(idx_4):
        # 计算目标单元格的行和列索引
        row = 'H'
        column = value + 4

        # 将内容 1 写入目标单元格
        worksheet[f"{row}{column}"] = 4

```

```
# 保存工作簿到文件
workbook.save('../E 题/数据/表 4-答案文件.xlsx')
def calculate_difference(row):
    first_valid = row.first_valid_index()  # 获取第一个非 NaN 值的索引
    last_valid = row.last_valid_index()  # 获取最后一个非 NaN 值的索引
    if first_valid is None or last_valid is None:  # 如果没有非 NaN 值，则返回 NaN
        return np.nan
    else:
        return row[first_valid] - row[last_valid]  # 计算差值
Y_train['水肿体积差'] = Y_train.apply(calculate_difference, axis=1)
table_1_issue2 = pd.concat([table_1.iloc[:100, -10:-3], Y_train['水肿体积差']], axis=1)
table_1_issue2
# 计算相关系数矩阵
corr_matrix = table_1_issue2.corr()
# 创建遮盖矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
# 绘制相关系数矩阵的热力图
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
plt.title('Correlation Matrix')
plt.show()
table_2_issue2 = table_2_issue1.iloc[:100, 1::2]
table_2_issue2['血肿体积差'] = table_2_issue2.apply(calculate_difference, axis=1)
table_issue2 = pd.concat([table_1.iloc[:100, -10:-3], table_2_issue2['血肿体积差'], Y_train['水肿体积差']], axis=1)
table_issue2
# 计算相关系数矩阵
corr_matrix = table_issue2.corr()

# 绘制相关系数矩阵的热力图
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Correlation Matrix')
plt.show()
```

```
train_data
table_1.shape
table_11 = pd.concat([table_1.iloc[:, 1:-3], table1['90 天 mRS']], axis=1)
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# 创建数据
data = table_11.iloc[:, :]

# 计算相关系数矩阵
corr_matrix = data.corr()

# 创建遮盖矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# 绘制相关矩阵
plt.figure(figsize=(16, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
plt.title('Correlation Matrix')
plt.show()
# 读表 1 的数据
table_1_issue3 = table_11[['年龄', '糖尿病史', '冠心病史', '吸烟史', '饮酒史', '脑室引流', '90 天 mRS']].iloc[:, :]
table_1_issue3
table_22 = pd.concat([table_2.iloc[:, 2:24], table_1_issue3['90 天 mRS']], axis=1)
table_22

# 计算相关系数矩阵
corr_matrix = table_22.corr()
# 创建遮盖矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
# 绘制相关系数矩阵的热力图
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
plt.title('Correlation Matrix')
plt.show()
```

```
    # 读表1的数据
    table_2_issue3 = table_22[['HM_volume', 'ED_volume']]
    table_2_issue3
    # 计算相关系数矩阵
    corr_matrix = table_3.iloc[:160, 2:].corr()
    # 创建遮盖矩阵
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
    # 绘制相关系数矩阵的热力图
    plt.figure(figsize=(10, 8))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
    plt.title('Correlation Matrix')
    plt.show()
    table_3_issue3 = table_3[['NCCT_original_firstorder_90Percentile', 'NCCT_original_firstorder_InterquartileRange', 'NCCT_original_firstorder_MeanAbsoluteDeviation', 'NCCT_original_firstorder_Range', 'NCCT_original_firstorder_RobustMeanAbsoluteDeviation']].iloc[:160, :]
    table_3_issue3.shape
    train_data3, test_data3 = table_issue3.iloc[:100, :-1], table_issue3.iloc[100:, :-1]
    train_label3, test_label3 = table_issue3.iloc[:100, -1:], table_issue3.iloc[100:, -1:]
    train_data3.shape, test_data3.shape, train_label3.shape, test_label3.shape
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score

    # 创建逻辑回归模型对象
    logreg = LogisticRegression(multi_class='ovr')

    # 训练逻辑回归模型
    logreg.fit(train_data3, train_label3)

    # 在测试集上进行预测
    logreg_pred = logreg.predict(train_data3)

    # 计算逻辑回归模型的精度
    logreg_accuracy = sum(train_label3.iloc[:, 0] == logreg_pred) / len(logreg_pred)
    print("逻辑回归模型精度：", logreg_accuracy)
    from sklearn.ensemble import RandomForestClassifier
```

```
1172.
1173.   # 创建随机森林模型
1174.   rf = RandomForestClassifier(n_estimators=8)
1175.
1176.   # 训练随机森林模型
1177.   rf.fit(train_data3, train_label3)
1178.
1179.   # 在测试集上进行预测
1180.   rf_pred = rf.predict(train_data3)
1181.   # 计算随机森林模型的精度
1182.   rf_accuracy = sum(train_label3.iloc[:, 0] == rf_pred) / len(r
      f_pred)
1183.   print("随机森林模型精度：", rf_accuracy)
1184.   from sklearn.neighbors import KNeighborsClassifier
1185.   from sklearn.metrics import accuracy_score
1186.
1187.   # 创建K 临近算法模型对象
1188.   knn = KNeighborsClassifier(n_neighbors=2)
1189.
1190.   # 训练K 临近算法模型
1191.   knn.fit(train_data3, train_label3)
1192.
1193.   # 在测试集上进行预测
1194.   knn_pred = knn.predict(train_data3)
1195.
1196.   # 计算K 临近算法模型的精度
1197.   knn_accuracy = sum(train_label3.iloc[:, 0] == knn_pred) / len
      (knn_pred)
1198.   print(f"K 临近算法模型精度：", knn_accuracy)
1199.   mRS_list = rf.predict(table_issue3.iloc[:, :-1])
1200.   mRS_list
1201.   from openpyxl import load_workbook
1202.
1203.   # 打开答案文件.xlsx
1204.   workbook = load_workbook('../E 题/数据/表 4-答案文件.xlsx')
1205.
1206.   # 选择活动的工作表
1207.   worksheet = workbook.active
1208.
1209.   # 获取起始单元格的列号和行号
1210.   prob_start_column = 'I'
1211.   start_row = 4
```

```
1212.
1213.   # 逐行写入 result 列表的元素
1214.   for i in range(len(mRS_list)):
1215.       # 定位
1216.       prob_current_column = prob_start_column
1217.       prob_cell_coordinate = f"{prob_current_column}{start_row
    + i}"
1218.       # 写入
1219.       worksheet[prob_cell_coordinate] = mRS_list[i]
1220.
1221.
1222.   # 保存工作簿到文件
1223.   workbook.save('../E 题/数据/表 4-答案文件.xlsx')
1224.   table = pd.concat([table_1.iloc[:100, :], table_2.iloc[:100,
    1:], table_3.iloc[:100, :]], axis=1)
1225.   table.shape
1226.   table = table.fillna(table.mean())
1227.   # 数据清洗
1228.   # 使用均值进行缺失值填充
1229.   table.fillna(table.mean(), inplace=True)
1230.   # 使用 Z-score 方法检测和处理异常值
1231.   from scipy import stats
1232.   z_scores = stats.zscore(table)
1233.   df = table[(z_scores < 3).all(axis=1)]
1234.
1235.   null_counts = df.isnull().sum()
1236.   print(null_counts)
1237.   print(table.dtypes)
1238.   table_1.head()
1239.   # 创建数据
1240.   data = table_1[['年龄', '性别', '吸烟史', '饮酒史
    ']].assign(mRS=train_label3)  # 假设 train_data 的标签位于第一列
1241.
1242.   # 计算相关系数矩阵
1243.   corr_matrix = data.corr()
1244.
1245.   # 创建遮盖矩阵
1246.   mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
1247.
1248.   # 绘制相关矩阵
1249.   plt.figure(figsize=(16, 8))
```

```
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
    plt.title('mRS 和个人史相关矩阵')
    plt.show()
    table_1 = table_1.drop(columns=['吸烟史', '饮酒史'])
    table_1
    # 创建数据
    data = table_1.iloc[:100, 4:9].assign(mRS=train_label3)  # 假设train_data 的标签位于第一列

    # 计算相关系数矩阵
    corr_matrix = data.corr()

    # 创建遮盖矩阵
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    # 绘制相关矩阵
    plt.figure(figsize=(16, 8))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
    plt.title('mRS 和疾病史相关矩阵')
    plt.show()
    # 创建数据
    data = table_1.iloc[:100, 10:-3].assign(mRS=train_label3)  # 假设train_data 的标签位于第一列

    # 计算相关系数矩阵
    corr_matrix = data.corr()

    # 创建遮盖矩阵
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    # 绘制相关矩阵
    plt.figure(figsize=(16, 8))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True, mask=mask)
    plt.title('mRS 和治疗方法相关矩阵')
    plt.show()
    table_2
```