古代玻璃制品的成分分析与鉴别

摘要

本文依据提供古代玻璃类型、表面风化与化学成分等数据对玻璃风化成分含量变化规律进行探索，对相关化学含量进行相关性分析。利用各类聚类方法、统计方法进行数据计算与预测。

针对问题一，本文首先进行数据预处理，对表单 1、2 编码后进行数据清洗，利用**最近邻填补法**填补空缺值并剔除异常值，同时进行数据整合、标准化。对于第一小问，分析文物样品表面风化与颜色玻璃类型以及纹饰间的关系，本文在对数据进行初步比例筛选后，对三组关系“表面风化-类型”，“表面风化-纹饰”，“表面风化-颜色”建立 **Pearson 卡方检验**，进行 Yates 校正判断，针对期望频率不同情况的数据进行了检验。对于第二小问，分析有无风化样品化学成分含量的统计规律。本文分别进行描述性统计量、正态性检验计算，利用偏度系数、峰度系数等进行数据分布判断。同时利用 Q-Q 图反应各成分数据分布类型。对于第三小问，进行风化前化学含量成分预测，本文通过利用**层次分析法**将已知数据再聚类划分为四个时间点，构建时间序列，通过时间序列，建立回归方程进行成分含量预测。

针对问题二，第一小问，分析玻璃类型的分类规律，根据已知数据，本文利用有监督学习算法进行分类，利用**决策树**分别对未风化点数据与风化点数据进行聚类分析。利用召回率、精确率、F1 分数三个评价指标进行模型评估，得到评估指标均为 1，体现了模型性能的良好。第二小问，需对每个类别进行亚类划分，本文采用了 **kmeans 聚类算法**，进行数据聚类。利用**肘部法**进行 k 值选择。同时，利用 **kmeans++ 算法**进行初始聚类中心优化，最终得到聚类结果，将玻璃分为 11 个亚类。利用 CH 指数、轮廓系数、DBI 指数、DVI 指数对模型进行评估，说明了模型敏感性良好。

针对问题三，对表单中数据分析鉴别其玻璃类型，选择利用 XGboost 算法与决策树算法预测进行对比，分别进行玻璃类型与亚类预测。对比分析对玻璃类型预测模型敏感性良好。对玻璃亚类分析方面有一定的缺陷，后续进行模型优化。

针对问题四，分析不同类玻璃类型的化学成分间相关性。基于第一问统计量计算，本文选择 **Spearman 相关系数**对四个大类玻璃进行相关性分析并绘制热力图，得到结果显示高钾无风化与有风化成分含量多数具有较强相关性，相关系数分布集中。铅钡玻璃类型中相关系数较分散。后对图表结果进行实际分析，反应玻璃化学成分的稳定性及反应性等。

关键词： **决策树　Kmeans 聚类　肘部法　卡方检验　Spearman 相关**

一、问题背景与重述

1.1 问题背景

玻璃是古代中西方文化交流贸易往来的宝贵物证。其主要原料为石英砂，主要化学成分为二氧化硅。由于纯石英砂的熔点较高，在炼制时需要添加助溶剂。古代常用有草木灰，天然泡碱，硝石和铅矿石等，同时添加石灰石作为稳定剂。添加助溶剂不同，玻璃化学成分也不同。

古代玻璃极易受埋藏环境的影响而风化，在风化过程中，内部元素与环境元素进行大量交换，导致其成分比例发生变化，影响对玻璃类别的正确判断。

1.2 问题重述

考古工作者依据已知玻璃制品的相关成分比例数据及其他检测手段将其划分为高钾玻璃和铅钡玻璃两种类型。附件表单 1 提供文物分类信息；附件表单 2 提供样品采样点主要成分比例。要求根据附表数据，建立数学模型解决以下问题：

问题一：分析玻璃文物表面风化与玻璃类型、纹饰和颜色的关系；结合玻璃类型分析，分析文物样品表面有无风化化学成分含量的统计规律，根据风化点检测数据，预测其风化前的化学成分含量。

问题二：要求基于附件数据分析高钾玻璃和铅钡玻璃的分类规律，并对每个类别选择合适的化学成分进行亚类划分。同时，需要给出具体的划分方法、划分结果，并对分类结果的合理性和敏感性进行分析。

问题三：对附件表单 3 中未知类别玻璃文物的化学成分进行分析，依据问题二所建模型，鉴别其所属类别并对分类结果的敏感性进行分析。

问题四：针对不同类别的玻璃文物样品，分析其化学成分之间的关联关系，并比较不同类别之间的化学成分，关联关系的差异性。

二、问题分析

2.1 问题一的分析

针对问题一，第一小问，分析玻璃文物表面风化与玻璃类型、纹饰和颜色的关系，首先需要对附件数据进行预处理，对表单一进行编码后利用最近邻插补法进行缺失值补充，将分类变量进行量化处理后进行 Person 卡方检验。第二小问，分析文物样品有无风化化学成分含量统计规律，首先对表单 2 进行预处理，剔除异常值，补充空缺值，对表单 1、2 进行特征集成，计算各成分统计量并进行相关性分析，后可根据统计规律对文物采样点重新聚类，划分为四个时间点，构建时间序列，利用时间序列建立回归方程进行风化前化学成分预测。

2.2 问题二的分析

针对问题二，第一小问，基于附件表单 2 高钾玻璃和铅钡玻璃成分含量进行玻璃分类规律总结，首先利用决策树对玻璃分类规律进行观测，依据表单 3 成分及是否风化共 15 个变量对风化与未风化两种类型玻璃进行分析，并对结果利用评估参数进行评价。第二小问，对玻璃进行亚类划分，依据已知数据可将数据集划分为四类，对铅钡玻璃有风化，铅钡玻璃无风化，高钾玻璃有风化，高钾玻璃无风化，基于此利用 K-means 聚类进行后续划分，选择所有成分含量及表面风化情况、玻璃类型分别进行 14 特征聚类，通过肘部法则进行聚类数确定，根据划分出的种类成分组成，对所有亚类进行命名，确定最终的亚类划分。将划分结果利用无监督算法评估指标进行敏感性评价。

2.3 问题三的分析

针对问题三，要求对附件表单 3 中未知类别玻璃文物的化学成分进行分析，可利用 XGboost 算法与决策树算法预测进行对比，分别预测玻璃类型与玻璃亚类，对比分析敏感性。

2.4 问题四的分析

针对问题四，要求分析不同玻璃类别的化学成分间的相关性。依据问题一统计量分析基础，选择 Spearman 相关系数进行铅钡风化，铅钡无风化，高钾风化，高钾无风化四类相关性分析，绘制热力图分析各成分含量见的关联。

三、模型假设

1. 假设玻璃文物风化过程中没有剧烈环境变化。
2. 假设同一文物风化部位与未风化部位无互相影响。
3. 假设玻璃文物形状均相同且对风化结果及含量变化无影响。

四、符号说明

符号	说明
d_{ij}	两文物数据间距离
x_{ik}、x_{jk}	第 i/j 个文物数据向量 $i \neq j$
$O_{i,j}$	数据检验观测值（实际值）标准化因子
$E_{i,j}$	数据检测理论值（期望值）
$\bar{y_\tau}$	化学成分含量均值
β_s	偏度系数
β_k	峰度系数
CV	变异系数
v_n	n 阶中心矩（$j = 1...7$）
$WCSS$	簇内平方和（$j = 1...7$）
C_i	第 i 个簇
x	簇 C_i 中的一个样本点
μ_i	簇 C_i 的聚类中心
i	样本（行号）
j	成分含量（列号）
s_i	标准化因子
$C^s_{i,j}$	标准化后成分含量
ρ	Spearman 相关系数

五、数据预处理

概览题设所给出的附件信息，可以发现表格信息有大量缺失值。对其进行整理关联，数据清洗，结构化，特征集成等初步工作能有效帮助后续数据处理，分析与预测。

5.1 数据结构化

在数据处理与分析的领域中，对表单变量（尤其是包含分类或名义变量的数据）进行编码量化是一项至关重要的预处理步骤。这一过程旨在将原始数据中的非数值

型信息转换为数值型数据，以便于后续的计算、建模及统计分析。本文采用 Label Encoding（标签映射）方法作为主要编码方式，为每个类别分配一个从 0 开始的连续整数。具体编码如表 1：

表 1 表单一变量编码

编码	0	1	2	3	4	5	6	7
纹饰	A	B	C	——	——	——	——	——
类型	铅钡	高钾	——	——	——	——	——	——
颜色	浅绿	浅蓝	深绿	深蓝	紫	绿	蓝绿	黑
表面风化	无风化	风化	——	——	——	——	——	——

5.2 数据清洗

对表单 1、2 空缺值与异常值进行处理

- 空缺值处理

  表单 1 为文物编号，纹饰，玻璃类型，颜色与表面风化情况。缺失值为文物编号 19、40、48、58 的玻璃样品颜色，选择**最近邻插补法 Nearest Neighbor** 进行数据补全。假设表面风化、玻璃类型，纹饰，颜色之间具有一定相关关系, 绘制散点图，采用欧式距离计算，将距离颜色缺失值最近的点颜色值作为填补值。将每个文物数据转化为一个 4 维向量，进行距离计算：

$$d_{ij} = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2} \tag{1}$$

  最终对数据完成颜色填补。

  表单 2 成分含量数据均为数值数据，空缺值采用 0 补齐。

- 异常值剔除

  题目要求将成分比例累加和介于 85% 到 105% 之间的数据视为有效数据。研究表单 2 发现，文物采样点 15 与 17 成分比例累加和均低于 85% 属于无效数据, 进行剔除。

5.3 数据集成

将表单 1 文物数据对应到表单 2，为后续计算提供全面的数据和统一的数据视图。同时对数据进行标准化处理。计算每个样本的标准化因子，并将每个样本的化学成分含量乘以标准化因子，使得所有样本的各成分的总和都标准化为 100%。

对于每个样本 i，$C_{i,j}$ 代表该样本成分含量值则：

标准化因子：

$$s_i = \frac{100}{C_{i,16}} \tag{2}$$

标准化后的化学成分含量：

$$C^s_{i,j} = C_{i,j} \times s_i \quad \text{其中} \quad j = 2, 3, ..., 15 \tag{3}$$

利用标准化后数据进行计算处理。

六、模型的建立与求解

6.1 问题一模型的建立与求解

6.1.1 问题一第一小问

卡方检验（Chi-square test）是一种统计检验方法，主要用于比较观察频数与期望频数之间的差异是否显著，从而判断两个或多个分类变量之间是否存在关联或独立性。由于表单 1 中展示玻璃纹饰、类型、颜色、表面风化均为分类变量，因此，本文设计了“表面风化—类型”、“表面风化—纹饰”、“表面分化—颜色”三组 Pearson 卡方检验用于分析指标间的相关性。

数据校正判断：

Pearson 卡方检验适用于两个分类变量的独立性检验，以及样本数据与理论数据的拟合优度检验。其特点是简单直观。一般来说，最小的期望频数不应小于 5。如果期望频数太小，卡方检验的结果可能不准确，小样本数据更容易受到随机波动的影响。当期望频数小于 5 时，需要使用 Yates 校正来提高检验的准确性。绘制列联表，判断可知“表面风化-纹饰”、“表面风化-颜色”需进行校正.。

表 2 表面风化—类型列联表

表面风化 \ 类型	0	1	合计
0	12（16.55）	12（7.45）	24
1	28（23.45）	6（10.55）	34
合计	40	18	58

表 3 表面风化-纹饰列联表

表面风化 \ 纹饰	0	1	2	合计
0	11（9.10）	0（2.48）	13（12.41）	24
1	11（12.90）	6（3.52）	17（17.59）	34
合计	22	6	30	58

表 4 表面风化-颜色列联表

表面风化 \ 颜色	0	1	2	3	4	5	6	7	合计
0	2（1.24）	8（9.10）	3（2.90）	2（0.83）	2（2.48）	1（0.41）	6（6.20）	0（0.82）	24
1	1（1.76）	14（12.90）	4（4.10）	0（1.17）	4（3.52）	0（0.59）	9（8.79）	2（1.17）	34
合计	3	22	7	2	6	1	15	2	58

Person 卡方检验：

- **表面风化—类型检验**

  提出假设：

  原假设 H_0 ：表面风化与玻璃类型相互独立不相关备择假设 H_1：表面风化与玻璃类型有关联

  计算卡方检验统计量，无需进行校正：

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \tag{4}$$

计算结果如下表：

表 5 表面风化-类型检验结果

卡方值	6.88
P 值	0.01
自由度	1

由结果可知，P 值 =0.01<0.05, 拒绝原假设, 接受备择假设，表面风化与玻璃类型有关联。

- **表面风化—纹饰检验**

表面风化-纹饰检验需对数据进行校正，计算公式为：

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c} \frac{(|O_{i,j} - E_{i,j}| - \frac{1}{2})^2}{E_{i,j}} \tag{5}$$

表 6 表面风化-纹饰检验结果

卡方值	4.96
P 值	0.08
自由度	2

由结果，P 值 =0.08>0.05, 故不能拒绝原假设，没有理由证明表面风化与纹饰有关。

- **表面风化—颜色检验**

该检验步骤与上相同，均进行有校正检验，检验结果为：由结果，P 值 =0.44>0.05,

表 7 表面风化-纹饰检验结果

卡方值	6.86
P 值	0.44
自由度	7

故不能拒绝原假设，没有理由证明表面风化与颜色有关。

综上所述玻璃文物表面风化主要与玻璃类型有关，与纹饰，颜色不相关。

6.1.2 问题一第二小问

- **描述性统计量**

题目要求，结合玻璃类型研究有无风化样品化学成分含量的统计规律。依据表格信息, 初步将所有样品分为四大类, 分别讨论四类玻璃文物化学成分含量统计规律。计算平均值、中位数、标准差、偏度系数、峰度系数、变异系数描述统计量。

（1）基本统计量：

均值：反映了数据集中所有数值的平均水平，是度量数据集中趋势最常用的指标。

$$\bar{y_\tau} = \frac{\sum_{i=1}^{n} y_\tau i}{n} \tag{6}$$

方差：衡量了数据与其平均值的偏离程度，即数据的离散程度。

$$\sigma_\tau^2 = \frac{\sum_{i=1}^{n}(y_{\tau i} - \bar{y_\tau})^2}{n-1} \tag{7}$$

标准差：标准差是方差的平方根，它与原始数据具有相同的量纲，因此更容易理解和解释。标准差同样衡量了数据的离散程度。

中位数：中位数不受极端值的影响，能够较好地反映数据的中心位置。（2）变异系数:

标准差与平均数的比值，用于比较不同数据集的离散程度。由于它是无量纲的，因此可以用来比较具有不同单位或量级的变量的变异程度。

$$CV = \frac{\sigma_\tau}{\bar{y_\tau}} \times 100\% \tag{8}$$

（3）偏度系数：

偏度系数用于描述数据分布形态的偏斜方向和程度。若偏度大于零，则认为处于均值右边的数据居多。若偏度小于零，则认为处于均值左边的数据偏多，当偏度接近于零时，则认为样本数据是对称分布的。

$$\beta_s = \frac{v_3}{v_2^{\frac{3}{2}}} = \frac{E(y_{\tau i} - \bar{y_\tau})^3}{{\sigma_\tau}^{\frac{3}{2}}} \tag{9}$$

（4）峰度系数:

峰度系数用于描述数据分布形态的尖峭或扁平程度，与正态分布相比较。若 β_k 大于零，则说明该分布相比于正态分布顶部更加尖锐或尾部更加粗，若 β_k 小于零，则说明该分布相比于正态分布顶部更加平坦或者尾部更加细。当 β_k 等于零时表示该分布与标准正态分布顶部尖锐程度和尾部粗细相当。计算公式如下：

$$\beta_k = \frac{v_4}{v_2^2} = \frac{E(y_{\tau i} - \bar{y_\tau})^4}{{\sigma_\tau}^2} - 3 \tag{10}$$

通过 Python 计算统计量，在此处展示铅钡无风化计算结果，其他三类统计量计算见附录：

表 8 铅钡无风化描述性统计量表

	平均值	中位数	标准差	偏度系数	峰度系数	变异系数
二氧化硅	54.69	55.98	14.37	-0.22	1.76	0.15
氧化钠	0.79	0.00	1.58	1.63	4.18	-
氧化钾	0.27	0.15	0.41	2.01	6.07	0.76
氧化钙	1.25	0.85	1.48	1.58	4.02	1.70
氧化镁	0.50	0.52	0.55	0.66	2.47	2.80
氧化铝	3.30	3.15	1.50	0.43	2.08	0.77
氧化铁	0.96	0.00	1.47	1.41	3.96	5.55
氧化铜	1.62	0.54	2.58	1.86	5.19	1.65
氧化铅	24.12	22.92	9.21	0.41	2.04	-
氧化钡	10.88	10.20	7.48	1.36	3.73	-
五氧化二磷	0.97	0.20	1.77	2.60	8.75	6.28
氧化锶	0.30	0.30	0.32	0.78	2.57	-
氧化锡	0.07	0.00	0.16	1.93	4.74	-
二氧化硫	0.28	0.00	1.02	3.18	11.08	-

通过计算结果可知。铅钡玻璃未风化类别，只有一组成分指标，偏度系数小于 0，其余偏度系数均大于 0，说明落在均值右侧的数据均偏多。其中二氧化硅偏度系数接

近于 0，说明二氧化硅数据接近对称分布。在该类别中，所有成分峰度系数均大于 0，表明数据分布相比于正态分布顶部更加尖锐或尾部更加粗。

- **正态性检验**

  为了深入探究不同玻璃类型及其风化程度在特定成分含量上的分布特性，确保数据分析的科学性与准确性，进行正态性检验。这一步骤旨在直观验证数据是否服从正态分布。具体而言，针对每种玻璃类型及其风化状态，利用 14 个特征变量进行了 Q-Q 图绘制。Q-Q 图通过将样本数据的分位数与相应正态分布的分位数进行比较，能够直观地展示数据分布与正态分布的拟合程度。若数据点紧密围绕在参考线（即斜率为 1 的直线）周围，则表明数据接近正态分布；反之，若数据点显著偏离此线，则提示数据可能存在非正态性。铅钡未风化 Q-Q 图 1 如图所示（其余三组见附录）：

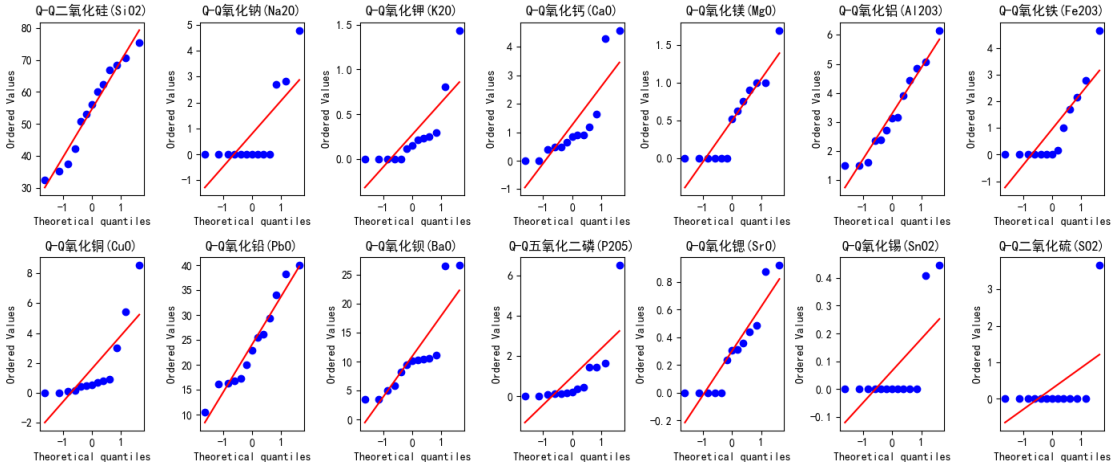

图 1 铅钡未风化玻璃 Q-Q 图

观察铅钡未风化玻璃 QQ 图，可以直观发现各成分分布近似围绕参考线，接近正态分布，其中 Na_2O,SnO_2,SO_2 数据点由于原始缺失值较多，由于进行补 0 处理，分布较分散，为非正态分布。铅钡风化玻璃，高钾无风化均在 Na_2O,SnO_2,SO_2，SrO,BaO 存在缺失值，高钾有风化玻璃存在较多缺失值。综上据图可分析，表单中详细给出的成分含量数据均满足正态分布，在有较多缺失值的成分中存在非正态分布。同时根据 QQ 图数据分布，可以进一步验证偏态系数计算正确性，如果数据曲线向左下方弯曲，说明样本数据存在正偏态（右偏），即数据的高值部分较为集中，此时偏态系数大于 0。如果曲线向右上方弯曲，说明样本数据存在负偏态（左偏），即数据的低值部分较为集中，此时偏态系数小于 0。通过结果对比，检验偏态系数正确。

6.1.3 问题一第三小问

本题要求根据统计规律分析，对风化玻璃风化前各成分含量进行预测。采用层次聚类方法对表单 1 数据进行聚类，将风化程度细分为“无风化”，“轻度风化”，“中度风化”，“重度风化”，后依据分类建立时间序列，进行拟合回归，对风化前成分含量进行预测。

- **数据处理** 表单二中的数据详细记录了文物采样点的风化属性及成分情况，其中每个采样点均为对应编号文物表面随机选取的，确保了其风化特性与附件表单 1 中相应文物的整体风化状况一致。此外区分文物造型上不同的两个部位，这两处可能因位置差异而具有不同的成分与含量。未风化点取自文物表面未风化区域，

用于对比研究；同时，严重风化点则是直接从文物的风化层中采集。因此在进行预测时，无需关注采样点是否来自同意文物，同时对各成分无需进行同一建模，单独对特定化学成分进行处理，通过找到化学成分在风化点前后的变换关系，进行数据预测。

- **层次聚类**

选择适用于成分性数据层次聚类方法进行分析，根据成分含量将风化情况细分为四个风化时间即“未风化”,“轻度风化”,“中度风化”,“重度风化”。首先对铅钡和高钾玻璃进行二分类，划分出有风化和无风化数据点，在进一步进行四分类。利用 Gower 距离矩阵及逆行计算，聚类结果如下：

表 9 铅钡玻璃二分类结果

类别	组别
风化数据	08、08 严重风化点、20、24、26、26 严重风化点
未风化数据	02 、11、19、23 未风化点、25 未风化点、28 未风化点、 29 未风化点、30 部位 1、30 部位 2、31、32、33、34、35、36、37、 38、39、40、41、42 未风化点 1、42 未风化点 2、43 部位 1、43 部位 2、 44 未风化点、45、46、47、48、49、49 未风化点、50、50 未风化点、 51 部位 1、51 部位 2、52、53 未风化点、54、54 严重风化点、55、56、57、58

表 10 高钾玻璃二分类结果

类别	组别
风化数据	01、04、05、13、14、16
未风化数据	03 部位 1、03 部位 2、06 部位 1、06 部位 2、07、09、10、12、18、21、22、27

- **时间序列建立**

对二分类后结果进一步聚类，分别对高钾玻璃和铅钡玻璃进行分析，分为 4 个时间点，聚类结果如下。

表 11 铅钡玻璃时间序列建立

时间点	组别
未风化	11、19、25 未风化点、32、34、35、36、37、38、39、40、50、50 未风化点、50、52、54、55、56、57、58
轻度风化	02、30 部位 1、30 部位 2、41、43 部位 1、43 部位 2、49、9 未风化点、51 部位 1、51 部位 2、54 严重风化点
中度风化	23 未风化点、28 未风化点、29 未风化点、31、33、42 未风化点 1、42 未风化点 2、44 未风化点、45、46、47、48、53 未风化点
重度风化	08、08 严重风化点、20、24、26、26 严重风化点

- **数据聚合**

通过以上时间序列建立回归方程，以上分析同一化学成分随时间变化对应多组数据，存在重复观测点及测量误差，因此使用平均值来聚合数据。

表 12 高钾玻璃时间序列建立

时间点	组别
未风化	03 部位 2、06 部位 1、06 部位 2、21
轻度风化	13、14、16
中度风化	03 部位 1、07、09、10、12、18、22、27
重度风化	01、04、05

表 13 铅钡玻璃平均值聚合

成分	未风化	轻度风化	中度风化	重度风化
二氧化硅 (SiO_2)	38.296490	28.775970	60.497446	20.496355
氧化钠 (Na_2O)	0.534865	0.000000	2.713149	0.000000
氧化钾 (K_2O)	0.090954	0.296870	0.194180	0.200586
氧化钙 (CaO)	1.775844	4.084818	1.190500	1.610176
氧化镁 (MgO)	0.267830	1.297280	1.017576	0.000000
氧化铝 (Al_2O_3)	2.266608	4.345993	6.254364	2.021222
氧化铁 (Fe_2O_3)	0.363782	1.529219	0.592699	0.284659
氧化铜 (CuO)	1.408726	1.136754	1.043451	6.963417
氧化铅 (PbO)	41.597336	45.462269	18.045867	26.881417
氧化钡 (BaO)	9.685705	5.682651	7.681583	30.569406
五氧化二磷 (P_2O_5)	3.212157	6.806676	0.364355	4.519931
氧化锶 (SrO)	0.307033	0.459028	0.285290	0.483661
氧化锡 (SnO_2)	0.000000	0.122472	0.119540	0.000000
二氧化硫 (SO_2)	0.192670	0.000000	0.000000	5.969170

表 14 高钾玻璃平均值聚合

成分	未风化	轻度风化	中度风化	重度风化
二氧化硅 (SiO_2)	67.285357	63.154168	91.842848	67.805518
氧化钠 (Na_2O)	0.000000	2.820758	0.000000	0.000000
氧化钾 (K_2O)	6.932678	13.308705	2.267981	10.549069
氧化钙 (CaO)	4.047684	8.527212	0.906185	7.167524
氧化镁 (MgO)	1.528205	0.398356	0.344971	1.449765
氧化铝 (Al_2O_3)	8.321242	7.293276	2.352915	6.167195
氧化铁 (Fe_2O_3)	3.279262	1.288807	0.199426	2.213947
氧化铜 (CuO)	3.304979	2.127227	1.273824	3.207474
氧化铅 (PbO)	0.749321	0.582714	0.031250	0.000000
氧化钡 (BaO)	1.817110	0.000000	0.000000	0.000000
五氧化二磷 (P_2O_5)	2.650726	0.485229	0.468260	0.998348
氧化锶 (SrO)	0.083435	0.013549	0.008997	0.020723
氧化锡 (SnO_2)	0.000000	0.000000	0.303342	0.000000
二氧化硫 (SO_2)	0.000000	0.000000	0.000000	0.420437

- **回归方程建立**

表 15 铅钡玻璃回归方程

名称	回归方程	R^2 值
二氧化硅 (SiO_2)	$y = 40.0770057919906 + 27.9076 \times t + -4.9765 * t^2$	0.2835496334
氧化钠 (Na_2O)	$y = -2.115568772734332 + 3.2439 \times t + -0.7052 * t^2$	0.4000000000
氧化钾 (K_2O)	$y = 10.693822350954399 + -2.4005 \times t + 0.4763 * t^2$	0.0132925791
氧化钙 (CaO)	$y = 6.95479213453069 + -2.0534 \times t + 0.4455 * t^2$	0.0272282025
氧化镁 (MgO)	$y = 3.7958044919966363 + -2.8222 \times t + 0.5587 * t^2$	0.9997335291
氧化铝 (Al_2O_3)	$y = 14.93708973027865 + -7.1931 \times t + 1.2106 * t^2$	0.6064487974
氧化铁 (Fe_2O_3)	$y = 7.822913568375175 + -5.4348 \times t + 1.0012 * t^2$	0.9530782013
氧化铜 (CuO)	$y = 6.6541077548712515 + -4.0038 \times t + 0.7779 * t^2$	0.8912751069
氧化铅 (PbO)	$y = 1.2098747713859042 + -0.4491 \times t + 0.0338 * t^2$	0.9063564955
氧化钡 (BaO)	$y = 4.088497268640949 + -2.8165 \times t + 0.4543 * t^2$	0.9333333333
五氧化二磷 (P_2O_5)	$y = 5.763646550102782 + -3.8669 \times t + 0.6739 * t^2$	0.9596978809
氧化锶 (SrO)	$y = 0.1818639464725556 + -0.1213 \times t + 0.0204 * t^2$	0.9669582009
氧化锡 (SnO_2)	$y = -0.379177377892032 + 0.4095 \times t + -0.0758 * t^2$	0.4000000000
二氧化硫 (SO_2)	$y = 0.3153277910267387 + -0.3994 \times t + 0.1051 * t^2$	0.9333333333

表 16 高钾玻璃回归方程

名称	回归方程	R^2 值
二氧化硅 (SiO_2)	$y = 40.0770057919906 + 27.9076 \times t + -4.9765 * t^2$	0.2835496334
氧化钠 (Na_2O)	$y = -2.115568772734332 + 3.2439 \times t + -0.7052 * t^2$	0.4000000000
氧化钾 (K_2O)	$y = 10.693822350954399 + -2.4005 \times t + 0.4763 * t^2$	0.0132925791
氧化钙 (CaO)	$y = 6.95479213453069 + -2.0534 \times t + 0.4455 * t^2$	0.0272282025
氧化镁 (MgO)	$y = 3.7958044919966363 + -2.8222 \times t + 0.5587 * t^2$	0.9997335291
氧化铝 (Al_2O_3)	$y = 14.93708973027865 + -7.1931 \times t + 1.2106 * t^2$	0.6064487974
氧化铁 (Fe_2O_3)	$y = 7.822913568375175 + -5.4348 \times t + 1.0012 * t^2$	0.9530782013
氧化铜 (CuO)	$y = 6.6541077548712515 + -4.0038 \times t + 0.7779 * t^2$	0.8912751069
氧化铅 (PbO)	$y = 1.2098747713859042 + -0.4491 \times t + 0.0338 * t^2$	0.9063564955
氧化钡 (BaO)	$y = 4.088497268640949 + -2.8165 \times t + 0.4543 * t^2$	0.9333333333
五氧化二磷 (P_2O_5)	$y = 5.763646550102782 + -3.8669 \times t + 0.6739 * t^2$	0.9596978809
氧化锶 (SrO)	$y = 0.1818639464725556 + -0.1213 \times t + 0.0204 * t^2$	0.9669582009
氧化锡 (SnO_2)	$y = -0.379177377892032 + 0.4095 \times t + -0.0758 * t^2$	0.4000000000
二氧化硫 (SO_2)	$y = 0.3153277910267387 + -0.3994 \times t + 0.1051 * t^2$	0.9333333333

通过表识 15 和 16, 可知一些化学成分拟合优度 R^2 方相对较低，可以认为该化学成分不受风化影响。因此，在后续预测过程中，仅对拟合优度 R^2 较高的化学成分进行预测。

- **预测方法** 基于平均值回归模型调整预测文物风化前化学成分含量，通过上述回归方程建立，预测文物化学成分随时间的变化趋势。为了更准确地预测特定文物风化前的化学成分含量，需要对基础回归模型进行平移调整。平移的距离基于文物在特定风化阶段的化学成分含量与平均回归方程中对应成分含量的差异。根据文物在风化时间点的聚类结果，分别计算各分类中文物化学成分的平均值。将这些分类平均值与平均回归方程中相应时间点的预测值进行比较，计算出每个分

类的平移距离。考虑到异常值的影响，可能采用更稳健的统计量（如中位数或截尾均值）来计算平移距离，以提高预测的准确性。由于数据已经过标准化处理，因此在应用平移距离时，需要将平移距离也除以相应的标准化因子，以确保平移操作在标准化后的数据空间内正确执行。将调整后的平移距离应用于基础回归模型，得到针对特定文物风化前化学成分含量的预测值。这一预测值通过平移基础回归模型的预测曲线来实现，以反映文物在不同风化阶段下的实际化学成分变化。

预测结果如表所示：

表 17 铅钡玻璃预测

成分	二氧化硅 (SiO2)	氧化钠 (Na2O)	氧化钾 (K2O)	氧化钙 (CaO)	氧化镁 (MgO)	...	氧化锡 (SnO2)	二氧化硫 (SO2)
02	36.28	0.00	1.05	2.34	2.42	...	0.00	0.27
08	20.14	0.00	0.00	1.48	2.73	...	0.00	1.496578
08 严重风化点	4.61	0.00	0.00	3.19	2.69	...	0.00	3.81
20	37.36	0.00	0.71	0.00	2.428	...	0.00	0.58
26 严重风化点	3.72	0.00	0.40	3.01	2.73	...	0.000	3.96
30 部位 1	34.34	0.00	1.41	4.49	2.27	...	0.40	0.27
30 部位 2	36.93	0.00	0.00	4.24	1.87	...	0.44	0.27
31	65.91	0.00	0.000	1.60	2.24	...	0.00	0.27
33	75.51	0.00	0.15	0.64	2.33	...	0.00	0.272042
41	18.46	0.00	0.44	4.96	2.92	...	0.00	0.255032
43 部位 1	12.41	0.00	0.00	5.24	2.16	...	0.00	0.26
43 部位 2	21.70	0.00	0.00	6.40	2.23	...	0.00	0.26
...	...	...	...	...	...	...	...	..
45	61.28	2.66	0.11	0.84	2.13	...	0.00	0.267633
46	55.21	0.00	0.25	0.00	2.65	...	0.00	0.27
47	51.54	4.66	0.29	0.87	1.99	...	0.00	0.26
48	53.33	0.80	0.32	2.82	2.60	...	1.31	0.27
49	28.79	0.00	0.00	4.58	2.49	...	0.00	0.26
51 部位 1	24.61	0.00	0.00	3.58	2.35	...	0.47	0.259441
54 严重风化点	17.11	0.00	0.000	0.000	2.33	...	0.00	0.26

表 18 高钾玻璃预测

成分	二氧化硅 (SiO2)	氧化钠 (Na2O)	氧化钾 (K2O)	氧化钙 (CaO)	氧化镁 (MgO)	...	氧化锡 (SnO2)	二氧化硫 (SO2)
01	69.33	0.00	9.99	6.32	2.22	...	0.00	0.03
03 部位 1	87.05	0.00	5.19	2.01	2.13	...	0.00	0.27
04	65.88	0.00	9.67	7.12	1.88	...	0.00	0.17
05	61.58	0.00	10.95	7.35	2.08	...	0.00	0.31
07	92.63	0.00	0.00	1.07	2.12	...	0.00	0.27
09	95.02	0.00	0.59	0.62	2.13	...	0.00	0.27

成分	二氧化硅 (SiO2)	氧化钠 (Na2O)	氧化钾 (K2O)	氧化钙 (CaO)	氧化镁 (MgO)	...	氧化锡 (SnO2)	二氧化硫 (SO2)
10	96.77	0.00	0.92	0.21	2.13	...	0.00	0.27
12	94.29	0.00	1.01	0.72	2.12	...	0.00	0.27
13	59.01	2.86	12.530	8.70	2.11	...	0.00	0.27
14	62.47	3.38	12.28	8.23	2.04	...	0.00	0.27
16	65.18	2.10	14.52	8.27	1.88	...	0.00	0.26
18	79.46	0.00	9.42	0.00	2.56	...	2.36	0.26
22	92.35	0.00	0.74	1.66	2.06	...	0.00	0.270
27	92.72	0.00	0.00	0.94	1.94	...	0.00	0.27

6.2 问题二模型的建立与求解

6.2.1 问题二第一小问

本题要求分析铅钡玻璃，高钾玻璃分类规律。由表单 1、2 数据整合，两类玻璃对应成分含量及是否风化均对应给出，可以利用解释性强的、合适的有监督的分类模型来得出分类的统计规律。本文对风化与未风化分别进行分析。由于两种类别中, 高钾和铅钡两种类型的玻璃的数量并不平衡, 因此采用决策树来探究分类规律。

决策树：

决策树算法是一种逼近离散函数值的方法。它是一种典型的分类方法，首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。决策树是一种树形结构，由节点（包括根节点、内部节点和叶节点）和边组成。每个内部节点表示一个特征属性，用于对数据集进行划分。每条边代表特征的一个取值，连接内部节点和子节点。每个叶节点代表一个类别（分类问题）或一个预测值（回归问题）。

决策树计算结果：

- **未风化点数据结果**

  针对未风化数据集取 70% 数据作为训练，30% 数据作为测试集。得到如图所示决策树。由图可知未分化点玻璃类型分类主要由氧化铅含量决定，当该玻璃中氧化铅含量小于或等于 6.08 时将其归于高钾玻璃类，当该玻璃中氧化铅含量大于 6.08 时，将其归于铅钡玻璃类。

  依据数据特性选择评价指标如下：

  （1）敏感性：也称为召回率（Recall），它衡量的是模型正确识别的正类样本占所有实际正类样本的比例。高敏感性意味着模型能够捕捉到大部分的正类样本。

  （2）精确率：指在所有被模型预测为正类（positive class）的样本中，实际也为正类的比例。换句话说，它衡量的是模型预测为正类中真正的正类样本的比例，即预测正确的正类样本占所有预测为正类样本的比例。

  （3）F1 分数：是统计学中用来衡量二分类模型精确度的一种指标。他同时兼顾了分类模型的精确率和召回率。F1 分数可以看作是模型精确率和召回率的一种加权平均，它的最大值为 1，最小值为 0

  对该模型评估结果如表所示：

  通过表 8 可知，模型召回率，F1 分数，精确率均为 1，可见模型性能良好。

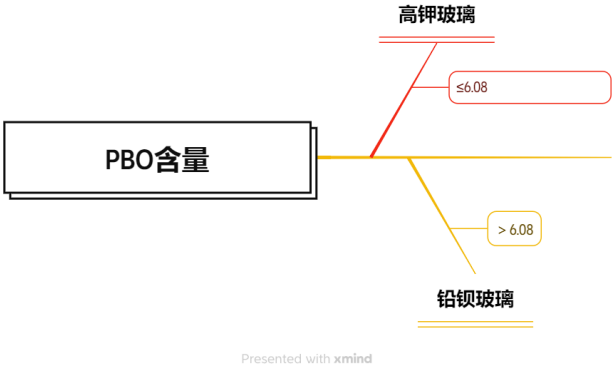

图 2 未风化点决策树

表 19 未风化数据点结果评估

评价指标	召回率	精确率	F1 分数
数值	1.0	1.0	1.0

- **风化数据结果**

  针对风化数据集同样取 70% 数据作为训练，30% 数据作为测试集。得到如图所示决策树，由图可知未分化点玻璃类型分类主要由氧化铅含量决定，当该玻璃中氧化铅含量小于或等于 6.34 时将其归于高钾玻璃类，当该玻璃中氧化铅含量大于 6.34 时，将其归于铅钡玻璃类。

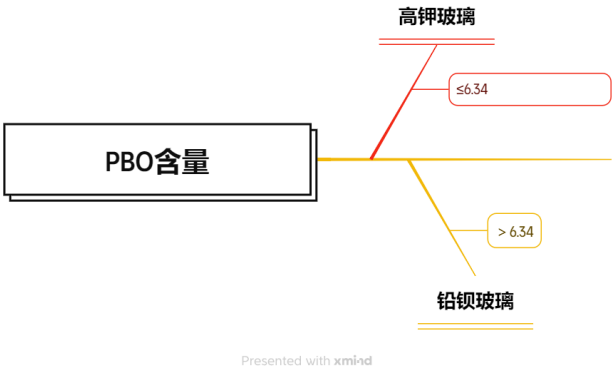

图 3 风化点决策树

通过表 9 可知，模型召回率，F1 分数，精确率均为 1，可见模型性能良好。综上所述，模型计算结果具有代表性。

- **总结分析**

  （1）玻璃文物类型主要与 *PbO* 含量有关，高钾玻璃 *PbO* 含量较低，铅钡玻璃含量较高。

  （2）通过对比有无风化玻璃 *PbO* 含量边界值，可知是否风化会影响其玻璃类型划分。

表 20 风化数据点结果评估

评价指标	召回率	精确率	F1 分数
数值	1.0	1.0	1.0

6.2.2 问题二第二小问

本问要求对每个类别选择合适的化学成分进行亚类划分。基于表单数据可以初步将数据划分为“铅钡风化”，“铅钡未风化”，“高钾风化”和“高钾未风化”四类。对四类玻璃分别进行 14 个特征的 Kmeans 聚类，将样本数据划分为 K 个类别，使得每个数据点与其所属类别的聚类中心之间的距离最小化，从而达到聚类的目的。

- **肘部法 K 值选择**

  肘部法则原理的核心思想是通过绘制不同 K 值下模型的误差与 k 值的关系曲线，找到曲线上的”肘部”点，即最佳的 K 值。通过观察聚类误差平方和 SSE 即每个样本到簇内中心点的距离偏差之和。计算簇内平方和 WCSS 公式如下：

$$WCSS = SSE = \sum_{i=1}^{k} \sum_{x \in c_i} \|x - \mu_i\|^2 \tag{11}$$

  随 k 值变化的曲线来确定最优聚类数，当 k 值较小时，增加 k 值会显著降低 SSE；当 k 值达到某个阈值后，在增加 k 值对 SSE 的降低效不再明显，这个阈值对应的 k 值即为最优聚类数。绘制肘部图如下：

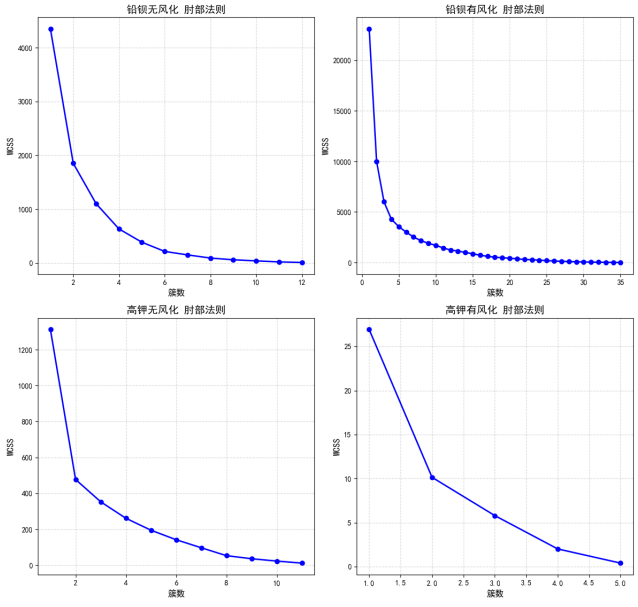

图 4 肘部图

  由图像观察折点，得到 k 值，分别为铅钡无风化-2，铅钡有风化-5，高钾无风化-2，高钾有风华-2。据此将每个类别继续分为对应亚类数，共分为 11 个亚类。

- **初始聚类中心选择**

  利用 **K-means++** 算法进行初始聚类中心选择优化。K-means++ 通过一种更合理的方式来选择初始聚类中心，使得算法更有可能找到全局最优解，而不是陷入局部最优。

- **聚类结果**

得到亚类划分成分含量如表 10 所示：

表 21 亚类划分成分含量表

大类	铅钡无风化		铅钡有风化					高钾无风化		高钾有风化	
亚类	簇 0	簇 1	簇 0	簇 1	簇 2	簇 3	簇 4	簇 0	簇 1	簇 0	簇 1
二氧化硅（SiO_2）	63.15	37.92	51.10	28.67	12.65	19.35	61.82	63.62	81.06	95.36	92.57
氧化铅（PbO）	19.80	29.66	24.09	42.23	30.15	59.40	14.49	0.41	0.41	0.00	0.00
氧化钡（BaO）	7.27	15.67	8.38	9.46	32.39	4.70	5.52	0.93	0.66	0.00	0.00
氧化铝 (Al_2O_3)	3.10	3.34	3.86	3.05	1.08	2.25	10.29	7.35	4.43	0.00	2.66
氧化钠（Na_2O）	0.92	0.54	3.61	0.32	0.00	0.00	1.56	0.93	0.00	0.00	0.00
氧化铜 (CuO)	0.77	2.82	1.78	1.49	6.93	1.53	0.41	2.82	1.35	1.35	1.78
氧化铁 (Fe_2O_3)	0.72	1.27	0.47	0.85	0.00	0.23	0.60	2.31	0.79	0.29	0.24
氧化钙 (CaO)	0.71	2.066	1.19	2.77	2.28	2.76	2.01	6.36	2.24	0.52	1.22
氧化镁 (MgO)	0.61	0.30	0.78	0.71	0.00	0.79	1.03	1.13	0.92	0.00	0.40
五氧化二磷 (P_2O_5)	0.51	1.530	1.82	5.72	5.08	5.01	0.51	1.52	1.04	0.17	0.39
二氧化硫 (SO_2)	0.46	0.00	0.00	0.00	8.88	0.00	0.00	0.14	0.00	0.00	0.00
氧化锶 (SrO)	0.27	0.35	0.24	0.32	0.49	0.66	0.23	0.05	0.02	0.00	0.00
氧化钾 (K_2O)	0.16	0.42	0.13	0.16	0.10	0.05	0.24	10.82	4.87	0.840	0.25
氧化锡 ($SnO2$)	0.00	0.17	0.00	0.031	0.00	0.00	0.31	0.00	0.79	0.00	0.00

可根据各亚类主要成分进行命名。铅钡玻璃类型均含有大量的氧化铅、氧化钡与二氧化硅。除去此三种主要成分对铅钡类玻璃亚类进行命名。高钾玻璃大部分还有高量氧化钾、二氧化硅。除去此两种成分对高钾类玻璃进行命名，并依据表单中文物采样点找到每个亚类所对应的文物，绘制树状图如图 5所示。

对计算结果进行指标评估，由于利用无监督算法进行计算，所以选择 CH 指数、轮廓系数 SC、DBI 指数、DVI 指数进行性能评估。

（1）CH 指数

即 Calinski-Harabasz 指数，通过计算簇内样本的协方差矩阵的迹与簇间样本的协方差矩阵的迹之比来评估聚类效果。CH 指数的值越大，表示聚类效果越好。

（2）轮廓系数 SC

它计算每个样本点到其所属簇内其他点的平均距离（凝聚度）和到最近簇的所有点的平均距离（分离度），然后用分离度减去凝聚度并归一化。能够衡量聚类效果的紧密性和分离度。轮廓系数的值范围在-1 到 1 之间，值越大表示聚类效果越好。

（3）DBI 指数

计算任意两个簇的类内距离平均距离之和与这两簇中心点距离的最大值来评估聚类效果。DBI 越小，表示聚类效果越好，分类越合理。

（3）DVI 指数

计算任意两个簇元素的最短距离（类间）除以任意簇中的最大距离（类内）来评估聚类效果。DVI 越大，表示聚类效果越好，分类越合理。

指标计算结果如表 22所示

表 22 亚类划分结果指标评估

种类	CH 指数	轮廓系数 SH	DBI 指数	DVI 指数
铅钡无风化	14.87	0.46	0.82	0.25
铅钡有风化	42.66	0.36	0.89	0.31
高钾无风化	17.51	0.53	0.65	0.85
高钾有风化	6.62	0.41	0.72	0.66

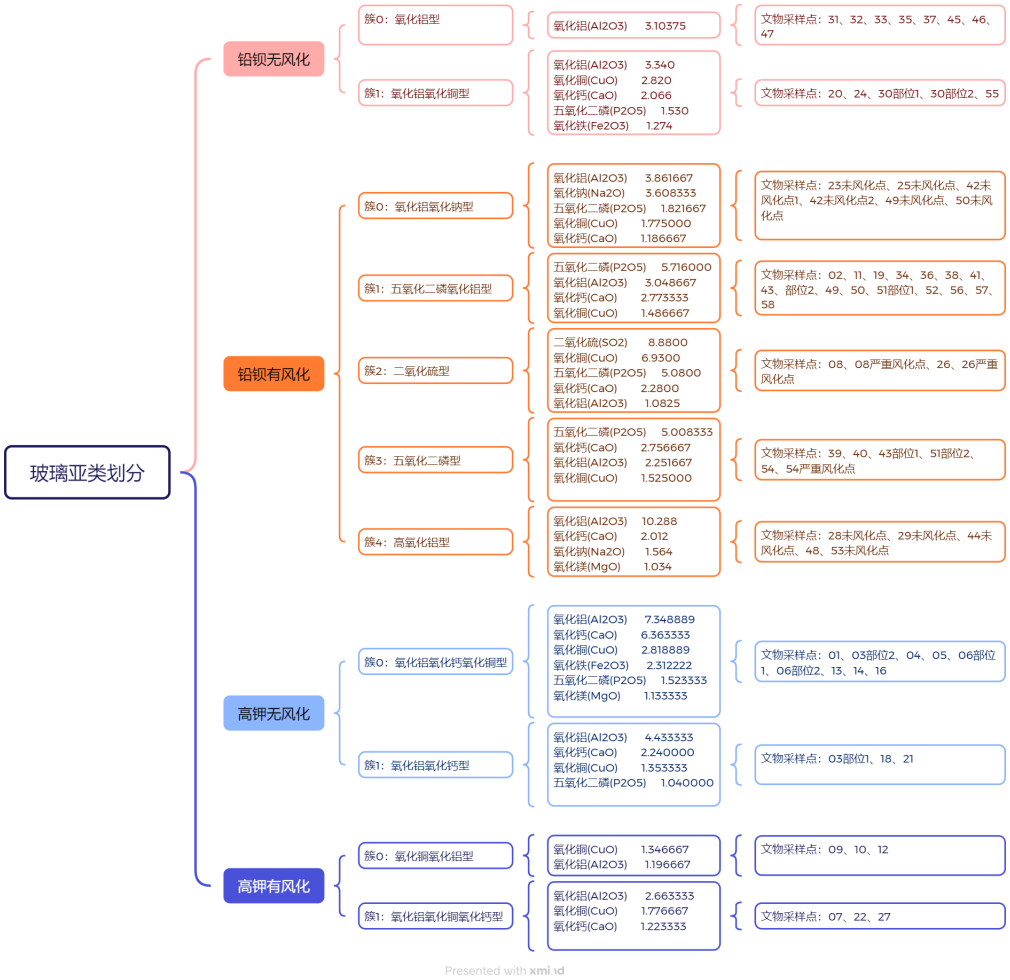

图 5 玻璃亚类划分树状图

由表中数据，高钾无风化在 CH 指数、轮廓系数 SH 和 DVI 指数上都表现出较好的聚类效果，意味着 kmeans 聚类在该类别上表现最为优秀。本文 kmens++ 算法进行了初始簇心的选择优化，经过多次运行，修改初始参数结果不变，敏感性较强。

6.3 问题三模型的建立与求解

本问要求对表单 3 中的数据进行分析，鉴别玻璃所属类型并对分类结果进行敏感性分析。采用两种分类模型决策树，XGboost，同时进行预测，XGBoost 的基本组成元素为决策树，但同时对损失函数进行了二阶泰勒展开，增加了精度且更灵活。目标函数中加入了正则项，能够防止过拟合。同时 XGBoost 也有计算复杂，可解释性差的缺点，故选择两种算法进行结果对比。

XGBoost 的目标函数可以表示为:

$$\text{obj}(\theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{12}$$

其中：

$L(y_i, \hat{y}_i)$ 是损失函数，用于衡量第 i 个样本的预测值 $\hat{y}_i$ 与真实值 y_i 之间的差异。

$\Omega(f_k)$ 是正则化项，用于控制第 k 个弱学习器 (即决策树) 的复杂度。

表 23 玻璃类型预测结果

文物编号	表面风化	玻璃类型		亚类类型	
		XGboost 预测	决策树预测	XGboost 预测	决策树预测
A1	无风化	1	1	1	1
A2	风化	0	0	0	1
A3	无风化	0	0	0	1
A4	无风化	0	0	0	1
A5	风化	0	0	0	4
A6	风化	1	1	1	0
A7	风化	1	1	1	1
A8	无风化	0	0	0	0

n 是样本数量，K 是弱学习器的数量。

计算结果如表 23：

由表观察可知，对玻璃类型，XGboost 预测与决策树预测结果完全相同，结果均为 A_1、A_6、A_7 为高钾玻璃,A_2、A_3、A_4、A_5、A_8 为铅钡玻璃，模型具有较高敏感性。而对玻璃亚类类型的预测，两种预测方法结果明显不同，算法敏感性较差，可进行后期模型优化。

6.4 问题四模型的建立与求解

问题四要求分析不同玻璃类别的化学成分间的相关性，本文选择 Spearman 相关系数对四大类玻璃的化学成分进行相关性分析。

据问题一中正态检验结果，面对部分不服从正态分布数据，选取适用条件广泛的 Spearman 相关系数进行相关性分析。其利用两变量秩次大小做线性相关分析，对原始变量分布不做要求，属于非参数统计方法。因此，其适用范围较广，如果数据中没有重复值，并且当两个变量完全单调相关时，Spearman 相关系数则为 ±1。同时，spearman 相关系数受异常值影响较小。计算公式：

$$\rho = \frac{\sum_i (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2 \sum_i (b_i - \bar{b})^2}}. \tag{13}$$

其中 a, b 分别代表两个特征的样本值。根据结果绘制绘制热力图，此处展示铅钡无风化如图 6。

在热力图中，颜色越深表示相关性越强。在铅钡未风化玻璃中可以观察到：

I. **氧化钡与二氧化硅、氧化锡与氧化铜、氧化铅与二氧化硅、具有较高相关性且为负相关。** II. **氧化锡与氧化钙、氧化铅具有一定相关性，并成正相关。**

III. **二氧化硫与其他各成分含量之间相关系数较低，可见其相关性较弱。**

IV. 可以观察到，二氧化硅、氧化锡、氧化锶、氧化钡均易受到多种其他成分含量变化影响而变化，**主要反映了这些成分在玻璃中的化学稳定性和反应性**。这些成分在玻璃中可能以较不稳定的化学状态存在，容易受到外界环境（如水分、二氧化碳、硫化物等）的影响而发生变化。或具有较强的化学活性，能够与其他成分或外界环境中的物质发生化学反应。

同时对其他三类玻璃热力图进行结果分析：在**铅钡风化类型玻璃**中，氧化铅与二氧化硅、氧化锶与二氧化硅、五氧化二磷与氧化钠成分含量变化具有较高相关性，且为负相关。

氧化铝与二氧化硅、五氧化二磷与氧化钙、氧化铝与氧化镁均呈正相关关系。

在玻璃风化中，氧化铝可以通过提高熔点和减缓晶化来稳定玻璃结构，而氧化镁

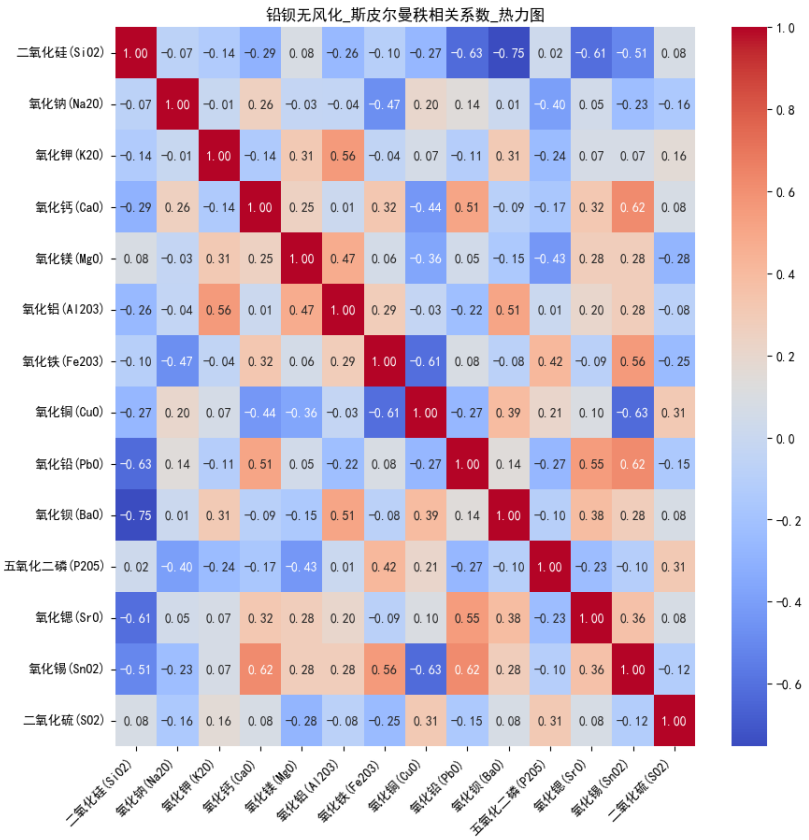

图 6 铅钡未风化玻璃热力图

可以通过助溶和降低析晶倾向来改善玻璃性能。两者之间呈较强正相关性。在玻璃分化过程中含量往往同增同长。

在**高钾无风化**热力图中，氧化钙与氧化钾、氧化钠，氧化钾与氧化钠均具有较强的正相关性。氧化铁与二氧化硅、氧化钾与二氧化硅具有较强的负相关性。

在**高钾风化**热力图中存在大量缺失值。但在已有数据中氧化铝与氧化钙呈高度正相关，其相关性系数为 0.94。两者随所以是时间变化含量变化趋势近乎相同。氧化铝与二氧化硅、氧化钾与五氧化二磷呈高度负相关。五氧化二磷在玻璃制造中通常。水的形式。磷酸盐存在。两者之间溶解性和稀疏性方面存在差异。这种差异在分化中得到体现使两者含量变化呈现负相关。

七、模型优缺点

7.1 模型优点

1. 计算效率：

   本文对数据进行充分预处理，进行相关的正态性检验为后续相关性分析等操作提供了前提条件，使得分析结果更加可靠。

2. 本文选择利用利用自适应 k 值选择优化聚类方法，提高了效率与准确性。

7.2 模型缺点

1. 本文在进行玻璃亚类划分时，构建模型敏感性存在缺陷，后续应进行进一步优化。

附录 A

附录 1：数据预处理 Python 代码

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.neighbors import KNeighborsRegressor,
    KNeighborsClassifier
from sklearn.model_selection import GridSearchCV,
    cross_val_score
from sklearn.metrics import accuracy_score
path = '..\Question\附件.xlsx'
excel1 = pd.read_excel(path,sheet_name = 0)
# 对各分类变量Label Encoding编码量化
le = LabelEncoder()
excel1_encoding = excel1.copy(deep=True)
excel1_columns_lst = list(excel1.columns)
excel1_columns_lst.remove('文物编号')
for column in excel1_columns_lst:
excel1_encoding[column] = le.fit_transform(excel1_encoding[
    column])
print(le.classes_)
# 将颜色中种类为8替换为NaN
excel1_encoding.loc[excel1_encoding['颜色'] == 8, '颜色'] = np
    .nan

excel1_encoding.l# KNN插补的特征列
X = excel1_encoding[['纹饰', '类型', '表面风化']]
y = excel1_encoding['颜色']
# 分离出有缺失和没有缺失的行
non_missing = ~excel1_encoding['颜色'].isna()
X_train, y_train = X[non_missing], y[non_missing]

# 初始化KNN回归器 KNeighborsRegressor连续，
    KNeighborsClassifier离散
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

# 预测缺失的颜色
missing_indices = excel1_encoding['颜色'].isna()
X_missing = X[missing_indices]
predicted_color = knn.predict(X_missing)

# 将预测值放回DataFrame
excel1_encoding.loc[missing_indices, '颜色'] = predicted_color
    oc[18,list(excel1_encoding.columns)]
excel1_encoding.loc[18,list(excel1_encoding.columns)]
```

```
excel1_encoding.to_excel('excel1_encoding.xlsx',index=False)
# 读取数据
excel2 = pd.read_excel(path,sheet_name=1)

excel2.head()

 以0填补空缺值
excel2_filled = excel2.fillna(0)

excel2_filled.head()

 计算成分比例累加和
excel2_filled['成分比例累加和'] = excel2_filled.iloc[:,2:].sum(
    axis=1)

excel2_filled.iloc[15:20,:]

# 85%~105%之间的数据视为有效数据，剔除异常数据
com_sum = excel2_filled['成分比例累加和']
outliers_data = excel2_filled[(com_sum < 85) | (com_sum >
    105)]
outliers_data
excel2_filled.drop(outliers_data.index,inplace=True)

excel2_filled.iloc[15:20,:]

merged_excel = pd.merge(excel2_filled, excel1_encoding[['文物
    编号', '表面风化',"类型"]], on='文物编号', how='left')
standard_excel = merged_excel.copy()
standard_excel["standard"] = None
# 将数据中的所有化学成分含量都乘以标准化因子
for key, item in standard_excel.iloc[:,16].items():
# 标准化因子
standard = 100 / standard_excel.iloc[key,16]
# 存储标准化因子
standard_excel.loc[key,"standard"] = standard
# 标准化
standard_excel.iloc[key,2:16] = standard_excel.iloc[key,2:16] *
    standard
standard_excel.iloc[key,16] = standard_excel.iloc[key,2:16].sum
    ()

standard_excel

with pd.ExcelWriter("merged_excel.xlsx") as writer:
merged_excel.to_excel(writer, sheet_name="origin",index=
    False)
standard_excel.to_excel(writer, sheet_name="standard",index
```

```
                =False)
```

附录 2. 问题一统计量计算即正态性检验代码

```
import pandas as pd
import numpy as np
from scipy.stats import skew,kurtosis,probplot,spearmanr
import matplotlib.pyplot as plt
import seaborn as sns
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
path = 'merged_excel.xlsx'
data = pd.read_excel(path,sheet_name=1,)
data.columns
# 提取铅钡无风化00，铅钡有风化01，高钾无风化10，高钾有风化
    11
bariumLead_non = data[(data['类型'] == 0) & (data['表面风化']
    == 0)]
bariumLead = data[(data['类型'] == 0) & (data['表面风化'] == 1)]
highK_non = data[(data['类型'] == 1) & (data['表面风化'] == 0)]
highK = data[(data['类型'] == 1) & (data['表面风化'] == 1) ]
bariumLead_non.head()

columns = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '氧
    化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)']
indexs = ['mean','median','std','skewness','kurtosis','coefficient']
statistics_bariumLead_non = pd.DataFrame(index=indexs,columns
    =columns)
statistics_bariumLead = pd.DataFrame(index=indexs,columns=
    columns)
statistics_highK_non = pd.DataFrame(index=indexs,columns=
    columns)
statistics_highK = pd.DataFrame(index=indexs,columns=columns)

for column in columns:
# 均值
statistics_bariumLead_non.loc['mean',column] = bariumLead_non[
    column].mean()
statistics_bariumLead.loc['mean',column] = bariumLead[column].
    mean()
statistics_highK_non.loc['mean',column] = highK_non[column].
    mean()
statistics_highK.loc['mean',column] = highK[column].mean()
# 中位数
statistics_bariumLead_non.loc['median',column] =
```

```
    bariumLead_non[column].median()
statistics_bariumLead.loc['median',column] = bariumLead[column].
    median()
statistics_highK_non.loc['median',column] = highK_non[column].
    median()
statistics_highK.loc['median',column] = highK[column].median()
# 标准差
statistics_bariumLead_non.loc['std',column] = bariumLead_non[
    column].std()
statistics_bariumLead.loc['std',column] = bariumLead[column].std()
statistics_highK_non.loc['std',column] = highK_non[column].std()
statistics_highK.loc['std',column] = highK[column].std()
# 变异系数CV
statistics_bariumLead_non.loc['coefficient',column] =
    bariumLead_non[column].std() / highK[column].mean()
statistics_bariumLead.loc['coefficient',column] = bariumLead[
    column].std() / highK[column].mean()
statistics_highK_non.loc['coefficient',column] = highK_non[column
    ].std() / highK[column].mean()
statistics_highK.loc['coefficient',column] = highK[column].std() /
    highK[column].mean()
# 偏度系数
statistics_bariumLead_non.loc['skewness',column] = skew(
    bariumLead_non[column])
statistics_bariumLead.loc['skewness',column] = skew(bariumLead[
    column])
statistics_highK_non.loc['skewness',column] = skew(highK_non[
    column])
statistics_highK.loc['skewness',column] = skew(highK[column])
# 峰度系数
statistics_bariumLead_non.loc['kurtosis',column] = kurtosis(
    bariumLead_non[column]) + 3
statistics_bariumLead.loc['kurtosis',column] = kurtosis(bariumLead
    [column]) + 3
statistics_highK_non.loc['kurtosis',column] = kurtosis(highK_non[
    column]) + 3
statistics_highK.loc['kurtosis',column] = kurtosis(highK[column]) +
     3

statistics_bariumLead_non.to_csv('铅钡无风化描述性统计规律.txt
    ',sep='\t')
statistics_bariumLead.to_csv('铅钡风化描述性统计规律.txt',sep='
    \t')
statistics_highK_non.to_csv('高钾无风化描述性统计规律.txt',sep
    ='\t')
statistics_highK.to_csv('高钾风化描述性统计规律.txt',sep='\t')

statistics_bariumLead_non.to_latex()
```

```
def qqplot(data):
fig, axes = plt.subplots(2, 7, figsize=(14, 6))
for i, col in enumerate(columns):
# 确定子图的位置
row = i // 7
col_idx = i % 7
ax = axes[row, col_idx]
probplot(data[col], dist="norm", plot=ax)
ax.set_title(f'Q-Q{col}')
# 调整子图布局
plt.tight_layout()
# 显示图形
plt.show()

qqplot(bariumLead_non)
qqplot(bariumLead)
qqplot(highK_non)
qqplot(highK)
```

附录 2：卡方检验代码

```
# 导入第三方库
import pandas as pd
import numpy as np
from pprint import pprint
from scipy.stats import chi2_contingency

# 加载数据
excelpath = "excel1_encoding.xlsx"
date_cul = pd.read_excel(excelpath, sheet_name="Sheet1")

# 首先为了进行卡方检验需要构建列联表
# 对于表面风化-玻璃类型
type_windroses_table = pd.crosstab(date_cul['表面风化'],
    date_cul['类型'])

# 对于表面风化-纹饰
decoration_windroses_table = pd.crosstab(date_cul['表面风化'
    ], date_cul['纹饰'])

# 对于表面风化-颜色
color_windroses_table = pd.crosstab(date_cul['表面风化'],
    date_cul['颜色'])

# 进行卡方检验
def perform_chi_square(test_table):
chi2, p, dof, expected = chi2_contingency(test_table,correction
    =False)
```

```
pprint(f"卡方值:␣{chi2:.2f},␣p值:␣{p:.2f},␣自由度:␣{dof}")
print(f"期望值:\n{expected}")
# 检查期望值
expected_no_suit = expected[expected < 5] # 期望值小于5的
    需要进行校正
pprint(f"{len(expected_no_suit)}个单元格的期望计数小于5")
print("−"*20)
return chi2, p, dof,expected

# 执行检验
perform_chi_square(type_windroses_table)
perform_chi_square(decoration_windroses_table)
perform_chi_square(color_windroses_table)

 进行Yates校正
def perform_chi_square(test_table):
chi2, p, dof, expected = chi2_contingency(test_table,correction
    =True)
pprint(f"卡方值:␣{chi2:.2f},␣p值:␣{p:.2f},␣自由度:␣{dof}")
return chi2, p, dof,expected

perform_chi_square(decoration_windroses_table)
perform_chi_square(color_windroses_table)
```

附录 3：利用 BFGS 算法求解位置时间代码

```
def solve_v2(data, methods='BFGS'):
"""
求解方程
Args:
data:四组数据列表
methods:求解算法 默认为‘BFGS’
Returns:
result:[x,y,z,t]求解结果
"""
def objective(unkonwn_vars):
"""
残差平方和目标函数
Args:
unkonwn_vars: 待求解变量
Returns:
residuals: 残差平方和
"""
x, y, z, t = unkonwn_vars
residuals = [] # 残差平方和
for xi, yi, zi, ti in data:
xm = LONGITUDE * (x − xi)
ym = LATITUDE * (y − yi)
zm = (z − zi)
```

```
distance = np.sqrt(xm**2 + ym**2 + zm**2)
ti_predict = t + distance/C
residuals.append(ti_predict - ti)

# x,y,z,t的期望求解范围为[100,150],[0,50],[10000,None],[0,
    min_ti]
# 边界条件惩罚
longitude_range = [100, 150]
latitude_range = [0, 50]
min_altitude = 10000
arrive_time_range = [0, min(ti for _, _, _, ti in data)]

penalty = [0, 0, 0, 0]
if not (longitude_range[0] <= x <= longitude_range[1]):
penalty[0] = (x - min(longitude_range, key=lambda v: abs(v
    - x))) ** 2
if not (latitude_range[0] <= y <= latitude_range[1]):
penalty[1] = (y - min(latitude_range, key=lambda v: abs(v -
    y))) ** 2
if z < min_altitude:
penalty[2] = (z - min_altitude) ** 2
if not (arrive_time_range[0] <= t <= arrive_time_range[1]):
penalty[3] = (t - min(arrive_time_range, key=lambda v: abs(
    v - t))) ** 2
total_penalty = sum(penalty)
return np.sum(np.array(residuals) ** 2) + total_penalty

# 初始化
x0 = [110.5, 27.6, 10000, 45]

# bounds = [[100, 150], [0, 50], [10000, None], [0, min(ti for _,
    _, _, ti in data)]]

# 求解
result = minimize(objective, np.array(x0),method=methods,
    options={'maxiter': 10000})

# 检查是否成功
if result.success:
return result.x
else:
return None
```

附录 4：时间序列预测代码

```
from sklearn.preprocessing import RobustScaler
from sklearn.cluster import AgglomerativeClustering
from gower import gower_matrix
```

```
import numpy as np
import pandas as pd
excelpath = ”merged_excel.xlsx”
dateGlass = pd.read_excel(excelpath, sheet_name=”standard”
    )

def dateProcess(dateGlass,type:int):

dateGlass_value = (dateGlass[dateGlass[”类型”] == type]
.iloc[:,3:17]
.reset_index(drop=True))
dateGlass_sample = (dateGlass[dateGlass[”类型”] == type]
.iloc[:,1]
.reset_index(drop=True))
dateGlass_standard = (dateGlass[dateGlass[”类型”] == type]
.iloc[:,-1]
.reset_index(drop=True))
dateGlass_all = pd.concat([dateGlass_sample,
    dateGlass_standard, dateGlass_value],axis=1)

return dateGlass_all,dateGlass_value

# 提取出不同类型的 文物数据 的 成分性数据
dateGlass_pb_all, dateGlass_pb_value = dateProcess(
    dateGlass,0) # 铅钡
dateGlass_hk_all, dateGlass_hk_value = dateProcess(
    dateGlass,1) # 高钾

def clusterClassify_2(dateGlass):

# 计算Gower距离矩阵
gower_dist = gower_matrix(dateGlass)

# 创建StandardScaler实例
scaler = RobustScaler()

# 标准正态分布缩放Gower距离矩阵
ilr_data_scaled = scaler.fit_transform(gower_dist)

# 使用层次聚类算法，选择簇数为 2
clustering_model = AgglomerativeClustering(n_clusters=2,
    linkage='ward')

# 拟合模型并预测簇标签
cluster_labels = clustering_model.fit_predict(ilr_data_scaled)
# 返回聚类结果
return cluster_labels
```

```
cluster_labels_pb_2 = clusterClassify_2(dateGlass_pb_value)
cluster_labels_hk_2 = clusterClassify_2(dateGlass_hk_value)

# 先对结果进行筛选
print("铅钡-是否风化分类结果")
print("层次聚类数据")
# 根据array数组提取风化和无风化数据的索引
weather_indices = np.where(cluster_labels_pb_2 == 1)[0] #
    风化数据的索引
no_weather_indices = np.where(cluster_labels_pb_2 == 0)[0]
     # 无风化数据的索引
# 提取风化数据
weather_data = np.asarray(dateGlass_pb_all.iloc[
    weather_indices].iloc[:,0])
# 提取无风化数据
no_weather_data = np.asarray(dateGlass_pb_all.iloc[
    no_weather_indices].iloc[:,0])
print(f"风化数据:\n{weather_data}")
print(f"无风化数据:\n{no_weather_data}")
# 先对结果进行筛选
print("高钾-是否风化分类结果")
print("层次聚类数据")
# 根据array数组提取风化和无风化数据的索引
weather_indices = np.where(cluster_labels_hk_2 == 1)[0] #
    风化数据的索引
no_weather_indices = np.where(cluster_labels_hk_2 == 0)[0]
     # 无风化数据的索引
# 提取风化数据
weather_data = np.asarray(dateGlass_hk_all.iloc[
    weather_indices].iloc[:,0])
# 提取无风化数据
no_weather_data = np.asarray(dateGlass_hk_all.iloc[
    no_weather_indices].iloc[:,0])
print(f"风化数据:\n{weather_data}")
print(f"无风化数据:\n{no_weather_data}")
def clusterClassify_4(dateGlass):

# 计算Gower距离矩阵
gower_dist = gower_matrix(dateGlass)

# 创建StandardScaler实例
scaler = RobustScaler()

# 标准正态分布 缩放Gower距离矩阵
ilr_data_scaled = scaler.fit_transform(gower_dist)

# 使用层次聚类算法，选择簇数为 2
clustering_model = AgglomerativeClustering(n_clusters=4,
```

```
        linkage='ward')

    # 拟合模型并预测簇标签
    cluster_labels = clustering_model.fit_predict(ilr_data_scaled)
    # 返回聚类结果
    return cluster_labels

cluster_labels_pb_4 = clusterClassify_4(dateGlass_pb_value)
cluster_labels_hk_4 = clusterClassify_4(dateGlass_hk_value)

# 先对结果进行筛选
print("铅钡—是否风化分类结果")
print("层次聚类数据")
# 根据array数组提取风化和无风化数据的索引
weather_indices_0 = np.where(cluster_labels_pb_4 == 0)[0]
    # 无风化数据的索引
weather_indices_1 = np.where(cluster_labels_pb_4 == 1)[0]
    # 轻度风化
weather_indices_2 = np.where(cluster_labels_pb_4 == 2)[0]
    # 中度风化
weather_indices_3 = np.where(cluster_labels_pb_4 == 3)[0]
    # 重度风化
# 提取风化数据
weather_data_0 = np.asarray(dateGlass_pb_all.iloc[
    weather_indices_0].iloc[:,0])
weather_data_1 = np.asarray(dateGlass_pb_all.iloc[
    weather_indices_1].iloc[:,0])
weather_data_2 = np.asarray(dateGlass_pb_all.iloc[
    weather_indices_2].iloc[:,0])
weather_data_3 = np.asarray(dateGlass_pb_all.iloc[
    weather_indices_3].iloc[:,0])
print(f"无风化数据:\n{weather_data_0}")
print(f"轻度风化数据:\n{weather_data_1}")
print(f"中度风化数据:\n{weather_data_2}")
print(f"重度风化数据:\n{weather_data_3}")

# 先对结果进行筛选
print("高钾—是否风化分类结果")
print("层次聚类数据")
# 根据array数组提取风化和无风化数据的s索引
weather_indices_0 = np.where(cluster_labels_hk_4 == 0)[0]
    # 无风化数据的索引
weather_indices_1 = np.where(cluster_labels_hk_4 == 1)[0]
    # 轻度风化
weather_indices_2 = np.where(cluster_labels_hk_4 == 2)[0]
    # 中度风化
weather_indices_3 = np.where(cluster_labels_hk_4 == 3)[0]
    # 重度风化
```

```
# 提取风化数据
weather_data_0 = np.asarray(dateGlass_hk_all.iloc[
    weather_indices_0].iloc[:,0])
weather_data_1 = np.asarray(dateGlass_hk_all.iloc[
    weather_indices_1].iloc[:,0])
weather_data_2 = np.asarray(dateGlass_hk_all.iloc[
    weather_indices_2].iloc[:,0])
weather_data_3 = np.asarray(dateGlass_hk_all.iloc[
    weather_indices_3].iloc[:,0])
print(f"无风化数据:\n{weather_data_0}")
print(f"轻度风化数据:\n{weather_data_1}")
print(f"中度风化数据:\n{weather_data_2}")
print(f"重度风化数据:\n{weather_data_3}")

dateGlass_pb_all["weather_time"] = cluster_labels_pb_4
dateGlass_hk_all["weather_time"] = cluster_labels_hk_4
dateGlass_pb_all["weather_or_no"] = cluster_labels_pb_2
dateGlass_hk_all["weather_or_no"] = cluster_labels_hk_2

with pd.ExcelWriter("date_weather_type.xlsx") as writer:
dateGlass_pb_all.to_excel(writer, sheet_name="pb")
dateGlass_hk_all.to_excel(writer, sheet_name="hk")
```

附录 5：决策树聚类代码

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text,
    export_graphviz
from sklearn.metrics import confusion_matrix,precision_score,
    recall_score, f1_score

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from IPython.display import Image

excelpath = "merged_excel.xlsx"
dateGlass = pd.read_excel(excelpath, sheet_name="standard"
    )

# 提取出不同类型的 文物数据
dateGlass_not_wind = dateGlass[dateGlass["表面风化"] == 0]
     # 未风化
dateGlass_wind = dateGlass[dateGlass["表面风化"] == 1] #
    风化

# 分割数据集
def GetDate(dateglass):
```

```
# 获取 X 数据
date_x = np.asarray(dateGlass.iloc[:,2:16])
# 获取对应的 Y 数据
date_y = np.asarray(dateGlass.iloc[:,-2])
# 划分数据
X_train, X_test, y_train, y_test = train_test_split(date_x,
    date_y,test_size=0.2)
# 数据库的决策名称表
names = list(dateGlass.iloc[:,2:16].columns)
return X_train, X_test, y_train, y_test, names

ef DecisionTreeClassify(X_train, y_train, X_test, names):
# 创建实例
clf = DecisionTreeClassifier(random_state=42)
# 训练模型
clf.fit(X_train, y_train)
# 进行预测
y_pred = clf.predict(X_test)
# 绘制树结构
rules = export_text(clf, feature_names=names)
return clf, y_pred, rules

# 未风化 决策树模型
X_train_nw, X_test_nw, y_train_nw, y_test_nw, names_nw
    = GetDate(dateGlass_not_wind)
clf_nw, y_pred_nw, rules_nw = DecisionTreeClassify(
    X_train_nw, y_train_nw, X_test_nw,names_nw)
print(rules_nw)

# 风化 决策树模型
X_train_w, X_test_w, y_train_w, y_test_w, names_w =
    GetDate(dateGlass_wind)
clf_w, y_pred_w, rules_w = DecisionTreeClassify(X_train_w,
    y_train_w, X_test_w,names_w)
print(rules_w)

def evalTreeModel(y_test, y_pred):
# 混淆矩阵
conf_matrix = confusion_matrix(y_test, y_pred)
# 计算召回率
recall = recall_score(y_test, y_pred)
# 计算精确率
precision = precision_score(y_test, y_pred)
# 计算F1分数
f1 = f1_score(y_test, y_pred)
# 统计表格
modelEstValue = {
    "recall": recall,
```

```
        "f1_score":f1,
        "precision":precision
    }
    modelEstValueDf = pd.DataFrame(modelEstValue,index=["
        value"])
    return conf_matrix, modelEstValueDf

    def plot_conf_matrix(conf_matrix):
    # 绘制混淆矩阵
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.show()

    # 计算未风化 决策树模型的评价指标
    conf_matrix_nw, modelEstValueDf_nw = evalTreeModel(
        y_test_nw,y_pred_nw)
    # 计算风化 决策树模型的评价指标
    conf_matrix_w, modelEstValueDf_w = evalTreeModel(
        y_test_w,y_pred_w)

    plot_conf_matrix(conf_matrix_nw),plot_conf_matrix(
        conf_matrix_w)

    def ImageTreeGraph(clf,feature_names, class_names=["铅钡","
        高钾"]):
    # 假定clf是已经训练好的决策树模型，iris是加载的数据集
    dot_data = export_graphviz(clf, out_file=None,
    feature_names = feature_names,
    class_names = class_names,
    filled=True, rounded=True,
    special_characters=True)

    # 使用pydotplus将dot数据转换为图像
    graph = pydotplus.graph_from_dot_data(dot_data)
    return graph

    graph_nw = ImageTreeGraph(clf_nw, names_nw)
    graph_w = ImageTreeGraph(clf_w, names_w)
    png_image_nw = graph_nw.create_png()
    png_image_w = graph_w.create_png()
```

附录 6：kmeans 聚类

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
# Matplotlib中设置字体—黑体，解决Matplotlib中文乱码问题
plt.rcParams['font.sans-serif'] = ['SimHei']
# 解决Matplotlib坐标轴负号'-'显示为方块的问题
plt.rcParams['axes.unicode_minus'] = False
path = 'merged_excel.xlsx'
data = pd.read_excel(path)
data.columns
data.head()
# ['无风化' '风化'], ['铅钡' '高钾']
# 提取铅钡无风化00，铅钡有风化01，高钾无风化10，高钾有风化11
bariumLead_non = data[(data['类型'] == 0) & (data['表面风化'] == 0)]
bariumLead = data[(data['类型'] == 0) & (data['表面风化'] == 1)]
highK_non = data[(data['类型'] == 1) & (data['表面风化'] == 0)]
highK = data[(data['类型'] == 1) & (data['表面风化'] == 1) ]
data_lst = [bariumLead_non, bariumLead, highK_non, highK]
data_name_lst = ['铅钡无风化', '铅钡有风化', '高钾无风化', '高钾有风化']
feature_lst = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '氧化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)', '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)', '二氧化硫(SO2)']
fig = plt.figure(figsize=(8, 8)) # 可以设置图形的大小
for index, data in enumerate(data_lst):
X = data[feature_lst]
n = X.shape[0]

# 使用肘部法则来确定最佳的簇数
wcss = []
for i in range(1, n):
kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
kmeans.fit(X)
wcss.append(kmeans.inertia_)

ax = fig.add_subplot(2, 2, index+1)
ax.plot(range(1, n), wcss)
ax.set_title(f'{data_name_lst[index]}肘部法则')
ax.set_xlabel('簇数')
ax.set_ylabel('SSE')
ax.grid(True)

# 将 plt.show() 移动到循环外部
plt.tight_layout() # 调整子图之间的间距
plt.show()
```

```
data_lst = [bariumLead_non, bariumLead, highK_non, highK]
data_name_lst = ['铅钡无风化', '铅钡有风化', '高钾无风化', '高钾
    有风化']
feature_lst = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '
    氧化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)']
# 创建一个新的图形，并设置大小
fig, axes = plt.subplots(2, 2, figsize=(12, 12))

# 遍历数据和对应的子图
for index, (data, ax) in enumerate(zip(data_lst, axes.flat)):
X = data[feature_lst]
n = X.shape[0]

# 肘部法则
wcss = []
for i in range(1, n):
kmeans = KMeans(n_clusters=i, init='k-means++', max_iter
    =300, n_init=10, random_state=0)
kmeans.fit(X)
wcss.append(kmeans.inertia_)

# 绘制图形
ax.plot(range(1, n), wcss, marker='o', linestyle='-', color='b',
    linewidth=2)

# 美化图形
ax.set_title(f'{data_name_lst[index]}␣肘部法则', fontsize=14,
    fontweight='bold')
ax.set_xlabel('簇数', fontsize=12)
ax.set_ylabel('WCSS', fontsize=12)
ax.grid(True, linestyle='--', alpha=0.5)
ax.tick_params(axis='both', which='major', labelsize=10)

# 调整子图之间的间距
plt.tight_layout(rect=[0, 0, 1, 0.95])

# 显示图形
plt.show()

best_k = [2, 5, 2, 2]
data_lst = [bariumLead_non, bariumLead, highK_non, highK]
data_name_lst = ['铅钡无风化', '铅钡有风化', '高钾无风化', '高钾
    有风化']
feature_lst = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '
```

```
    氧化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)']
cluster_stats = {}
# {'铅钡无风化':{'簇0':df1,'簇1':df2},'铅钡有风化':{},}
for index,data in enumerate(data_lst):
kmeans = KMeans(n_clusters=best_k[index], init='k-means++',
    max_iter=300, n_init=10, random_state=0)
X = data[feature_lst]
data[f'{data_name_lst[index]}亚类分类结果'] = kmeans.fit_predict
    (X)
cluster_stats[data_name_lst[index]] = {}
for i in range(best_k[index]):
claster_data = data[data[f'{data_name_lst[index]}亚类分类结果']
    == i]
claster_stat = claster_data[feature_lst].describe()
cluster_stats[data_name_lst[index]][f'簇{i}'] = claster_stat

from scipy.spatial.distance import pdist, squareform
from sklearn.metrics import calinski_harabasz_score,
    silhouette_score, davies_bouldin_score
import numpy as np

# 计算DVI片评价指标函数
def dunn_index(X, labels):
unique_labels = np.unique(labels)
n_clusters = len(unique_labels)
diameters = []
min_sep = float('inf')
for i, label in enumerate(unique_labels):
cluster_points = X[labels == label]
distances = pdist(cluster_points)
diameters.append(np.max(distances))
for j, label2 in enumerate(unique_labels):
if i < j:
cluster_points2 = X[labels == label2]
# 计算两个簇之间的最小距离
sep = np.min(squareform(pdist(np.vstack([cluster_points,
    cluster_points2])))[:len(cluster_points), len(cluster_points):])
min_sep = min(min_sep, sep)
return min_sep / np.max(diameters)

best_k = [2, 5, 2, 2]
data_lst = [bariumLead_non, bariumLead, highK_non, highK]
data_name_lst = ['铅钡无风化', '铅钡有风化', '高钾无风化', '高钾
    有风化']
```

```
feature_lst = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '
    氧化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)']

cluster_evaluation = {}
for index,data in enumerate(data_lst):
kmeans = KMeans(n_clusters=best_k[index], init='k-means++',
    max_iter=300, n_init=10, random_state=0)
X = data[feature_lst]
y = kmeans.fit_predict(X)
cluster_evaluation[f'{data_name_lst[index]}评价指标'] = {}
ch = calinski_harabasz_score(X,y) # CH指数
sh = silhouette_score(X,y) # SH轮廓系数
dbi = davies_bouldin_score(X,y) # DBI指数
dvi = dunn_index(X,y) # DVI指数
cluster_evaluation[f'{data_name_lst[index]}评价指标']['CH指数']
    = ch
cluster_evaluation[f'{data_name_lst[index]}评价指标']['轮廓系数
    SH'] = sh
cluster_evaluation[f'{data_name_lst[index]}评价指标']['DBI指数']
    = dbi
cluster_evaluation[f'{data_name_lst[index]}评价指标']['DVI指数']
    = dvi


evaluation_df = pd.DataFrame(cluster_evaluation)
evaluation_df.head()

mean_df = pd.DataFrame()
for kind, subkind_data in cluster_stats.items():
for subkind, df in subkind_data.items():
mean_row = df.loc['mean']
mean_df[f'{kind}_{subkind}'] = mean_row

mean_df.head(20)

for col in mean_df.columns:
print(mean_df[col].sort_values(ascending=False))
columns = ['文物采样点', '文物编号', '二氧化硅(SiO2)', '氧化钠(
    Na2O)', '氧化钾(K2O)', '氧化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)',
'成分比例累加和', '表面风化', '类型', '铅钡无风化亚类分类结果','
```

```
    铅钡有风化亚类分类结果','高钾无风化亚类分类结果','高钾有风
    化亚类分类结果']
index = range(len(bariumLead_non) + len(bariumLead) + len(
    highK_non) + len(highK))
merge_data = pd.DataFrame(index=index, columns=columns)

start_idx = 0
for df, cls_group in [(bariumLead_non, '铅钡无风化亚类分类结果')
    , (bariumLead, '铅钡有风化亚类分类结果'), (highK_non, '高钾
    无风化亚类分类结果'),(highK,'高钾有风化亚类分类结果')]:
end_idx = start_idx + len(df)
merge_data.iloc[start_idx:end_idx, :19] = df[['文物采样点', '文物
    编号', '二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '氧化
    钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)',
'成分比例累加和', '表面风化', '类型']]
merge_data.iloc[start_idx:end_idx, columns.index(cls_group)] = df
    [cls_group]
start_idx = end_idx

merge_data.head()

merge_data.to_excel('excel2_subkind_result.xlsx',index=False)

data_lst = [bariumLead_non, bariumLead, highK_non, highK]
data_name_lst = ['铅钡无风化', '铅钡有风化', '高钾无风化', '高钾
    有风化']
for index,data in enumerate(data_lst):
data.to_excel(f'{data_name_lst[index]}.xlsx',index=False)
```

附录 7：问题三预测代码

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text,
    export_graphviz
def DecisionTreeClassify(X_train, y_train, X_test, names):
# 创建实例
clf = DecisionTreeClassifier(random_state=42)
# 训练模型
clf.fit(X_train, y_train)
# 进行预测
y_pred = clf.predict(X_test)
# 绘制树结构
rules = export_text(clf, feature_names=names)
```

```
return clf, y_pred, rules
train_data = pd.read_excel('merged_excel.xlsx',sheet_name
    =1)
feature_lst = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O
    )', '氧化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(
    CuO)', '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(
    SnO2)', '二氧化硫(SO2)']

train_Notwind = train_data[train_data['表面风化'] == 0]
train_wind = train_data[train_data['表面风化'] == 1]

train_Notwind_X = train_Notwind[feature_lst]
train_Notwind_y = train_Notwind['类型']

train_wind_X = train_wind[feature_lst]
train_wind_y = train_wind['类型']
pre_data = pd.read_excel('E:\python\GitHub_teamProject\
    Problem_2022C\Question\附件.xlsx',sheet_name=2)

pre_data = pre_data.fillna(0)
pre_Notwind = pre_data[pre_data['表面风化']== '无风化']
pre_Notwind_X = pre_Notwind[feature_lst]

pre_wind = pre_data[pre_data['表面风化']== '风化']
pre_wind_X = pre_wind[feature_lst]

import xgboost as xgb

# 未风化XGBoost预测
# 初始化模型
model_nw = xgb.XGBClassifier(use_label_encoder=False,
    eval_metric='mlogloss')
# 训练模型
model_nw.fit(train_Notwind_X, train_Notwind_y)
# 进行预测
y_pred_nw = model_nw.predict(pre_Notwind_X)
# 还原数据
pre_Notwind.loc[:,'XGboost预测_玻璃类型'] = list(y_pred_nw
    )
# 风化XGBoost预测
# 初始化模型
model_w = xgb.XGBClassifier(use_label_encoder=False,
    eval_metric='mlogloss')
# 训练模型
model_w.fit(train_wind_X, train_wind_y)
# 进行预测
```

```
y_pred_w = model_nw.predict(pre_wind_X)
# 还原数据
pre_wind.loc[:,'XGboost预测_玻璃类型'] = list(y_pred_w)

clf_nw, y_pred_nw, rules_nw = DecisionTreeClassify(
    train_Notwind_X, train_Notwind_y, pre_Notwind_X,
    feature_lst)
clf_w, y_pred_w, rules_w = DecisionTreeClassify(
    train_wind_X, train_wind_y, pre_wind_X, feature_lst)

pre_Notwind.loc[:,'决策树预测_玻璃类型'] = list(y_pred_nw)
pre_wind.loc[:,'决策树预测_玻璃类型'] = list(y_pred_w)

merged_PreKind = pd.concat([pre_wind,pre_Notwind])

merged_PreKind = merged_PreKind.sort_index()
merged_PreKind
```

附录 8：问题四相关性分析

```
def heatmap_corr_plot(data, feature , title):
"""
绘制斯皮尔曼秩相关系数热力图
Args:
data: 原始数据
feature: 特征
tittle: 热力图名称
"""
# 计算斯皮尔曼秩相关系数
corr_matrix = data[feature].corr(method='spearman')
# 绘制热力图
plt.figure(figsize=(10, 10)) # 设定图形大小
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f
    ")
# 旋转x轴标签
plt.xticks(rotation=45, ha="right")
plt.title(f'{title}_斯皮尔曼秩相关系数_热力图')
plt.show()

columns = ['二氧化硅(SiO2)', '氧化钠(Na2O)', '氧化钾(K2O)', '氧
    化钙(CaO)',
'氧化镁(MgO)', '氧化铝(Al2O3)', '氧化铁(Fe2O3)', '氧化铜(CuO)',
    '氧化铅(PbO)',
'氧化钡(BaO)', '五氧化二磷(P2O5)', '氧化锶(SrO)', '氧化锡(SnO2)'
    , '二氧化硫(SO2)']
heatmap_corr_plot(bariumLead_non,columns,'铅钡无风化')
heatmap_corr_plot(bariumLead,columns,'铅钡风化')
heatmap_corr_plot(highK_non,columns,'高钾无风化')
heatmap_corr_plot(highK,columns,'高钾风化')
```

附录 B

附录 1：铅钡风化统计量

化学成分	平均值	中位数	标准差	偏度系数	峰度系数	变异系数
二氧化硅 (SiO2)	34.518015	30.459082	17.335801	0.341391	2.163827	0.183776
氧化钠 (Na2O)	0.975706	0.000000	1.972145	2.295428	7.604319	inf
氧化钾 (K2O)	0.145070	0.000000	0.216857	2.198255	9.373954	0.398210
氧化钙 (CaO)	2.431059	2.250788	1.699309	0.617111	2.676670	1.945968
氧化镁 (MgO)	0.724796	0.740032	0.683843	0.779356	3.863212	3.458109
氧化铝 (Al2O3)	3.929714	3.173733	3.424143	1.853208	6.038707	1.766984
氧化铁 (Fe2O3)	0.573456	0.310988	0.716177	1.315432	4.238286	2.693399
氧化铜 (CuO)	2.043532	1.022487	2.507747	2.268851	7.927416	1.598887
氧化铅 (PbO)	38.131900	41.027666	15.851296	0.039861	2.067873	inf
氧化钡 (BaO)	10.787697	8.4531	8.971232	1.476958	4.532929	inf
五氧化二磷 (P2O5)	4.310358	3.354788	4.322561	0.688037	2.345496	15.367961
氧化锶 (SrO)	0.376974	0.349538	0.253616	0.842285	4.017148	inf
氧化锡 (SnO2)	0.056862	0.000000	0.23457	4.719488	25.146625	inf
二氧化硫（SO2）	0.994862	0.000000	3.640783	3.756872	15.415144	inf

化学成分	平均值	中位数	标准差	偏度系数	峰度系数	变异系数
二氧化硅 (SiO2)	34.518015	30.459082	17.335801	0.341391	2.163827	0.183776
氧化钠 (Na2O)	0.975706	0.000000	1.972145	2.295428	7.604319	inf
氧化钾 (K2O)	0.145070	0.000000	0.216857	2.198255	9.373954	0.398210
氧化钙 (CaO)	2.431059	2.250788	1.699309	0.617111	2.676670	1.945968
氧化镁 (MgO)	0.724796	0.740032	0.683843	0.779356	3.863212	3.458109
氧化铝 (Al2O3)	3.929714	3.173733	3.424143	1.853208	6.038707	1.766984
氧化铁 (Fe2O3)	0.573456	0.310988	0.716177	1.315432	4.238286	2.693399
氧化铜 (CuO)	2.043532	1.022487	2.507747	2.268851	7.927416	1.598887
氧化铅 (PbO)	38.131900	41.027666	15.851296	0.039861	2.067873	inf
氧化钡 (BaO)	10.787697	8.453152	8.971232	1.476958	4.532929	inf
五氧化二磷 (P2O5)	4.310358	3.354788	4.322561	0.688037	2.345496	15.367961
氧化锶 (SrO)	0.042378	0.020323	0.049065	0.481896	1.602342	inf
氧化锡 (SnO2)	0.202228	0.000000	0.700538	3.015113	10.090909	inf
二氧化硫（SO2）	0.105109	0.000000	0.191805	1.216134	2.577149	inf

附录 2：各类玻璃成分含量热力图

化学成分	平均值	中位数	标准差	偏度系数	峰度系数	变异系数
二氧化硅 (SiO2)	94.330974	94.266927	1.675202	0.376459	2.060185	0.017759
氧化钠 (Na2O)	0.000000	0.000000	0.000000	NaN	NaN	NaN
氧化钾 (K2O)	0.544579	0.665680	0.446422	-0.389343	1.488367	0.819757
氧化钙 (CaO)	0.873246	0.837215	0.487931	0.348674	2.465796	0.558756
氧化镁 (MgO)	0.197751	0.000000	0.307777	0.736312	1.582415	1.556391
氧化铝 (Al2O3)	1.937846	1.726131	0.966892	0.548939	2.166156	0.498952
氧化铁 (Fe2O3)	0.265901	0.275874	0.069112	-0.231015	1.667644	0.259915
氧化铜 (CuO)	1.568432	1.556060	0.938137	0.879160	2.895792	0.598137
氧化铅 (PbO)	0.000000	0.000000	0.000000	NaN	NaN	NaN
氧化钡 (BaO)	0.000000	0.000000	0.000000	NaN	NaN	NaN
五氧化二磷 (P2O5)	0.281271	0.280403	0.210838	0.284910	2.257537	0.749590
氧化锶 (SrO)	0.000000	0.000000	0.0000	NAN	NAN	NAN
氧化锡 (SnO2)	0.000000	0.000000	0.000000	NAN	NAN	NAN
二氧化硫（SO2）	0.000000	0.000000	0.0000000	NAN	NAN	NAN

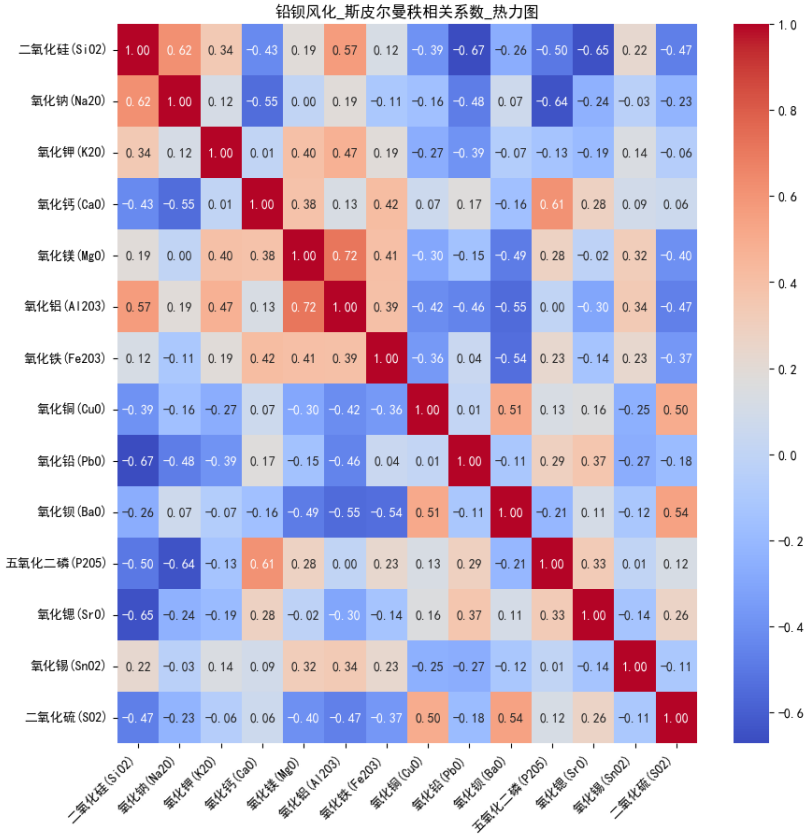

图 7 铅钡风化玻璃热力图

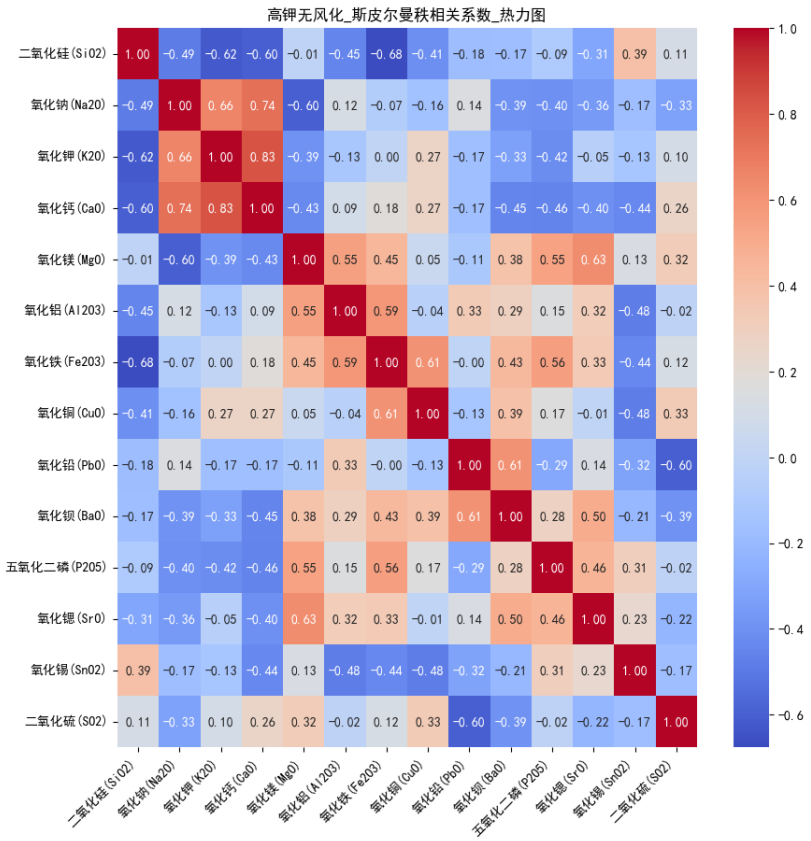

图 8 高钾未风化玻璃热力图

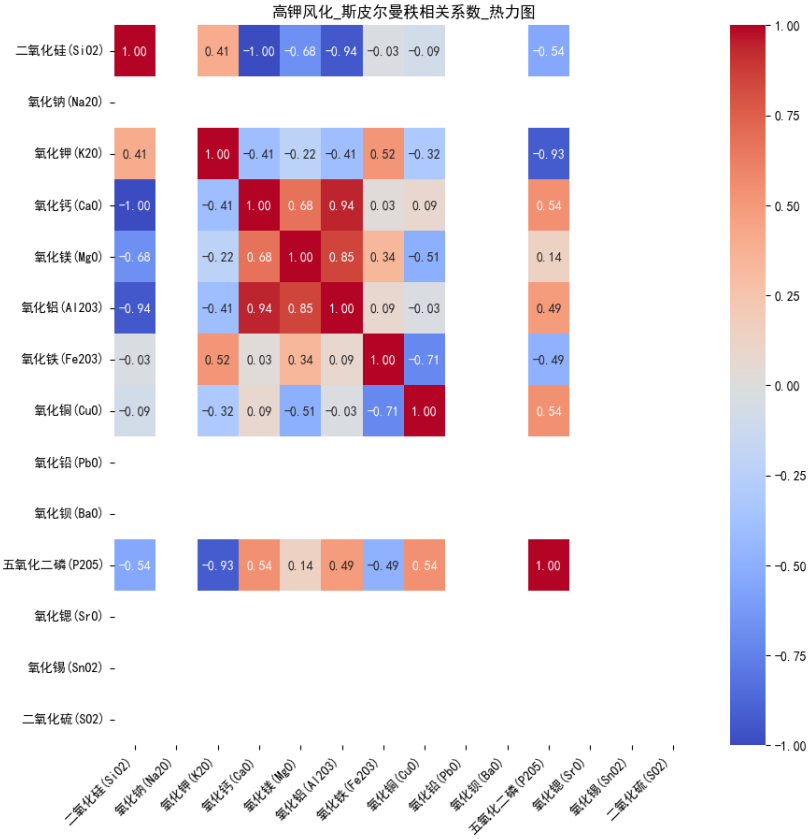

图 9 高钾风化玻璃热力图