基于相关关系的古代玻璃文物的成分分析与预测

摘要

古代中西方文化交流过程中，玻璃是贸易往来的宝贵物证，具有极大研究价值。但古代玻璃的化学成分差异较大，玻璃文物的分类标准、风化前后化学成分含量的差异。

针对问题一，对于表面风化与其玻璃类型、纹饰和颜色的相关关系，本文采用**卡方分析**和 ***spearman* 系数**对差异性关系和相关性关系进行分析，得知仅有类型与风化与否具有相关关系，并用**二元 *logistic* 回归**量化了其线性关系的大小；对于不同类型有无风化化学成分含量的统计规律，本文对每种类型不同的化学物质绘制了**核密度图**，通过分析风化前后均值与方差的变化，将化学物质分为了增长型、下降型和平稳型，并对数据的离散程度进行分析；对于其风化前的化学成分含量的预测，本文构建了基于**风化前后概率密度分布**的预测模型，综合考虑均值与方差，得出风化前后化学物质含量的一般规律。

针对问题二，对于高钾玻璃、铅钡玻璃的分类规律，本文结合 ***spearman* 相关系数**，总结分类规律大致为：风化前 *K* 元素含量高的倾向于分为高钾玻璃，*Pb*、*Ba* 元素含量高的倾向于分为铅钡玻璃；风化后 *Si* 元素含量高的倾向于分为高钾玻璃，*Pb*、*Ba*、*Sr* 元素含量高的倾向于分为铅钡玻璃。

对于不同类别的亚类划分，可采用 ***k-means* 聚类算法**分别对风化前和风化后高钾、铅钡玻璃进行聚类，根据轮廓系数图确定聚类数目，并通过**聚类项的重要性指标**得出亚分类的分类指标。风化前高钾玻璃可以分为 *Ba* 与低 *Ba* 两个亚类，铅钡玻璃可以分为高 *Cu-K* 和低 *Cu-K* 两个亚类；风化后高钾玻璃可以分为高 *Al* 与低 *Al* 两个亚类，铅钡玻璃可以分为高 *Ca-Fe* 和低 *Ca-Fe* 两个亚类。最后，我们通过去除某个化学物质重新计算轮廓系数，对化学物质的敏感性进行分类，得出风化前后高钾玻璃都对 SiO_2 敏感，而铅钡玻璃在风化前没有特别敏感的化学元素，风化后对 *CaO* 和 *BaO* 敏感。

针对问题三，结合问题二相关系数的分析，本文对与类别相关系数较大的几种化学物质采用**加权软投票法**，对每种化学物质构建**生成类别概率的二层决策树**作为基分类器，以相关系数作为基分类器的权重进行投票。对原有数据进行检验，识别正确率到达 100%，证明了该集成算法的有效性。以此对未知文物进行预测，正文对预测结果进行说明。本文通过改变预测模型分析**对风化条件的敏感性**，其次通过对各化学元素值进行放缩分析**对化学元素含量的敏感性**，得出该预测模型对风化条件和氧化铅具有敏感性。

针对问题四，本问绘制出不同类别样品风化前后各化学成分的**相关关系**图和**偏相关关系**图，通过同一类型相关系数图和偏相关图的对比，总结化学成分之间的正负线性关系、线性关系强弱、共线性关系；同时对比不同类型化学成分的偏自相关图，总结化学成分在不同玻璃类型中正负关系与关系强弱的差异性。

关键词：*spearman* 系数 二元 *logistic* 回归 加权软投票法 相关关系 偏相关关系

一．问题重述

1.1 问题的背景

玻璃在早期贸易往来中常被制作成珠形饰品传入我国，其主要原料是石英砂，主要化学成分是二氧化硅。制作玻璃的过程中，由于纯石英砂的熔点较高，常使用助熔剂来降低熔化温度，添加的助熔剂不同，会导致玻璃的化学成分有所差异。

我国自己发明的玻璃品种——铅钡玻璃，烧制过程中以铅矿石作为助熔剂，使玻璃中的氧化铅和氧化钡的含量较高；制作钾玻璃的过程中，加入了含钾量高的物质如草木灰作为助熔剂，因此钾玻璃中氧化钾的含量较高。

古代玻璃在长时间的埋藏过程中，极易受周围环境影响而风化。风化过程中，玻璃的内部元素与环境元素进行大量交换，导致其化学成分比例发生变化，玻璃表面会变暗、透明度降低、晕色或是在其外部形成风化产物结壳，风化严重的玻璃几乎无法辨认其原貌，从而无法正确判断其类别。

1.2 问题的重述

现给出一批我国古代玻璃制品的相关数据，附件给出了文物的风化程度、分类信息及对应的化学成分比例。根据问题背景及附件需要解决以下问题：

问题一：分析研究玻璃文物的表面风化与其玻璃类型、纹饰和颜色的关系；根据玻璃的类型，统计分析玻璃样品表面在风化前后的化学成分含量的变化规律，并结合其风化点的检测数据预测其风化前的化学成分含量。

问题二：根据附件数据分析高钾玻璃和铅钡玻璃的分类规律；选择合适的化学成分对玻璃类别进行亚类划分，给出具体的划分方法及划分结果，并分析其合理性和敏感性。

问题三：分析附件表单 3 中的玻璃文物的化学成分，判断其玻璃类型，并分析结果的敏感性。

问题四：针对不同类别的玻璃文物，分析其化学成分之间的关联关系，并对其差异性进行比较。

二．问题分析

全文主要针对类别、风化、化学成分三个要素进行分析与预测。问题一首先分析了类别与风化的相关关系，其次分析化学物质与风化的相关关系，并根据化学物质含量对已风化玻璃风化前的物质含量进行预测；问题二分析了化学物质与类型的关系，并由此给出了亚类划分；问题三沿袭问题二的思路，根据化学物质含量对玻璃类型进行了预测；问题四则针对各个化学物质，对不同类型化学物质之间的关系进行分析。全文分析思路如图 1 所示：

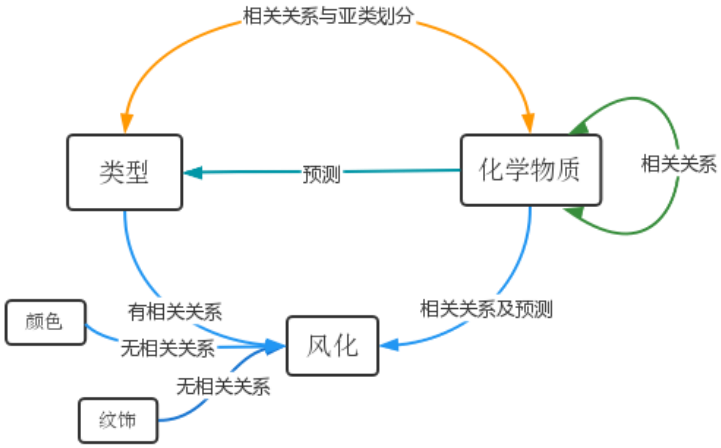

图 1 思路分析图

2.1 问题一分析

第一小问要求根据附件信息分析玻璃文物的表面风化与其玻璃类型、纹饰和颜色的关系。对此需要进行相关性分析、差异性分析和影响程度分析。相关性分析可采用 *Spearman* 相关系数来判断变量之间的相关关系，并结合卡方检验表面风化对各个因素的差异性。基于相关性分析，对变量进行二元 *logistic* 回归，分析自变量对因变量的影响程度。

第二小问需要根据玻璃类型分析玻璃样品表面在风化前后化学成分含量的变化规律，我们对每种类型不同的化学物质绘制核密度图，通过观察风化前后核密度图的均值和方差变化概括化学物质的变化规律。

第三小问需要根据风化点的检测数据预测其风化前的化学物质含量，我们假设化学物质的含量分布基本为无偏分布，通过结合第二小问的均值和方差找到风化前后概率密度分布规律对风化前的化学含量进行预测。

2.2 问题二分析

第一小问需要分析高钾玻璃和铅钡玻璃的分类规律。我们结合 *Spearman* 相关系数，将与类型相关系数最大的几种化学物质作为高钾玻璃和铅钡玻璃的大致分类标准。第二小问要求对玻璃类别进行亚类划分。本题的难点在于如何选择合适的化学成分进行划分。对此可先采用 *K-means* 聚类的轮廓系数图确定划分类别数，再利用聚类项的重要性指标得出亚分类的分类标准。对于模型的合理性分析，本问采用亚分类分类物质分类的散点图进行说明；对于模型的敏感性分析，本问通过减少某一化学物质再次聚类后轮廓系数的改变，分析模型对于化学物质的敏感性。

2.3 问题三分析

附件表单 3 中给出了未知类别玻璃文物的相关信息，通过对化学成分的分析鉴别其所属类型。本问的难点在于如何量化各化学成分对玻璃类型的影响程度。对此可建立基于二层决策树和加权软投票法的分类模型，确定与玻璃类型相关的影响因子，并利用相关系数求解每个影响因子的权重；类别概率可利用影响因子的取值与分界值的关系确定；之后根据权重和类别概率得到判别函数；最后通过比较高钾玻璃和铅钡玻璃的得分确定玻璃文物的类型。

2.4 问题四分析

问题四需要化学物质之间的关联关系及不同类别之间的差异性。本问的难点在于如何分析化学物质之间的关联关系。本问仍然延续第一、二问的相关系数分析思路，我们绘制相关系数图分析化学物质之间的局部线性相关关系，再利用偏相关系数分析控制变量条件下化学物质之间的线性相关关系。通过比较同一种类相关系数图与偏相关系数图、不同种类的相关系数图与偏相关系数图找出化学物质之间的关联关系及差异性。

三．模型假设

1. 各化学成分的测量无误差或误差不大。
2. 不存在表中未提及的化学物质会对本模型的结论具有重大影响。
3. 不存在除化学物质、玻璃类型、风化之外因素的影响关系。
4. 假设相同类型的玻璃风化后化学物质的变化规律具有一般性。

四．符号说明

符号	含义
B_{ij}^{k}	类型为 k，文物采样点编号为 j，化学物质为 i 的含量
p	事件发生的显著性
E_{i}^{k}	风化前后化学物质为 i 的含量均值
S_{i}^{k}	风化前后化学物质为 i 的含量方差
S	轮廓系数
K	聚类类别数目
Ω_i	类别概率
y_i	加权软投票法的判别函数

五．模型的建立与求解

5.1 建模前的准备

5.1.1 钾玻璃

钾玻璃（K_2O-SiO_2），色调以蓝绿色为主，包括湖蓝色、深蓝色、浅蓝色等，其主要助熔剂为氧化钾（K_2O），且氧化钾的含量一般不低于 10%；同时，钾玻璃中的二氧化硅（SiO_2）含量高，硅氧四面体[SiO_4]互相连接程度大，化学稳定性好，使玻璃具有较好的抗风化性能；此外钾玻璃还含有一定数量的氧化铝（Al_2O_3），它形成的[AlO_4]四面体对硅氧网络起补网作用，对提高化学稳定性有所帮助。

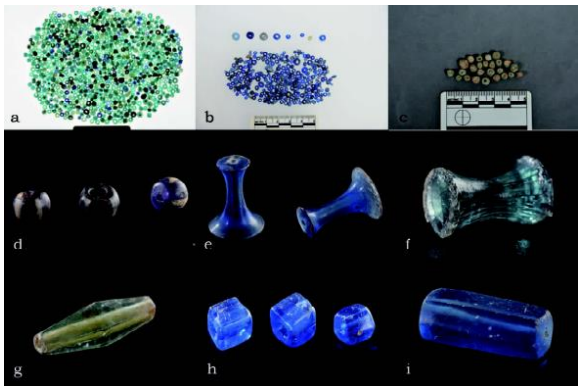

图 2 钾玻璃文物

钾玻璃中三种主要成分——SiO_2、K_2O、Al_2O_3，它们的含量虽有所差异，但均在一定范围内波动。广州出土的汉代钾玻璃制品中的 SiO_2 含量平均为 74.69%，大部分样品的 K_2O 含量在 13.72%以上，Al_2O_3 的含量大概为 7.15%；西安出土的战国至东汉时期的钾玻璃制品中，SiO_2、K_2O 和 Al_2O_3 的含量分布范围分别为 81.66%-82.73%、8.51%-8.74%和 2.28%-2.40%。钾玻璃的主要化学成分含量如下图所示。从图 3 中可以看到，大部分钾玻璃的主要成分及其含量是大致相同的。

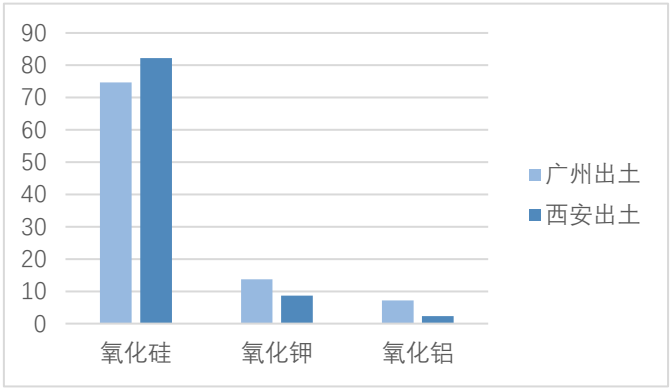

图 3 钾玻璃主要成分含量图

钾玻璃经过长时间的埋藏，极易受周围环境影响而风化，表层中的 K_2O 会大量流失，SiO_2 的含量显著提高，从而在玻璃表面形成一层薄层风化层，风化层中的 K_2O 含量会急剧降低。在文献《一批中国汉墓出土钾玻璃的研究》中，给出了其研究的钾玻璃样品表面风化层化学组成变化的结果，如表 1 所示。通过表中数据可以初步看到，风化前后 SiO_2 和 Al_2O_3 的含量升高，而 K_2O 的含量下降。

成分 / 样品		二氧化硅	氧化钾	氧化铝
样品一	风化前	78.48%	16.75%	1.91%
	风化后	89.52%	1.29%	4.4%
样品二	风化前	78.34%	15.93%	4.6%
	风化后	90.02%	1.68%	5.9%
样品三	风化前	77.87%	16.97%	1.55%
	风化后	81.16%	1.41%	11.55%

表 1 风化前后成分含量变化

5.1.2 铅钡玻璃

铅钡玻璃（$PbO\text{-}BaO\text{-}SiO_2$），其主要成分为二氧化硅（$SiO_2$）、氧化铅（$PbO$）和氧化钡（$BaO$），颜色主要为透明/半透明蓝色或绿色，其显色元素为 Cu 和 Fe，是我国古代自己发明的玻璃。

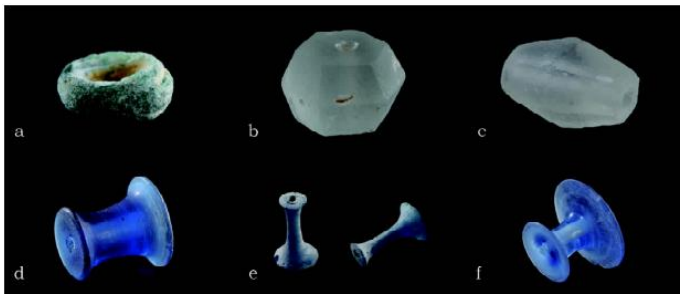

图 4 铅钡玻璃文物

据统计，铅钡玻璃中的主要化学成分基本一致，但含量有所不同。广州汉墓玻璃珠饰中的铅钡玻璃样品，其 PbO、BaO 的含量范围分别为 26.89%-40.48%、3.02%-11.37%；西安出土的东汉时期的铅钡玻璃样品，大部分保存完好，PbO 和 BaO 含量分布范围分别是 13.29%-51.13%和 0.31%-10.02%；湖南出土的战国时期的铅钡玻璃制品，其中 PbO 的含量在 29%-48%，BaO 的含量为 5%-21%。铅钡玻璃的主要化学成分含量如图 5 所示。虽然成分含量的范围有所不同，在一定范围内波动，但其含量的均值差异不大，具有参考价值。

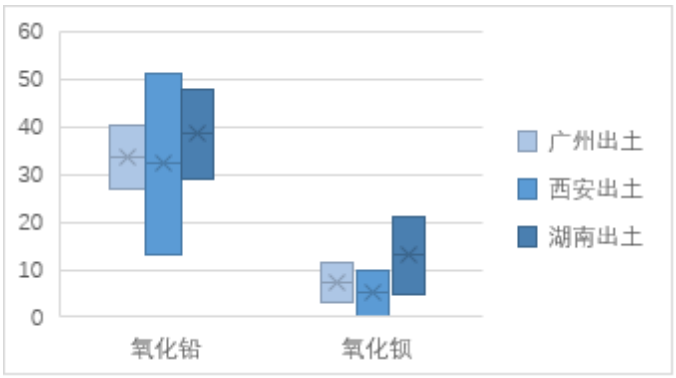

图 5 铅钡玻璃主要成分含量图

铅钡玻璃制品极易受环境影响而产生风化。风化过程中，玻璃内部的元素与环境中的元素大量交换，Si 由里至外顺着风化层的走向流失，在最外层的结壳中含量非常低；Ba 在最外层结壳中积聚形成夹心结构在最外层；S 在最外层积聚；Mg、Ca 等元素在越靠近最内层未风化部位的位置含量越高；Pb 的含量由外到内呈逐步降低的趋势；主要的显色元素 Cu 向外流失，Fe 元素则在外层积聚。经风化后的铅钡玻璃制品在外层结壳中生成了大量的 $PbCO_3$，其是由玻璃中高含量的 Pb 与外界环境中的 CO_2、水蒸汽反应生成的，是导致玻璃风化的主要原因。

5.2 问题一的建模与求解

5.2.1 表面风化的因素分析

问题一中需要分析研究玻璃样品表面风化与其玻璃类型、纹饰和颜色的关系，对此分别利用卡方分析和 spearman 相关系数对其进行差异性分析和相关性分析，研究随机变量之间的关联关系及线性关系；基于相关系数，对随机变量进行二元 $logistic$ 回归

分析，研究玻璃类型、纹饰和颜色对表面风化的线性关系大小。

5.2.1.1 相关性分析及检验

（1）卡方检验

为研究表面风化对颜色、类型和纹饰的差异关系，可利用卡方检验对数据进行分析。卡方值作为非参数检验中的一个统计量，主要用于检验数据的相关性。卡方分析的根本思想是利用样本的实际频数和理论频数的吻合度，来判断样本是否符合预期。若卡方值的显著性小于 0.05，说明两个变量是显著相关的。

将整理好的数据导入 *SPSSAU* 后，可得到相应的卡方值与 p 值，如表 2 所示。

		表面风化		x^2	*p*
		无风化	风化		
纹饰	A	45.83%	32.35%	4.957	0.084
	B	0.00%	17.65%		
	C	54.17%	50.00%		
玻璃类型	铅钡	50.00%	82.35%	6.880	0.009
	高钾	50.00%	17.65%		
颜色	浅绿	8.33%	2.94%	9.432	0.307
	浅蓝	33.33%	35.29%		
	深绿	12.50%	11.76%		
	深蓝	8.33%	0.00%		
	空	0.00%	11.76%		
	紫	8.33%	5.88%		
	绿	4.17%	0.00%		
	蓝绿	25.00%	26.47%		
	黑	0.00%	5.88%		

表 2 卡方值及置信度

从上表可知，由于三个因素中纹饰和颜色的置信度均大于 0.05，表明表面风化对纹饰和颜色不会展现出显著性，对纹饰以及多种颜色均表现出一致型，并没有太大差异。而表面风化对玻璃类型呈现出显著性（$p<0.05$），意味着表面风化对不同的玻璃类型会产生差异性。就表中数据而言，在表面风化的玻璃中，铅钡玻璃占比 82.35%，而高钾玻璃占比 17.65%；在无风化的玻璃中，铅钡玻璃和高钾玻璃各占比 50%，从中可以发现表面风化选择铅钡玻璃的比例 82.35%远高于无风化的选择比例 50%，因此可以认为相对于铅钡玻璃，高钾玻璃的抗风化能力更强。

综合上述的分析和检验，可知“表面风化与玻璃类型有显著相关关系，与颜色和纹饰几乎无相关关系”。

（2）相关分析

相关分析是研究两个或两个以上处于同等地位的随机变量间的相关关系的统计分析，侧重于发现随机变量间的种种相关特性。对于相关关系，按相关程度可分为完全相关、不完全相关和不相关，按相关形式可分为线性相关和非线性相关，按相关方向分为正相关和负相关，按影响因素的多少分为单相关和复相关。

根据题目要求，可将玻璃类型、纹饰和颜色作为自变量，表面风化作为因变量。

通过统计附件表单 1 中的数据，可得到表面风化与玻璃类型、纹饰和颜色相关图。

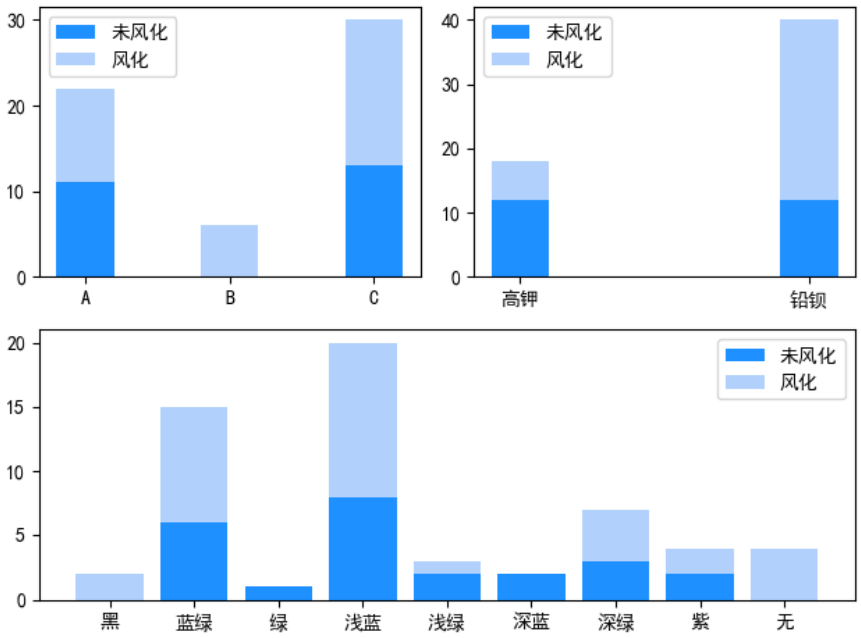

图 6 表面风化与纹饰、颜色、类型的相关图

基于上述表面风化与玻璃类型、纹饰和颜色的统计数据，可以计算因变量与自变量之间的相关系数。相关系数是用来反映变量之间线性相关程度的统计指标，主要有 *Pearson* 相关系数和 *Spearman* 相关系数。*Pearson* 相关系数要求统计资料是连续变量且符合正态分布，由于附件中的数据分布并不严格服从正态分布，故本问采用 *Spearman* 相关系数来进行分析。

Spearman 相关系数也称秩相关系数，可以反映两个变量之间的关联程度与相关方向。相关系数的绝对值越接近 1 时，两变量越接近单调相关，相关系数为 0 时表明变量之间无相关关系。

将整理后的数据导入 *SPSSAU* 之后，对数据进行相关性分析，可得到 *Spearman* 相关系数、置信度以及相关图。

因素	选项	表面风化
纹饰	相关系数	0.037
	p 值	0.781
类型	相关系数	-0.344
	p 值	0.008
颜色	相关系数	0.086
	p 值	0.519

表 3 *Spearman* 相关系数及置信度

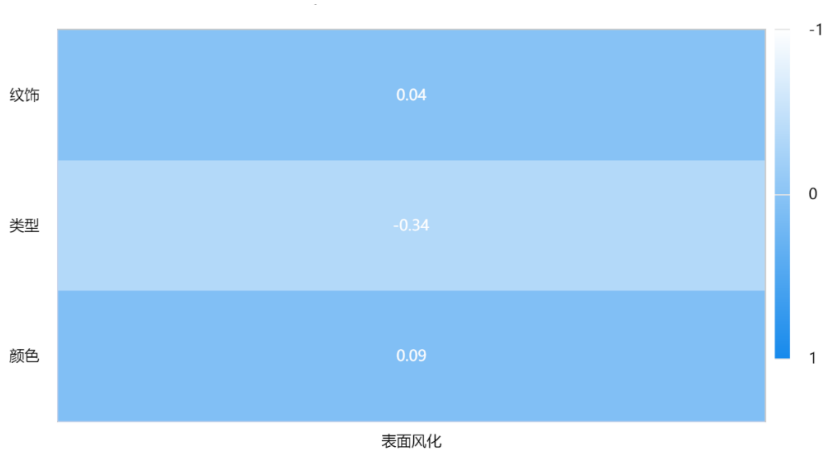

图 7 *Spearman* 相关图

由表中数据可知，表面风化与纹饰和颜色的相关系数极小且置信度较低，表明玻璃表面风化与纹饰和颜色之间没有相关关系。

表面风化与玻璃类型之间的相关系数为-0.344，并呈现出 0.01 水平的显著性，表明玻璃表面风化与玻璃类型之间有显著的负相关关系。

5.2.1.2 二元 *logistic* 回归分析

基于上述对表面风化与纹饰、颜色、玻璃类型的相关性分析，可以利用二元 *logistic* 回归分析研究玻璃类型、纹饰和颜色对表面风化的线性影响程度，以颜色、纹饰和玻璃类型作为自变量，玻璃表面风化情况的 0-1 变量为因变量，0 表示玻璃表面未风化，1 表示玻璃表面风化。

基于 *logistic* 回归分析的理论基础，可以得到关于玻璃表面风化概率的表达式，即：

$$\ln\frac{P}{1-P}=\beta_0+\beta_1x_1+\cdots+\beta_nx_n+\varepsilon \tag{1}$$

其中，β_i 为各变量的回归系数，x_i 为第 i 个变量，P 的值表示结果为玻璃表面风化的概率。

将整理好的数据导入 *SPSSAU* 后，可以得到表面风化与纹饰、颜色、玻璃类型的关系表达式：

$$\ln(\frac{P}{1-P})=1.911+0.284\cdot x_1\text{-}2.378\cdot x_2+0.158\cdot x_3 \tag{2}$$

其中，x_1 表示颜色，x_2 表示玻璃类型，x_3 表示纹饰

表面风化与纹饰、颜色、玻璃类型的 *logistic* 回归分析结果如表 4 所示。

因素	回归系数	*p* 值	OR 值
类型	-2.378	0.004	0.093
纹饰	0.158	0.624	1.172
截距	1.911	0.104	6.759

表 4 *logistic* 回归分析结果

由表中数据可知，颜色和纹饰的 p 值大于 0.05，表示这两个因素没有意义，不予考虑；玻璃类型的 p 值小于 0.01，呈现 0.01 水平的显著性；且类型的回归系数小于 0 且 OR 值小于 1，说明玻璃类型对表面风化呈负相关，即相对于高钾玻璃，铅钡玻璃更容易风化。

5.2.2 风化前后化学成分含量分析

（1）数据清洗

根据题目要求，仅将附件二文物中各成分比例累加和介于 85%-105%之间的数据视为有效数据，因此需要对附件的数据进行清洗。通过统计数据发现，文物编号为 15 和 17 的各成分比例累加和小于 85%，不予考虑。

（2）化学成分的统计规律

在制作玻璃时采用的助熔剂不同，会导致玻璃样品的化学成分有较大差异，因此需要根据玻璃类型将样品划分为高钾玻璃和铅钡玻璃来进行化学成分的分析。

●高钾玻璃

根据对附件表单 2 中数据的统计分析，高钾玻璃在风化前后的化学成分含量比较如图 8 所示。

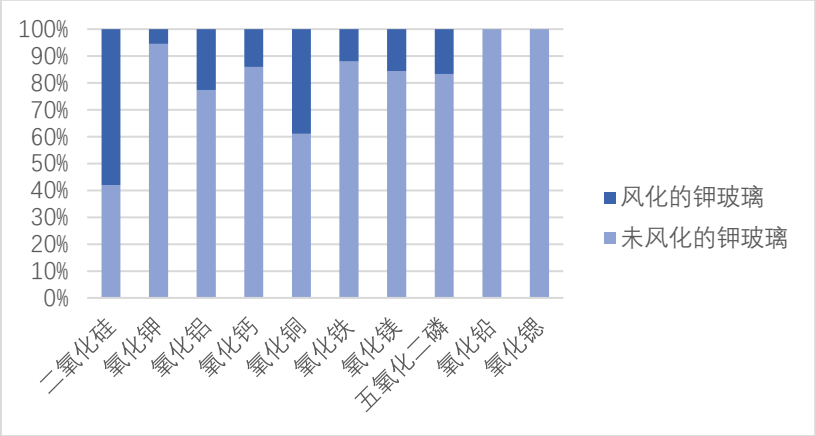

图 8 风化前后高钾玻璃化学成分含量比较图

由于部分化学成分（Na_2O、BaO、SnO 和 SO_2）只有少数高钾玻璃含有，且风化后含量均为 0，不具有普遍意义，在本问题中不进行分析。对于图中的十种化学成分，可以明显看到在风化前后仅有 SiO_2 的含量有所增加，其余九种化学成分的含量均有不同程度的下降。为更加清晰地看到每种成分的变化规律，可采用核密度函数来展现每种物质在风化前后的分布情况，并给出风化前后物质含量的均值和方差，如表 5 所示。对于化学成分的变化，可分为两种情况——增长型和下降型。

化学成分	风化前（含量百分比）		风化后（含量百分比）	
	平均值	方差	平均值	方差
二氧化硅	67.98	70.26	93.96	2.50
氧化钠	0.70	1.52	0.00	0.00
氧化钾	9.33	14.09	0.54	0.17
氧化钙	5.33	8.77	0.87	0.20
氧化镁	1.08	0.42	0.20	0.08
氧化铝	6.62	5.69	1.93	0.78
氧化铁	1.93	2.55	0.27	0.00
氧化铜	2.45	2.53	1.56	0.73

氧化铅	0.40	0.32	0.00	0.00
氧化钡	0.60	0.88	0.00	0.00
五氧化二磷	1.40	1.88	0.28	0.04
氧化锶	0.0	0.00	0.00	0.00
氧化锡	0.20	0.43	0.00	0.00
二氧化硫	0.10	0.03	0.00	0.00

表 5 高钾玻璃各成分含量均值和方差

①增长型

通过上述分析以及表格数据可知，SiO_2 的含量变化属于增长型，其核密度函数如图 9 所示，其中 0 表示表面未风化，1 表示表面风化。

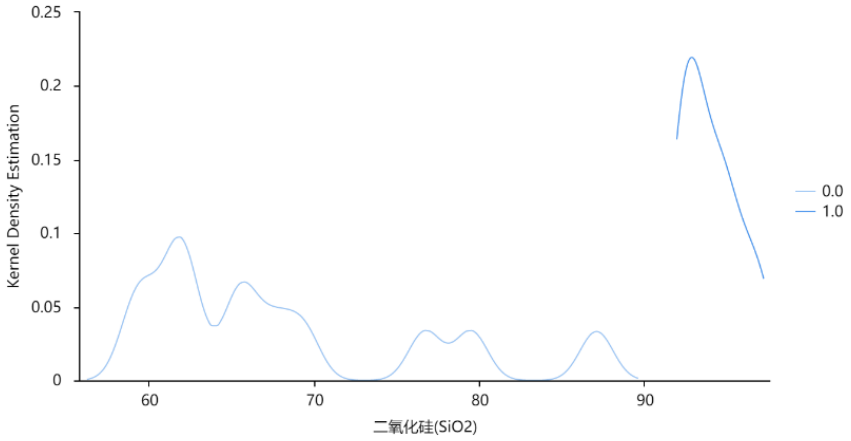

图 9 二氧化硅核密度函数图

由图可以明显看到风化后的 SiO_2 核密度函数向右聚拢，分布更集中，方差急剧下降，且平均含量从 67.98%增长到了 93.96%。

②下降型

通过表格中的数据可知，K_2O、Al_2O_3、CaO、CuO、Fe_2O_3、MgO、P_2O_5、PbO、SrO 的变化均属于下降型，但降低的程度与集中度有所差异。

对于 K_2O、Al_2O_3、CaO、Fe_2O_3、MgO 和 CuO 来说，它们的核密度函数均向左聚拢，并且分布更加集中，方差更小，但 K_2O、Al_2O_3、CaO 含量下降幅度较大，Fe_2O_3、MgO 和 CuO 的含量下降幅度较小。而对于 P_2O_5、PbO、SrO 而言，由于数据量较少，无法利用核密度函数进行分析，但它们的含量在风化后均有所降低，接近于 0。图 10 给出了氧化钾和氧化铜的核密度函数。

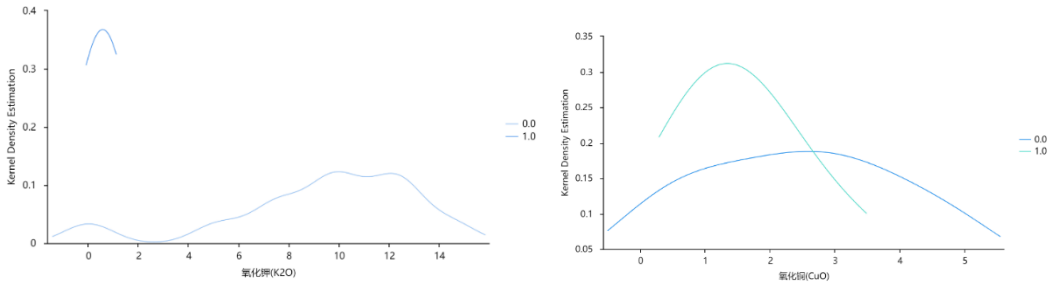

图 10 氧化钾和氧化铜的核密度函数图

●铅钡玻璃

根据对附件表单 2 中数据的统计分析，发现化学成分 SnO 和 SO_2 在风化前后均只有少数铅钡玻璃含有，样本数量较少，无法分析得到其变化规律，在本问题中不进行分析。利用统计的数据，可得到风化前后物质含量的均值和方差，如表 6 所示。根据化学成分的含量变化分为增长型、下降型和平稳型。

化学成分	风化前（含量百分比）		风化后（含量百分比）	
	平均值	方差	平均值	方差
二氧化硅	54.66	133.83	24.91	108.15
氧化钠	1.68	5.38	0.22	0.30
氧化钾	0.22	0.09	0.13	0.06
氧化钙	1.32	1.58	2.70	2.65
氧化镁	0.64	0.29	0.65	0.48
氧化铝	4.46	10.18	2.97	6.67
氧化铁	0.74	1.28	0.58	0.52
氧化铜	1.43	3.71	2.28	7.65
氧化铅	22.08	64.55	43.31	143.83
氧化钡	9.00	32.46	11.81	95.74
五氧化二磷	1.05	3.26	5.28	16.93
氧化锶	0.27	0.06	0.42	0.07
氧化锡	0.05	0.02	0.07	0.07
二氧化硫	0.16	0.56	1.37	17.01

表 6 铅钡玻璃各成分含量均值和方差

①增长型

通过化学成分含量的核密度函数分析，可以将 CaO、PbO、CuO 、BaO、P_2O_5 划分为增长型。PbO 的平均含量增长幅度大，从 22.08%增长到 33.35%，但由于方差变大，因此含量分布更加分散；CaO、BaO 和 CuO 的平均含量增长幅度小，但方差变化不大，结合核密度函数发现风化曲线变化的规律基本一致；P_2O_5 的平均含量增长约为 4%，但方差变大，核密度函数分布更加分散。图 11 给出了氧化铅和氧化铜的核密度函数。

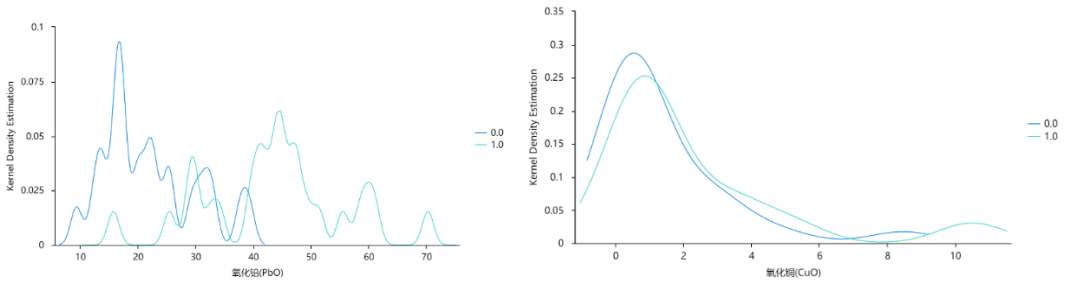

图 11 氧化铅和氧化铜的核密度函数图

②下降型

通过化学成分含量的核密度函数研究，可以将 SiO_2、Al_2O_3、Na_2O 划分为下降型。SiO_2 的平均含量下降幅度大，约为 30%，而 Al_2O_3、Na_2O 平均含量下降幅度小，仅有 1.5%。它们的方差变小，核密度函数均有一定程度的收缩，分布相对更加集中。图 12 给出了二氧化硅和氧化钠的核密度函数。

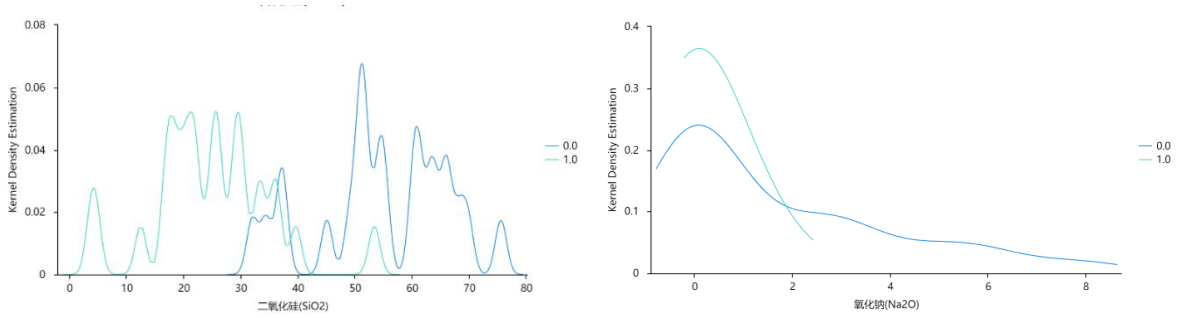

图 12 二氧化硅和氧化钠的核密度函数图

③平稳型

通过对化学成分含量的核密度函数分析，可将 K_2O、MgO、Fe_2O_3、SrO 划分为平稳型。它们风化前后的核密度函数基本一致，同时它们的平均含量变化很小，几乎可以忽略。图 13 给出了氧化铁和氧化钾的核密度函数。

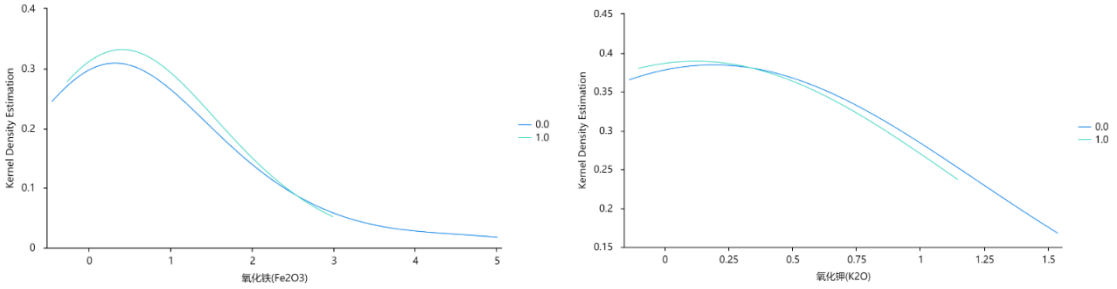

图 13 氧化铁和氧化钾的核密度函数图

5.2.3 基于风化前后概率密度分布的预测模型

玻璃文物样品在经过长时间的埋藏后，会受到环境因素影响而风化，但由于玻璃样品的化学成分不同，受风化的影响也会有所不同，从而导致不同玻璃类型的样品的化学成分含量产生差异。因此在对玻璃样品预测其风化前的化学成分含量时，需根据玻璃类型对样品划分为高钾玻璃和铅钡玻璃。

B_{ij}^{k} 表示玻璃类型为 k，文物编号为 j，化学物质为 i 的含量，当 k=0 时，玻璃类型为铅钡；k=1 时，玻璃类型为高钾。

U_j 表示编号为 j 的文物采样点的风化属性，U_j=0 表示风化前，U_j=1 表示风化后。

$E_{前i}^{k}$ 和 $E_{后i}^{k}$ 分别表示风化前后玻璃类型为 k、化学物质为 i 的均值。

$S_{前i}^{k}$ 和 $S_{后i}^{k}$ 分别表示风化前后玻璃类型为 k、化学物质为 i 的标准差。

$J_{前i}^{k}$ 和 $J_{后i}^{k}$ 分别表示风化前后玻璃类型为 k、化学物质为 i 的含量。

风化前后化学物质为 i 的含量均值可表示为：

$$E_{前i}^{k}=\frac{\sum_{j=1}^{n}\left|1-U_j\right|\cdot B_{ij}^{k}}{\sum_{j=1}^{n}\left|1-U_j\right|},\quad E_{后i}^{k}=\frac{\sum_{j=1}^{n}U_j\cdot B_{ij}^{k}}{\sum_{j=1}^{n}U_j} \tag{3}$$

$$S_{前i}^{k}=\sqrt{\frac{\sum_{j=1}^{n}(B_{ij}^{k}-E_{前i}^{k})\cdot\left|1-U_{j}\right|}{\sum_{j=1}^{n}\left|1-U_{j}\right|}},\quad S_{后i}^{k}=\sqrt{\frac{\sum_{j=1}^{n}(B_{ij}^{k}-E_{后i}^{k})\cdot U_{j}}{\sum_{j=1}^{n}U_{j}}} \tag{4}$$

在同种玻璃类型的样品中，由于制作工艺和化学组成的相似性，可以认为同种化学成分的变化趋势是一致的，即在风化前后同一种化学成分的核密度函数中，某一样品对应的成分含量的分位数是近似相同的，因此可以得到化学成分含量在风化前后的关系为：

$$\frac{J_{前i}^{k}-E_{前i}^{k}}{S_{前i}^{k}}=\frac{J_{后i}^{k}-E_{后i}^{k}}{S_{后i}^{k}} \tag{5}$$

5.2.4 预测模型求解

通过对预测模型的求解，得到了风化后样品在风化前的化学成分含量预测，具体数据见附件。图 14、15 给出了预测后高钾玻璃和铅钡玻璃的各化学成分平均含量的饼状图。由图可以清晰的看到，高钾玻璃中二氧化硅的含量最高，占比约 69%，氧化钾含量次之，占比 10%左右；铅钡玻璃中二氧化硅的含量最高，占比约为 56%，氧化铅含量次之，占比 23%，具体的化学成分含量预测见附件。预测出来的未风化的各化学成分的含量与文献中查阅的相关数据吻合，说明了预测模型的合理性。

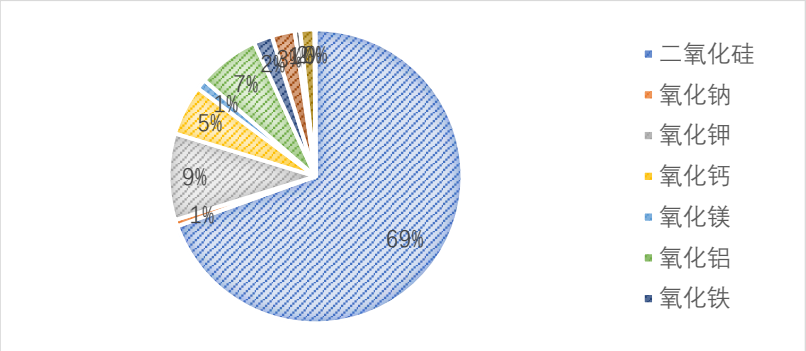

图 14 高钾玻璃各成分的预测值

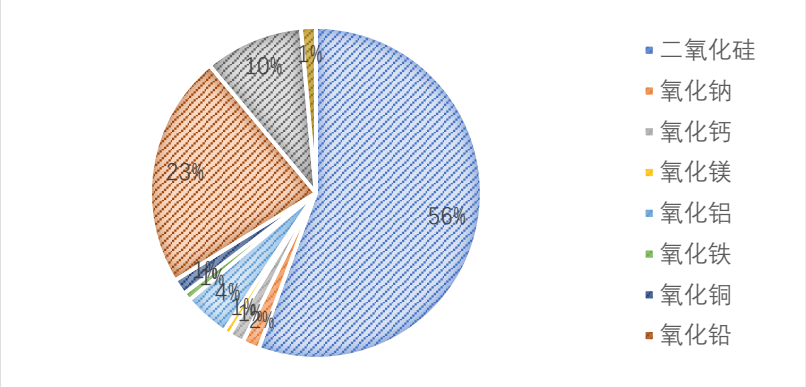

图 15 铅钡玻璃各成分的预测值

此外，通过将各化学成分的预测值与未风化的平均含量进行比对，可以得到差值的分布折线图，如图 16、17 所示。从图中可以看到，对于高钾玻璃来说，除二氧化硅外，其余成分的波动范围均在 5%以内，而由于二氧化硅在高钾玻璃中的含量占比最大，因此波动范围在 10%以内，符合实际情况，故高钾玻璃的各化学成分含量预测正确合理；对于铅钡玻璃而言，除二氧化硅、氧化铅和氧化钡之外，其余的差值波动范围均在 5%以内，而由于二氧化硅、氧化铅和氧化钡是铅钡玻璃中的三种主要化学成分，含量占比较大，且仅有个别样本点的预测值差值较大，因此可以认为铅钡玻璃的各化学成分含量预测合理。

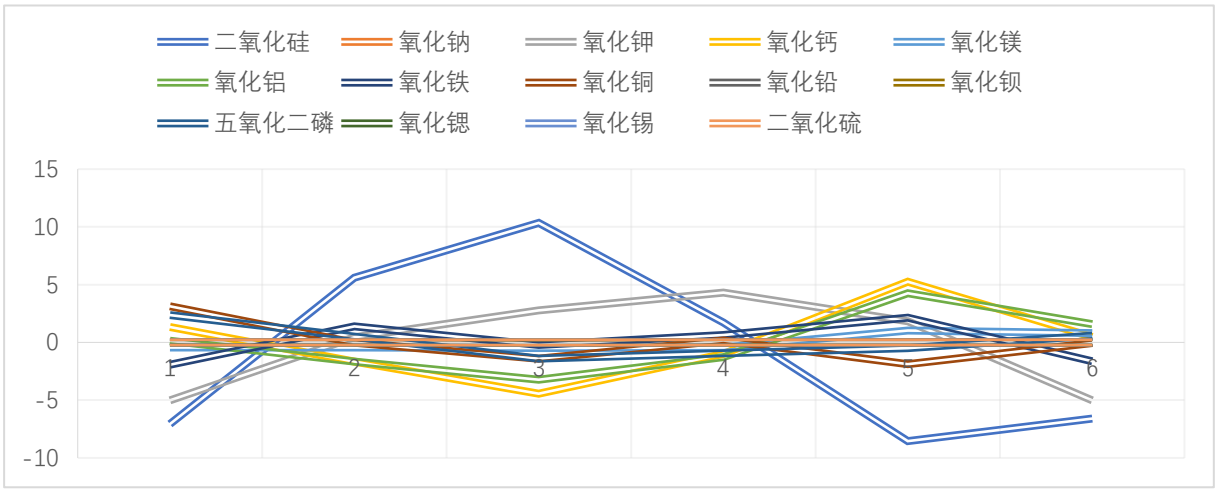

图 16 高钾玻璃差值波动图

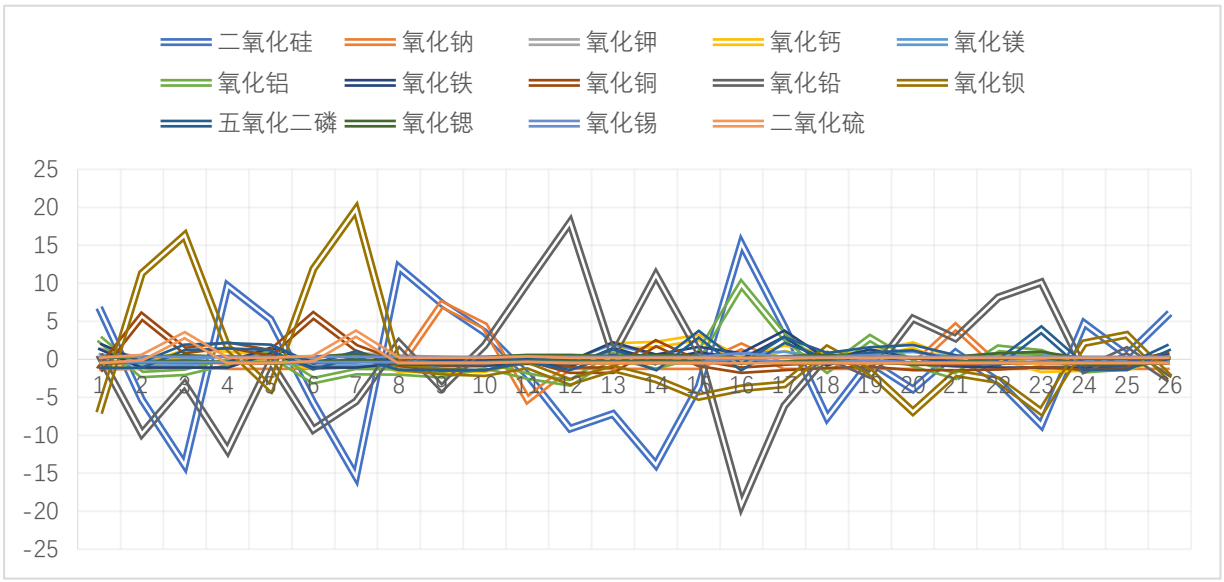

图 17 铅钡玻璃差值波动图

5.3 问题二的建模与求解

5.3.1 玻璃类型的分类规律探索

本题需要利用附件信息分析高钾玻璃和铅钡玻璃的分类规律。考虑到不同类型的玻璃所含的化学成分不同，并且某些化学成分的含量有着显著差异，因此可以利用各化学成分对玻璃类型的影响程度来分析玻璃类型的分类规律。此外，由于风化前后各

化学成分的含量有所差异，对玻璃类型的分类可能产生影响，因此需要分别讨论风化前后的玻璃类型分类规律。

（1）风化前的玻璃分类规律

相关分析需要考虑相关系数和置信度，它们共同反映两个变量之间的相关关系。置信度反映的是结果是否具有显著性，当 p 值大于 0.05 时，说明没有显著相关关系；当 p 值小于 0.05 时，呈现 0.05 水平的显著性；当 p 值小于 0.01 时，呈现 0.01 水平的显著性。

根据附件表单 2 中的数据，可以得到未风化的十四种化学成分与玻璃类型之间的置信度，如表 7 所示。

化学成分	二氧化硅	氧化钠	氧化钾	氧化钙	氧化镁	氧化铝	氧化铁
置信度 p	0.003	0.243	0	0.001	0.052	0.006	0.009
化学成分	氧化铜	氧化铅	氧化钡	五氧化二磷	氧化锶	氧化锡	二氧化硫
置信度 p	0.037	0	0	0.094	0.02	0.758	0.090

表 7 未风化的化学成分与类型的置信度

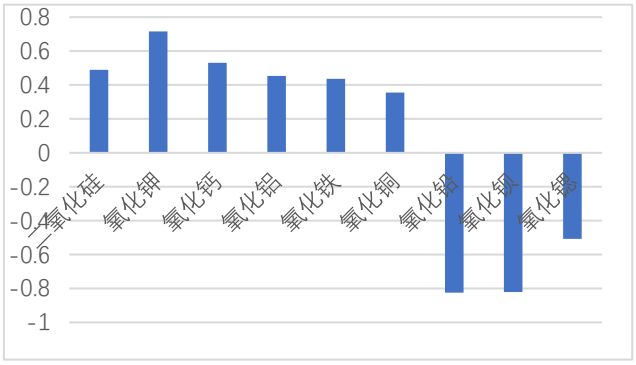

图 18 未风化的化学成分与类型的相关关系图

由 p 值可以确定与玻璃类型相关的九种化学成分。其中 SiO_2、K_2O、CaO、Fe_2O_3、CuO、Al_2O_3 含量越高，样品是高钾玻璃的概率越高；PbO、SrO、BaO 含量越高，样品是铅钡玻璃的概率越高。

由相关系数图知，K_2O 与玻璃类型呈最大正相关关系，PbO 和 BaO 与玻璃类型呈最大负相关关系，即风化前 K_2O 含量高的倾向于为高钾玻璃，PbO 和 BaO 含量高的倾向于为铅钡玻璃。

（2）风化后的玻璃分类规律

由于风化前后化学成分的含量不同，化学物质与玻璃类型的相关系数、置信度以及相关程度会有所不同，因此需要重新确定与玻璃类型相关的影响因子。利用附件表单 2 中风化后的化学物质含量数据计算相关系数和置信度，如表 8 所示。

化学成分	二氧化硅	氧化钠	氧化钾	氧化钙	氧化镁	氧化铝	氧化铁
置信度 p	0	0.321	0.032	0.006	0.128	0.355	0.848
化学成分	氧化铜	氧化铅	氧化钡	五氧化二磷	氧化锶	氧化锡	二氧化硫
置信度 p	0.603	0	0	0.003	0	0.499	0.321

表 8 风化后的化学成分与类型的置信度

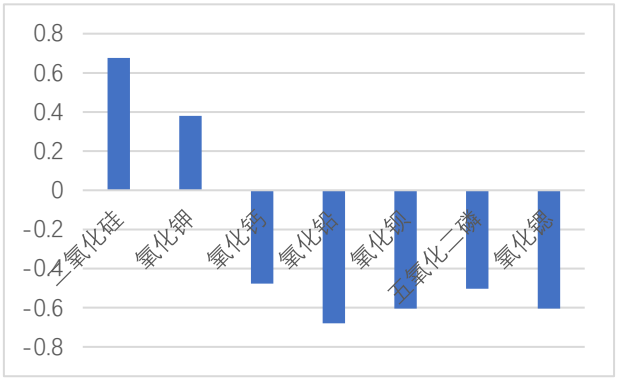

图 19 风化后化学成分与类型的相关关系图

由 p 值可以确定与玻璃类型相关的七种化学成分。其中 SiO_2、K_2O 的含量越高，样品是高钾玻璃的概率越高；CaO、PbO、SrO、BaO 和 P_2O_5 的含量越高，样品是铅钡玻璃的概率越高。

由相关系数图知，SiO_2 与玻璃类型呈最大正相关关系，PbO、BaO 和 SrO 与玻璃类型呈最大负相关关系，即风化后 SiO_2 含量高的倾向于为高钾玻璃，PbO、BaO 和 SrO 含量高的倾向于为铅钡玻璃。

5.3.2 基于聚类的亚类划分模型

同一种类型的玻璃由于制作过程的差异，会造成各种化学成分含量的不同，其中某些化学成分的含量是存在显著差异的，为探究各化学物质对玻璃的分类影响，采用聚类方法对高钾玻璃和铅钡玻璃进行亚类划分。由于样本数量较少且划分类别数未知，故采用 *K-means* 聚类算法对玻璃进行亚类划分。*K-means* 聚类算法可以根据各化学物质含量差异，将同一玻璃样品分为合适的 k 个亚类。

给定的一个包含 n 个 d 维数据点的数据集 $Z=\{z_1,z_2,...,z_j,...,z_n\}$，其中 $z_j\in R^d$，要生成的数据子集的数目为 K，*K-means* 聚类算法将数据对象划分为 K 个簇 $C=\{c_i,i=1,2,...,K\}$，每个簇中均有一个聚类中心 μ_i。选取欧式距离作为相似性和距离判断的准则，计算簇中各点到聚类中心 μ_i 的距离平方和为：

$$J(c_i)=\sum_{z_j\in c_i}\left\|z_j-\mu_i\right\|^2 \tag{6}$$

聚类的目标是使各类总的距离平方 $J(C)=\sum_{i=1}^{K}J(c_i)$ 最小，则 $J(C)$ 可表示为：

$$J(C)=\sum_{i=1}^{K}\sum_{z_j\in c_i}\left\|z_j-\mu_i\right\|^2=\sum_{i=1}^{K}\sum_{j=1}^{n}d_{ij}\left\|z_j-\mu_i\right\|^2 \tag{7}$$

其中，$d_{ij}=\begin{cases}1, z_j\in c_i\\ 0, z_j\notin c_i\end{cases}$。

显然，根据最小二乘法和拉格朗日原理，聚类中心 μ_i 应该取为类别 c_i 中各个数据点的平均值 。

5.3.3 模型求解

针对亚类划分模型的求解步骤如下图所示。

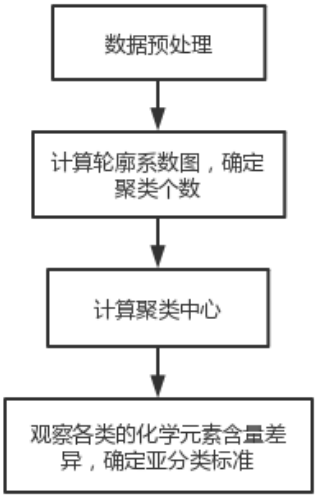

图 20 模型求解步骤图

（1）数据处理

附件表单 2 中的数据给出了样本点风化前或风化后的数据，由于风化前和风化后的玻璃类型分类规律有所不同，需要对风化前和风化后的玻璃分别进行亚类划分，因此需要利用问题一中的预测模型分别求出风化后样品在风化前的化学物质含量以及未风化的样品在风化后的化学物质含量，之后对所有风化前的样本和所有风化后的样本分别进行聚类。

在进行 *K-means* 聚类时需要计算样本点之间的欧氏距离，由于样本点的数据之间差异较大，需要对数据进行标准化处理。

（2）确定聚类数目 K

在 *K-means* 聚类中，最核心的参数是聚类数目 K，可引入轮廓系数进行确定。*K-means* 的聚类目标是希望簇中内部差异小，簇外差异大，而轮廓系数结合了聚类的凝聚度和分离度，是用于描述类内外差异的关键指标。轮廓系数 S 的计算公式为：

$$S(j)=\frac{b(j)-a(j)}{\max\{a(j),b(j)\}} \tag{8}$$

*a(j)*表示了样本点 j 到同簇内其它点不相似程度的平均值，体现了簇内的凝聚度，计算公式如下：

$$a(j)=\frac{1}{m-1}\sum_{j\neq i}^{m} distance(i,j) \tag{9}$$

其中，i 代表与样本点 j 在同一个簇中的其它样本点，m 表示簇中所有的样本点数，*distance(i,j)*表示样本点 i 与 j 之间的欧式距离。

*b(j)*表示了样本点 j 到其它簇的平均不相似程度的最小值，体现了簇外的分离度。通过遍历求得样本点 j 到其它簇的距离，可以得到多个 b 值，取其中的最小值作为 *b(j)*。

根据上述对轮廓系数公式的分析，可以发现当 *a(j)<b(j)*时，即簇内的距离小于簇间距离，则聚类结果更加紧凑，*S(j)*会趋近于 1；相反，当 *a(j)>b(j)*时，即簇内的距离大于簇间距离，则聚类结果比较松散，*S(j)*会趋近于-1。因此，轮廓系数的取值范围为[-1,1]，轮廓系数越大则聚类效果越好。

本问中高钾玻璃和铅钡玻璃的数据集中分别包含了 18 个和 49 个样本，每个样本中包含 14 个数据点。考虑到数据点较少，类别数不应过大，因此利用 *K-means* 聚类在区间[2,6]中遍历 K 值，计算对应的轮廓系数，并得到对应的轮廓系数折线图。从图中可以看到，风化前的高钾玻璃和铅钡玻璃的 K 值均为 2，风化后的高钾玻璃和铅钡玻璃的 K 值分别为 2 和 3。

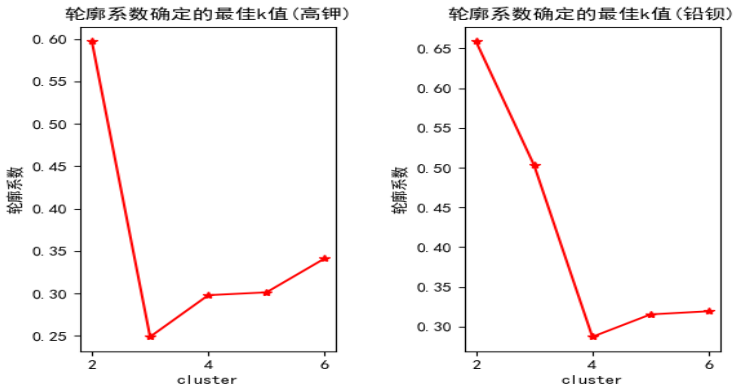

图 21 风化前两种玻璃的轮廓系数图

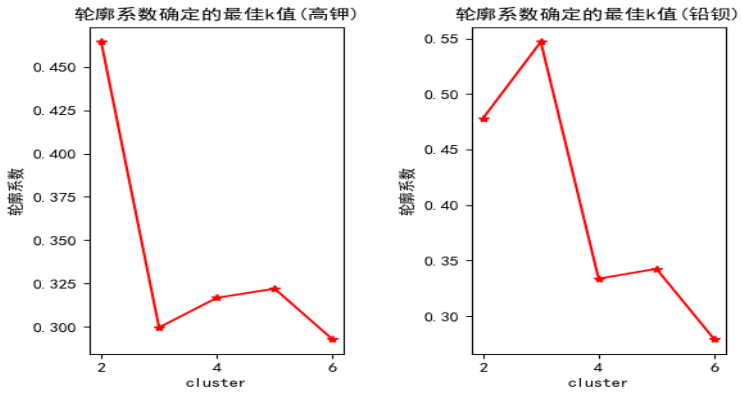

图 22 风化后两种玻璃的轮廓系数图

但在实际聚类过程中，风化后的铅钡玻璃分三类聚类的样本数为 30：18：1，样本并不均匀分布于各个聚类，可解释性较差，因此我们仍然采用 2 作为 K 值。

（3）计算聚类中心

通过 *K-means* 聚类算法，可以计算得到聚类中心、具体的划分情况如下表所示。划分的文物编号见附件。

玻璃类型及状态		聚类中心	样本数
风化前	高钾玻璃	[0.044，0.191，0.182，0.125，-0.185，-0.187，-0.229，-0.142，-0.199，-0.433，-0.247，-0.301，0.102，0.194]	14
		[-0.154，-0.669，-0.637，-0.437，0.648，0.653，0.802，0.498，0.696，1.517，0.865，1.052，-0.358，-0.679]	4
	铅钡玻璃	[0.333，-0.025，0.058，0.117，0.201，0.181，0.199，-0.394，0.002，-0.374，-0.025，-0.34，0.123，-0.268]	35
		[0.063，-0.145，-0.292，-0.502，-0.453，-0.497，0.985，-0.006，0.936，0.063，0.849，-0.309，0.671]	14
风化后	高钾玻璃	[-0.659，0，1.389，1.659，1.361，-0.428，1.268]	3
		[0.132，0，-0.278，-0.332，-0.272，0.086，-0.254]	15

	铅钡玻璃	[-0.097，0.186，-0.202，-0.512，-0.307，-0.289，-0.406，-0.147，-0.354，-0.499，0.239，0.039，0.335，0.117]	32
		[0.183，-0.351，0.38，0.963，0.577，0.544，0.764，-0.277，0.666，0.94，-0.45，-0.073，-0.63，-0.219]	17

表 9 各种玻璃的聚类中心

注：聚类中心的值为标准化后的数据。

特别说明的是，由于风化后的高钾玻璃有较多的化学成分含量为 0，会对聚类分析的结果造成影响，因此在对风化后的高钾玻璃进行亚类划分时，不考虑含量为 0 的化学成分，即氧化钠、氧化钾、氧化铅、氧化钡、氧化锶、氧化锡和二氧化硫。故风化后的高钾玻璃亚类划分的聚类中心分别对应 SiO_2、CaO、MgO、Al_2O_3、Fe_2O_3、CuO、P_2O_5。其余的聚类中心分别对应 SiO_2、Na_2O、K_2O、CaO、MgO、Al_2O_3、Fe_2O_3、CuO、PbO、BaO、P_2O_5、SrO、SnO_2、SO_2。

（4）确定亚类划分标准

确定每一类中具体的文物样品后，可以得到每种化学成分聚类项重要性比对图，聚类项重要性表示的是化学物质对玻璃亚类划分的贡献度，聚类项重要性值越高，则化学物质对划分结果的贡献越多，以此来确定划分标准。

●风化前的高钾玻璃

通过对聚类项重要性比对图的观察，可以看到氧化钡对风化前的高钾玻璃亚类划分贡献更多，而其余化学成分对划分的贡献度远小于氧化钡，因此可以将风化前的高钾玻璃划分为高钡和低钡两个亚类。

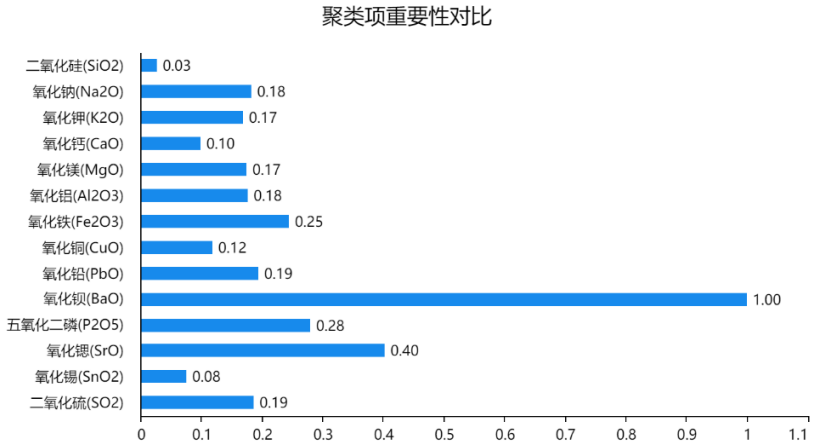

图 23 风化前高钾玻璃的聚类项重要性比对图

●风化前的铅钡玻璃

通过观察聚类项重要性比对图，可以发现氧化铜和氧化钡对风化前的铅钡玻璃亚类划分贡献更多，接近于 1，同时考虑到亚类划分类别不应过多，故将氧化铜和氧化钡作为划分风化前的铅钡玻璃的标准。

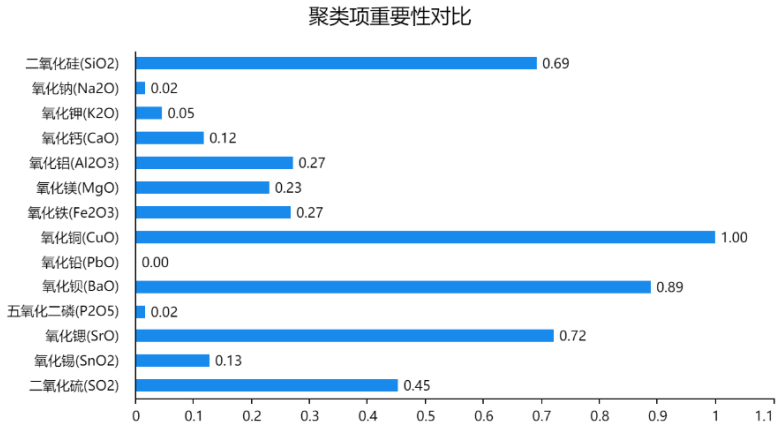

图 24 风化前铅钡玻璃的聚类项重要性比对图

●风化后的高钾玻璃

从聚类项重要性比对图中可以看到氧化铝对风化后的高钾玻璃的划分贡献度接近于 1，故可以将氧化铝作为划分风化后的高钾玻璃的标准。

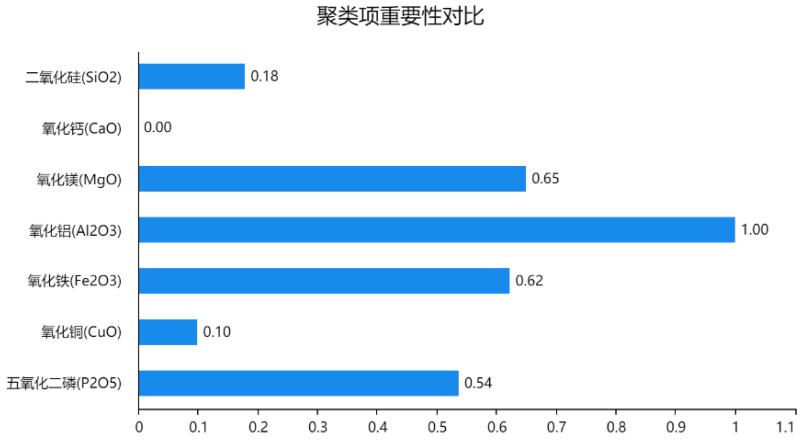

图 25 风化后高钾玻璃的聚类项重要性比对图

●风化后的铅钡玻璃

聚类项重要性比对图中氧化钙和氧化铁的贡献度接近于 1，其余的化学成分对划分的贡献度较低，因此将氧化钙和氧化铁作为划分风化后铅钡玻璃的标准。

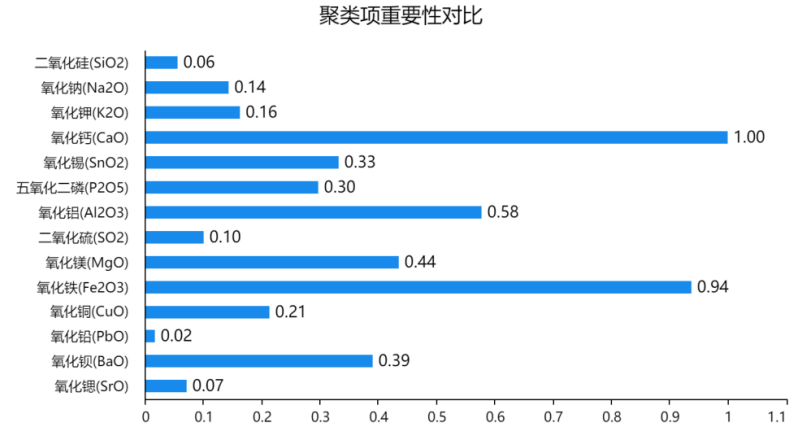

图 26 风化后铅钡玻璃的聚类项重要性比对图

综上所述，风化前后铅钡与高钾玻璃亚类划分的标准总结为：

	高钾玻璃	铅钡玻璃
风化前	氧化钡	氧化铜和氧化钡
风化后	氧化铝	氧化钙及氧化铁

表 10 划分标准总结

（5）结果分析

●合理性分析

通过对聚类分项重要性比对图的分析，确定了风化前后高钾玻璃和铅钡玻璃的划分标准。为使结果更具有说服力，可利用化学成分的样品散点图来进行检验。

风化前高钾玻璃的划分标准为氧化钡的含量，下图给出了关于氧化钡的样品散点图，样本点共有 18 个，由于部分样品的氧化钡含量为 0，故在图中无法显示出来。

通过观察样品散点图可以清晰发现，第一类样品属于低钡玻璃，第二类样品属于高钡的玻璃。因此可以将氧化钡作为划分风化前高钾玻璃的标准。

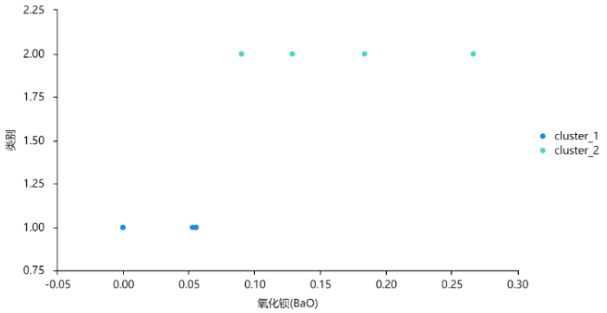

图 27 氧化钡和类别的散点图

对于风化前铅钡玻璃的亚类划分，其划分标准为氧化钡和氧化铜的含量。根据样品类别的散点图可以看到，虽然部分样品的化学成分含量重叠，但利用氧化钡和氧化铜可以将大部分铅钡玻璃划分为两类，故风化前的铅钡玻璃可以划分为低钡低铜玻璃和高钡高铜玻璃。

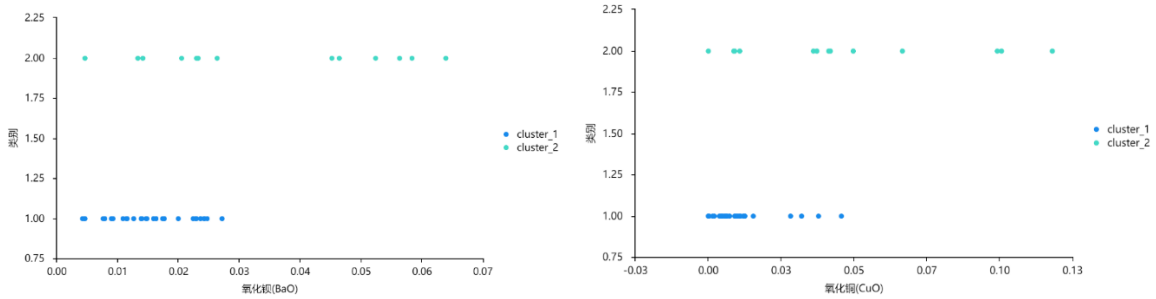

图 28 氧化钡、氧化铜和类别的散点图

由图可以看到，风化后的高钾玻璃根据氧化铝的含量可以划分成两类，第一类为高铝玻璃，第二类为低铝玻璃，而且不存在化学成分含量重叠的样品，说明氧化铝可以作为划分风化后高钾玻璃的标准。

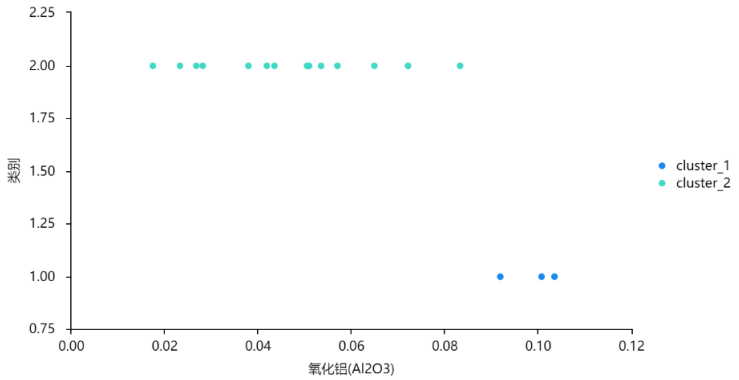

图 29 氧化铝和类别的散点图

风化后的铅钡玻璃利用氧化钙和氧化铁进行划分，由图可知，虽然部分样品的化学成分含量重叠，但大部分的样品可以基于氧化钙和氧化铁划分成两类，分别为高铁高钙玻璃和低铁低钙玻璃，准确性较高。

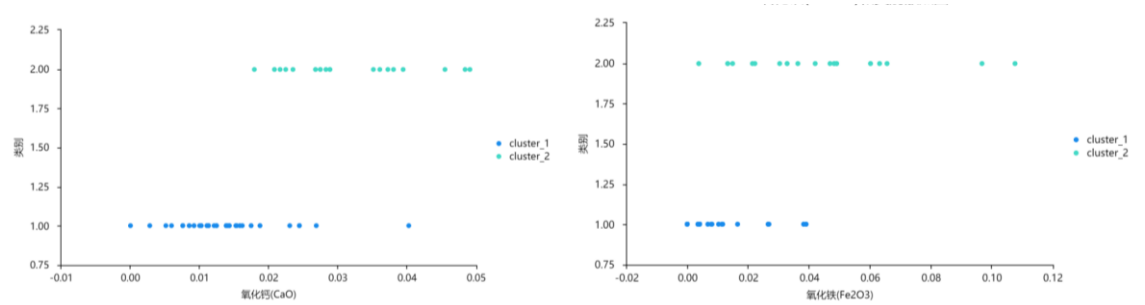

图 30 氧化钙、氧化铁和类别的散点图

●灵敏度分析

灵敏度分析是用来研究和分析模型的输出变化对参数或周围条件变化的敏感程度。在对高钾玻璃和铅钡玻璃进行亚类划分时，各个化学成分的含量在其中起到了重要作用。对此，可研究化学成分的存在对聚类效果的影响，即分析轮廓系数对化学成分的敏感性。

每一种化学成分均参与了轮廓系数的求解，为研究各化学成分对轮廓系数的影响，可通过控制变量法，求解当一种化学成分不存在时对应的轮廓系数，并得到对应的折线图，如下图所示。

对于风化前的高钾玻璃来说，当所有化学成分均存在时，轮廓系数约为 0.6。当二氧化硅不存在时，对应的轮廓系数最低，接近 0.3，故轮廓系数对二氧化硅最为敏感；当氧化钾不存在时，其轮廓系数最高，约为 0.5，故氧化钾对轮廓系数不会产生较大影响；当其余化学成分不存在时，由图可知，对应的轮廓系数在 0.45 附近波动，故轮廓系数对它们较为敏感。

对于风化前的铅钡玻璃而言，当所有化学成分均存在时，轮廓系数为 0.65.当各化学成分不存在时，由图可知，它们对应的轮廓系数均在区间[0.35,0.4]内波动，与原始的轮廓系数值偏差较大，故各化学成分均会对聚类效果产生一定程度的影响。

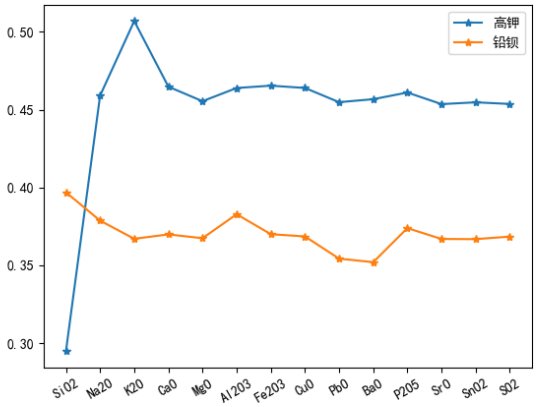

图 31 风化前元素与轮廓系数的灵敏度分析图

对于风化后的高钾玻璃而言，当所有化学物质均存在时，轮廓系数为 0.48。观察下图可知，当氧化铝或氧化铜不存在时，对应的轮廓系数均接近 0.45，与原始的轮廓系数的差值较小，故可以认为氧化铝和氧化铜对聚类效果几乎不会产生影响。而当氧化硅不存在时，对应的轮廓系数约为 0.1，偏离原始的轮廓系数较大，因此轮廓系数对二氧化硅最为敏感。对于其它的化学物质，当它们不存在时对应的轮廓系数均约为 0.42，与原始的轮廓系数差值较小，因此它们会对聚类效果产生轻微影响。

对于风化后的铅钡玻璃来说，当所有化学物质均存在时，轮廓系数为 0.55。通过观察下图可知，氧化钡和氧化钙对应的轮廓系数与原始的轮廓系数偏离最多，二氧化硅次之，故轮廓系数对氧化钙和氧化钡最为敏感，对二氧化硅比较敏感。而其它化学成分所对应的轮廓系数与原始值相比偏差在 0.1 左右，因此它们对聚类效果有轻微影响。

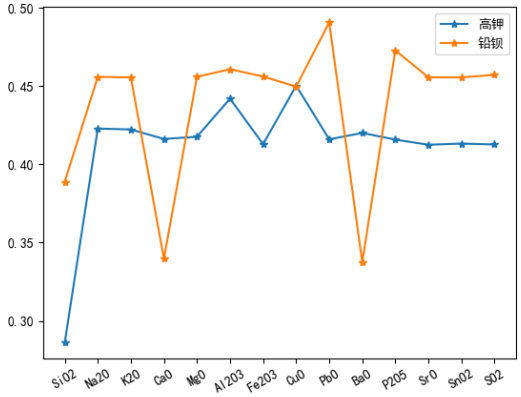

图 32 风化后元素与轮廓系数的灵敏度分析图

5.4 问题三的建模与求解

5.4.1 基于二层决策树和加权软投票法的分类模型

基于问题二中对高钾玻璃和铅钡玻璃的分类规律的研究，要对玻璃文物的类别进行鉴别，可建立基于二层决策树和加权软投票法的分类模型，构建二层决策树作为分类器，并采用加权软投票法量化各化学成分对玻璃类型的影响程度，以此来鉴别玻璃文物的类别。

（1）确定影响因子

问题二中研究了风化前后与玻璃类型相关的化学成分及其相关系数。对于风化前的玻璃样品，确定了九种与玻璃类型相关的化学成分，其中 SiO_2、K_2O、CaO、Fe_2O_3、CuO、Al_2O_3与玻璃类型呈正相关；PbO、SrO、BaO与玻璃类型呈负相关；对于风化后的玻璃样品，确定了七种与玻璃类型相关的化学成分，SiO_2、K_2O 为正相关因素，CaO、PbO、SrO、BaO 和 P_2O_5 为负相关因素，由此可以将与玻璃类型相关的化学成分作为影响因子。 （其中 1 表示高钾玻璃，0 表示铅钡玻璃）

考虑各化学成分在高钾玻璃和铅钡玻璃中均存在，因此需给化学成分含量确定分界值，利用分界值来区分不同类型的玻璃。设共有 n 个影响因子，其分界值为x_i，其中 $i=1,2,\cdots,n$。对于风化前的样品 $n=9$，对于风化后的样品 $n=7$。

（2）构建决策树

对于不同的影响因子，可以将其作为决策树，通过判断取值与分界值的大小，对样品的玻璃类型进行划分。对于与玻璃类型呈正相关的影响因子，当样品的取值高于对应分界值时，则样品大概率为高钾玻璃；对于与玻璃类型呈负相关的影响因子，当样品的取值低于对应分界值时，则样品大概率为铅钡玻璃。

（3）建立基于加权软投票的模型

Step1：确定权重

以各个影响因子作为决策树——分类器，由于分类器的判断标准不同、性能不同，采用投票法时需要对不同的分类器进行加权。因此当影响因子与玻璃类型相关性强时，需赋予较高的权重；当影响因子与玻璃类型相关性弱时，需赋予较高的权重。

在本问中，由于玻璃类型与 n 个影响因子之间的相关程度不同，相关系数越偏离 0，则相关程度越高，故可用相关系数来计算每个影响因子的权重。设第 i 个影响因子的相关系数为 r_i，权重为 w_i，则权重可表示为：

$$w_i=\frac{|r_i|}{\sum_{i=1}^{n}|r_i|} \tag{10}$$

Step2：确定类别概率

类别概率可以认为是后验概率的估计，其范围区间为[0,1]。在本问中可以用影响因子的取值与分界值之间的差值来表征，当差值越大时，则类别概率越大，即越容易鉴别玻璃的类型。

设标准化后的样品的 n 个影响因子取值为 $A=(A_1,A_2,...,A_i,...,A_n)$，由于玻璃样品中各影响因子取值差异较大，故数据需要标准化。

设类别概率为Ω_i，当影响因子的取值高于分界值时，类别概率表示为：

$$\Omega_i=\frac{|A_i-x_i|}{|A_i^{\max}-x_i|} \tag{11}$$

其中，$A_i^{\max}$ 表示第 i 个影响因子的最大取值。

当影响因子的取值低于分界值时，类别概率表示为：

$$\Omega_i=\frac{|A_i-x_i|}{|A_i^{\min}-x_i|} \tag{12}$$

故类别概率可以表示为：

$$\Omega_i=\begin{cases}\dfrac{|A_i-x_i|}{|A_i^{\max}-x_i|}, & A_i>x_i\\ \dfrac{|A_i-x_i|}{|A_i^{\min}-x_i|}, & A_i<x_i\end{cases} \tag{13}$$

Step3：确定判别函数

确定了分类器的权重和类别概率后，由此可写出判别函数。

设判别函数为 y_i，则表达式为：$y_i=\Omega_i*w_i$。

设对于高钾玻璃的得分为 z_1，对于铅钡玻璃的得分为 z_2，两者的得分初始化为 0. 当第 i 个影响因子与高钾玻璃呈正相关时，将 y_i 加到高钾玻璃的得分中；当第 i 个影响因子与铅钡玻璃呈正相关时，将 y_i 加到铅钡玻璃的得分。将所有的影响因子的判别函数根据它们的正负相关性分别加到对应的玻璃类型的得分中，最终比较两者的得分，得分高的玻璃类型即为对应样品的预测类型。

5.4.2 风化前后的分类模型求解

（1）数据分类

由于附件表单 3 中给出的玻璃文物的风化情况不同，故需要对玻璃文物进行分类，对于未风化的玻璃文物，其影响因子为 9 个；对于已风化的玻璃文物，其影响因子为 7 个，对未风化文物和风化文物需采用不同的影响因子对玻璃类型进行判断。

（2）参数求解

风化前的九种化学成分含量的分界值和权重如下表所示。

	SiO_2	K_2O	CaO	Al_2O_3	Fe_2O_3	CuO	PbO	BaO	SrO
分界值	87.05	14.52	10.59	14.34	6.04	8.46	40.10	28.73	0.91
权重	0.17	0.17	0.08	0.05	0.07	0.01	0.18	0.13	0.13

表 11 风化前化学成分的分界值和权重

风化后的七种化学成分含量的分界值和权重如下表所示。

	SiO_2	K_2O	CaO	PbO	BaO	P_2O_5	SrO
临界值	6.217	0.018	0.053	0.011	0.033	0.027	0.002
权重	0.18	0.19	0.09	0.20	0.14	0.07	0.14

表 12 风化后化学成分的分界值和权重

基于问题二的分类规律模型以及亚类划分模型，可以将附件表单 3 中的玻璃文物进行划分，结果如下表所示。

文物编号	玻璃类型
A1	高钾
A2	铅钡
A3	铅钡

A4	铅钡
A5	高钾
A6	高钾
A7	高钾
A8	铅钡

表 13 文物类型预测

（3）灵敏度分析

对于灵敏度的分析可从两方面进行探究，首先是判断分类讨论风化情况对分类结果的影响。在本问中，对于未风化和风化的玻璃文物采用了不同的判断标准进行鉴别。对此，我们将风化前后的两种判断标准分别对表单 3 中的所有文物进行种类预测，得到的结果与本问的结果进行比对，应用风化前的判断标准得到的结果正确率为 87.5%，应用风化后的判断标准得到的结果正确率为 100%。

由于使用风化后的判断标准得到的结果与本问中的结果一致，因此需进一步对风化后的判断标准进行灵敏度分析。通过改变文物中所含元素的含量，观测正确率的变化情况。将文物中的化学成分含量从 0%到 200%进行放缩，通过同时改变一个元素、改变两个元素、改变四个元素，观察正确率的变化，其中 0-7 编号分别对应二氧化硅、氧化钾、氧化钙、氧化铅、氧化钡、五氧化二磷和氧化锶。

图 33 给出了四种情况的正确率变化曲线，由图可知，本问所建立的分类标准保证了分析结果不受单个元素含量变化的影响，而对于多个元素含量同时变化的情况，建立的分类标准对于 3 号元素（氧化铅）的敏感性最强，其含量变化将极大的影响模型判断正确率

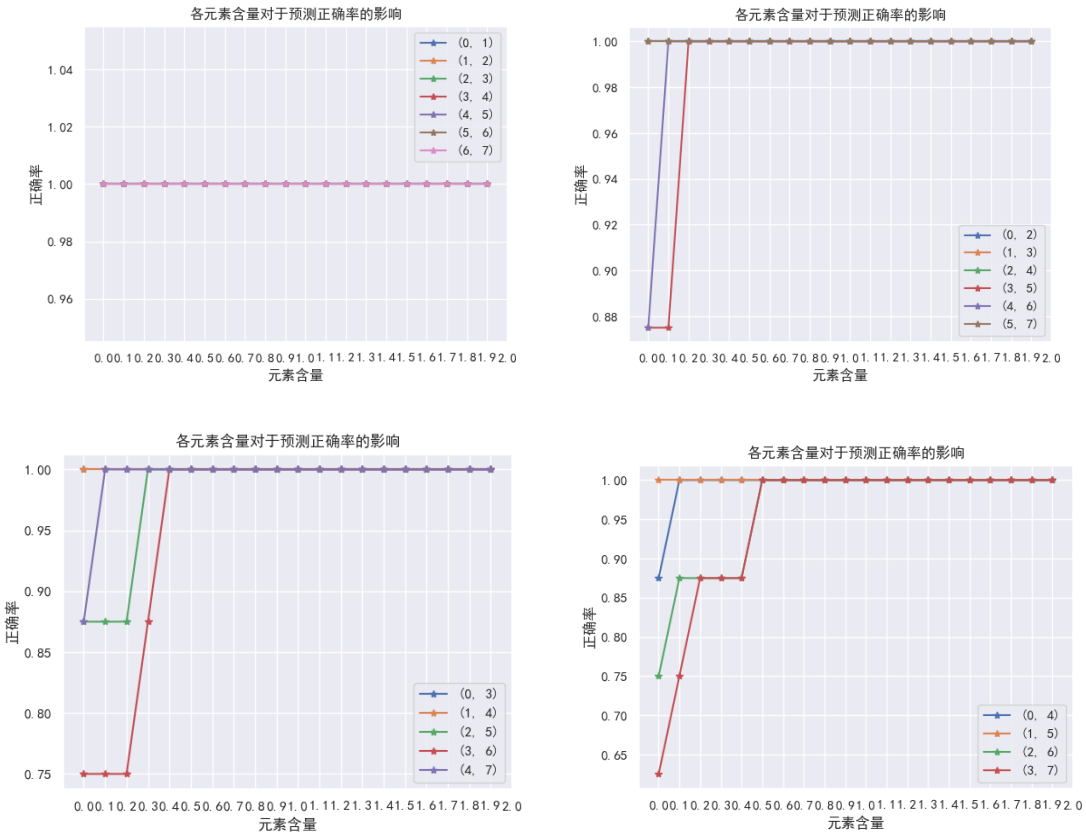

图 33 各化学成分对预测正确性的影响图

5.5 问题四的建模与求解

5.5.1 基于相关系数图与偏相关系数图的关联关系分析

相关系数是用以反映变量之间相关关系密切程度的统计指标，但在多元回归中，相关系数只是两变量局部的线性的单相关系数，而并非整体的性质。而偏相关系数在消除其他变量影响的条件下，计算某两变量之间的相关系数。因此，通过对相关系数图及偏相关系数图进行对比，可以得出两化学物质之间的相关关系及其受其他化学物质的影响。

本问将相关关系的强弱定义如下：

0.6<p	强线性相关
0.3<p<0.6	弱线性相关
0<p<0.3	无线性相关

表 14 相关关系定义

（1）高钾玻璃

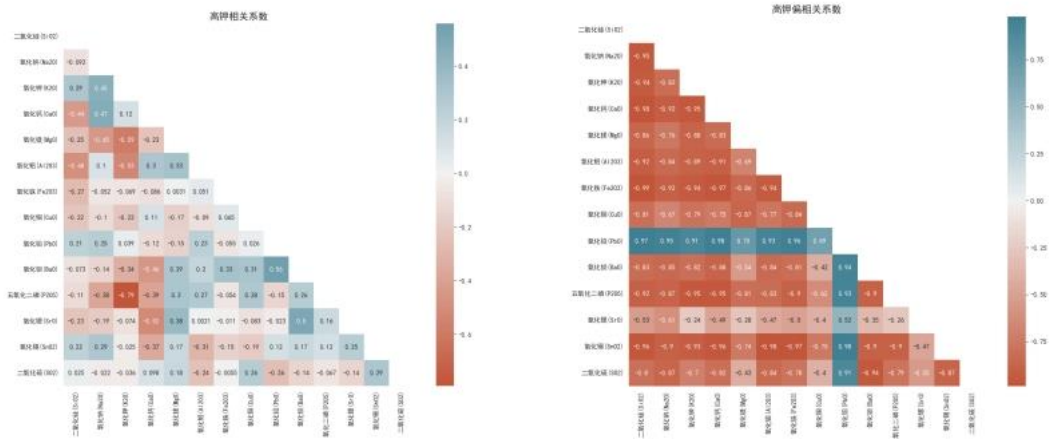

图 34 高钾玻璃各化学成分的相关系数图及偏相关系数图

高钾玻璃的相关系数图中，化学元素之间的局部线性关系正负相关比例相差不大，并且线性关系较弱；但偏自相关图中，除了氧化铅之外，所有化学物质之间呈负相关，且仅有氧化锶、氧化铜的负相关关系较弱，其余的负相关关系均非常强烈。对此，我们总结化学物质关系如下：

① 氧化铅与其余物质呈线性正相关；其余化学物质之间呈线性负相关。

② 氧化锶、氧化铜与其余物质呈现较弱的线性关系；其余化学物质之间呈现较强的线性关系。

③ 化学物质之间单相关的关系并不明显，但控制其余变量之后两两相关关系相较强烈，证明高钾玻璃的各化学物质含量之间有较强的共线性关系。

（2）铅钡玻璃

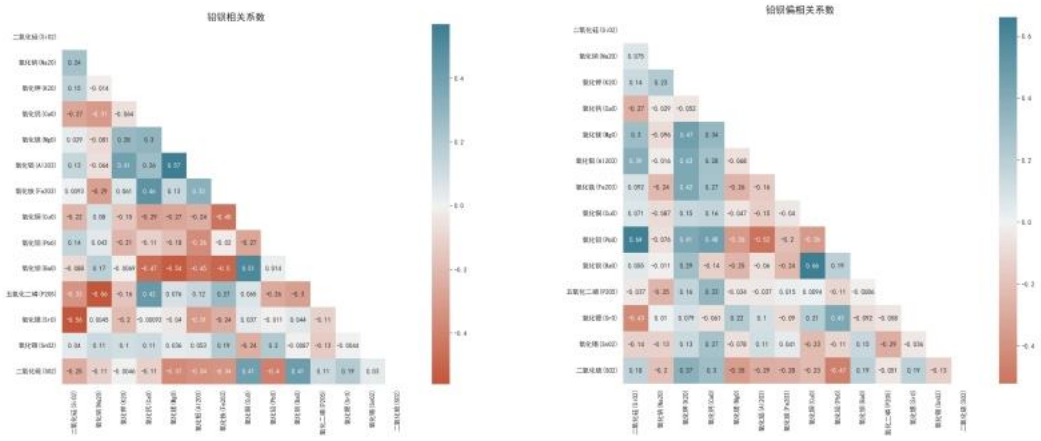

图 35 铅钡玻璃各化学成分的相关系数图及偏相关系数图

铅钡玻璃的相关系数图中，化学元素之间的局部线性关系正负相关比例相差不大，但线性相关关系总体较强。而偏自相关图中，仅有氧化钾与其余物质的正负相关关系改变较大，其余各化学物质的正负相关关系并未发生强烈改变，但相关关系相较较弱。对此，我们总结化学物质关系如下：

① 化学物质之间单相关的关系明显，但控制其余变量之后两两相关关系相较减弱，证明高钾玻璃的各化学物质含量之间共线性关系较弱。

② 二氧化硅、氧化钾对其余物质呈正线性关系较多，其余化学物质之间的正负关系较为复杂。

5.5.2 基于偏相关图分析不同种类玻璃化学物质关联关系的差异性

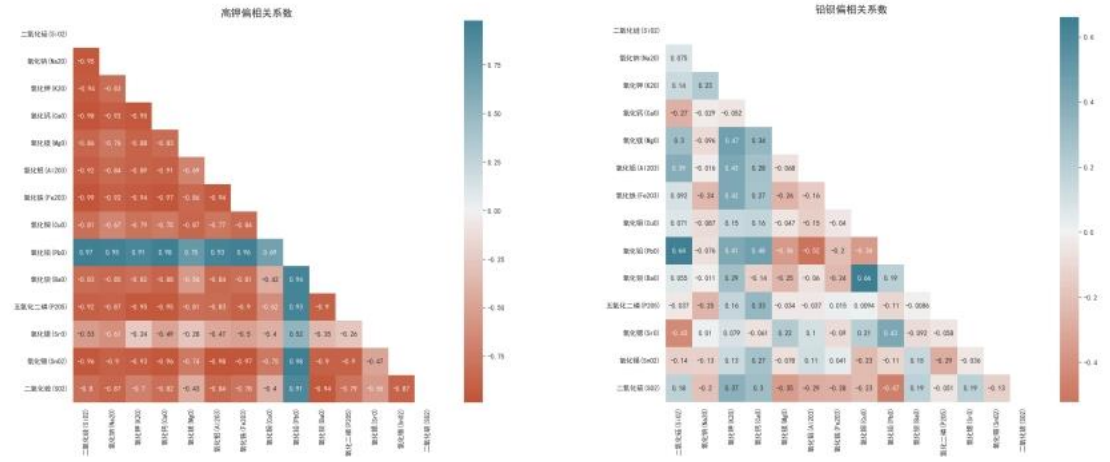

图 36 高钾玻璃和铅钡玻璃各化学成分的偏相关系数图

结合偏自相关图的正负关系与关系大小，可以总结总体差异为：高钾玻璃各化学成分之间以负相关关系为主，铅钡玻璃各化学成分之间的线性关系正负关系基本相等。

对于特定化学成分，有以下化学成分具有明显的差异性：

①氧化铅在高钾玻璃中与其余化学成分呈强烈的正线性相关，但在铅钡玻璃中仅与二氧化硅呈强烈的正线性相关，与其他元素呈弱线性相关。

②二氧化硅、氧化钾和二氧化钙在高钾玻璃中与其余化学成分呈强烈的负相关关系，但在铅钡玻璃中，普遍与其余元素呈弱负线性相关。

六．模型评价

模型优点：

（1）问题二和问题三的类型分类及亚类划分中，应用了问题一中的预测模型，将所有数据转为未风化和已风化的数据，分类讨论了未风化和已风化的分类标准，使模型可以广泛适用。

（2）问题三中采用加权投票算法与类别概率投票算法相结合的方式，

模型缺点：

（1）样本数据集较小，如风化后的高钾玻璃只有 6 组数据，难以对其总结一般性规律。

（2）样本数据集异常值较多，且部分样品的氧化钠、氧化铅、氧化钡、氧化锶、氧化锡的含量均为 0，难以对风化前后的数据进行预测与分析。

（3）由于颜色与化学物质成分之间存在因果关系，如文物颜色受氧化铁(Fe_2O_3)、氧化铜(CuO)、氧化铅(PbO)等化学物质的影响，因此颜色对风化与否、玻璃类型的判断会存在间接影响。

七．模型推广与改进

模型改进：

二层决策树虽然具有易解释、易给出类别概率等优点，但三层或多层决策树可能对数据具有更好的分类效果。

模型推广：

（1）本文对化学物质成分与类型、风化的相关关系分析可以推广到其他分类问题及预测模型中。

（2）本文对高钾、铅钡玻璃的亚类划分模型可以推广到其余亚类划分模型中。

八．参考文献

[1]王承遇,陶瑛.硅酸盐玻璃的风化[J].硅酸盐学报,2003(01):78-85.

[2]史美光,何欧里,周福征.一批中国汉墓出土钾玻璃的研究[J].硅酸盐学报,1986(03):307-313.

[3]王婕,李沫,马清林,张治国,章梅芳,王菊琳.一件战国时期八棱柱状铅钡玻璃器的风化研究[J].玻璃与搪瓷,2014,42(02):6-13.DOI:10.13588/j.cnki.g.e.1000-2871.2014.02.002.

[4]赵志强.湖南里耶麦茶战国墓地出土玻璃制品的检测与分析[J].湖南考古辑刊,2020(00):288-301.

九. 附录

解题主框代码

```
import os.path
import pprint

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import style
from pylab import mpl

mpl.rcParams['font.sans-serif'] = ['STZhongsong']  # 指定默认字体：解决plot不能显示中文问题
mpl.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问题
sns.set_style({"font.sans-serif": ['simhei', 'Droid Sans Fallback']})

"""
    数据读取
"""
data_path = r'data/题目原始数据/附件.xlsx'
# 表单1
data_sheet_1 = pd.read_excel(data_path, index_col=0, sheet_name='表单1')
# 表单2
data_sheet_2 = pd.read_excel(data_path, index_col=0, sheet_name='表单2')
# 表单3
data_sheet_3 = pd.read_excel(data_path, index_col=0, sheet_name='表单3')

"""
    数据处理
"""
# 表单1的索引列
sheet_1_index = data_sheet_1.index
# 表单2的索引列
sheet_2_index = data_sheet_2.index
# 表单3的索引列
sheet_3_index = data_sheet_3.index

```

```
# 表单1进行数据清洗,空缺项填0
sheet_1_columns = data_sheet_1.columns
for row_index in sheet_1_index:
    for col_index in sheet_1_columns:
        data = data_sheet_1.loc[row_index][col_index]
        if isinstance(data, str):
            continue
        data_sheet_1.loc[row_index][col_index] = 0

# 对表单2进行数据清洗,将空缺的数据补成0
sheet_2_columns = data_sheet_2.columns
for row_index in sheet_2_index:
    for col_index in sheet_2_columns:
        data = data_sheet_2.loc[row_index][col_index]
        if np.isnan(data):
            data_sheet_2.loc[row_index][col_index] = 0

# 表单3进行数据清洗
sheet_3_columns = data_sheet_3.columns
data_sheet_3_copy = data_sheet_3.copy()

sheet_2_data_valid = {}
for row_index in sheet_2_index:
    data = data_sheet_2.loc[row_index]
    data_sum = sum(data)
    if 85 <= data_sum <= 105:
        sheet_2_data_valid[row_index] = 1
    else:
        sheet_2_data_valid[row_index] = 0
sheet_2_data_valid_df = pd.DataFrame(sheet_2_data_valid.values(),
index=sheet_2_data_valid.keys(), columns=['是否有效'])
sheet_2_data_valid_df.to_excel('data/第一问数据/表单2数据有效
性.xlsx')

"""
    表单1数据统计
"""
no_weather_dict = {}
weathered_dict = {}
weathered_index = []  # 风化的ID
no_weather_index = []  # 未风化的ID
for row_index in sheet_1_index:
    data = data_sheet_1.loc[row_index]
    weather_condition = data['表面风化']
```

```
    if weather_condition == '无风化':
        no_weather_index.append(row_index)
        no_weather_dict[row_index] = data
    else:
        weathered_index.append(row_index)
        weathered_dict[row_index] = data

weather_info = {
    '纹饰': {'A': 0, 'B': 0, 'C': 0},
    '类型': {'高钾': 0, '铅钡': 0},
    '颜色': {
        '黑': 0, '蓝绿': 0, '绿': 0, '浅蓝': 0, '浅绿': 0, '深蓝': 0, '深绿': 0, '紫': 0, '无': 0
    }
}
for index in weathered_index:
    data = weathered_dict[index]
    decoration = data[0]
    kind = data[1]
    color = data[2]
    weather_info['纹饰'][decoration] += 1
    weather_info['类型'][kind] += 1
    if color == 0:
        weather_info['颜色']['无'] += 1
    else:
        weather_info['颜色'][color] += 1

no_weather_info = {
    '纹饰': {'A': 0, 'B': 0, 'C': 0},
    '类型': {'高钾': 0, '铅钡': 0},
    '颜色': {
        '黑': 0, '蓝绿': 0, '绿': 0, '浅蓝': 0, '浅绿': 0, '深蓝': 0, '深绿': 0, '紫': 0, '无': 0
    }
}

for index in no_weather_index:
    data = no_weather_dict[index]
    decoration = data[0]
    kind = data[1]
    color = data[2]
    no_weather_info['纹饰'][decoration] += 1
    no_weather_info['类型'][kind] += 1
    if color == 0:
```

```
            no_weather_info['颜色']['无'] += 1
        else:
            no_weather_info['颜色'][color] += 1

    """
        表单 1 堆叠柱状图
    """
    # 纹饰图
    name_list = no_weather_info['纹饰'].keys()
    grid = plt.GridSpec(2, 4)
    plt.figure()
    plt.subplot(grid[0, 0:2])
    plt.bar(range(len(name_list)), no_weather_info['纹饰
    '].values(), label='未风化', fc='dodgerblue', width=0.4)
    plt.bar(range(len(name_list)), weather_info['纹饰
    '].values(), bottom=list(no_weather_info['纹饰'].values()),
            label='风化
    ', tick_label=list(name_list), fc='xkcd:powder blue', width=0.4)
    plt.legend()

    # 类型图
    name_list = no_weather_info['类型'].keys()
    plt.subplot(grid[0, 2:4])
    plt.bar(range(len(name_list)), no_weather_info['类型
    '].values(), label='未风化', fc='dodgerblue', width=0.2)
    plt.bar(range(len(name_list)), weather_info['类型
    '].values(), bottom=list(no_weather_info['类型'].values()),
            label='风化
    ', tick_label=list(name_list), fc='xkcd:powder blue', width=0.2)
    plt.legend()

    # 颜色图
    name_list = no_weather_info['颜色'].keys()
    plt.subplot(grid[1, :])
    plt.bar(range(len(name_list)), no_weather_info['颜色
    '].values(), label='未风化', fc='dodgerblue')
    plt.bar(range(len(name_list)), weather_info['颜色
    '].values(), bottom=list(no_weather_info['颜色'].values()),
            label='风化
    ', tick_label=list(name_list), fc='xkcd:powder blue')
    plt.legend()
    plt.tight_layout()
    plt.savefig(r'picture/第一问图像/风化程度与类型、纹饰、颜色关系图/
    合并图.png')
```

```
154.
155.  """
156.      结合玻璃类型，分析文物样品表面有无风化化学成分含量的统计规律
157.      可用数据：
158.          sheet_2_data_valid:  判断表单二中的数据是否有效
159.          sheet_2_data_weather: 表单二中每个表项是否风化
160.
161.  """
162.  sheet_2_data_weather = {}
163.  sheet_2_data_kind = {}
164.  sheet_2_data_color = {}
165.  for index in sheet_2_index:
166.      index_1 = int(index[0:2])
167.      sheet_2_data_weather[index] = data_sheet_1.loc[index_1]['表
     面风化']
168.      sheet_2_data_kind[index] = data_sheet_1.loc[index_1]['类型']
169.      sheet_2_data_color[index] = data_sheet_1.loc[index_1]['颜色']
170.  """
171.      未风化与风化的所有的元素含量
172.  """
173.  # 未风化与风化的所有的元素含量
174.  no_weather_df = []
175.  no_weather_index_2 = []
176.  weather_df = []
177.  weather_index_2 = []
178.  for index in sheet_2_index:
179.      data_valid = sheet_2_data_valid[index]
180.      # 若为无效数据则过滤
181.      if data_valid == 0:
182.          continue
183.      data_weather = sheet_2_data_weather[index]
184.      if data_weather == '无风化':
185.          no_weather_index_2.append(index)
186.          no_weather_df.append(data_sheet_2.loc[index].values)
187.      else:
188.          weather_index_2.append(index)
189.          weather_df.append(data_sheet_2.loc[index].values)
190.
191.  weather_df = pd.DataFrame(weather_df, index=weather_index_2, col
     umns=data_sheet_2.columns)
192.  weather_df.to_excel(r'data/第一问数据/表面有无风化的统计规律/化的所
     有文物数据.xlsx')
193.  no_weather_df = pd.DataFrame(no_weather_df, index=no_weather_ind
     ex_2, columns=data_sheet_2.columns)
```

```
194.  no_weather_df.to_excel(r'data/第一问数据/表面有无风化的统计规律/未风化的所有文物数据.xlsx')
195.
196.  """
197.      表单 2 所有元素标注，类型和风化或未风化
198.  """
199.  data_sheet_2_copy = data_sheet_2.copy()
200.  sheet_2_no_weather = []  # 未风化
201.  sheet_2_weather = []  # 风化
202.  sheet_2_severe_weather = []  # 严重风化
203.  sheet_2_color = []
204.  sheet_2_kind = []  # 种类
205.  for index in sheet_2_index:
206.      data_valid = sheet_2_data_valid[index]
207.      # 是否风化
208.      data_weather = sheet_2_data_weather[index]
209.      # 颜色
210.      data_color = sheet_2_data_color[index]
211.      sheet_2_color.append(data_color)
212.      # 种类
213.      data_kind = sheet_2_data_kind[index]
214.      sheet_2_kind.append(data_kind)
215.      if len(index) > 2:
216.          judge_word = index[2]
217.          if judge_word == '未':
218.              sheet_2_no_weather.append(1)
219.              sheet_2_weather.append(0)
220.              sheet_2_severe_weather.append(0)
221.          elif judge_word == '严':
222.              sheet_2_no_weather.append(0)
223.              sheet_2_weather.append(1)
224.              sheet_2_severe_weather.append(1)
225.          else:
226.              sheet_2_no_weather.append(0 if data_weather == '风化' else 1)
227.              sheet_2_weather.append(1 if data_weather == '风化' else 0)
228.              sheet_2_severe_weather.append(0)
229.      else:
230.          sheet_2_no_weather.append(0 if data_weather == '风化' else 1)
231.          sheet_2_weather.append(1 if data_weather == '风化' else 0)
232.          sheet_2_severe_weather.append(0)
```

```
233.  # 插入列
234.  data_sheet_2_copy.insert(data_sheet_2_copy.shape[1], '无风化
   ', sheet_2_no_weather)
235.  data_sheet_2_copy.insert(data_sheet_2_copy.shape[1], '风化
   ', sheet_2_weather)
236.  data_sheet_2_copy.insert(data_sheet_2_copy.shape[1], '严重风化
   ', sheet_2_severe_weather)
237.  data_sheet_2_copy.insert(data_sheet_2_copy.shape[1], '颜色
   ', sheet_2_color)
238.  data_sheet_2_copy.insert(data_sheet_2_copy.shape[1], '种类
   ', sheet_2_kind)
239.  # 清洗无用数据
240.  for index in sheet_2_index:
241.      data_valid = sheet_2_data_valid[index]
242.      if data_valid == 0:
243.          data_sheet_2_copy.drop(index, inplace=True)
244.  data_sheet_2_copy.to_excel(r'data/第一问数据/风化前的含量预测/带风
   化分类与颜色的数据（删去无效数据）.xlsx')
245.
246.  """
247.      统计风化的文物中每一个元素的百分比
248.      新加的列
249.      data_sheet_2_copy: 是经过处理的数据
250.      sheet_2_no_weather = []     # 未风化
251.      sheet_2_weather = []        # 风化
252.      sheet_2_severe_weather = [] # 严重风化
253.      sheet_2_color = []          # 颜色
254.      sheet_2_kind = []           # 种类
255.  """
256.  sheet_2_pro_path = r'data/第一问数据/风化前的含量预测/带风化分类与
   颜色的数据（删去无效数据）.xlsx'
257.  sheet_2_pro = pd.read_excel(sheet_2_pro_path, index_col=0, heade
   r=0)
258.  # 读取处理过的数据
259.  sheet_2_no_weather = sheet_2_pro.loc[:, '无风化']
260.  sheet_2_weather = sheet_2_pro.loc[:, '风化']
261.  sheet_2_severe_weather = sheet_2_pro.loc[:, '严重风化']
262.  sheet_2_color = sheet_2_pro.loc[:, '颜色']
263.  sheet_2_kind = sheet_2_pro.loc[:, '种类']
264.
265.  sheet_2_pro_index = sheet_2_pro.index
266.  weather_ingredients_dict = {}
267.  no_weather_ingredients_dict = {}
268.  for num in range(len(sheet_2_pro_index)):
```

```
        index = sheet_2_pro_index[num]
        # 风化情况
        data_weather = sheet_2_weather[num]
        # 文物种类
        data_kind = sheet_2_kind[num]

        # 无风化情况
        if data_weather == 0:
            for col_index in sheet_2_columns:
                data_col = sheet_2_pro.loc[index][col_index]
                if col_index not in no_weather_ingredients_dict:
                    no_weather_ingredients_dict[col_index] = {
                        '高钾': [],
                        '铅钡': []
                    }
                no_weather_ingredients_dict[col_index][data_kind].append(data_col)
        else:
            for col_index in sheet_2_columns:
                data_col = sheet_2_pro.loc[index][col_index]
                if col_index not in weather_ingredients_dict:
                    weather_ingredients_dict[col_index] = {
                        '高钾': [],
                        '铅钡': []
                    }
                weather_ingredients_dict[col_index][data_kind].append(data_col)

    """
        绘制每个元素的折线图与核密度图
    """
    for ingredient in weather_ingredients_dict.keys():
        # 有风化高钾数据
        weather_ingredient_kind_1 = weather_ingredients_dict[ingredient]['高钾']
        # 有风化铅钡数据
        weather_ingredient_kind_2 = weather_ingredients_dict[ingredient]['铅钡']
        # 无风化高钾
        no_weather_ingredient_kind_1 = no_weather_ingredients_dict[ingredient]['高钾']
        # 无风化铅钡
        no_weather_ingredient_kind_2 = no_weather_ingredients_dict[ingredient]['铅钡']
```

```
307.
308.     """
309.         处理数据，最大值，平均值，最小值
310.     """
311.     weather_kind_1_dict = {
312.         'max': max(weather_ingredient_kind_1),
313.         'mean': np.mean(weather_ingredient_kind_1),
314.         'min': min(weather_ingredient_kind_1)
315.     }
316.     weather_kind_2_dict = {
317.         'max': max(weather_ingredient_kind_2),
318.         'mean': np.mean(weather_ingredient_kind_2),
319.         'min': min(weather_ingredient_kind_2)
320.     }
321.     no_weather_kind_1_dict = {
322.         'max': max(no_weather_ingredient_kind_1),
323.         'mean': np.mean(no_weather_ingredient_kind_1),
324.         'min': min(no_weather_ingredient_kind_1)
325.     }
326.     no_weather_kind_2_dict = {
327.         'max': max(no_weather_ingredient_kind_2),
328.         'mean': np.mean(no_weather_ingredient_kind_2),
329.         'min': min(no_weather_ingredient_kind_2)
330.     }
331.     # 绘制高钾图
332.     plt.figure()
333.     sns.set(font='SimHei')
334.     plt.title('{}含量与风化关系图(高钾）'.format(ingredient))
335.
336.     sns.kdeplot(weather_ingredient_kind_1, label='有风化高钾')
337.     sns.kdeplot(no_weather_ingredient_kind_1, label='无风化高钾')
338.     plt.legend()
339.     plt.savefig(r'picture/第一问图像/核密度图/高钾/{}含量与风化关系
    图(高钾）.png'.format(ingredient))
340.
341.     # 绘制铅钡图
342.     plt.figure()
343.     sns.set(font='SimHei')
344.     plt.title('{}含量与风化关系图(铅钡）'.format(ingredient))
345.     sns.kdeplot(weather_ingredient_kind_2, label='有风化铅钡')
346.     sns.kdeplot(no_weather_ingredient_kind_2, label='无风化铅钡')
347.     plt.legend()
348.     plt.savefig(r'picture/第一问图像/核密度图/铅钡/{}含量与风化关系
    图（铅钡）.png'.format(ingredient))
```

```
    """
        根据风化点检测数据，预测其风化前的化学成分含量。
    """
    # 计算变化率
    change_rate_dict = {}
    for ingredient in weather_ingredients_dict.keys():
        # 风化后高钾和铅钡的百分比含量
        weather_kind_1_mean = np.mean(weather_ingredients_dict[ingredient]['高钾'])
        weather_kind_2_mean = np.mean(weather_ingredients_dict[ingredient]['铅钡'])
        # 风化前高钾和铅钡的百分比含量
        no_weather_kind_1_mean = np.mean(no_weather_ingredients_dict[ingredient]['高钾'])
        no_weather_kind_2_mean = np.mean(no_weather_ingredients_dict[ingredient]['铅钡'])

        # 风化前方差计算
        weather_kind_1_std = np.std(weather_ingredients_dict[ingredient]['高钾'])
        weather_kind_2_std = np.std(weather_ingredients_dict[ingredient]['铅钡'])
        # 风化后方差计算
        no_weather_kind_1_std = np.std(no_weather_ingredients_dict[ingredient]['高钾'])
        no_weather_kind_2_std = np.std(no_weather_ingredients_dict[ingredient]['铅钡'])

        # 高级方案
        kind_1_change_rate = {
            'before mean': no_weather_kind_1_mean,
            'before std': no_weather_kind_1_std,
            'after mean': weather_kind_1_mean,
            'after std': weather_kind_1_std
        }

        kind_2_change_rate = {
            'before mean': no_weather_kind_2_mean,
            'before std': no_weather_kind_2_std,
            'after mean': weather_kind_2_mean,
            'after std': weather_kind_2_std
        }

        change_rate_dict[ingredient] = {
```

```
            '高钾': kind_1_change_rate,
            '铅钡': kind_2_change_rate
        }

    change_rate_dict_df = pd.DataFrame(change_rate_dict)

    sheet_2_predict_no_weather = pd.DataFrame(index=sheet_2_pro_index, columns=sheet_2_pro.columns)
    for num in range(len(sheet_2_pro_index)):
        index = sheet_2_pro_index[num]
        # 风化情况
        data_weather = sheet_2_weather[num]
        # 文物种类
        data_kind = sheet_2_kind[num]

        row_data = sheet_2_predict_no_weather.loc[index]
        for column_index in range(len(sheet_2_pro.columns)):
            column = sheet_2_pro.columns[column_index]
            cur_data = sheet_2_pro.loc[index][column]
            # 风化属性与颜色属性
            if 14 <= column_index <= 18:
                row_data.loc[column] = cur_data
            else:
                # 未风化的情况
                if data_weather == 0:
                    row_data.loc[column] = cur_data
                # 风化情况
                else:
                    # 高级方案
                    before_mean = change_rate_dict[column][data_kind]['before mean']
                    before_std = change_rate_dict[column][data_kind]['before std']
                    after_mean = change_rate_dict[column][data_kind]['after mean']
                    after_std = change_rate_dict[column][data_kind]['after std']

                    if after_std == 0:
                        row_data.loc[column] = before_mean
                    else:
                        row_data.loc[column] = ((cur_data - after_mean) / after_std) * before_std + before_mean

```

```
                if row_data.loc[column] < 0:
                    row_data.loc[column] = 0
sheet_2_predict_no_weather.to_excel(r'data/第二问模型/预测数据（未经标准化）/预测数据（无风化）.xlsx')

"""
    表单 2 计算所有的风化后的数据
"""

sheet_2_predict_weather = pd.DataFrame(index=sheet_2_pro_index, columns=sheet_2_pro.columns)
for num in range(len(sheet_2_pro_index)):
    index = sheet_2_pro_index[num]
    # 风化情况
    data_weather = sheet_2_weather[num]
    # 文物种类
    data_kind = sheet_2_kind[num]

    row_data = sheet_2_predict_weather.loc[index]
    for column_index in range(len(sheet_2_pro.columns)):
        column = sheet_2_pro.columns[column_index]
        cur_data = sheet_2_pro.loc[index][column]
        # 风化属性与颜色属性
        if 14 <= column_index <= 18:
            row_data.loc[column] = cur_data
        else:
            # 风化的情况
            if data_weather == 1:

                row_data.loc[column] = cur_data
            # 未风化情况
            else:
                # 高级方案
                before_mean = change_rate_dict[column][data_kind]['before mean']
                before_std = change_rate_dict[column][data_kind]['before std']
                after_mean = change_rate_dict[column][data_kind]['after mean']
                after_std = change_rate_dict[column][data_kind]['after std']

                if after_std == 0:
                    row_data.loc[column] = before_mean
```

```
                else:
                    if after_std == 0:
                        row_data.loc[column] = before_mean
                    else:
                        row_data.loc[column] = ((cur_data - before_mean) / before_std) * after_std + after_mean

                if row_data.loc[column] < 0:
                    row_data.loc[column] = 0

    sheet_2_predict_weather.to_excel(r'data/第二问模型/预测数据（未经标准化）/预测数据（有风化）.xlsx')


    """
        对 sheet_2_predict 进行归 100 化
        sheet_2_predict
    """
    std_num = 100
    for index in sheet_2_predict_no_weather.index:
        column_data = sheet_2_predict_no_weather.loc[index][:14]
        data_sum = sum(column_data)
        if 85 <= data_sum <= 105:
            continue
        else:
            std_rate = data_sum / std_num
            for data_index in range(len(column_data)):
                sheet_2_predict_no_weather.loc[index][data_index] /= std_rate
    sheet_2_predict_no_weather.to_excel(r'data/第二问模型/预测数据（经过标准化）/预测数据（无风化）(经过标准化）.xlsx')

    """
        对 sheet_2_no_weather 进行归 100 化
    """
    for index in sheet_2_predict_weather.index:
        column_data = sheet_2_predict_weather.loc[index][:14]
        data_sum = sum(column_data)
        if 85 <= data_sum <= 105:
            continue
        else:
            std_rate = data_sum / std_num
            for data_index in range(len(column_data)):
```

```
500.             sheet_2_predict_weather.loc[index][data_index] /= st
    d_rate
501.  sheet_2_predict_weather.to_excel(r'data/第二问模型/预测数据（经过
    标准化）/预测数据(经过标准化）(风化后）.xlsx')
502.
503.  """
504.      预测数据归一化
505.  """
506.  sheet_2_predict_normalize = pd.DataFrame(index=sheet_2_pro_index
    , columns=sheet_2_pro.columns)
507.  for index in sheet_2_predict_normalize.index:
508.
509.      for column_index in range(len(sheet_2_pro.columns)):
510.          column = sheet_2_pro.columns[column_index]
511.          if 14 <= column_index <= 18:
512.              sheet_2_predict_normalize.loc[index][column] = sheet
    _2_predict_no_weather.loc[index][column]
513.          else:
514.              # 标准化方案2
515.              max = np.max(sheet_2_predict_no_weather.loc[:][colum
    n])
516.              min = np.min(sheet_2_predict_no_weather.loc[:][colum
    n])
517.              sheet_2_predict_normalize.loc[index][column] = (shee
    t_2_predict_no_weather.loc[index][column] - min) / (
518.                      max - min)
519.
520.      row_sum = sum(sheet_2_predict_normalize.loc[index][:14])
521.      if row_sum < 0.85 or row_sum > 1.05:
522.          sheet_2_predict_normalize.loc[index][:14] /= row_sum
523.
524.  # 列归一化
525.  sheet_2_predict_normalize_copy = sheet_2_predict_normalize.copy()
526.
527.  for index in sheet_2_pro.index:
528.      for column_index in range(len(sheet_2_pro.columns)):
529.          column = sheet_2_pro.columns[column_index]
530.          if column_index < 14:
531.              col_sum = sum(sheet_2_predict_normalize_copy.loc[:][
    column])
532.              if col_sum == 0:
533.                  continue
534.              sheet_2_predict_normalize.loc[index][column] /= col_
    sum
```

```
535.  sheet_2_predict_normalize.to_excel(
536.      r'data/第二问模型/预测数据（经过按列归一化）/经过标准化和归一化的预测数据(无风化）.xlsx')
537.
538.  """
539.      风化预测数据归一化
540.  """
541.  sheet_2_predict_weather_normalize = pd.DataFrame(index=sheet_2_pro_index, columns=sheet_2_pro.columns)
542.  for index in sheet_2_predict_normalize.index:
543.      for column_index in range(len(sheet_2_pro.columns)):
544.          column = sheet_2_pro.columns[column_index]
545.          if 14 <= column_index <= 18:
546.              sheet_2_predict_weather_normalize.loc[index][column] = sheet_2_predict_weather.loc[index][column]
547.          else:
548.              # 标准化方案2
549.              max = np.max(sheet_2_predict_weather.loc[:][column])
550.              min = np.min(sheet_2_predict_weather.loc[:][column])
551.              sheet_2_predict_weather_normalize.loc[index][column] = (sheet_2_predict_weather.loc[index][
552.
            column] - min) / (max - min)
553.      row_sum = sum(sheet_2_predict_weather_normalize.loc[index][:14])
554.      if row_sum < 0.85 or row_sum > 1.05:
555.          sheet_2_predict_weather_normalize.loc[index][:14] /= row_sum
556.
557.  # 列归一化
558.  sheet_2_predict_weather_normalize_copy = sheet_2_predict_weather_normalize.copy()
559.
560.  for index in sheet_2_pro.index:
561.      for column_index in range(len(sheet_2_pro.columns)):
562.          column = sheet_2_pro.columns[column_index]
563.          if column_index < 14:
564.              col_sum = sum(sheet_2_predict_weather_normalize_copy.loc[:][column])
565.              if col_sum == 0:
566.                  continue
567.              sheet_2_predict_weather_normalize.loc[index][column] /= col_sum
568.
```

```
sheet_2_predict_weather_normalize.to_excel(
    r'data/第二问模型/预测数据（经过按列归一化）/经过标准化和归一化的预测数据(全部为风化后）.xlsx')

"""
    分类划分（无归一化）
"""
# 风化前的相关元素表
correlation_cofficient_dict = dict.fromkeys(sheet_2_pro.columns[:14])
# 1 代表正相关，0 代表无关，-1 代表负相关
correlation_cofficient_list = [1, 0, 1, 1, 0,
                               1, 1, 1, -1, -1,
                               0, -1, 0, 0]

for index in range(len(correlation_cofficient_dict.keys())):
    key = list(correlation_cofficient_dict.keys())[index]
    correlation_cofficient_dict[key] = correlation_cofficient_list[index]

# 风化后的相关元素表
correlation_cofficient_dict_weather = dict.fromkeys(sheet_2_pro.columns[:14])
correlation_cofficient_list_weather = [1, 0, 1, 1, 0, 0,
                                       0, 0, 1, 1, 1, 1,
                                       0, 0]
for index in range(len(correlation_cofficient_dict_weather.keys())):
    key = list(correlation_cofficient_dict.keys())[index]
    correlation_cofficient_dict_weather[key] = correlation_cofficient_list_weather[index]

"""
    计算分界线相关数据
"""
# 构造绘图数据(风化前）
relevant_ingredient_dict = {}
for index in sheet_2_predict_no_weather.index:
    data_kind = sheet_2_predict_no_weather.loc[index]['种类']

    for ingredient in correlation_cofficient_dict.keys():
        correlation_cofficient = correlation_cofficient_dict[ingredient]
        # 无关
```

```
            if correlation_cofficient == 0:
                continue
            # 有关
            else:
                if ingredient not in relevant_ingredient_dict.keys():
                    relevant_ingredient_dict[ingredient] = {
                        '高钾': [],
                        '铅钡': []
                    }
                ingredient_data = sheet_2_predict_no_weather.loc[index][ingredient]
                relevant_ingredient_dict[ingredient][data_kind].append(ingredient_data)

    # 画图数据（风化后）
    relevant_ingredient_dict_weather = {}
    for index in sheet_2_predict_no_weather.index:
        data_kind = sheet_2_predict_weather.loc[index]['种类']

        for ingredient in correlation_cofficient_dict_weather.keys():
            correlation_cofficient = correlation_cofficient_dict_weather[ingredient]
            # 无关
            if correlation_cofficient == 0:
                continue
            # 有关
            else:
                if ingredient not in relevant_ingredient_dict_weather.keys():
                    relevant_ingredient_dict_weather[ingredient] = {
                        '高钾': [],
                        '铅钡': []
                    }
                ingredient_data = sheet_2_predict_weather.loc[index][ingredient]
                relevant_ingredient_dict_weather[ingredient][data_kind].append(ingredient_data)

    """
        分类划分，经过归一化数据
    """
    # 风化前数据
    relevant_ingredient_dict_normalize = {}
    for index in sheet_2_predict_no_weather.index:
```

```
644.      data_kind = sheet_2_predict_normalize.loc[index]['种类']
645.
646.      for ingredient in correlation_cofficient_dict.keys():
647.          correlation_cofficient = correlation_cofficient_dict[ing
   redient]
648.          # 无关
649.          if correlation_cofficient == 0:
650.              continue
651.          # 有关
652.          else:
653.              if ingredient not in relevant_ingredient_dict_normal
   ize.keys():
654.                  relevant_ingredient_dict_normalize[ingredient] =
    {
655.                      '高钾': [],
656.                      '铅钡': []
657.                  }
658.              ingredient_data = sheet_2_predict_normalize.loc[inde
   x][ingredient]
659.              relevant_ingredient_dict_normalize[ingredient][data_
   kind].append(ingredient_data)
660.
661.  # 风化后数据
662.  relevant_ingredient_dict_weather_normalize = {}
663.  for index in sheet_2_predict_no_weather.index:
664.      data_kind = sheet_2_predict_normalize.loc[index]['种类']
665.
666.      for ingredient in correlation_cofficient_dict_weather.keys():
667.          correlation_cofficient = correlation_cofficient_dict_wea
   ther[ingredient]
668.          # 无关
669.          if correlation_cofficient == 0:
670.              continue
671.          # 有关
672.          else:
673.              if ingredient not in relevant_ingredient_dict_weathe
   r_normalize.keys():
674.                  relevant_ingredient_dict_weather_normalize[ingre
   dient] = {
675.                      '高钾': [],
676.                      '铅钡': []
677.                  }
678.              ingredient_data = sheet_2_predict_weather_normalize.
   loc[index][ingredient]
```

```
679.                 relevant_ingredient_dict_weather_normalize[ingredien
   t][data_kind].append(ingredient_data)
680.
681.  """
682.      计算分界值
683.  """
684.  # 风化前
685.  best_divide_dict = {}
686.  for ingredient in relevant_ingredient_dict.keys():
687.      kind_1_data = relevant_ingredient_dict_normalize[ingredient]
   ['高钾']
688.      kind_2_data = relevant_ingredient_dict_normalize[ingredient]
   ['铅钡']
689.
690.      # 区间范围，两个平均值之间，步长为0.01
691.      kind_1_mean = np.mean(kind_1_data)
692.      kind_2_mean = np.mean(kind_2_data)
693.      if kind_1_mean < kind_2_mean:
694.          divide_line_list = np.arange(kind_1_mean, kind_2_mean, 0
   .01)
695.      else:
696.          divide_line_list = np.arange(kind_2_mean, kind_1_mean, 0
   .01)
697.
698.      # 以平均值的平均值作为最佳分界线初始值
699.      best_divide = (kind_1_mean + kind_2_mean)
700.      if kind_1_mean < kind_2_mean:
701.          kind_1_sel = [sel_data for sel_data in kind_1_data if se
   l_data > best_divide]
702.          kind_2_sel = [sel_data for sel_data in kind_2_data if se
   l_data < best_divide]
703.      else:
704.          kind_1_sel = [sel_data for sel_data in kind_1_data if se
   l_data < best_divide]
705.          kind_2_sel = [sel_data for sel_data in kind_2_data if se
   l_data > best_divide]
706.
707.      # 超过分界线的值到分界线的距离和
708.      best_distance = sum(abs(data -
    best_divide) for data in kind_1_sel) \
709.                      + sum(abs(data -
    best_divide) for data in kind_2_sel)
710.
711.      for divide_line in divide_line_list:
```

```
712.          if kind_1_mean < kind_2_mean:
713.              kind_1_sel = [sel_data for sel_data in kind_1_data i
    f sel_data > divide_line]
714.              kind_2_sel = [sel_data for sel_data in kind_2_data i
    f sel_data < divide_line]
715.          else:
716.              kind_1_sel = [sel_data for sel_data in kind_1_data i
    f sel_data < divide_line]
717.              kind_2_sel = [sel_data for sel_data in kind_2_data i
    f sel_data > divide_line]
718.
719.          distance = sum(abs(data -
    divide_line) for data in kind_1_sel) \
720.                     + sum(abs(data -
    divide_line) for data in kind_2_sel)
721.
722.          # 当找到更好的就更新
723.          if distance < best_distance:
724.              best_distance = distance
725.              best_divide = divide_line
726.
727.      # 遍历结束，写入结果
728.      best_divide_dict[ingredient] = {
729.          'num': best_divide,
730.          'up': '铅钡' if kind_1_mean < kind_2_mean else '高钾',
731.          'down': '高钾' if kind_1_mean < kind_2_mean else '铅钡'
732.      }
733.
734.      # 画图
735.      plt.figure()
736.      plt.title('{}含量分类图'.format(ingredient))
737.      plt.plot(kind_1_data, label='高钾')
738.      plt.plot(kind_2_data, label='铅钡')
739.      plt.hlines(best_divide, -5, 60, linestyles='dashed')
740.      plt.legend()
741.      plt.savefig(r'picture/第三问图像/分界线数据/{}(风化前数
    据）.png'.format(ingredient))
742.
743.  # 输出到excel
744.  best_divide_dict_df = pd.DataFrame(index=best_divide_dict.keys()
    , columns=['num', 'up', 'down'])
745.  for index in best_divide_dict_df.index:
746.      for column in best_divide_dict_df.columns:
```

```
747.          best_divide_dict_df.loc[index][column] = best_divide_dic
   t[index][column]
748.  best_divide_dict_df.to_excel(r'data/第三问数据/分界值/分界值（归一
   化）(风化前）.xlsx')
749.
750.  # 风化后
751.  best_divide_dict_weather = {}
752.  for ingredient in relevant_ingredient_dict_weather.keys():
753.      kind_1_data = relevant_ingredient_dict_weather_normalize[ing
   redient]['高钾']
754.      kind_2_data = relevant_ingredient_dict_weather_normalize[ing
   redient]['铅钡']
755.
756.      # 区间范围，两个平均值之间，步长为0.01
757.      kind_1_mean = np.mean(kind_1_data)
758.      kind_2_mean = np.mean(kind_2_data)
759.      if kind_1_mean < kind_2_mean:
760.          divide_line_list = np.arange(kind_1_mean, kind_2_mean, 0
   .01)
761.      else:
762.          divide_line_list = np.arange(kind_2_mean, kind_1_mean, 0
   .01)
763.
764.      # 以平均值的平均值作为最佳分界线初始值
765.      best_divide = (kind_1_mean + kind_2_mean)
766.      if kind_1_mean < kind_2_mean:
767.          kind_1_sel = [sel_data for sel_data in kind_1_data if se
   l_data > best_divide]
768.          kind_2_sel = [sel_data for sel_data in kind_2_data if se
   l_data < best_divide]
769.      else:
770.          kind_1_sel = [sel_data for sel_data in kind_1_data if se
   l_data < best_divide]
771.          kind_2_sel = [sel_data for sel_data in kind_2_data if se
   l_data > best_divide]
772.
773.      # 超过分界线的值到分界线的距离和
774.      best_distance = sum(abs(data -
    best_divide) for data in kind_1_sel) \
775.                      + sum(abs(data -
    best_divide) for data in kind_2_sel)
776.
777.      for divide_line in divide_line_list:
778.          if kind_1_mean < kind_2_mean:
```

```
779.                 kind_1_sel = [sel_data for sel_data in kind_1_data i
    f sel_data > divide_line]
780.                 kind_2_sel = [sel_data for sel_data in kind_2_data i
    f sel_data < divide_line]
781.             else:
782.                 kind_1_sel = [sel_data for sel_data in kind_1_data i
    f sel_data < divide_line]
783.                 kind_2_sel = [sel_data for sel_data in kind_2_data i
    f sel_data > divide_line]
784.
785.             distance = sum(abs(data -
     divide_line) for data in kind_1_sel) \
786.                        + sum(abs(data -
     divide_line) for data in kind_2_sel)
787.
788.             # 当找到更好的就更新
789.             if distance < best_distance:
790.                 best_distance = distance
791.                 best_divide = divide_line
792.
793.         # 遍历结束，写入结果
794.         best_divide_dict_weather[ingredient] = {
795.             'num': best_divide,
796.             'up': '铅钡' if kind_1_mean < kind_2_mean else '高钾',
797.             'down': '高钾' if kind_1_mean < kind_2_mean else '铅钡'
798.         }
799.
800.         # 画图
801.         plt.figure()
802.         plt.title('{}含量分类图'.format(ingredient))
803.         plt.plot(kind_1_data, label='高钾')
804.         plt.plot(kind_2_data, label='铅钡')
805.         plt.hlines(best_divide, -5, 60, linestyles='dashed')
806.         plt.legend()
807.         plt.savefig(r'picture/第三问图像/分界线数据/{}(风化后数
    据）.png'.format(ingredient))
808.
809.  # 输出到excel
810.  best_divide_dict_weather_df = pd.DataFrame(index=best_divide_dic
    t_weather.keys(), columns=['num', 'up', 'down'])
811.  for index in best_divide_dict_weather_df.index:
812.         for column in best_divide_dict_weather_df.columns:
813.             best_divide_dict_weather_df.loc[index][column] = best_di
    vide_dict_weather[index][column]
```

```
best_divide_dict_weather_df.to_excel(r'data/第三问数据/分界值/分界值（归一化）(风化后）.xlsx')

"""
    计算权重
"""

# 计算权重
ingredient_num = [
    -0.693670815, 0.119156192, -0.717030234, -0.344828276, -0.095775661, -0.202264054, -0.272875667,
    -0.054967783, 0.757008765, 0.534965174, 0.285351187, 0.53499528, -0.096593581, 0.121150082
]
# 每种元素的权重(风化前）
ingredient_weight = {}
for ingredient_index in range(len(correlation_cofficient_dict.keys())):
    ingredient = list(correlation_cofficient_dict.keys())[ingredient_index]
    relevant = correlation_cofficient_dict[ingredient]
    if relevant == 0:
        continue
    else:
        ingredient_weight[ingredient] = ingredient_num[ingredient_index]

# 权重归一化
weight_sum = sum(abs(value) for value in ingredient_weight.values())
for ingredient in ingredient_weight.keys():
    value = ingredient_weight[ingredient]
    ingredient_weight[ingredient] = value / weight_sum
ingredient_weight_df = pd.DataFrame(ingredient_weight.values(), index=ingredient_weight.keys(), columns=['权重'])
ingredient_weight_df.to_excel(r'picture/第三问图像/分界线权重/权重（风化前）.xlsx')

# 每种元素的权重(风化后）
ingredient_weight_weather = {}
for ingredient_index in range(len(correlation_cofficient_dict_weather.keys())):
```

```
    ingredient = list(correlation_cofficient_dict_weather.keys()
)[ingredient_index]
    relevant = correlation_cofficient_dict_weather[ingredient]
    if relevant == 0:
        continue
    else:
        ingredient_weight_weather[ingredient] = ingredient_num[i
ngredient_index]

# 权重归一化
weight_sum = sum(abs(value) for value in ingredient_weight_weath
er.values())
for ingredient in ingredient_weight_weather.keys():
    value = ingredient_weight_weather[ingredient]
    ingredient_weight_weather[ingredient] = value / weight_sum

ingredient_weight_weather_df = pd.DataFrame(ingredient_weight_we
ather.values(), index=ingredient_weight_weather.keys(),
                                            columns=['权重'])
ingredient_weight_weather_df.to_excel(r'picture/第三问图像/分界线
权重/权重（风化后）.xlsx')

"""
    风化前的规律验证
"""
judge_result = {}
correct_num = 0
for index in sheet_2_predict_no_weather.index:

    kind_1_score = 0  # 高钾得分
    kind_2_score = 0  # 铅钡得分

    for ingredient in ingredient_weight.keys():
        # 分界线
        divide_line = best_divide_dict[ingredient]['num']
        # 对应值
        ingredient_data = sheet_2_predict_normalize.loc[index][i
ngredient]
        # 上方对应种类
        up_kind = best_divide_dict[ingredient]['up']
        # 下方对应种类
        down_kind = best_divide_dict[ingredient]['down']
        # 权重(绝对值)
        weight = abs(ingredient_weight[ingredient])
```

```
884.
885.          # 确定种类
886.          if ingredient_data >= divide_line:
887.              kind = up_kind
888.          else:
889.              kind = down_kind
890.
891.          # 对应加分
892.          if kind == '高钾':
893.              kind_1_score += weight * abs(ingredient_data -
    divide_line)
894.          else:
895.              kind_2_score += weight * abs(ingredient_data -
    divide_line)
896.
897.      # 遍历完所有元素
898.      if kind_1_score < kind_2_score:
899.          predict_kind = '铅钡'
900.      else:
901.          predict_kind = '高钾'
902.      judge_result[index] = {
903.          '预测种类': predict_kind,
904.          '高钾得分': kind_1_score,
905.          '铅钡得分': kind_2_score
906.      }
907.      real_kind = sheet_2_predict_normalize.loc[index]['种类']
908.      if predict_kind == real_kind:
909.          correct_num += 1
910.
911.  judge_result_df = pd.DataFrame(judge_result.values(), index=judg
    e_result.keys(),
912.                                 columns=['预测种类', '高钾得分', '
    铅钡得分'])
913.  judge_result_df.to_excel(r'data/第三问数据/模型验证数据/模型检验数
    据（风化前）.xlsx')
914.
915.  """
916.      风化后的规律验证
917.  """
918.  judge_result_weather = {}
919.  correct_num = 0
920.  for index in sheet_2_predict_no_weather.index:
921.
922.      kind_1_score = 0  # 高钾得分
```

```
923.      kind_2_score = 0  # 铅钡得分
924.
925.      for ingredient in ingredient_weight_weather.keys():
926.          # 分界线
927.          divide_line = best_divide_dict_weather[ingredient]['num']
928.          # 对应值
929.          ingredient_data = sheet_2_predict_weather_normalize.loc[
    index][ingredient]
930.          # 上方对应种类
931.          up_kind = best_divide_dict_weather[ingredient]['up']
932.          # 下方对应种类
933.          down_kind = best_divide_dict_weather[ingredient]['down']
934.          # 权重(绝对值)
935.          weight = abs(ingredient_weight_weather[ingredient])
936.
937.          # 确定种类
938.          if ingredient_data >= divide_line:
939.              kind = up_kind
940.          else:
941.              kind = down_kind
942.
943.          # 对应加分
944.          if kind == '高钾':
945.              kind_1_score += weight * abs(ingredient_data -
    divide_line)
946.          else:
947.              kind_2_score += weight * abs(ingredient_data -
    divide_line)
948.
949.      # 遍历完所有元素
950.      if kind_1_score < kind_2_score:
951.          predict_kind = '铅钡'
952.      else:
953.          predict_kind = '高钾'
954.      judge_result_weather[index] = {
955.          '预测种类': predict_kind,
956.          '高钾得分': kind_1_score,
957.          '铅钡得分': kind_2_score
958.      }
959.      real_kind = sheet_2_predict_normalize.loc[index]['种类']
960.      if predict_kind == real_kind:
961.          correct_num += 1
962.
```

```
963.  judge_result_df = pd.DataFrame(judge_result_weather.values(), in
   dex=judge_result.keys(),
964.                                      columns=['预测种类', '高钾得分', '
   铅钡得分'])
965.  judge_result_df.to_excel(r'data/第三问数据/模型验证数据/模型检验数
   据（风化后）.xlsx')
966.
967.  """
968.      亚类数据的合理性分析和灵敏度分析(另一份代码里）
969.  """
970.
971.  """
972.      第三问鉴别种类
973.      可用数据：
974.          best_divide_dict = {
975.              'num': best_divide,
976.              'up': '铅钡' if kind_1_mean < kind_2_mean else '高钾
   ',
977.              'down': '高钾' if kind_1_mean < kind_2_mean else '铅
   钡'
978.          }
979.          ingredient_weight: 权重
980.  """
981.  """
982.      对表格 3 进行归一化处理
983.  """
984.  data_sheet_3_normalize = pd.DataFrame(index=data_sheet_3.index,
   columns=data_sheet_3.columns)
985.  for index in data_sheet_3_normalize.index:
986.      for column in data_sheet_3_normalize.columns:
987.          if column == '表面风化':
988.              data_sheet_3_normalize.loc[index][column] = data_she
   et_3.loc[index][column]
989.              continue
990.          mean = np.mean(data_sheet_3.loc[:][column])
991.          std = np.mean(data_sheet_3.loc[:][column])
992.          data_sheet_3_normalize.loc[index][column] = (data_sheet_
   3.loc[index][column] - mean) / std
993.
994.  predict_result_3 = {}
995.  for index in data_sheet_3.index:
996.      kind_1_score = 0
997.      kind_2_score = 0
998.
```

```
999.      data_kind = data_sheet_3_normalize.loc[index]['表面风化']
1000.     # 无风化的分界标准
1001.     if data_kind == '无风化':
1002.         for ingredient in ingredient_weight.keys():
1003.             # 分界线
1004.             divide_line = best_divide_dict[ingredient]['num']
1005.             # 上方对应种类
1006.             up_kind = best_divide_dict[ingredient]['up']
1007.             # 下方对应种类
1008.             down_kind = best_divide_dict[ingredient]['down']
1009.             # 对应值
1010.             ingredient_data = data_sheet_3_normalize.loc[index][
    ingredient]
1011.             # 权重(绝对值)
1012.             weight = abs(ingredient_weight[ingredient])
1013.             # 确定种类
1014.             if ingredient_data >= divide_line:
1015.                 kind = up_kind
1016.             else:
1017.                 kind = down_kind
1018.             # 对应加分
1019.             if kind == '高钾':
1020.                 kind_1_score += weight * abs(ingredient_data -
     divide_line)
1021.             else:
1022.                 kind_2_score += weight * abs(ingredient_data -
     divide_line)
1023.
1024.     # 有风化的分界标准
1025.     else:
1026.         for ingredient in ingredient_weight_weather.keys():
1027.             # 分界线
1028.             divide_line = best_divide_dict_weather[ingredient]['
    num']
1029.             # 上方对应种类
1030.             up_kind = best_divide_dict_weather[ingredient]['up']
1031.             # 下方对应种类
1032.             down_kind = best_divide_dict_weather[ingredient]['do
    wn']
1033.             # 对应值
1034.             ingredient_data = data_sheet_3_normalize.loc[index][
    ingredient]
1035.             # 权重(绝对值)
1036.             weight = abs(ingredient_weight_weather[ingredient])
```

```
            # 确定种类
            if ingredient_data >= divide_line:
                kind = up_kind
            else:
                kind = down_kind
            # 对应加分
            if kind == '高钾':
                kind_1_score += weight * abs(ingredient_data - divide_line)
            else:
                kind_2_score += weight * abs(ingredient_data - divide_line)

    # 遍历完所有元素
    if kind_1_score < kind_2_score:
        predict_kind = '铅钡'
    else:
        predict_kind = '高钾'

    predict_result_3[index] = {
        '预测种类': predict_kind,
        '高钾得分': kind_1_score,
        '铅钡得分': kind_2_score
    }

predict_result_3_df = pd.DataFrame(predict_result_3.values(), index=data_sheet_3.index,
                                   columns=['预测种类', '高钾得分', '铅钡得分'])
predict_result_3_df.to_excel(r'data/第三问数据/预测结果/第三问预测结果(归一化）.xlsx')

"""
    敏感性分析
    分别对每一类数据进行正负百分之 100 的调整，判断分类结果是否有变化(只对有风化进行含量改变）
"""
result_dict = {}
change_num = 1
for change_ingredient_index in range(len(ingredient_weight_weather.keys()) - change_num + 1):

    # 改动元素列表
```

```
    change_ingredient_list = list(ingredient_weight_weather.keys())[
                                change_ingredient_index:(change_ingredient_index + change_num)]

    # 复制数据原始表
    cur_data_sheet_3_normalize = data_sheet_3_normalize.copy()

    # 改动率列表
    change_rate_list = np.arange(0, 2, 0.1)

    cur_result_dict = {}  # 同一元素每个变化率的影响
    # 数据改动
    for change_rate in change_rate_list:

        correct_num = 0
        # 预测分类
        for index in cur_data_sheet_3_normalize.index:

            kind_1_score = 0
            kind_2_score = 0

            data_kind = data_sheet_3_normalize.loc[index]['表面风化']
            # 实际方案
            # # 无风化
            # if data_kind == '无风化':
            #     for ingredient in ingredient_weight.keys():
            #         # 分界线
            #         divide_line = best_divide_dict[ingredient]['num']
            #         # 对应值
            #         if ingredient == change_ingredient:
            #             ingredient_data = change_data.loc[index]
            #         else:
            #             ingredient_data = cur_data_sheet_3_normalize.loc[index][ingredient]
            #         # 上方对应种类
            #         up_kind = best_divide_dict[ingredient]['up']
            #         # 下方对应种类
            #         down_kind = best_divide_dict[ingredient]['down']
```

```
1109.             #         # 权重(绝对值)
1110.             #         weight = abs(ingredient_weight[ingredient])
1111.             #
1112.             #         # 确定种类
1113.             #         if ingredient_data >= divide_line:
1114.             #             kind = up_kind
1115.             #         else:
1116.             #             kind = down_kind
1117.             #
1118.             #         # 对应加分
1119.             #         if kind == '高钾':
1120.             #             kind_1_score += weight * abs(ingredien
    t_data - divide_line)
1121.             #         else:
1122.             #             kind_2_score += weight * abs(ingredien
    t_data - divide_line)
1123.             # # 有风化
1124.             # else:
1125.             #     for ingredient in ingredient_weight_weather.ke
    ys():
1126.             #         # 分界线
1127.             #         divide_line = best_divide_dict_weather[ing
    redient]['num']
1128.             #         # 对应值
1129.             #         if ingredient == change_ingredient:
1130.             #             ingredient_data = change_data.loc[inde
    x]
1131.             #         else:
1132.             #             ingredient_data = cur_data_sheet_3_nor
    malize.loc[index][ingredient]
1133.             #         # 上方对应种类
1134.             #         up_kind = best_divide_dict_weather[ingredi
    ent]['up']
1135.             #         # 下方对应种类
1136.             #         down_kind = best_divide_dict_weather[ingre
    dient]['down']
1137.             #         # 权重(绝对值)
1138.             #         weight = abs(ingredient_weight_weather[ing
    redient])
1139.             #
1140.             #         # 确定种类
1141.             #         if ingredient_data >= divide_line:
1142.             #             kind = up_kind
1143.             #         else:
```

```
1144.             #             kind = down_kind
1145.             #
1146.             #         # 对应加分
1147.             #         if kind == '高钾':
1148.             #             kind_1_score += weight * abs(ingredien
   t_data - divide_line)
1149.             #         else:
1150.             #             kind_2_score += weight * abs(ingredien
   t_data - divide_line)
1151.             # 只用无风化的分界线预测
1152.             for ingredient in ingredient_weight_weather.keys():
1153.                 # 分界线
1154.                 divide_line = best_divide_dict_weather[ingredien
   t]['num']
1155.                 # 对应值
1156.                 if ingredient in change_ingredient_list:
1157.                     ingredient_data = cur_data_sheet_3_normalize
   .loc[index][ingredient] * change_rate
1158.                 else:
1159.                     ingredient_data = cur_data_sheet_3_normalize
   .loc[index][ingredient]
1160.                 # 上方对应种类
1161.                 up_kind = best_divide_dict_weather[ingredient]['
   up']
1162.                 # 下方对应种类
1163.                 down_kind = best_divide_dict_weather[ingredient]
   ['down']
1164.                 # 权重(绝对值)
1165.                 weight = abs(ingredient_weight_weather[ingredien
   t])
1166.
1167.                 # 确定种类
1168.                 if ingredient_data >= divide_line:
1169.                     kind = up_kind
1170.                 else:
1171.                     kind = down_kind
1172.
1173.                 # 对应加分
1174.                 if kind == '高钾':
1175.                     kind_1_score += weight * abs(ingredient_data
    - divide_line)
1176.                 else:
1177.                     kind_2_score += weight * abs(ingredient_data
    - divide_line)
```

```
1178.
1179.             # 遍历完所有元素
1180.             if kind_1_score < kind_2_score:
1181.                 predict_kind = '铅钡'
1182.                 # judge_result[index] = '铅钡'
1183.             else:
1184.                 predict_kind = '高钾'
1185.
1186.             # 计算正确率
1187.             if predict_kind == predict_result_3[index]['预测种类
    ']:
1188.                 correct_num += 1
1189.
1190.         # 对于每个元素，关于改动幅度的正确率表
1191.         cur_result_dict[change_rate] = correct_num / len(predict
    _result_3)
1192.
1193.     # 每个元素的结果写入
1194.     result_dict[change_ingredient_index, change_ingredient_index
     + change_num] = cur_result_dict
1195.
1196. result_dict_df = pd.DataFrame(result_dict.values(), index=result
    _dict.keys())
1197. result_dict_df.to_excel('data/第三问数据/灵敏度分析/灵敏度分析数据
    (消去{}个元素）.xlsx'.format(change_num))
1198.
1199. """
1200.     第三问灵敏度分析思路二
1201. """
1202. correct_num_1 = 0  # 无风化分界线的正确数
1203. correct_num_2 = 0  # 有风化分界线的正确数
1204. predict_result_3_2 = {}
1205. # 预测分类
1206. for index in data_sheet_3_normalize.index:
1207.
1208.     kind_1_score = 0
1209.     kind_2_score = 0
1210.
1211.     data_kind = data_sheet_3_normalize.loc[index]['表面风化']
1212.     # 只用无风化的分界线预测
1213.     for ingredient in ingredient_weight.keys():
1214.         # 分界线
1215.         divide_line = best_divide_dict[ingredient]['num']
1216.         # 对应值
```

```
1217.         ingredient_data = data_sheet_3_normalize.loc[index][ingr
    edient]
1218.         # 上方对应种类
1219.         up_kind = best_divide_dict[ingredient]['up']
1220.         # 下方对应种类
1221.         down_kind = best_divide_dict[ingredient]['down']
1222.         # 权重(绝对值)
1223.         weight = abs(ingredient_weight[ingredient])
1224.
1225.         # 确定种类
1226.         if ingredient_data >= divide_line:
1227.             kind = up_kind
1228.         else:
1229.             kind = down_kind
1230.
1231.         # 对应加分
1232.         if kind == '高钾':
1233.             kind_1_score += weight * abs(ingredient_data -
    divide_line)
1234.         else:
1235.             kind_2_score += weight * abs(ingredient_data -
    divide_line)
1236.
1237.     # 遍历完所有元素
1238.     if kind_1_score < kind_2_score:
1239.         predict_kind = '铅钡'
1240.     else:
1241.         predict_kind = '高钾'
1242.
1243.     # 计算正确率
1244.     if predict_kind == predict_result_3[index]['预测种类']:
1245.         correct_num_1 += 1
1246.     # 有风化的分界线预测
1247.     for ingredient in ingredient_weight_weather.keys():
1248.         # 分界线
1249.         divide_line = best_divide_dict_weather[ingredient]['num']
1250.         # 对应值
1251.         ingredient_data = data_sheet_3_normalize.loc[index][ingr
    edient]
1252.         # 上方对应种类
1253.         up_kind = best_divide_dict_weather[ingredient]['up']
1254.         # 下方对应种类
1255.         down_kind = best_divide_dict_weather[ingredient]['down']
1256.         # 权重(绝对值)
```

```
            weight = abs(ingredient_weight_weather[ingredient])

            # 确定种类
            if ingredient_data >= divide_line:
                kind = up_kind
            else:
                kind = down_kind

            # 对应加分
            if kind == '高钾':
                kind_1_score += weight * abs(ingredient_data - divide_line)
            else:
                kind_2_score += weight * abs(ingredient_data - divide_line)

        # 遍历完所有元素
        if kind_1_score < kind_2_score:
            predict_kind = '铅钡'
        else:
            predict_kind = '高钾'

        # 计算正确率
        if predict_kind == predict_result_3[index]['预测种类']:
            correct_num_2 += 1

predict_result_3_2 = {
    '风化前': correct_num_1 / len(predict_result_3),
    '风化后': correct_num_2 / len(predict_result_3),
    '均使用': 1
}
"""
    第三问灵敏度分析绘图
"""

import matplotlib.ticker as ticker

plt.figure()
plt.title('各元素含量对于预测正确率的影响')
for ingredient in result_dict.keys():
    y = list(result_dict[ingredient].values())
    plt.plot(y, marker='*', label=ingredient)

```

```
plt.gca().xaxis.set_major_formatter(ticker.FormatStrFormatter('%.2f'))
x = np.linspace(0, 2, 21)
x_1 = [float('{:2f}'.format(i)) for i in x]
plt.xticks(range(len(x)), x_1)

plt.xlabel('元素含量')
plt.ylabel('正确率')
plt.legend()
plt.savefig(r'picture/第三问图像/灵敏度分析/灵敏度分析（有风化分界线）（改变{}个元素）.png'.format(change_num))

"""
    第四问
"""
import pingouin as pg
sheet_2_predict_normalize_kind_1 = sheet_2_predict_normalize[sheet_2_predict_normalize['种类'] == '高钾']
sheet_2_predict_normalize_kind_2 = sheet_2_predict_normalize[sheet_2_predict_normalize['种类'] == '铅钡']
sheet_2_predict_normalize_kind_1 = sheet_2_predict_normalize_kind_1.iloc[:, :14]
sheet_2_predict_normalize_kind_2 = sheet_2_predict_normalize_kind_2.iloc[:, :14]
sheet_2_predict_normalize_kind_1 = pd.DataFrame(sheet_2_predict_normalize_kind_1).astype(float)
sheet_2_predict_normalize_kind_2 = pd.DataFrame(sheet_2_predict_normalize_kind_2).astype(float)
corr_kind_1 = sheet_2_predict_normalize_kind_1.corr('spearman')
corr_kind_2 = sheet_2_predict_normalize_kind_2.corr('spearman')

writer = pd.ExcelWriter('data/第四问数据/相关系数.xlsx')

corr_kind_1.to_excel(writer, sheet_name='高钾')
corr_kind_2.to_excel(writer, sheet_name='铅钡')
writer.save()

# 偏相关系数
partial_corr_kind_1 = sheet_2_predict_normalize_kind_1.pcorr()
# print(partial_corr_kind_1)
partial_corr_kind_2 = sheet_2_predict_normalize_kind_2.pcorr()

writer = pd.ExcelWriter('data/第四问数据/偏相关系数.xlsx')
partial_corr_kind_1.to_excel(writer, sheet_name='高钾')
```

```
partial_corr_kind_2.to_excel(writer, sheet_name='铅钡')
writer.save()
"""
    热力图绘制
"""


def heat_map(data, title, path):
    # 绘图风格
    style.use('ggplot')
    sns.set_style('whitegrid')
    sns.set_style({"font.sans-serif": ['simhei', 'Droid Sans Fallback']})
    # 设置滑板尺寸
    fig = plt.figure(figsize=(12, 10))

    # 画热力图
    mask = np.zeros_like(data, dtype=bool)
    mask[np.triu_indices_from(mask)] = True

    sns.heatmap(data,
                mask=mask,
                annot=True,
                cmap=sns.diverging_palette(20, 220, n=200),
                center=0)

    plt.title('{}'.format(title), fontsize=15)
    dest = os.path.join(path, title)
    plt.savefig(dest)
    plt.cla()
    plt.close(fig)

heat_map(corr_kind_1, '高钾相关系数', 'picture/第四问热力图')
heat_map(corr_kind_2, '铅钡相关系数', 'picture/第四问热力图')
heat_map(partial_corr_kind_1, '高钾偏相关系数', 'picture/第四问热力图')
heat_map(partial_corr_kind_2, '铅钡偏相关系数', 'picture/第四问热力图')
```

问题二、问题三代码

```
import pprint
import numpy as np
from sklearn.cluster import KMeans, k_means
import pandas as pd
```

```
from sklearn.metrics import silhouette_score
from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

path_weather = r'data/第一问数据/预测数据(经过标准化）(风化后）.xlsx'
path_no_weather = r'data/第一问数据/预测数据(经过标准化）.xlsx'
data = pd.read_excel(path_weather, index_col=0, header=0)

ingredients = list(data.columns[:14])

"""
    数据标准化
"""
# 高钾数据标准化
data_kind_1 = data[data['种类'] == '高钾']
data_kind_2 = data[data['种类'] == '铅钡']
new_data_kind_1 = pd.DataFrame(index=data_kind_1.index, columns=data_kind_1.columns)
for index in data_kind_1.index:
    for col_index in range(len(data_kind_1.columns)):
        column = data_kind_1.columns[col_index]
        if 14 <= col_index <= 18:
            new_data_kind_1.loc[index][column] = data_kind_1.loc[index][column]
        else:
            col_sum = sum(data_kind_1.loc[:][column])
            new_data_kind_1.loc[index][column] = data_kind_1.loc[index][column] / col_sum
data_kind_1 = new_data_kind_1

# 铅钡数据标准化
new_data_kind_2 = pd.DataFrame(index=data_kind_2.index, columns=data_kind_2.columns)
for index in data_kind_2.index:
    for col_index in range(len(data_kind_2.columns)):
        column = data_kind_2.columns[col_index]
        if 14 <= col_index <= 18:
            new_data_kind_2.loc[index][column] = data_kind_2.loc[index][column]
        else:
            col_sum = sum(data_kind_2.loc[:][column])
```

```
            new_data_kind_2.loc[index][column] = data_kind_2.loc[index][column] / col_sum
data_kind_2 = new_data_kind_2

data_kind_1.to_excel('data/第二问模型/亚类划分结果/高钾（风化后）.xlsx')
data_kind_2.to_excel('data/第二问模型/亚类划分结果/铅钡（风化后）.xlsx')

data_list_kind_1 = []
kind_1_index = []
data_list_kind_2 = []
kind_2_index = []

# 高钾数据
for index in data_kind_1.index:
    # 数据输入
    cur_data_list_kind_1 = []
    for ingredient in ingredients:
        cur_data = data_kind_1.loc[index][ingredient]
        cur_data_list_kind_1.append(cur_data)

    kind_1_index.append(index)
    data_list_kind_1.append(cur_data_list_kind_1)
# 铅钡数据
for index in data_kind_2.index:
    # 数据输入
    cur_data_list_kind_2 = []
    for ingredient in ingredients:
        cur_data = data_kind_2.loc[index][ingredient]
        cur_data_list_kind_2.append(cur_data)

    kind_2_index.append(index)
    data_list_kind_2.append(cur_data_list_kind_2)
# 计算轮廓系数
score_kind_1 = []
score_kind_2 = []
for i in range(2, 7):
    model_1 = k_means(data_list_kind_1, n_clusters=i)
    model_2 = k_means(data_list_kind_2, n_clusters=i)
    score_kind_1.append(silhouette_score(data_list_kind_1, model_1[1]))
    score_kind_2.append(silhouette_score(data_list_kind_2, model_2[1]))
```

```
82.
83.
84.
85.
86.plt.figure()
87.plt.subplot(1, 2, 1)
88.plt.plot(range(2, 7), score_kind_1, 'r*-')
89.plt.xlabel('cluster')
90.plt.ylabel('轮廓系数')
91.plt.title('轮廓系数确定的最佳 k 值(高钾)')
92.
93.# plt.figure()
94.plt.subplot(1, 2, 2)
95.plt.plot(range(2, 7), score_kind_2, 'r*-')
96.plt.xlabel('cluster')
97.plt.ylabel('轮廓系数')
98.plt.title('轮廓系数确定的最佳 k 值(铅钡)')
99.plt.subplots_adjust(wspace=0.5)
100.  plt.savefig(r'picture/第二问图像/轮廓系数/最佳 k 值(高钾与铅钡)（风化后）.png')
101.
102.  # 假如我要构造一个聚类数为 3 的聚类器
103.  estimator_kind_1 = KMeans(n_clusters=3)  # 构造聚类器
104.  estimator_kind_1.fit(data_list_kind_1)  # 聚类
105.  label_pred = estimator_kind_1.labels_  # 获取聚类标签
106.  centroids = estimator_kind_1.cluster_centers_  # 获取聚类中心
107.  inertia = estimator_kind_1.inertia_  # 获取聚类准则的总和
108.  writer = pd.ExcelWriter('data/第一问数据/亚类划分结果/划分结果（归一化）（风化后）.xlsx')
109.
110.  divide_dict_kind_1 = {}
111.  for num in range(len(kind_1_index)):
112.      index = kind_1_index[num]
113.      divide_dict_kind_1[index] = label_pred[num]
114.
115.  divide_dict_df = pd.DataFrame(divide_dict_kind_1.values(), index=divide_dict_kind_1.keys(), columns=['分类'])
116.  divide_dict_df.to_excel(writer, sheet_name='高钾分类')
117.
118.  estimator_kind_2 = KMeans(n_clusters=2)  # 构造聚类器
119.  estimator_kind_2.fit(data_list_kind_2)  # 聚类
120.  label_pred = estimator_kind_2.labels_  # 获取聚类标签
121.  centroids = estimator_kind_2.cluster_centers_  # 获取聚类中心
122.  inertia = estimator_kind_2.inertia_  # 获取聚类准则的总和
```

```
123.
124.  divide_dict_kind_2 = {}
125.  for num in range(len(kind_2_index)):
126.      index = kind_2_index[num]
127.      divide_dict_kind_2[index] = label_pred[num]
128.
129.  divide_dict_df = pd.DataFrame(divide_dict_kind_2.values(), index
    =divide_dict_kind_2.keys(), columns=['分类'])
130.  divide_dict_df.to_excel(writer, sheet_name='铅钡分类')
131.  writer.save()
132.
133.
134.  """
135.      灵敏度分析
136.  """
137.  ingredients_list = []
138.  ingredients = list(data.columns[:14])
139.  for num in range(14):
140.      cur_list = ingredients.copy()
141.      cur_list.pop(num)
142.      ingredients_list.append(cur_list)
143.
144.  scores_kind_1 = []
145.  scores_kind_2 = []
146.  for ingredients in ingredients_list:
147.
148.      data_list_kind_1 = []
149.      kind_1_index = []
150.      data_list_kind_2 = []
151.      kind_2_index = []
152.      for index in data.index:
153.          cur_data_list_kind_1 = []
154.          cur_data_list_kind_2 = []
155.          data_kind = data.loc[index]['种类']
156.          if data_kind == '高钾':
157.              kind_1_index.append(index)
158.              for ingredient in ingredients:
159.                  cur_data = data.loc[index][ingredient]
160.                  cur_data_list_kind_1.append(cur_data)
161.              data_list_kind_1.append(cur_data_list_kind_1)
162.          else:
163.              kind_2_index.append(index)
164.              for ingredient in ingredients:
165.                  cur_data = data.loc[index][ingredient]
```

```
                cur_data_list_kind_2.append(cur_data)
            data_list_kind_2.append(cur_data_list_kind_2)

    # 计算轮廓系数
    score_kind_1 = []
    score_kind_2 = []
    for i in range(2, 7):
        model_1 = k_means(data_list_kind_1, n_clusters=i)
        model_2 = k_means(data_list_kind_2, n_clusters=i)
        # model = cluster.SpectralClustering(x_1, n_clusters=i + 2)
        score_kind_1.append(silhouette_score(data_list_kind_1, model_1[1]))
        score_kind_2.append(silhouette_score(data_list_kind_2, model_2[1]))

    scores_kind_1.append(max(score_kind_1))
    scores_kind_2.append(max(score_kind_2))

x_label = ['SiO2', 'Na2O', 'K2O', 'CaO', 'MgO', 'Al2O3', 'Fe2O3', 'CuO',
           'PbO', 'BaO', 'P2O5', 'SrO', 'SnO2', 'SO2']
plt.figure()
plt.title('元素与轮廓系数的灵敏度')
plt.xticks(range(len(ingredients) + 1), x_label, rotation=30)
plt.plot(scores_kind_1, label='高钾', marker='*')
plt.plot(scores_kind_2, label='铅钡', marker='*')
plt.legend()
plt.savefig(r'picture/第二问图像/灵敏度分析/分类的灵敏度分析(风化后）.png')
```