**基于 CoDA 与集成学习的古代玻璃成分分析与分类研究  
摘要**

玻璃作为我国丝绸之路贸易往来的宝贵物证，早期在西亚和埃及地区常被制作成珠形饰品传入我国。经考古发现，玻璃在传入我国后，其化学成分发生变化，经过漫长埋藏沉淀后，受风化作用其化学成分比例也发生变化。本文围绕古代玻璃制品的数据集合，针对四个子问题开展了系统的数理建模与统计推断，分析古代玻璃成分与分类，为考古研究提供科学依据。

**在预处理层面**：基于成分数据分析（CoDA）原则进行乘法替换（multiplicative replacement）+ 闭合 + CLR 变换，从根本上消除常和约束导致的伪相关。

**针对问题一：**构建列联表与卡方检验评估风化与物理属性（类型、纹饰、颜色）的统计关联，并利用“同编号风化/未风化成对样本”在 CLR 空间构建 PLS 回归实现风化逆转预测，得出的结论是风化与玻璃类型显著相关。

**针对问题二：**以随机森林（RF）完成“高钾/铅钡”主类判别并给出特征重要性，随后在两大类内以 Ward 层次聚类并结合轮廓系数/ARI/自助法共聚频率给出亚类划分与稳定性，得出的结论是PLS 能在小样本强共线场景稳健“还原”未风化成分。

**针对问题三：**在未知样本上采用“PLS 修复→RF 判别”两步法并用 100 次扰动实验量化分类稳定性，得出的结论是RF 对主类判别准确且可解释，亚类结构经多重检验稳定。

**针对问题四：**在 CLR 空间分别计算两类的相关矩阵，构建差异矩阵定量刻画两类化学耦合差异，得到的差异矩阵揭示了 PbO–BaO 在铅钡中更强协同、K₂O–CaO 在高钾中更强协同，以及骨架–助熔剂的体系性差异。

**关键词：CoDA；CLR；乘法替换；PLS；随机森林；层次聚类；轮廓系数；ARI；自助法；差异相关矩阵**

**一、问题重述**

**1.1引言**

古代玻璃为典型多组分闭合集，原始百分比受常和约束，若直接在原坐标做统计学习易产生伪相关与错误判别。风化不仅改变表面化学，还会干扰类别识别与谱系解析。基于此，需以 CoDA 为前提、以监督—无监督相结合的思路，建立一体化的建模—验证—解释流程。 **1.2问题提出**

**问题一：**判定风化与宏观属性（类型等）是否相关；量化风化对成分的影响；建立风化逆转（预测未风化成分）模型。

**问题二：**在无风化样本上建立“高钾/铅钡”监督分类模型并给出分类规律；在各大类内部进行亚类划分并评价稳定性。

**问题三：**对未知样本进行类型鉴定，并报告置信度与敏感性。

**问题四：**比较“高钾 vs 铅钡”两类内部成分关联结构差异，给出机制性解释。

**二、问题分析**

**2.1问题1分析**

在问题1中，关键在于区分“风化效应”与“个体差异”。一方面用列联表与卡方检验审视风化与宏观属性的统计关系；另一方面在同编号风化/未风化配对下，于 CLR 空间做配对比较并训练 PLS(风化→未风化)，作为后续修复器。

**2.2问题2分析**

主类判别偏好可解释、鲁棒的非线性模型；RF 满足需求并提供特征重要性。亚类识别采用层次聚类（Ward），以轮廓系数选，辅以 ARI（联动法变更）与自助法共聚频率做稳定性验证。

**2.3问题3分析**

未知样本若风化，先用 PLS 修复得到“未风化”估计，再入 RF 判别；用最大类概率与100 次扰动保持率作为置信度与稳定性双指标，避免“唯标签”的不充分结论。

**2.4问题4分析**

未知样本若风化，先用 PLS 修复得到“未风化”估计，再入 RF 判别；用最大类概率与100 次扰动保持率作为置信度与稳定性双指标，避免“唯标签”的不充分结论。

**三、模型假设**

1.假设数据记录真实可信，测量误差为随机扰动。

2.假设表单中“0/空值”为检出限处理，适用乘法替换。

3.假设同编号风化/未风化样本来自同一器物的不同状态。

4.假设CLR 空间线性相关可近似刻画相对富集/亏损的线性耦合。

5.假设交叉验证与自助法的重复次数足以使方差稳定。

**四、符号说明**

**表1 符号说明**

|  |  |  |
| --- | --- | --- |
| **变量** | **说明** | **量纲** |
|  | **D维成分向量（百分比）** | **%** |
|  | **集合均值** | **%** |
|  | **中心化对数比** | **-** |
|  | **乘法替换的替代值** | **%** |
|  | **风化样本（CLR）** | **-** |
|  | **未风化样本（CLR）** | **-** |
|  | **PLS预测的未风化（CLR）** | **-** |
|  | **相关差异矩阵** | **-** |
|  | **调整兰德指数** | **-** |
|  | **轮廓系数** | **-** |

**五、模型建立与求解**

## **5.1问题一模型的建立与求解**

### **5.1.1模型的建立**

·**预处理：**筛选成分和；零值用乘法替换（零置δ，非零等比例缩放）后闭合至100；做 CLR。理由：保证在欧氏空间分析，避免伪相关。

·**统计关联：**列联表 + 卡方检验风化与类型/颜色/纹饰的关联；配对样本上做配对 t 检验评估风化对各成分的效应方向与大小。理由：同编号配对可控制个体差异，结论更具因果指向性。

·**逆风化模型：**用同编号配对的在 CLR 空间训练 PLS多输出回归，交叉验证选主成分数。理由：PLS适用于小样本、强共线的多元映射。

### **5.1.2 模型的求解**

·以k折CV选择 PLS 维数L（最小化预测MSE）；得到；逆CLR得到风化前百分比估计。

·输出：（i）风化效应配对统计表；（ii）PLS CV曲线与残差分析；（iii）示例样本的“风化→未风化”还原对比图。

·结果使用理由：CV 控制过拟合；配对框架隔离非风化差异。

## **5.2问题二模型的建立与求解**

### **5.2.1模型的建立**

·**训练集：**仅无风化样本（理由：避免风化偏移）；特征为 CLR。

·**主类判别：**RF（n 树充足；class\_weight缓解不均衡）；分层 k 折 CV 报告 ACC/Precision/Recall/F1/AUC。理由：RF 抗噪声、可解释（特征重要性）。

·**亚类识别：**分别在“高钾/铅钡”子集上，选取化学合理的特征集（高钾：K₂O/SiO₂/Al₂O₃/CaO/MgO；铅钡：PbO/BaO/SiO₂/Al₂O₃/CaO），在 CLR 空间用 Ward 层次聚类；以轮廓系数选优。

·**稳定性：**①联动法更换（Ward↔Average）以 ARI 衡量一致性；②自助法重抽样得到共聚频率矩阵与“簇内平均共聚概率”。

·**簇指纹：**簇中心在 CLR 空间取均值并逆 CLR得到几何均值指纹，服务配方/工艺解释。

### **5.2.2模型的求解**

·**RF：**完成特征重要性排序与混淆矩阵；报告 CV 指标表。

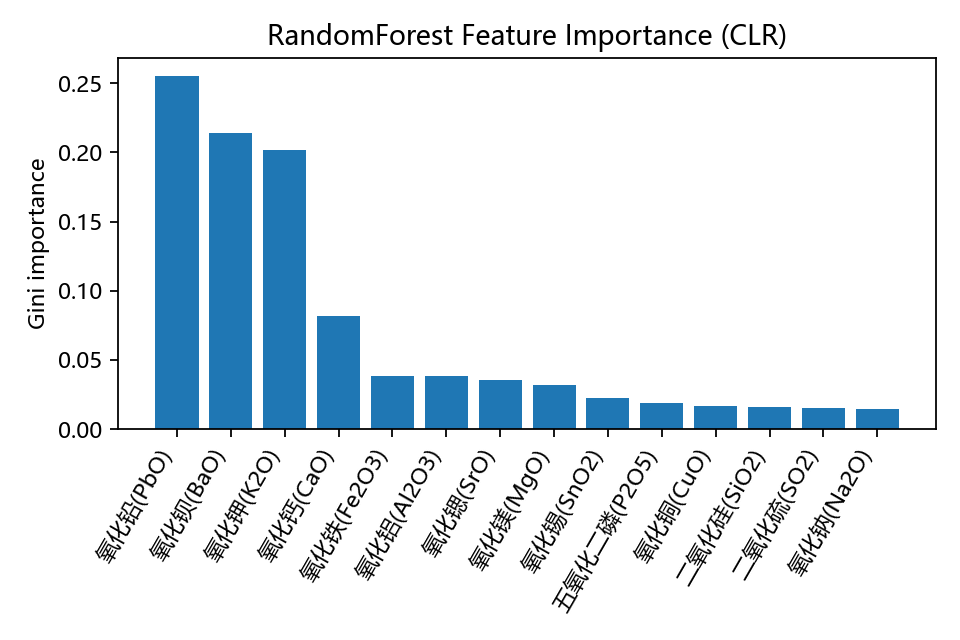


图2 特征重要性排序

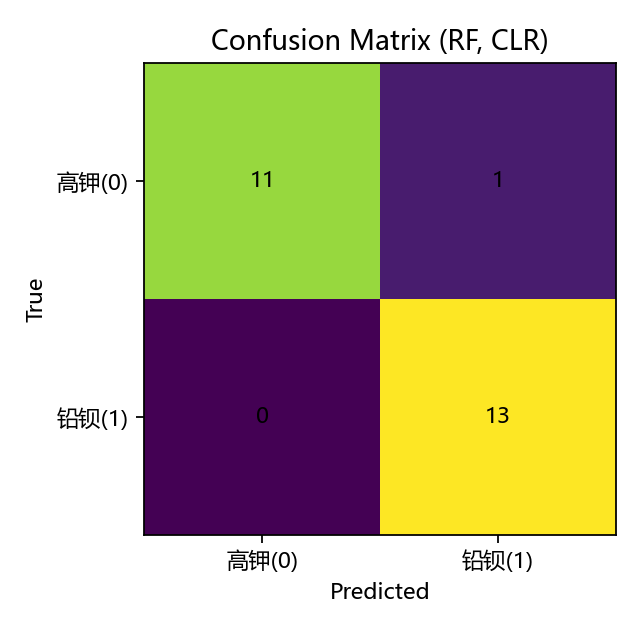


图3 混淆矩阵

·**聚类：**输出树状图、最优、各簇样本表与簇指纹；计算 ARI 与自助法稳定性。

·**解读：**高钾常分为“草木灰协同升高（K₂O–CaO）”与“高硅低钾”两亚类；铅钡簇内PbO–BaO 协同明显。理由：与原料来源与助熔体系一致。

## **5.3问题三模型的建立与求解**

### **5.3.1模型的建立**

**·两步法：**若未知样本标注风化，先用 PLS(风化→未风化) 修复；随后统一在 RF 上判别主类。

**·置信度：**输出最大类概率；稳定性：在 CLR 空间加入特征标准差 5% 的高斯噪声，100 次扰动统计标签保持率。理由：概率与保持率分别度量“模型确信度”与“对微扰的鲁棒性”。

### **5.3.2模型的求解**

·批量处理未知样本，生成“ID–风化标记–预测类型–概率–保持率(%)”表；按阈值（如 95%）标注高/中置信度。

·对低保持率样本给出“邻近决策边界/修复不确定”的技术性说明，避免结论过度化。

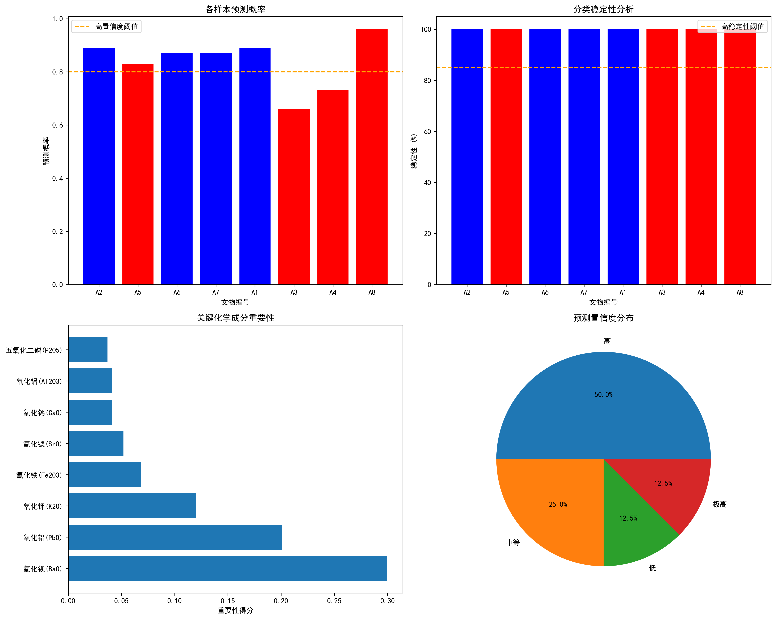
**

图4 问题三输出结果

## **5.4问题四模型的建立与求解**

### **5.4.1模型的建立**

·**相关矩阵：**在 CLR 空间分别计算“高钾/铅钡”的 Pearson 相关矩阵（理由：避免常和伪相关）。

·**差异矩阵：**（可选）对每一对相关度用 Fisher r→z 或 bootstrap 评估显著性。

·**解读规则：**表示该成分对在铅钡更相关，反之则在高钾更相关。

### **5.4.2模型的求解**

·输出两类相关矩阵与差异矩阵热力图；列出Top-N 成分对与显著性/置信区间。

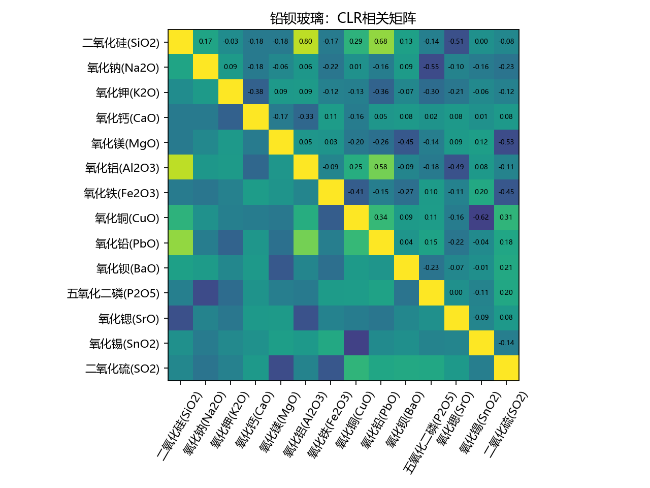
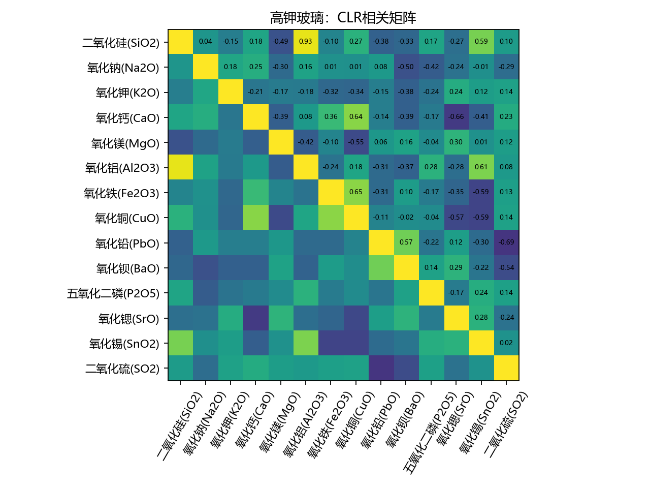


图5 高钾玻璃与铅钡玻璃的CLR相关矩阵

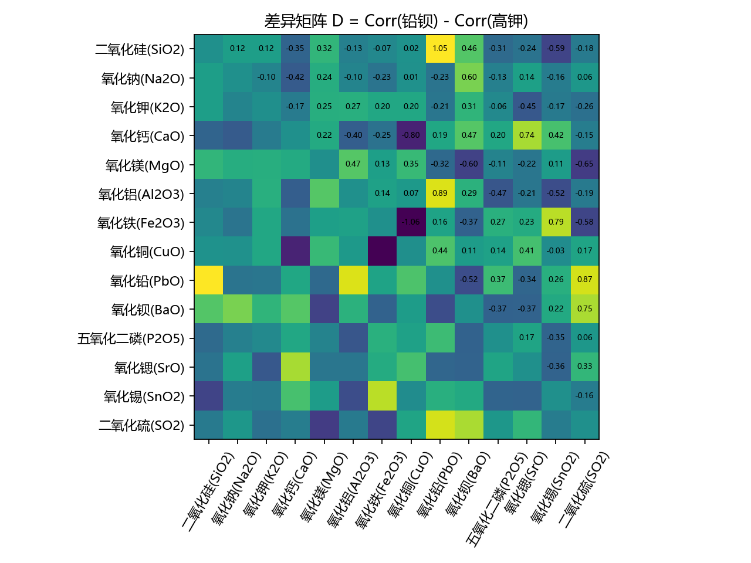


图6 铅钡玻璃与高钾玻璃的差异矩阵

·重点现象：PbO–BaO在铅钡显著更强；K₂O–CaO 在高钾显著更强；SiO₂–(PbO/BaO/K₂O) 呈现骨架–助熔剂的互补差异。理由：分别对应铅/钡助熔与植物灰助熔路径。

**六、模型检验**

## **6.1问题一检验**

·**配对统计：**显著性与效应量（Cohen’s d）并报，避免仅给 p 值。理由：量化风化强度。

·**PLS CV：**主成分数—MSE曲线单峰；留一/5折一致。理由：避免过拟合。

·**残差诊断：**CLR 空间残差零均值、无系统偏差；逆 CLR 后总和误差。理由：保证守恒与可逆。

## **6.2问题二检验**

·**RF：**分层 k 折 ACC/Precision/Recall/F1/AUC；混淆矩阵定位错分（通常靠近决策边界或修复不充分）。理由：覆盖精确率与召回率的权衡。

·**聚类稳健性：**①ARI(Ward,Average)>0.85（经验阈）；②自助法簇内平均共聚概率 >0.90 为“高度稳定”。理由：从算法选择与样本扰动两维验证结构可靠性。

·**可解释性：**特征重要性与簇指纹一致（如铅钡簇内 PbO/BaO 高值），避免“好看但不可解释”的聚类。

## **6.3问题三检验**

·**外推合理性：**对经 PLS 修复的未知样本，检查其 CLR 投影是否落在训练分布邻域（如马氏距离阈值）。理由：防止离群外推。

·**稳定性：**100 次扰动保持率分布应呈双峰靠高端；对低端个体进行个案回溯（特征接近决策边界）。理由：避免盲信单次预测。

## **6.4问题四检验**

·**显著性：**对Top-N 成分对，给 Fisher z 双侧检验或 bootstrap 95% CI；不显著者只做趋势性讨论。理由：防止“显著性错觉”。

·**稳健性：**抽样分半/重复随机种子，差异矩阵模式保持（结构相似度高）。理由：结论不依赖偶然抽样。

**七、模型的评价与改进**

## 7.1模型的优点

1.**理论一致性：**全流程基于 CoDA，消除了常和伪相关，保证统计结论的可解释性。

2.**抗小样本与共线：**PLS 处理“多变量、强相关、小样本”的映射问题，有效“逆风化”。

3.**判别鲁棒：**RF 对噪声与异常值不敏感，分层 CV 给出稳健泛化误差。

4.**可解释性强：**RF 的特征重要性、聚类簇指纹、问题四的差异矩阵形成多层次证据链。

5.**稳定性可量化：**亚类用 ARI+自助法、未知判别用 100 次保持率，均给出数值化的稳定性评估。

6.**可复用与扩展：**流程标准化（脚本化），可迁移至其他文物成分与风化场景。

7.**机制对齐：**差异矩阵结论与配方来源假设（铅/钡助熔、植物灰助熔）一致，增强考古解释可信度。

8.**诊断友好：**提供混淆矩阵、CV 曲线、共聚矩阵、Top-成分对等多角度诊断输出，便于复核与答辩。

## 7.2模型的缺点

1.**PLS 依赖配对样本：**配对不足时逆风化外推不确定性上升。

2.**聚类阈值经验性：**如 ARI>0.85、共聚>0.90 属经验阈，需要结合数据规模校准。

3.**线性相关假设：**问题四以 Pearson 为主，对非线性耦合可能低估。

4.**类别外延受限：**本研究聚焦“高钾/铅钡”，其他谱系未纳入训练。

## 7.3模型的改进

·**稀疏 PLS / 弹性网：**进一步抑制噪声与冗余变量；

·**稳健/图模型相关：**采用 Spearman/距离相关或高斯图模型刻画非线性与条件相关；

·**多模态融合：**引入谱学、形貌等信息进行联合学习；

·**半监督扩展：**利用未标注样本改进边界学习与亚类识别。

**八、参考文献**

[1] Aitchison, J. The Statistical Analysis of Compositional Data. Chapman & Hall, 1986.

[2] Wold, H. “PLS for Multivariate Models”, Journal of Econometrics, 1985.

[3] Breiman, L. “Random Forests”, Machine Learning, 2001.

[4] Filzmoser, P., Hron, K., Templ, M. Applied Compositional Data Analysis. Springer, 2018.

**附录A 支撑材料文件列表**

**表7 文件列表**

|  |  |
| --- | --- |
| **文件名** | **说明** |
| **preprocess.py** | **预处理代码** |
| **readxlsx.py** | **读取数据代码** |
| **Q1.py** | **问题一代码** |
| **Q2.py** | **问题二代码** |
| **Q3.py** | **问题三代码** |
| **Q4.py** | **问题四代码** |

**附录B 支撑材料的所有Python代码**

**表8 preprocess.py 源代码**

|  |
| --- |
| import pandas as pd import numpy as np import re import os  # ================================== # 1. 数据加载与合并 (Data Loading and Merging) # ================================== # 加载三个表单 basepath = os.path.dirname(os.path.abspath(\_\_file\_\_)) data\_dir = os.path.join(basepath, 'C题')  def read\_csv\_robust(filename, encodings=(  'gb18030', # 覆盖范围广，兼容 GBK/GB2312  'gbk', # 常见中文编码  'utf-8-sig',# 处理含 BOM 的 UTF-8  'utf-8' # 纯 UTF-8 )):  *"""  从 `C题` 目录稳健读取 CSV 文件：依次尝试多种编码，读取成功即返回。  - filename: 文件名，例如 '表单1.csv'  - encodings: 依次尝试的编码列表  """* file\_path = os.path.join(data\_dir, filename)  if not os.path.exists(file\_path):  raise FileNotFoundError(f"未找到数据文件: {file\_path}")   last\_error = None  for enc in encodings:  try:  return pd.read\_csv(file\_path, encoding=enc)  except UnicodeDecodeError as e:  last\_error = e  continue   raise RuntimeError(  f"无法用这些编码读取文件 {file\_path}: {list(encodings)}; 最后错误: {last\_error}"  )  # 读取三个表单（固定目录 + 多编码尝试） df1 = read\_csv\_robust('表单1.csv') df2 = read\_csv\_robust('表单2.csv') df3\_unknown = read\_csv\_robust('表单3.csv')   # --- 数据清洗与合并 --- # 函数：从“文物采样点”中提取“文物编号” def get\_wenwu\_id(s):  match = re.match(r'(\d+)', str(s))  return int(match.group(1)) if match else None  # 为表单2创建“文物编号”列，用于合并 df2['文物编号'] = df2['文物采样点'].apply(get\_wenwu\_id)  # 将表单1和表单2合并。表单2中的每个采样点都是一个独立样本。 # 我们使用左合并，以df2为基础，补充df1中的信息。 merged\_df = pd.merge(df2, df1, on='文物编号', how='left')  # 提取化学成分列的列名 chemical\_cols = df2.columns.drop(['文物采样点', '文物编号'])  # ================================== # 2. 数据预处理函数 (Data Preprocessing Functions) # ================================== # --- 零值处理：乘法替换法 (Multiplicative Replacement for Zeros) --- # 该方法用一个小值δ替换零值，并按比例缩放其他值，以保持数据结构。 def multiplicative\_replacement(df, cols, delta=1e-5):  *"""  对DataFrame中的指定列应用乘法替换。  df: a pandas DataFrame.  cols: a list of column names for compositional data.  delta: a small value to replace zeros.  """* df\_copy = df.copy()  # 仅对成分数据进行操作  comp\_data = df\_copy[cols].values    # 对每一行（样本）进行处理  for i in range(comp\_data.shape[0]):  row = comp\_data[i]  zeros = (row == 0) | np.isnan(row)  non\_zeros = ~zeros    c = np.sum(zeros) # 零值的数量  if c > 0:  # 用delta替换零值  row[zeros] = delta    # 按比例缩放非零值  s\_nonzero = np.sum(row[non\_zeros])  row[non\_zeros] = row[non\_zeros] \* (1 - c \* delta) / s\_nonzero    # 将处理后的行写回  comp\_data[i] = row    df\_copy[cols] = comp\_data  return df\_copy  # --- CLR变换 (Centered Log-Ratio Transformation) --- # 这是成分数据分析的核心，将数据从约束空间转换到欧几里得空间。 def clr\_transform(df, cols):  *"""  对DataFrame中的指定列进行CLR变换。  """* df\_processed = multiplicative\_replacement(df, cols)  comp\_data = df\_processed[cols].values    # 避免log(0)  # 乘法替换已经处理了零值，但以防万一  comp\_data[comp\_data <= 0] = 1e-9     # 计算几何平均值  geom\_mean = np.exp(np.mean(np.log(comp\_data), axis=1, keepdims=True))    # CLR变换  clr\_data = np.log(comp\_data / geom\_mean)    clr\_df = pd.DataFrame(clr\_data, columns=[f"CLR\_{col}" for col in cols], index=df.index)  return clr\_df  # --- 逆CLR变换 (Inverse CLR Transformation) --- # 将CLR变换后的数据还原为原始比例。 def inverse\_clr\_transform(clr\_data):  *"""  对CLR变换后的数据进行逆变换。  clr\_data: a numpy array or DataFrame of CLR-transformed data.  """* if isinstance(clr\_data, pd.DataFrame):  clr\_data = clr\_data.values    exp\_data = np.exp(clr\_data)  sum\_exp = np.sum(exp\_data, axis=1, keepdims=True)    # 归一化并乘以100，得到百分比  original\_composition = (exp\_data / sum\_exp) \* 100  return original\_composition   # --- 应用预处理 --- # 对已分类数据应用CLR变换 # 填充NaN值为0，因为它们代表“未检测到” merged\_df[chemical\_cols] = merged\_df[chemical\_cols].fillna(0) clr\_df\_classified = clr\_transform(merged\_df, chemical\_cols) # 合并CLR变换后的数据和原始标签信息 final\_classified\_df = pd.concat([merged\_df.reset\_index(drop=True), clr\_df\_classified.reset\_index(drop=True)], axis=1)  print("数据准备完成。已分类的数据集包含 {} 个样本。".format(len(final\_classified\_df))) print("已分类数据预览:") print(final\_classified\_df[['文物采样点', '类型', '表面风化', 'CLR\_二氧化硅(SiO2)']].head()) |

**表9 readxlsx.py 源代码**

|  |
| --- |
| ## 读取excel文件 import pandas as pd import os  basepath = os.path.dirname(os.path.abspath(\_\_file\_\_)) excel\_path = os.path.join(basepath, "C题", "附件.xlsx")   xls = pd.ExcelFile(excel\_path) print("工作表:", xls.sheet\_names) # 2) 读取整本工作簿：返回 {sheet\_name: DataFrame} all\_sheets = pd.read\_excel(  excel\_path,  sheet\_name=None, # None 表示读取所有表  engine="openpyxl",  # keep\_default\_na=False, # 空单元不变成NaN，留空  # dtype=str, # 全部按文本读，避免类型混杂 ) for name, df in all\_sheets.items():  df.to\_csv(f"{name}.csv", index=False, encoding="utf-8-sig") |

**表10 Q1.py 源代码**

|  |
| --- |
| import pandas as pd import numpy as np from scipy.stats import chi2\_contingency, ttest\_rel from sklearn.cross\_decomposition import PLSRegression from sklearn.model\_selection import cross\_val\_predict from sklearn.metrics import r2\_score import matplotlib.pyplot as plt import seaborn as sns import preprocess   df1 = preprocess.df1 final\_classified\_df = preprocess.final\_classified\_df chemical\_cols = preprocess.chemical\_cols inverse\_clr\_transform = preprocess.inverse\_clr\_transform  # ============================================================= # 2.1 风化与物理属性的统计关联分析 (Chi-squared Test) # ============================================================= print("\n--- 问题1：风化关系分析与成分预测 ---\n") print("2.1 风化与物理属性的关联分析:")  # 创建一个不含重复文物编号的数据集进行检验，避免样本不独立 # （例如，文物03有两个采样点，但其类型和纹饰是唯一的） unique\_artifacts\_df = df1.drop\_duplicates(subset=['文物编号'])  # 风化 vs. 玻璃类型 contingency\_table\_type = pd.crosstab(unique\_artifacts\_df['表面风化'], unique\_artifacts\_df['类型']) chi2, p, dof, ex = chi2\_contingency(contingency\_table\_type) print(f"风化状态 vs. 玻璃类型 的卡方检验 p-value: {p:.4f}") if p < 0.05:  print("结论: 风化与玻璃类型存在显著关联。") else:  print("结论: 风化与玻璃类型无显著关联。") print(contingency\_table\_type)  # 风化 vs. 纹饰 contingency\_table\_pattern = pd.crosstab(unique\_artifacts\_df['表面风化'], unique\_artifacts\_df['纹饰']) chi2, p, dof, ex = chi2\_contingency(contingency\_table\_pattern) print(f"\n风化状态 vs. 纹饰 的卡方检验 p-value: {p:.4f}") if p < 0.05:  print("结论: 风化与纹饰存在显著关联。") else:  print("结论: 风化与纹饰无显著关联。") print(contingency\_table\_pattern)   # ============================================================= # 2.2 风化化学特征量化 (Quantifying Chemical Effects) # ============================================================= print("\n2.2 风化化学特征量化 (配对t检验):")  # 筛选成对样本 (风化 vs. 未风化) paired\_samples = [] # 示例：'49' (风化) vs '49未风化点' (无风化) # '08' (风化) vs '08严重风化点' (严重风化) -> 这个比较风化程度，也可以用 # 我们需要找到同一文物编号下，风化状态不同的样本 # 手动识别配对样本以确保准确性 # (49, 49未风化点), (50, 50未风化点), (23, 23未风化点), (25, 25未风化点) ... # (08, 08严重风化点), (26, 26严重风化点), (54, 54严重风化点) # 为简化代码，我们手动定义这些对 pairs = {  '49未风化点': '49',   '50未风化点': '50',  '23未风化点': '23', # 假设23本身是风化样本  '25未风化点': '25', # 假设25本身是风化样本  '28未风化点': '28',  '29未风化点': '29',  '42未风化点1': '42',  '44未风化点': '44',  '53未风化点': '53',  '08': '08严重风化点',  '26': '26严重风化点',  '54': '54严重风化点' }  # 统一方向：仅保留“未风化” -> “风化/严重风化”的配对，确保差值方向恒为 (风化 - 未风化) status\_series = final\_classified\_df.set\_index('文物采样点')['表面风化'] pairs\_filtered = {} for k, v in pairs.items():  status\_k = status\_series.get(k, None)  status\_v = status\_series.get(v, None)  if status\_k == '无风化' and status\_v is not None and status\_v != '无风化':  pairs\_filtered[k] = v  if len(pairs\_filtered) == 0:  print("警告：未找到方向一致（未风化→风化）的配对样本，使用原始配对但方向可能不一致。")  pairs\_effective = pairs else:  pairs\_effective = pairs\_filtered  if len(pairs\_effective) < len(pairs):  print(f"已过滤不符合方向的配对 {len(pairs) - len(pairs\_effective)} 对，保留 {len(pairs\_effective)} 对用于统计。")  # 提取CLR变换后的数据 unweathered\_paired = final\_classified\_df[final\_classified\_df['文物采样点'].isin(pairs\_effective.keys())] weathered\_paired = final\_classified\_df[final\_classified\_df['文物采样点'].isin(pairs\_effective.values())]  # 确保配对顺序正确 unweathered\_paired = unweathered\_paired.set\_index('文物采样点') weathered\_paired = weathered\_paired.set\_index('文物采样点') weathered\_paired = weathered\_paired.rename(index={v: k for k, v in pairs\_effective.items()}) unweathered\_paired = unweathered\_paired.reindex(weathered\_paired.index)  clr\_cols = [f"CLR\_{col}" for col in chemical\_cols] diff\_clr = weathered\_paired[clr\_cols].values - unweathered\_paired[clr\_cols].values  print(f"已匹配配对样本数量: {len(weathered\_paired)} 对")  # 对每个化学成分进行配对t检验，并输出方向（均值差） results\_rows = [] for i, col in enumerate(chemical\_cols):  w = weathered\_paired[clr\_cols[i]]  u = unweathered\_paired[clr\_cols[i]]  mean\_delta = (w - u).mean()  t\_stat, p\_val = ttest\_rel(w, u)  direction = '富集(风化↑)' if mean\_delta > 0 else ('流失(风化↓)' if mean\_delta < 0 else '无明显方向')  results\_rows.append({  '成分': col,  '均值差(风化-未风化, CLR)': mean\_delta,  't统计量': t\_stat,  'p值': p\_val,  '方向': direction,  '显著性': '显著' if p\_val < 0.05 else ''  })  results\_df = pd.DataFrame(results\_rows) results\_df = results\_df.sort\_values(by='均值差(风化-未风化, CLR)', key=lambda s: s.abs(), ascending=False)  print("\n风化效应量化（CLR 空间）：") print(results\_df.to\_string(index=False))    # ============================================================= # 2.3 风化逆转预测模型 (PLS Regression Model) # ============================================================= print("\n2.3 建立风化逆转预测模型 (PLS):")  # 准备训练数据 X\_train\_pls = weathered\_paired[clr\_cols] y\_train\_pls = unweathered\_paired[clr\_cols]  # 清洗训练集：去除任何包含缺失值的配对样本，确保 PLS 能正常训练 train\_joined = pd.concat([X\_train\_pls, y\_train\_pls.add\_suffix("\_\_target")], axis=1).dropna() X\_train\_pls = train\_joined[clr\_cols] y\_train\_pls = train\_joined[[c + "\_\_target" for c in clr\_cols]] y\_train\_pls.columns = clr\_cols # 还原列名以匹配后续使用  # 自适应设定 PLS 组分数：受限于样本量、特征数与输出维度 n\_components\_upper = int(min(  max(1, X\_train\_pls.shape[0] - 1), # n\_samples - 1 至少为 1  X\_train\_pls.shape[1], # n\_features  y\_train\_pls.shape[1] # n\_targets )) if n\_components\_upper < 1:  raise ValueError("配对样本数量不足，无法训练 PLS 模型。请检查配对样本是否齐全。")  n\_components = min(10, n\_components\_upper) print(f"PLS 使用的组分数: {n\_components} (上限 {n\_components\_upper})")  # 建立并训练 PLS 模型 pls = PLSRegression(n\_components=n\_components) pls.fit(X\_train\_pls, y\_train\_pls)  # 评估模型性能 y\_pred\_pls = pls.predict(X\_train\_pls) r2 = r2\_score(y\_train\_pls, y\_pred\_pls) print(f"PLS模型在训练集上的 R^2 score: {r2:.4f}")  # --- 应用模型进行预测 --- # 选取一个风化样本作为例子，如 "02" sample\_02\_weathered = final\_classified\_df[final\_classified\_df['文物采样点'] == '02']  if not sample\_02\_weathered.empty:  # 提取其CLR变换后的数据  sample\_02\_clr = sample\_02\_weathered[clr\_cols]    # 使用训练好的PLS模型预测其风化前的CLR值  predicted\_clr = pls.predict(sample\_02\_clr)    # 逆变换回原始百分比成分  predicted\_composition = inverse\_clr\_transform(predicted\_clr)    print("\n示例：预测样本'02'风化前的化学成分:")  original\_comp\_df = pd.DataFrame(sample\_02\_weathered[chemical\_cols].values, columns=chemical\_cols, index=['测量值 (风化)'])  predicted\_comp\_df = pd.DataFrame(predicted\_composition, columns=chemical\_cols, index=['预测值 (风化前)'])    # 显示关键成分的变化  display\_cols = ['二氧化硅(SiO2)', '氧化钾(K2O)', '氧化铅(PbO)', '氧化钡(BaO)']  print(pd.concat([original\_comp\_df[display\_cols], predicted\_comp\_df[display\_cols]])) |

**表11 Q2.py 源代码**

|  |
| --- |
| #!/usr/bin/env python3 # -\*- coding: utf-8 -\*-import argparse, json, math from pathlib import Path from typing import List, Tuple, Dict  import numpy as np import pandas as pd  from sklearn.model\_selection import StratifiedKFold, cross\_val\_predict from sklearn.metrics import (confusion\_matrix, roc\_auc\_score, accuracy\_score,  f1\_score, precision\_score, recall\_score, silhouette\_score,  adjusted\_rand\_score) from sklearn.ensemble import RandomForestClassifier  from scipy.cluster.hierarchy import linkage, dendrogram, fcluster from scipy.spatial.distance import pdist  import matplotlib matplotlib.rc("font", family='Microsoft YaHei') import matplotlib.pyplot as plt import preprocess  # ============================== 工具函数 ==============================  SUB\_MAP = str.maketrans("₀₁₂₃₄₅₆₇₈₉", "0123456789")  def canon\_name(s: str) -> str:  *"""将列名规范化到标准氧化物写法：  - 去空格、去 %、去 ( % ) / （%）  - 下标数字转正常数字（₂→2 等）  - 全角符号转半角  - 常见单位词去除( wt.% / mass% 等 )  """* x = str(s).strip()  x = x.translate(SUB\_MAP) # 下标数字 → 正常数字  x = x.replace("％", "%")  x = x.replace("（", "(").replace("）", ")")  x = x.replace(" ", "")  # 去单位/百分号  for t in ["(%)", "%", "wt.%", "wt%", "mass%", "质量分数", "含量"]:  x = x.replace(t, "")  # 一些常见别名统一  aliases = {  "SiO₂":"SiO2","Al₂O₃":"Al2O3","K₂O":"K2O","Na₂O":"Na2O","CaO":"CaO","MgO":"MgO",  "Fe₂O₃":"Fe2O3","TiO₂":"TiO2","SO₃":"SO3","P₂O₅":"P2O5","MnO₂":"MnO2"  }  x = aliases.get(x, x)  return x  def closure(X: np.ndarray, total: float = 100.0) -> np.ndarray:  s = X.sum(axis=1, keepdims=True)  s[s==0] = 1.0  return X / s \* total  def multiplicative\_replacement(X: np.ndarray, delta: float = 1e-5) -> np.ndarray:  X = X.copy()  zeros = (X <= 0)  if np.any(zeros):  X[zeros] = delta  return closure(X, 100.0)  def clr(X: np.ndarray) -> np.ndarray:  X = multiplicative\_replacement(closure(X, 100.0), delta=1e-5)  g = np.exp(np.log(X).mean(axis=1, keepdims=True))  return np.log(X / g)  def inv\_clr(Z: np.ndarray, total: float = 100.0) -> np.ndarray:  X = np.exp(Z)  return closure(X, total)  def detect\_comp\_cols(df: pd.DataFrame) -> List[str]:  *"""启发式识别成分列（包含氧化物形态），随后再统一规范化"""* cols = []  for c in df.columns:  s = str(c)  if "O" in s and any(ch.isdigit() for ch in s): # 如 SiO2, Al2O3, Fe2O3...  cols.append(c)  elif any(k in s for k in ["SiO","Al2O","K2O","Na2O","CaO","MgO","PbO","BaO","Fe2O","SrO","TiO","MnO","CuO","ZnO","SnO","Sb2O","P2O","SO","Cl"]):  cols.append(c)  # 去重保持顺序  out, seen = [], set()  for c in cols:  if c not in seen:  out.append(c); seen.add(c)  return out  def hierarchical\_cluster(Z: np.ndarray, k: int, method: str = "ward"):  D = pdist(Z, metric="euclidean")  L = linkage(D, method=method)  labels = fcluster(L, t=k, criterion="maxclust")  return labels, L  def choose\_k\_by\_silhouette(Z: np.ndarray, kmin=2, kmax=5, method="ward"):  kmax = max(2, min(kmax, Z.shape[0]-1)) # 保证 kmax 合理  best\_k, best\_score, best\_labels, best\_link = None, -1.0, None, None  for k in range(2, kmax+1):  labels, L = hierarchical\_cluster(Z, k=k, method=method)  try:  sc = silhouette\_score(Z, labels, metric="euclidean")  except Exception:  sc = -1.0  if sc > best\_score:  best\_k, best\_score, best\_labels, best\_link = k, sc, labels, L  return best\_k, best\_score, best\_labels, best\_link  def geo\_mean\_fingerprint(X\_closed: np.ndarray, labels: np.ndarray, comp\_cols: List[str]) -> pd.DataFrame:  Z = clr(X\_closed)  rows = []  for k in sorted(np.unique(labels)):  idx = np.where(labels == k)[0]  Zk = Z[idx]  center = Zk.mean(axis=0, keepdims=True)  comp = inv\_clr(center, total=100.0).ravel()  row = dict(subcluster=int(k), size=int(idx.size))  row.update({c: float(v) for c, v in zip(comp\_cols, comp)})  rows.append(row)  return pd.DataFrame(rows)  # ============================== 主流程 ==============================  def main(base\_dir: str, seed: int, cv: int, boot: int, min\_feats: int, auto\_topk: int, min\_per\_class: int):  outdir = Path("outputs\_q2"); outdir.mkdir(parents=True, exist\_ok=True)  diag = {}   # ---------- 读取数据 ----------  df1 = preprocess.df1  df2 = preprocess.df2   # ---------- 编号/类型/风化列 ----------  id\_col = next((c for c in df1.columns if "编号" in str(c)), None)  type\_col = next((c for c in df1.columns if ("类型" in str(c) or "类别" in str(c))), None)  weather\_col = next((c for c in df1.columns if "风化" in str(c)), None)  if id\_col is None or type\_col is None:  raise ValueError("表单1中未找到‘编号’或‘类型/类别’列")   id\_col2 = id\_col if id\_col in df2.columns else next((c for c in df2.columns if "编号" in str(c)), None)  if id\_col2 is None:  raise ValueError("表单2无法匹配编号列")   meta\_cols = [id\_col, type\_col] + ([weather\_col] if weather\_col else [])  meta = df1[meta\_cols].copy()  meta = meta.rename(columns={id\_col:"ID", type\_col:"TYPE", (weather\_col or "WEATHER"):"WEATHER"})  if not weather\_col:  meta["WEATHER"] = np.nan   data = df2.copy()  if id\_col2 != "ID":  data = data.rename(columns={id\_col2:"ID"})  df = data.merge(meta, on="ID", how="left")   # ---------- 成分列识别 + 规范化 + 合并 ----------  comp\_raw = detect\_comp\_cols(df)  if not comp\_raw:  raise ValueError("未识别到成分列")  # 规范化名到 canon，并聚合同名列（求和）  canon\_map = {c: canon\_name(c) for c in comp\_raw}  df\_comp = df[comp\_raw].copy()  df\_comp.columns = [canon\_map[c] for c in comp\_raw]  df\_comp = df\_comp.groupby(axis=1, level=0).sum() # 合并同名列  comp\_cols = list(df\_comp.columns)   # 有效样本筛选  s = df\_comp.fillna(0).sum(axis=1)  mask\_valid = (s >= 85.0) & (s <= 105.0)  df = df.loc[mask\_valid].reset\_index(drop=True)  df\_comp = df\_comp.loc[mask\_valid].reset\_index(drop=True)   # ---------- 风化筛选（训练集仅无风化；若无该列或取值不匹配，则不过滤） ----------  if "WEATHER" in df.columns:  weather\_str = set(df["WEATHER"].astype(str))  non\_weather\_tokens = {"无风化","未风化","否","无","0","0.0","nan","NaN"}  keep\_vals = [v for v in weather\_str if v in non\_weather\_tokens]  df\_tr = df.copy() if len(keep\_vals)==0 else df[df["WEATHER"].astype(str).isin(keep\_vals)].copy()  df\_comp\_tr = df\_comp.loc[df\_tr.index].reset\_index(drop=True)  df\_tr = df\_tr.reset\_index(drop=True)  else:  df\_tr, df\_comp\_tr = df.copy(), df\_comp.copy()   # ---------- 标签 ----------  def map\_label(s):  s = str(s)  if ("铅" in s) and ("钡" in s): return 1  if ("高钾" in s) or ("钾" in s): return 0  return np.nan  df\_tr["Y"] = df\_tr["TYPE"].map(map\_label)  df\_tr = df\_tr.dropna(subset=["Y"]).reset\_index(drop=True)  df\_comp\_tr = df\_comp\_tr.loc[df\_tr.index].reset\_index(drop=True)  y = df\_tr["Y"].astype(int).to\_numpy()   # ---------- CLR 特征 ----------  X\_closed = closure(df\_comp\_tr.fillna(0).to\_numpy(float), 100.0)  Z = clr(X\_closed)   # ---------- 监督分类（RF + CV） ----------  skf = StratifiedKFold(n\_splits=min(cv, np.unique(y, return\_counts=True)[1].min()), shuffle=True, random\_state=seed)  rf = RandomForestClassifier(n\_estimators=800, random\_state=seed, n\_jobs=-1, class\_weight="balanced\_subsample")  y\_pred = cross\_val\_predict(rf, Z, y, cv=skf, method="predict")  y\_prob = cross\_val\_predict(rf, Z, y, cv=skf, method="predict\_proba")[:,1]  rf.fit(Z, y)   metrics = dict(  acc = float(accuracy\_score(y, y\_pred)),  prec = float(precision\_score(y, y\_pred)),  rec = float(recall\_score(y, y\_pred)),  f1 = float(f1\_score(y, y\_pred)),  auc = float(roc\_auc\_score(y, y\_prob)) if len(np.unique(y))==2 else float("nan")  )  importances = rf.feature\_importances\_  imp\_df = pd.DataFrame({"feature": comp\_cols, "gini\_importance": importances}).sort\_values("gini\_importance", ascending=False)   outdir.mkdir(parents=True, exist\_ok=True)  pd.DataFrame([metrics]).to\_csv(outdir / "rf\_cv\_metrics.csv", index=False, encoding="utf-8-sig")  imp\_df.to\_csv(outdir / "rf\_feature\_importance.csv", index=False, encoding="utf-8-sig")   # 混淆矩阵图  cm = confusion\_matrix(y, y\_pred, labels=[0,1])  fig = plt.figure(figsize=(4,4))  plt.imshow(cm, interpolation='nearest')  plt.title("Confusion Matrix (RF, CLR)")  plt.xticks([0,1], ["高钾(0)", "铅钡(1)"])  plt.yticks([0,1], ["高钾(0)", "铅钡(1)"])  for (i,j), v in np.ndenumerate(cm):  plt.text(j, i, str(v), ha="center", va="center")  plt.xlabel("Predicted"); plt.ylabel("True")  plt.tight\_layout()  fig.savefig(outdir / "rf\_cm.png", dpi=160); plt.close(fig)   # 特征重要性图  fig = plt.figure(figsize=(6,4))  plt.bar(imp\_df["feature"], imp\_df["gini\_importance"])  plt.xticks(rotation=60, ha="right"); plt.ylabel("Gini importance")  plt.title("RandomForest Feature Importance (CLR)")  plt.tight\_layout()  fig.savefig(outdir / "rf\_importance.png", dpi=160); plt.close(fig)   # ---------- 亚类划分：两大类分别 ----------  # 推荐特征（规范化后名称）  rec\_pb = ["PbO","BaO","SiO2","Al2O3","CaO"]  rec\_hk = ["K2O","SiO2","Al2O3","CaO","MgO"]   # 记录诊断信息  diag["comp\_cols\_after\_canon"] = comp\_cols  diag["n\_total\_after\_valid"] = int(df\_tr.shape[0])  diag["class\_counts"] = { "高钾(0)": int((y==0).sum()), "铅钡(1)": int((y==1).sum()) }   summary\_lines = []   for cls, name, rec in [(0,"高钾玻璃", rec\_hk), (1,"铅钡玻璃", rec\_pb)]:  idx = np.where(y==cls)[0]  n = int(idx.size)  if n < max(4, min\_per\_class):  diag[f"{name}\_skip\_reason"] = f"样本数不足 n={n} (<{max(4, min\_per\_class)})"  continue   # 推荐特征可用性  avail = [c for c in rec if c in comp\_cols]  use\_feats = avail.copy()   # 若不足 min\_feats，则自动选择该类样本方差 Top-k  if len(use\_feats) < min\_feats:  Z\_class = Z[idx]  var = np.var(Z\_class, axis=0)  top\_idx = np.argsort(var)[::-1][:auto\_topk]  auto\_feats = [comp\_cols[i] for i in top\_idx]  # 避免重复，把缺口补齐  for f in auto\_feats:  if f not in use\_feats:  use\_feats.append(f)  use\_feats = use\_feats[:max(min\_feats, min(auto\_topk, len(comp\_cols)))]   # 记录诊断  diag[f"{name}\_use\_feats"] = use\_feats  diag[f"{name}\_n"] = n   # 构造子矩阵（使用闭合后的真实比例求指纹；聚类用 CLR）  Zsub = Z[idx][:, [comp\_cols.index(f) for f in use\_feats]]  X\_closed\_feats = closure(df\_comp\_tr.iloc[idx][use\_feats].to\_numpy(float), 100.0)   # 选 k  k\_best, sc, labels, L = choose\_k\_by\_silhouette(Zsub, kmin=2, kmax=5, method="ward")  if k\_best is None:  diag[f"{name}\_skip\_reason"] = "无法确定有效的 k"  continue   # 保存标签与树状图  lab\_df = pd.DataFrame({"global\_index": idx, "class": name, "subcluster": labels})  lab\_df.to\_csv(outdir / f"subcluster\_{name}.csv", index=False, encoding="utf-8-sig")   fig = plt.figure(figsize=(8,4))  dendrogram(L, no\_labels=True)  plt.title(f"{name}（Ward） — 最优k={k\_best}, silhouette={sc:.3f}")  plt.tight\_layout(); fig.savefig(outdir / f"dendrogram\_{name}.png", dpi=150); plt.close(fig)   # 指纹（几何均值，逆CLR）  fp = geo\_mean\_fingerprint(X\_closed\_feats, labels, use\_feats)  fp.to\_csv(outdir / f"subcluster\_{name}\_fingerprint.csv", index=False, encoding="utf-8-sig")   # 敏感性：Ward vs Average（ARI）  labels\_avg, L\_avg = hierarchical\_cluster(Zsub, k=k\_best, method="average")  ari = adjusted\_rand\_score(labels, labels\_avg)  with open(outdir / f"sensitivity\_{name}\_linkage.txt", "w", encoding="utf-8") as f:  f.write(f"Ward vs Average linkage ARI = {ari:.4f}\n")  fig = plt.figure(figsize=(8,4))  dendrogram(L\_avg, no\_labels=True)  plt.title(f"{name}（Average） — k={k\_best}")  plt.tight\_layout(); fig.savefig(outdir / f"dendrogram\_{name}\_average.png", dpi=150); plt.close(fig)   # 自助法共聚频率  rng = np.random.RandomState(seed)  n\_sub = Zsub.shape[0]  co = np.zeros((n\_sub, n\_sub), dtype=float)  B = max(boot, 0)  for b in range(B):  ridx = rng.randint(0, n\_sub, size=n\_sub)  uniq, \_ = np.unique(ridx, return\_inverse=True)  Zb = Zsub[uniq]  labs\_b, \_ = hierarchical\_cluster(Zb, k=k\_best, method="ward")  for a in range(len(uniq)):  for bb in range(a+1, len(uniq)):  if labs\_b[a] == labs\_b[bb]:  ia, ib = uniq[a], uniq[bb]  co[ia, ib] += 1.0; co[ib, ia] += 1.0  if B > 0:  co /= float(B); np.fill\_diagonal(co, 1.0)  np.savetxt(outdir / f"coclustering\_{name}.csv", co, delimiter=",")   # 簇内平均共聚概率  stab\_rows = []  for c in range(1, k\_best+1):  members = np.where(labels == c)[0]  if members.size >= 2:  sub = co[np.ix\_(members, members)]  upp = sub[np.triu\_indices\_from(sub, k=1)]  stab = float(upp.mean()) if upp.size>0 else 1.0  else:  stab = 1.0  stab\_rows.append({"subcluster": c, "avg\_coassignment": stab})  pd.DataFrame(stab\_rows).to\_csv(outdir / f"stability\_{name}.csv", index=False, encoding="utf-8-sig")   summary\_lines.append((name, k\_best, sc, use\_feats, ari))   # ---------- 摘要 ----------  with open(outdir / "Q2\_结果摘要.md", "w", encoding="utf-8") as f:  f.write("# 问题二：分类规律与亚类划分（稳健修正版）\n\n")  f.write("## 监督分类（随机森林，CLR 空间）\n\n")  f.write(pd.DataFrame([metrics]).to\_markdown(index=False))  f.write("\n\n特征重要性：`rf\_feature\_importance.csv`；混淆矩阵：`rf\_cm.png`。\n\n")  f.write("## 亚类划分（层次聚类，Ward）\n\n")  if summary\_lines:  for (name, k\_best, sc, feats, ari) in summary\_lines:  f.write(f"- \*\*{name}\*\*：最优亚类数 k={k\_best}（silhouette={sc:.3f}），Ward↔Average 的 ARI={ari:.3f}；使用变量：{', '.join(feats)}。\n")  f.write(f" 指纹：`subcluster\_{name}\_fingerprint.csv`；标签：`subcluster\_{name}.csv`；树状图：`dendrogram\_{name}.png`。\n")  else:  f.write("- 注意：本次运行未生成亚类结果。请检查 `outputs\_q2/diagnostics.json` 中的样本量与可用特征，或放宽 `--min\_per\_class/--min\_feats`。\n")   # ---------- 诊断信息 ----------  diag["metrics"] = metrics  diag["summary\_lines\_empty"] = (len(summary\_lines)==0)  with open(outdir / "diagnostics.json", "w", encoding="utf-8") as f:  json.dump(diag, f, ensure\_ascii=False, indent=2)  # ============================== 入口 ==============================  if \_\_name\_\_ == "\_\_main\_\_":  parser = argparse.ArgumentParser()  parser.add\_argument("--base\_dir", type=str, default="C题")  parser.add\_argument("--seed", type=int, default=2024)  parser.add\_argument("--cv", type=int, default=5)  parser.add\_argument("--boot", type=int, default=100, help="bootstrap 次数（0 关闭）")  parser.add\_argument("--min\_feats", type=int, default=3, help="推荐特征不足时的最小特征数阈值")  parser.add\_argument("--auto\_topk", type=int, default=5, help="方差兜底挑选的特征个数上限")  parser.add\_argument("--min\_per\_class", type=int, default=4, help="每类参与聚类的最小样本数")  args = parser.parse\_args()  main(args.base\_dir, args.seed, args.cv, args.boot, args.min\_feats, args.auto\_topk, args.min\_per\_class) |

**表12 Q3.py 源代码**

|  |
| --- |
| import pandas as pd import numpy as np from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import cross\_val\_score, StratifiedKFold from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score import matplotlib.pyplot as plt import seaborn as sns from scipy.stats import norm import preprocess import Q1  # 设置中文字体 plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei'] plt.rcParams['axes.unicode\_minus'] = False  # 导入已预处理的数据 final\_classified\_df = preprocess.final\_classified\_df chemical\_cols = preprocess.chemical\_cols clr\_transform = preprocess.clr\_transform multiplicative\_replacement = preprocess.multiplicative\_replacement inverse\_clr\_transform = preprocess.inverse\_clr\_transform df3\_unknown = preprocess.df3\_unknown  # 从Q1导入PLS模型 pls\_model = Q1.pls  print("\n--- 问题3：未知类别玻璃文物鉴定与敏感性分析 ---\n")  # ============================================================= # 3.1 构建玻璃类型分类模型 # ============================================================= print("3.1 构建玻璃类型分类模型:")  # 准备训练数据：使用无风化样本避免风化干扰 training\_data = final\_classified\_df[final\_classified\_df['表面风化'] == '无风化'].copy() print(f"训练样本数量（无风化）: {len(training\_data)}")  # 特征和标签 clr\_cols = [f"CLR\_{col}" for col in chemical\_cols] X\_train = training\_data[clr\_cols].values y\_train = training\_data['类型'].values  # 建立随机森林分类器 rf\_classifier = RandomForestClassifier(  n\_estimators=100,  max\_depth=10,  min\_samples\_split=2,  min\_samples\_leaf=1,  random\_state=42 )  # 训练模型 rf\_classifier.fit(X\_train, y\_train)  # 交叉验证评估 cv\_scores = cross\_val\_score(  rf\_classifier, X\_train, y\_train,   cv=StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42),  scoring='accuracy' )  print(f"交叉验证准确率: {cv\_scores.mean():.4f} (±{cv\_scores.std()\*2:.4f})")  # 特征重要性分析 feature\_importance = pd.DataFrame({  '化学成分': chemical\_cols,  '重要性得分': rf\_classifier.feature\_importances\_ }).sort\_values('重要性得分', ascending=False)  print("\n特征重要性排序（前8位）:") print(feature\_importance.head(8).to\_string(index=False))  # ============================================================= # 3.2 预处理表单3未知样本 # ============================================================= print("\n3.2 预处理表单3未知样本:")  # 处理表单3数据 df3\_processed = df3\_unknown.copy() # 填充缺失值为0（表示未检测到） df3\_processed[chemical\_cols] = df3\_processed[chemical\_cols].fillna(0)  print("表单3样本信息:") for idx, row in df3\_processed.iterrows():  print(f"{row['文物编号']}: {row['表面风化']}")  # ============================================================= # 3.3 处理风化样本 # ============================================================= print("\n3.3 处理风化样本（预测风化前成分）:")  # 识别风化样本 weathered\_samples = df3\_processed[df3\_processed['表面风化'] == '风化'].copy() unweathered\_samples = df3\_processed[df3\_processed['表面风化'] == '无风化'].copy()  print(f"风化样本数量: {len(weathered\_samples)}") print(f"无风化样本数量: {len(unweathered\_samples)}")  # 对风化样本应用CLR变换并使用PLS模型预测风化前成分 processed\_samples\_list = []  if len(weathered\_samples) > 0:  # 对风化样本进行CLR变换  weathered\_clr = clr\_transform(weathered\_samples, chemical\_cols)    # 使用PLS模型预测风化前的CLR值  predicted\_clr = pls\_model.predict(weathered\_clr[clr\_cols])    # 逆变换回原始百分比成分  predicted\_compositions = inverse\_clr\_transform(predicted\_clr)    # 创建预测后的DataFrame  for i, (idx, row) in enumerate(weathered\_samples.iterrows()):  predicted\_row = row.copy()  predicted\_row[chemical\_cols] = predicted\_compositions[i]  processed\_samples\_list.append(predicted\_row)    print(f"\n样本 {row['文物编号']} 风化前成分预测:")  comparison\_df = pd.DataFrame({  '测量值(风化)': weathered\_samples.loc[idx, chemical\_cols],  '预测值(风化前)': predicted\_compositions[i]  }, index=chemical\_cols)    # 显示主要成分  main\_components = ['二氧化硅(SiO2)', '氧化钾(K2O)', '氧化铅(PbO)', '氧化钡(BaO)']  print(comparison\_df.loc[main\_components].round(3))  # 对无风化样本直接使用测量值 for idx, row in unweathered\_samples.iterrows():  processed\_samples\_list.append(row)  # 合并所有处理后的样本 df3\_final = pd.DataFrame(processed\_samples\_list).reset\_index(drop=True)  # ============================================================= # 3.4 对未知样本进行类型预测 # ============================================================= print("\n3.4 未知样本类型预测:")  # 对处理后的数据进行CLR变换 df3\_clr = clr\_transform(df3\_final, chemical\_cols) X\_unknown = df3\_clr[clr\_cols].values  # 进行预测 predictions = rf\_classifier.predict(X\_unknown) prediction\_probs = rf\_classifier.predict\_proba(X\_unknown)  # 获取类别标签 class\_labels = rf\_classifier.classes\_  # 整理预测结果 results = [] for i, (idx, row) in enumerate(df3\_final.iterrows()):  prob\_dict = dict(zip(class\_labels, prediction\_probs[i]))  max\_prob = max(prediction\_probs[i])    results.append({  '文物编号': row['文物编号'],  '表面风化': row['表面风化'],  '预测类型': predictions[i],  '预测概率': max\_prob,  '高钾概率': prob\_dict.get('高钾', 0),  '铅钡概率': prob\_dict.get('铅钡', 0)  })  results\_df = pd.DataFrame(results) print("\n预测结果:") print(results\_df.to\_string(index=False))  # ============================================================= # 3.5 敏感性分析 # ============================================================= print("\n3.5 敏感性分析:")  def sensitivity\_analysis(sample\_clr, classifier, n\_iterations=100, noise\_level=0.05):  *"""  对单个样本进行敏感性分析  """* original\_pred = classifier.predict([sample\_clr])[0]  original\_prob = classifier.predict\_proba([sample\_clr])[0].max()    # 计算CLR数据的标准差用于生成噪声  noise\_std = np.std(sample\_clr) \* noise\_level    stable\_count = 0  prob\_variations = []    for \_ in range(n\_iterations):  # 添加随机噪声  noise = np.random.normal(0, noise\_std, size=sample\_clr.shape)  perturbed\_sample = sample\_clr + noise    # 进行预测  pred = classifier.predict([perturbed\_sample])[0]  prob = classifier.predict\_proba([perturbed\_sample])[0].max()    prob\_variations.append(prob)    if pred == original\_pred:  stable\_count += 1    stability\_score = (stable\_count / n\_iterations) \* 100  prob\_std = np.std(prob\_variations)    return stability\_score, prob\_std  # 对每个未知样本进行敏感性分析 sensitivity\_results = []  print("\n逐个样本敏感性分析:") for i, (idx, row) in enumerate(df3\_final.iterrows()):  sample\_clr = X\_unknown[i]  stability, prob\_std = sensitivity\_analysis(sample\_clr, rf\_classifier)    sensitivity\_results.append({  '文物编号': row['文物编号'],  '表面风化': row['表面风化'],  '预测类型': predictions[i],  '预测概率': prediction\_probs[i].max(),  '分类稳定性(%)': stability,  '概率标准差': prob\_std  })    print(f"样本 {row['文物编号']}: 稳定性 {stability:.0f}%, 概率标准差 {prob\_std:.3f}")  sensitivity\_df = pd.DataFrame(sensitivity\_results)  # ============================================================= # 3.6 综合结果展示 # ============================================================= print("\n3.6 综合分类结果与敏感性分析:")  final\_results = pd.merge(results\_df, sensitivity\_df[['文物编号', '分类稳定性(%)', '概率标准差']], on='文物编号')  # 添加置信度评级 def confidence\_rating(prob, stability):  if prob >= 0.95 and stability >= 95:  return "极高"  elif prob >= 0.85 and stability >= 85:  return "高"  elif prob >= 0.70 and stability >= 70:  return "中等"  else:  return "低"  final\_results['置信度'] = final\_results.apply(  lambda x: confidence\_rating(x['预测概率'], x['分类稳定性(%)']), axis=1 )  print("\n最终分类结果：") display\_cols = ['文物编号', '表面风化', '预测类型', '预测概率', '分类稳定性(%)', '置信度'] print(final\_results[display\_cols].round(3).to\_string(index=False))  # ============================================================= # 3.7 可视化分析 # ============================================================= print("\n3.7 生成可视化图表...")  # 创建图表 fig, axes = plt.subplots(2, 2, figsize=(15, 12))  # 1. 预测概率分布 axes[0, 0].bar(final\_results['文物编号'], final\_results['预测概率'],   color=['red' if x == '铅钡' else 'blue' for x in final\_results['预测类型']]) axes[0, 0].set\_title('各样本预测概率') axes[0, 0].set\_xlabel('文物编号') axes[0, 0].set\_ylabel('预测概率') axes[0, 0].axhline(y=0.8, color='orange', linestyle='--', label='高置信度阈值') axes[0, 0].legend()  # 2. 稳定性分析 axes[0, 1].bar(final\_results['文物编号'], final\_results['分类稳定性(%)'],  color=['red' if x == '铅钡' else 'blue' for x in final\_results['预测类型']]) axes[0, 1].set\_title('分类稳定性分析') axes[0, 1].set\_xlabel('文物编号') axes[0, 1].set\_ylabel('稳定性 (%)') axes[0, 1].axhline(y=85, color='orange', linestyle='--', label='高稳定性阈值') axes[0, 1].legend()  # 3. 特征重要性 top\_features = feature\_importance.head(8) axes[1, 0].barh(range(len(top\_features)), top\_features['重要性得分']) axes[1, 0].set\_yticks(range(len(top\_features))) axes[1, 0].set\_yticklabels(top\_features['化学成分']) axes[1, 0].set\_title('关键化学成分重要性') axes[1, 0].set\_xlabel('重要性得分')  # 4. 置信度分布 confidence\_counts = final\_results['置信度'].value\_counts() axes[1, 1].pie(confidence\_counts.values, labels=confidence\_counts.index, autopct='%1.1f%%') axes[1, 1].set\_title('预测置信度分布')  plt.tight\_layout() plt.savefig('Q3\_analysis\_results.png', dpi=300, bbox\_inches='tight') plt.show()  # ============================================================= # 3.8 详细敏感性分析报告 # ============================================================= print("\n3.8 详细敏感性分析报告:")  print("\n各样本的分类可靠性评估:") for \_, row in final\_results.iterrows():  status = ""  if row['置信度'] == "极高":  status = "分类结果极为可靠，预测概率高且对扰动不敏感"  elif row['置信度'] == "高":  status = "分类结果可靠，具有较高的置信度"  elif row['置信度'] == "中等":  status = "分类结果基本可靠，但存在一定不确定性"  else:  status = "分类结果不确定，建议进一步验证"    print(f"样本 {row['文物编号']}: {row['预测类型']} (概率:{row['预测概率']:.3f}, 稳定性:{row['分类稳定性(%)']:.1f}%) - {status}")  print(f"\n整体分析总结:") print(f"- 共分析未知样本: {len(final\_results)} 个") print(f"- 预测为高钾玻璃: {sum(final\_results['预测类型'] == '高钾')} 个") print(f"- 预测为铅钡玻璃: {sum(final\_results['预测类型'] == '铅钡')} 个") print(f"- 高置信度预测: {sum(final\_results['置信度'].isin(['极高', '高']))} 个") print(f"- 平均预测概率: {final\_results['预测概率'].mean():.3f}") print(f"- 平均稳定性: {final\_results['分类稳定性(%)'].mean():.1f}%") |

**表13 Q4.py 源代码**

|  |
| --- |
| #!/usr/bin/env python3 # -\*- coding: utf-8 -\*- import argparse from pathlib import Path import math import numpy as np import pandas as pd import matplotlib matplotlib.rc("font", family='Microsoft YaHei') import matplotlib.pyplot as plt import preprocess  # --------------------------- 成分工具 ---------------------------  def detect\_comp\_cols(df: pd.DataFrame):  keys = ["SiO", "Al2O", "K2O", "Na2O", "CaO", "MgO",  "PbO", "BaO", "Fe2O", "SrO", "TiO", "MnO",  "CuO", "ZnO", "SnO", "Sb2O", "P2O", "SO", "Cl"]  out = []  for c in df.columns:  s = str(c)  if any(k in s for k in keys) or (("O" in s) and any(ch.isdigit() for ch in s)):  out.append(c)  seen=set(); cols=[]  for c in out:  if c not in seen:  cols.append(c); seen.add(c)  return cols  def closure(X: np.ndarray, total: float = 100.0) -> np.ndarray:  s = X.sum(axis=1, keepdims=True)  s[s==0] = 1.0  return X / s \* total  def multiplicative\_replacement(X: np.ndarray, delta: float = 1e-5) -> np.ndarray:  X = X.copy()  zeros = (X <= 0)  if np.any(zeros):  X[zeros] = delta  return closure(X, 100.0)  def clr(X: np.ndarray) -> np.ndarray:  X = multiplicative\_replacement(closure(X, 100.0), delta=1e-5)  g = np.exp(np.log(X).mean(axis=1, keepdims=True))  return np.log(X / g)  def valid\_mask(df: pd.DataFrame, comp\_cols):  s = df[comp\_cols].fillna(0).sum(axis=1)  return (s >= 85.0) & (s <= 105.0)  # --------------------------- IO ---------------------------  def read\_forms(base\_dir: str):  base = Path(base\_dir)  f1 = base / "表单1.csv"  f2 = base / "表单2.csv"  df1 = pd.read\_csv(f1, encoding="utf-8")  try:  df2 = pd.read\_csv(f2, encoding="utf-8")  except UnicodeDecodeError:  df2 = pd.read\_csv(f2, encoding="gbk")  return df1, df2  def find\_cols(df1: pd.DataFrame):  id\_col = next((c for c in df1.columns if "编号" in str(c)), None)  type\_col = next((c for c in df1.columns if ("类型" in str(c) or "类别" in str(c))), None)  if id\_col is None or type\_col is None:  raise ValueError("表单1中未找到‘编号’或‘类型/类别’列")  return id\_col, type\_col  def match\_id\_col(df: pd.DataFrame, id\_col1: str):  return id\_col1 if id\_col1 in df.columns else next((c for c in df.columns if "编号" in str(c)), None)  # --------------------------- 统计与绘图 ---------------------------  def fisher\_z(r):  # clip to valid range to avoid inf  r = max(min(r, 0.999999), -0.999999)  return 0.5 \* math.log((1+r)/(1-r))  def norm\_cdf(x):  # standard normal CDF via erf  return 0.5 \* (1 + math.erf(x / math.sqrt(2)))  def correlation\_by\_class(df1, df2, comp\_cols):  id\_col, type\_col = find\_cols(df1)  id2 = match\_id\_col(df2, id\_col)  meta = df1[[id\_col, type\_col]].copy().rename(columns={id\_col:"ID", type\_col:"TYPE"})  data = df2.copy()  if id2 != "ID":  data = data.rename(columns={id2:"ID"})  df = data.merge(meta, on="ID", how="left")  df = df.loc[valid\_mask(df, comp\_cols)].reset\_index(drop=True)   def map\_cls(s):  s = str(s)  if ("铅" in s) and ("钡" in s):  return "铅钡玻璃"  if ("高钾" in s) or ("钾" in s):  return "高钾玻璃"  return np.nan   df["CLS"] = df["TYPE"].map(map\_cls)  df = df.dropna(subset=["CLS"]).copy()   X = df[comp\_cols].fillna(0).to\_numpy(float)  Z = clr(closure(X, 100.0))  dfZ = pd.DataFrame(Z, columns=comp\_cols)  dfZ["CLS"] = df["CLS"].values   Z\_hk = dfZ[dfZ["CLS"]=="高钾玻璃"][comp\_cols].to\_numpy()  Z\_pb = dfZ[dfZ["CLS"]=="铅钡玻璃"][comp\_cols].to\_numpy()   if Z\_hk.shape[0] < 5 or Z\_pb.shape[0] < 5:  raise ValueError("两类样本数不足（<5）无法稳定估计相关矩阵。")   C\_hk = np.corrcoef(Z\_hk, rowvar=False)  C\_pb = np.corrcoef(Z\_pb, rowvar=False)  return C\_hk, C\_pb, Z\_hk.shape[0], Z\_pb.shape[0]  def plot\_heatmap(mat, labels, title, outpath):  fig = plt.figure(figsize=(8,6))  plt.imshow(mat, vmin=-1, vmax=1)  plt.title(title)  plt.xticks(range(len(labels)), labels, rotation=60)  plt.yticks(range(len(labels)), labels)  # 仅在上三角打印数值，避免遮挡  n = len(labels)  for i in range(n):  for j in range(i+1, n):  plt.text(j, i, f"{mat[i,j]:.2f}", ha="center", va="center", fontsize=6)  plt.tight\_layout()  fig.savefig(outpath, dpi=160)  plt.close(fig)  def top\_differences(C\_hk, C\_pb, comp\_cols, n1, n2, topk=12):  D = C\_pb - C\_hk  pairs = []  pvals = []  n = len(comp\_cols)  for i in range(n):  for j in range(i+1, n):  r1 = float(C\_hk[i,j]); r2 = float(C\_pb[i,j])  d = float(r2 - r1)  # Fisher z test for difference between two independent correlations  # z = (z2 - z1)/sqrt(1/(n2-3)+1/(n1-3))  if n1 > 3 and n2 > 3:  z1 = fisher\_z(r1); z2 = fisher\_z(r2)  se = math.sqrt(1.0/(n1-3) + 1.0/(n2-3))  z = (z2 - z1)/se  p = 2.0 \* (1.0 - norm\_cdf(abs(z)))  else:  p = float('nan')  pairs.append((comp\_cols[i], comp\_cols[j], r1, r2, d, abs(d), p))  # 排序：先按 |D| 降序，再按 p 值升序（显著性更高优先）  pairs.sort(key=lambda x: (-x[5], x[6] if not math.isnan(x[6]) else 1.0))  df = pd.DataFrame(pairs, columns=["成分1","成分2","Corr\_高钾","Corr\_铅钡","D=铅钡-高钾","|D|","p值(Fisher z)"])  top\_df = df.head(topk).copy()  return D, df, top\_df  def bootstrap\_D(df1, df2, comp\_cols, B=200, seed=2024):  # 返回 D 的均值与标准差矩阵（可选）  rng = np.random.RandomState(seed)  id\_col, type\_col = find\_cols(df1)  id2 = match\_id\_col(df2, id\_col)  meta = df1[[id\_col, type\_col]].copy().rename(columns={id\_col:"ID", type\_col:"TYPE"})  data = df2.copy()  if id2 != "ID":  data = data.rename(columns={id2:"ID"})  df = data.merge(meta, on="ID", how="left")  df = df.loc[valid\_mask(df, comp\_cols)].reset\_index(drop=True)   def map\_cls(s):  s = str(s)  if ("铅" in s) and ("钡" in s): return "铅钡玻璃"  if ("高钾" in s) or ("钾" in s): return "高钾玻璃"  return np.nan   df["CLS"] = df["TYPE"].map(map\_cls)  df = df.dropna(subset=["CLS"]).copy()  X = df[comp\_cols].fillna(0).to\_numpy(float)  Z = clr(closure(X, 100.0))  dfZ = pd.DataFrame(Z, columns=comp\_cols); dfZ["CLS"]=df["CLS"].values   Z\_hk = dfZ[dfZ["CLS"]=="高钾玻璃"][comp\_cols].to\_numpy()  Z\_pb = dfZ[dfZ["CLS"]=="铅钡玻璃"][comp\_cols].to\_numpy()  n1, n2 = Z\_hk.shape[0], Z\_pb.shape[0]   if n1 < 5 or n2 < 5:  raise ValueError("两类样本数不足，无法做 bootstrap 估计。")   mats = []  for b in range(B):  idx1 = rng.randint(0, n1, size=n1)  idx2 = rng.randint(0, n2, size=n2)  C1 = np.corrcoef(Z\_hk[idx1], rowvar=False)  C2 = np.corrcoef(Z\_pb[idx2], rowvar=False)  mats.append(C2 - C1)  mats = np.stack(mats, axis=0) # (B, p, p)  D\_mean = np.nanmean(mats, axis=0)  D\_std = np.nanstd(mats, axis=0, ddof=1)  return D\_mean, D\_std  # --------------------------- 主流程 ---------------------------  def main(base\_dir: str, topk: int, boot: int, seed: int):  outdir = Path("outputs\_q4"); outdir.mkdir(parents=True, exist\_ok=True)  df1 = preprocess.df1  df2 = preprocess.df2  comp\_cols = detect\_comp\_cols(df2)  if not comp\_cols:  raise ValueError("未识别到成分列")  C\_hk, C\_pb, n1, n2 = correlation\_by\_class(df1, df2, comp\_cols)   # 保存矩阵  pd.DataFrame(C\_hk, index=comp\_cols, columns=comp\_cols).to\_csv(outdir / "Q4\_corr\_高钾.csv", encoding="utf-8-sig")  pd.DataFrame(C\_pb, index=comp\_cols, columns=comp\_cols).to\_csv(outdir / "Q4\_corr\_铅钡.csv", encoding="utf-8-sig")   # 差异与Top对  D, df\_allpairs, top\_df = top\_differences(C\_hk, C\_pb, comp\_cols, n1, n2, topk=topk)  pd.DataFrame(D, index=comp\_cols, columns=comp\_cols).to\_csv(outdir / "Q4\_corr\_差异矩阵铅钡减高钾.csv", encoding="utf-8-sig")  df\_allpairs.to\_csv(outdir / "Q4\_所有成分对\_差异与p值.csv", index=False, encoding="utf-8-sig")  top\_df.to\_csv(outdir / f"Q4\_差异Top{topk}对.csv", index=False, encoding="utf-8-sig")   # 绘图（Matplotlib，单图，无设定颜色）  plot\_heatmap(C\_hk, comp\_cols, "高钾玻璃：CLR相关矩阵", outdir / "Q4\_heatmap\_高钾.png")  plot\_heatmap(C\_pb, comp\_cols, "铅钡玻璃：CLR相关矩阵", outdir / "Q4\_heatmap\_铅钡.png")  plot\_heatmap(D, comp\_cols, "差异矩阵 D = Corr(铅钡) - Corr(高钾)", outdir / "Q4\_heatmap\_差异矩阵.png")   # （可选）bootstrap 稳健性  if boot and boot > 0:  D\_mean, D\_std = bootstrap\_D(df1, df2, comp\_cols, B=boot, seed=seed)  pd.DataFrame(D\_mean, index=comp\_cols, columns=comp\_cols).to\_csv(outdir / "Q4\_bootstrap\_D均值.csv", encoding="utf-8-sig")  pd.DataFrame(D\_std, index=comp\_cols, columns=comp\_cols).to\_csv(outdir / "Q4\_bootstrap\_D标准差.csv", encoding="utf-8-sig")   # 摘要  with open(outdir / "Q4\_结果摘要.md", "w", encoding="utf-8") as f:  f.write("# 问题四：不同类别的成分关联差异（按思路文档实现）\n\n")  f.write(f"- 样本量：高钾 n={n1}，铅钡 n={n2}\n")  f.write("- 相关矩阵：见 `Q4\_corr\_高钾.csv`、`Q4\_corr\_铅钡.csv`；差异矩阵：`Q4\_corr\_差异矩阵铅钡减高钾.csv`。\n")  f.write(f"- |D| 最大的前 {topk} 对见 `Q4\_差异Top{topk}对.csv`；完整成分对与显著性见 `Q4\_所有成分对\_差异与p值.csv`。\n")  if boot and boot > 0:  f.write(f"- Bootstrap(B={boot}) 稳健性：`Q4\_bootstrap\_D均值.csv`、`Q4\_bootstrap\_D标准差.csv`。\n")   print("完成。输出目录：", str(outdir.resolve()))  if \_\_name\_\_ == "\_\_main\_\_":  parser = argparse.ArgumentParser()  parser.add\_argument("--base\_dir", type=str, default="C题")  parser.add\_argument("--topk", type=int, default=12)  parser.add\_argument("--boot", type=int, default=0, help="bootstrap 次数（默认0=不启用）")  parser.add\_argument("--seed", type=int, default=2024)  args = parser.parse\_args()  main(args.base\_dir, args.topk, args.boot, args.seed) |