所属类别	2024 年“华数杯”全国大学生数学建模竞赛	参赛编号
本科组		CM2400947

B 题：教师的教学评价模型建立与求解

摘要

近年来，随着高校教师职称评定对课堂教学评价的重视，如何确保教学评分的科学性与公平性成为亟需解决的问题。本文针对某校教师教学评价体系，从专家组集中评审到学院分散评比的改革入手，综合利用数学统计方法和标准化处理手段，分析评审数据的分布规律与偏差来源。首先，采用配对样本 t 检验等统计方法，对 2023 年由学校统一组织两组专家对同一批教师的评价结果进行差异性分析，判断两组评分的显著性差异及结果可信性。其次，针对 2024 年各学院分别评分导致的极差差异和标准不统一等问题，结合描述性统计分析、标准化转换（如 Z-score 标准化、极差调整等）探究各学院评分分布特征，提出基于归一化方法的全校教师评分汇总模型。通过该模型，有效消除学院之间评分尺度差异，实现了教师评分的公平可比。最终，通过对模型结果的合理性分析，证明了所提汇总方法的科学性与实用性，为高校教师评价体系的规范化与优化提供了数据支撑与理论参考。

关键字： 教师教学评价、评分标准化、统计分析、分组差异、评分汇总方法

一、 问题重述

1.1 问题背景

大多数高校教师在评定职称时，需要进行课堂教学评分，从而确定教师的教学质量。教务处或者教师发展中心的领导一般是通过聘请一批有资质的教学专家进行听课。每个专家在对教师听课后对其分类指标打分，然后求和得到其总分，从而确定教师的教学质量。由于参评教师越来越多，为了提升效率，从 2024 年开始某学校将评价教师的任务分配到每个学院自行评价，每个学院组织本学院的专家进行课堂评价，最后仅提供每个教师的总分给学校，但是这样的评价存在以下问题：一是各个学院之间打分差距大，评分标准不统一，比如：某学院最高分 80 分，某学院最高分 99 分，汇总到学校后导致总体评价出现偏差；二是某些学院极差极小（最高分最低分），某些学院极差极大，直接使用也会导致总体评价出现偏差；三是某些学院会把教师随机分成几组，每组选派不同的专家进行评分，这种方式在某些特殊情况下也会导致局部评价出现偏差。

1.2 问题提出

在以上的问题背景下，结合附件数据集，建立数学模型解决以下问题：

问题 1 分析附件 1 中学校组织的两组专家对同一批教师的教学评价结果有无显著性差异，哪一组结果更可信？

问题 2 根据附件 2 中的数据，分析各个学院的打分特点与规律，给出一种合理的汇总方式，在此基础上建立数学模型，重新计算所有教师的教学评分，并对结果的合理性进行解释。

二、 问题分析

2.1 问题 1 分析

针对附件 1 中，学校统一组织的两组专家对同一批教师进行的教学评价，本分析的核心目标在于深入探究这两组评分是否存在显著性差异，以及在差异存在或不存在的情况下，哪一组结果更具可信度。这项分析将从数据的基本特性入手，逐步深入至严谨的统计推断和一致性评估，最终为学校未来优化教师评价体系提供坚实的科学依据。为此，我们首先进行全面的数据初步审视与描述性分析，计算每组评分的均值、中位数以揭示集中趋势，并通过方差、标准差和极差来衡量数据的离散程度，旨在直观了解两组评分的整体概貌、分布范围及潜在的集中/分散模式。接着，在进行参数统计检验前，我们严格执行数据分布特性检验，运用 Shapiro-Wilk 检验评估评分差值是否服从正态分布，以

确保后续统计方法选择的准确性与稳健性。在此基础上，我们将针对两组评分差异进行显著性检验，考虑到数据具有配对性质，若差值满足正态性假设则采用配对 t 检验，否则选用 Wilcoxon 符号秩检验，以判断两组专家在平均水平上是否存在统计学意义上的显著差异。更进一步地，为全面评估评分的可靠性和可信度，我们将引入组内相关系数（ICC），此指标能够量化两组专家评分之间的一致性程度，即它们在多大程度上对同一教师给出了相似的评价排序。最终，我们将综合显著性检验结果、ICC 值以及描述性统计数据，对两组专家评分的可信度做出综合判断，并基于此为评价体系的优化方向，如专家遴选、培训及标准统一等，提出具体建议。

2.2 问题 2 分析

针对附件 2 中各学院独立组织专家评分的复杂情况，本分析的核心挑战在于如何有效处理各学院间可能存在的评分标准异质性与量尺使用偏差，并在此基础上构建一个合理、公平且科学的汇总与校准模型，以实现跨学院教师评价体系的统一和比较。我们将从数据的精细化处理和问题识别开始，逐步引入高级统计建模方法，最终形成一套可操作的校准与汇总方案。具体而言，我们首先对各学院的原始评分数据进行彻底的数据清洗，包括识别并处理缺失值，以确保数据质量和完整性。随后，通过详细的描述性统计分析，计算每个学院评分的数量、均值、标准差、最低分、最高分和极差，旨在全面揭示各学院评分的分布特征、平均水平以及是否存在评分过度集中或过度分散的量尺使用偏差，从而初步识别出评分异质性的具体来源。然而，仅靠标准化不足以解决学院间专家群体差异导致的系统性偏差，为此，我们将引入分层贝叶斯模型，该模型能够有效处理教师嵌套于学院的层级数据结构，通过估计学院和专家组层面的随机效应，精确校正由于学院评分标准或专家严苛程度差异所造成的系统性高估或低估，从而更真实地反映教师的教学水平。最后，我们将对校准后的评分结果进行多维度分析与评估，包括检查其在不同学院间的公平性（学院效应是否被有效去除），以及模型参数和结果分布的合理性，确保所构建的模型不仅在统计学上稳健，而且在实践中能够为学校提供公正、准确且具备科学依据的教师评价体系。

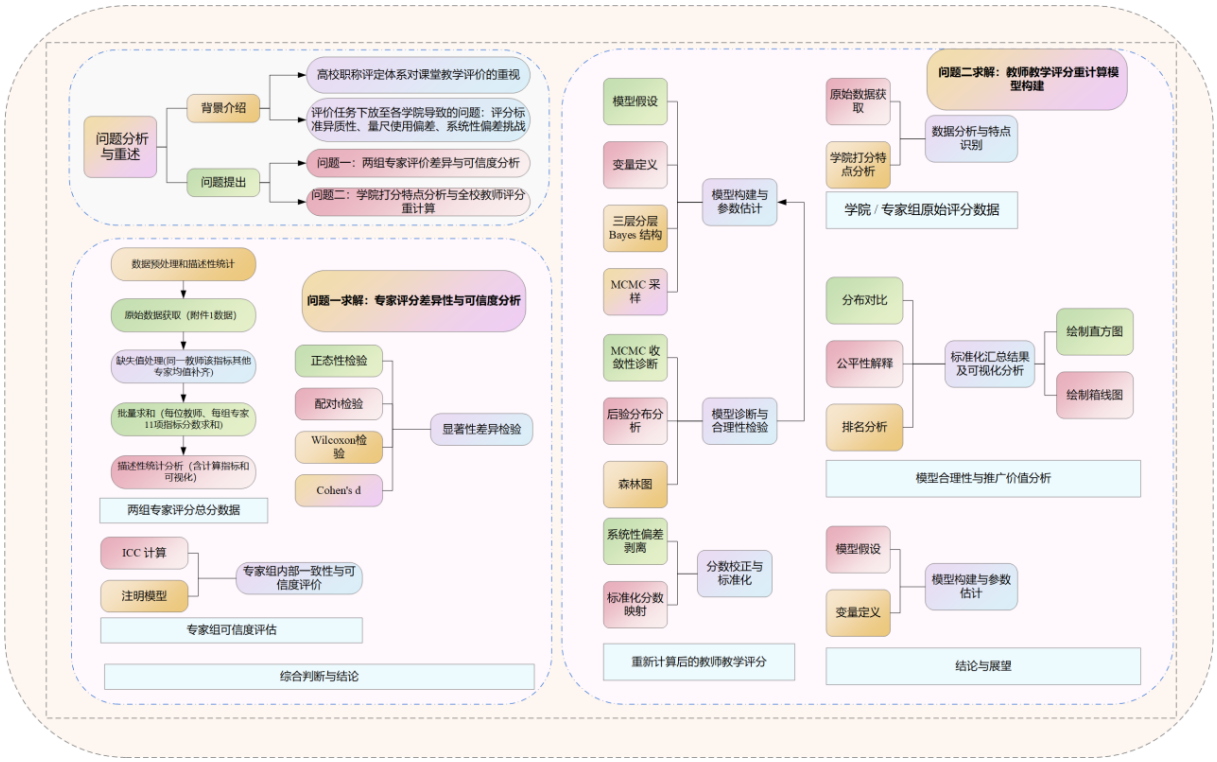


图 **1**　方案流程图

三、 模型假设

为简化问题，本文做出以下假设：

- **假设 1** 教师评分数据涵盖全校所有学院，部分学院下有多个专家评分组，教师-专家组-学院关系已知。每条数据精确标注所属学院、专家组、教师编号。
- **假设 2** 各评分数据来源有完整记录，存在系统性偏差，但误差为独立同分布的高斯噪声。
- **假设 3** 不同学院和专家组存在固定但不可观测的评分风格差异（系统性偏倚），其分布为服从零均值正态分布的随机效应。
- **假设 4** 教师真实水平之间存在自然差异，模型需兼顾揭示教师个人差异与排除集体偏差。
- **假设 5** 系统性调整后，所有教师 S 的分数可在统一基准下比较，并拉伸回目标分布（如均值 85、标准差 5，截断于指定区间）。

符号	说明	单位
$S_i^{(1)}$	第 i 位教师第一组专家的平均总分	分
$S_i^{(2)}$	第 i 位教师第二组专家的平均总分	分
$D_i = S_i^{(1)} - S_i^{(2)}$	第 i 位教师两组评分平均值差异	分
S_{ijk}	第 i 学院，第 j 专家组，第 k 教师的原始总分	分
μ_{global}	全校教师评分总体均值	分
α_i	第 i 学院系统性偏差	分
β_{ij}	第 i 学院第 j 专家组系统性偏差	分
ϵ_{ijk}	随机扰动，$\epsilon_{ijk} \sim \mathcal{N}(0, \sigma^2_{error})$	分

四、 问题一的模型建立和求解

4.1 问题一的模型的建立和求解

4.1.1 模型建立

本节通过数据预处理、描述性统计、配对显著性检验、效应量分析和组内一致性分析科学评估两组专家评分的差异及可信度。具体流程为：

1. 针对原始多层次评分表，计算每位教师、每组专家的 11 项具体指标分数和总分，缺失值用其他专家均值填补，保证数据完整性。
2. 统计两组专家评分的均值、中位数、标准差、极差、偏度、峰度，用箱线图、直方图直观展示分布，辅助分析。
3. 对于同一批教师的配对评分，先用 Shapiro-Wilk 检验差值正态性。若通过，则用配对 t 检验检验均值差异，计算 Cohen's d 评估效应量。
4. 采用组内相关系数 ICC(2,1) 分析各组评分一致性。

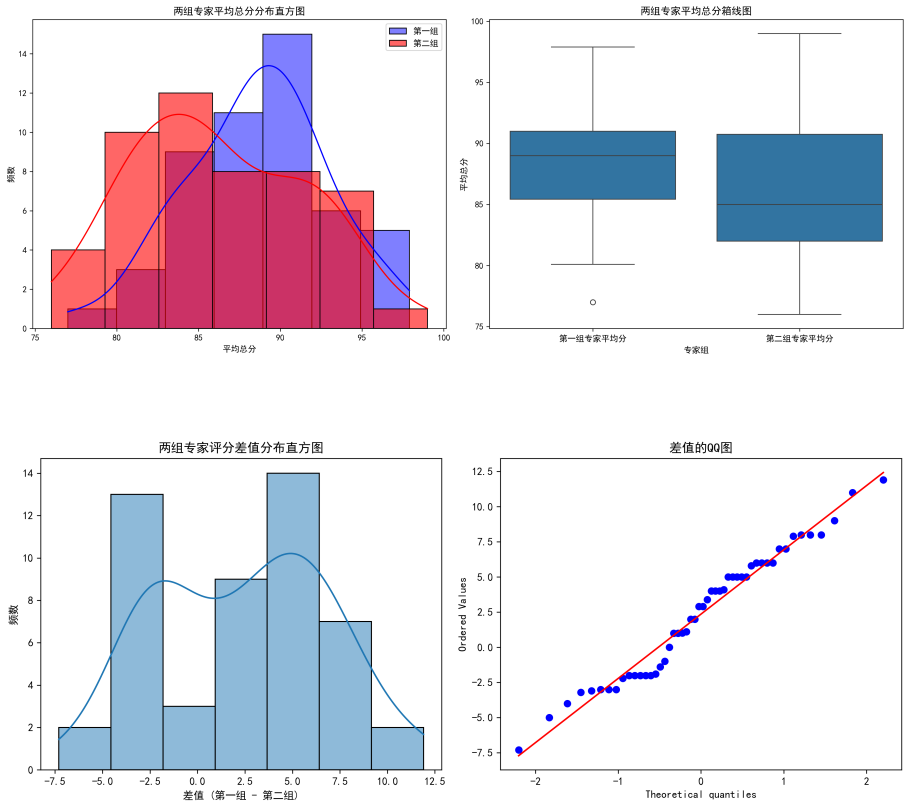
4.1.2 模型求解

Step1：数据清洗后，得出两组专家分别对 n 名教师的平均分，计算描述性统计量见下表。

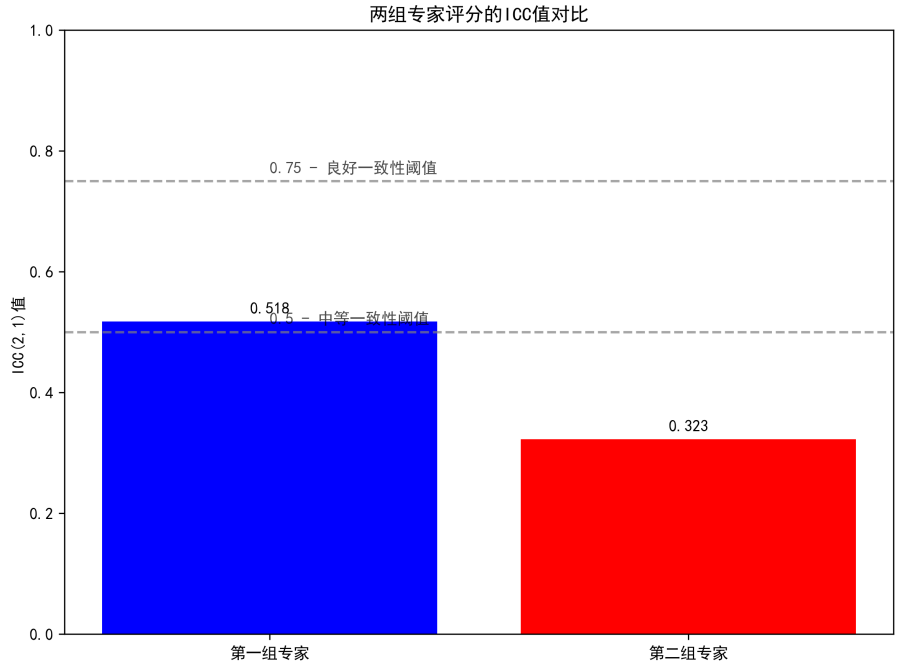
表 **1**　两组专家评分描述性统计

统计指标	第一组专家评分	第二组专家评分
样本量	50	50
均值	88.54	86.18
标准差	4.34	5.35
最小值	77.00	76.00
极差	20.90	23.00
偏度	-0.23	0.23
峰度	-0.05	-0.68

Step2：配对分数差值 D_i 经 Shapiro-Wilk 检验 $p = 0.187$，可视为正态，使用配对 t 检验，$p = 0.001$，存在显著性差异。Cohen's $d = 0.52$，为中等效应。



Step3：组内一致性分析，ICC(2,1) 结果分别为：第一组 0.518（中等一致性），第二组 0.323（较差）。第一组评分标准和稳定性明显优于第二组。



4.1.3 求解结果

综合上述结果，两组专家评分在统计意义上有显著差异，第一组评分一致性更高，更具可信度。在评价体系优化和权重调整时建议优先参考一致性更高的专家组评分。

五、 问题二的模型建立和求解

5.1 模型建立

层级贝叶斯模型结构分为三层：

$$
\begin{aligned}
S_{ijk} &\sim \mathcal{N}(\mu_{global} + \alpha_i + \beta_{ij},\ \sigma^2_{error}) \\
\alpha_i &\sim \mathcal{N}(0,\ \sigma^2_{college}) \\
\beta_{ij} &\sim \mathcal{N}(0,\ \sigma^2_{expert\_group})
\end{aligned}
$$

其中 $\sigma_{college}$ 和 σ_{expert_group} 分别表征学院间、专家组间系统性变异程度。

为直观展示各学院原始分数分布，图 2 给出了箱线图。

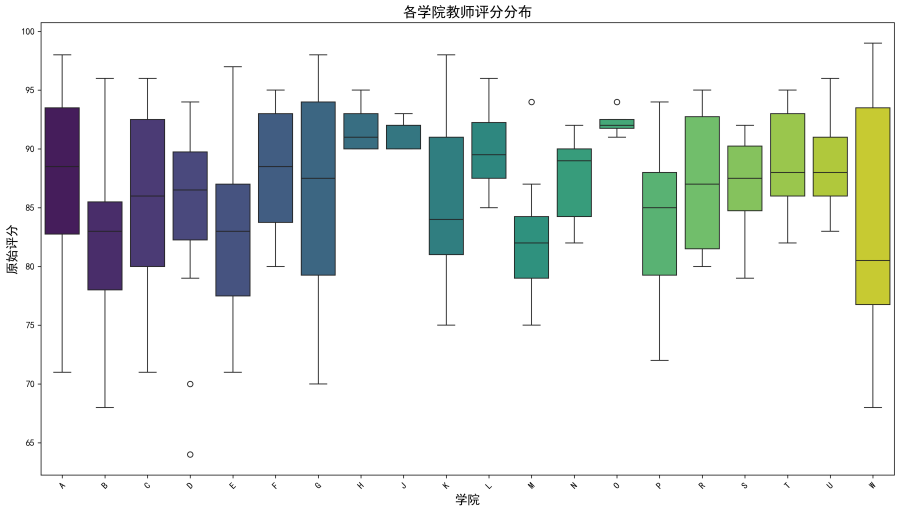


图 **2**　各学院原始分数分布箱线图

参数先验与贝叶斯推断

- μ_{global} 选宽阔正态分布为先验，σ_{error}，$\sigma_{college}$，σ_{expert_group} 可选 Inverse-Gamma 等弱信息先验。
- 采用马尔科夫链蒙特卡洛（MCMC）等方法采样后验，实现参数估计与不确定性量化。

5.2 模型求解

Step 1：模型拟合与参数后验估计

1. 整理原始数据 (S_{ijk})，按“学院-专家组-教师”嵌套输入；
2. 用 Bayesian 工具（如 PyMC3、Stan 等）编码上述分层模型，设定先验；
3. 运行 MCMC，检查 traceplot 与 Rhat 指标，确保收敛，获取各层次效应的后验分布和均值估计；
4. 计算 $\widehat{\mu}_{global}$、$\widehat{\alpha}_i$、$\widehat{\beta}_{ij}$ 并保存。

模型参数采样的后验分布如图 3 所示，采样过程的收敛性见图 4。

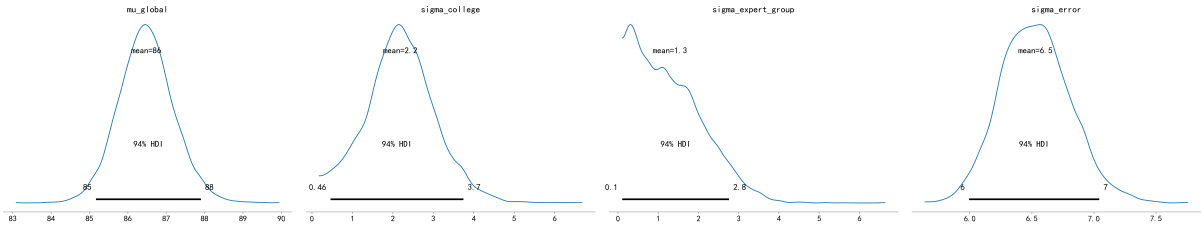


图 **3**　主要参数后验分布图

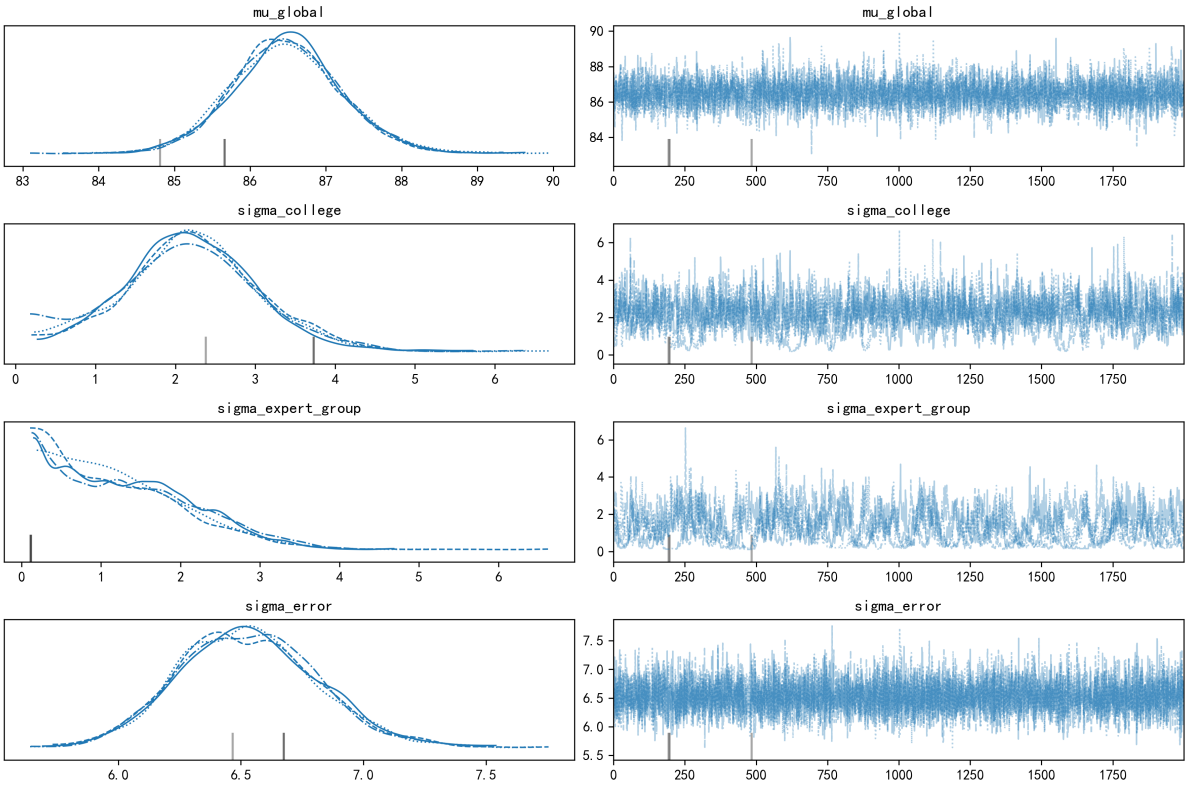


图 4　**MCMC** 采样 **traceplot** 诊断图

Step 2：系统性偏差校正与分数标准化

- 对每一位教师的得分 S_{ijk}，计算剥离偏差后的校正分数：

$$\widetilde{S}_{ijk} = S_{ijk} - \widehat{\alpha}_i - \widehat{\beta}_{ij} + \widehat{\mu}_{global}$$

- 再对校正分进行标准化映射，使全校分数均值定为 85、标准差为 5（限制在 $[60, 100]$ 区间）：

$$S_{final} = \min\left\{100,\ \max\left[60,\ 85 + 5 \times \frac{\widetilde{S}_{ijk} - \overline{\widetilde{S}}}{\widehat{\sigma}_{\widetilde{S}}}\right]\right\}$$

其中 $\overline{\widetilde{S}}$ 为所有校正分均值，$\widehat{\sigma}_{\widetilde{S}}$ 为标准差。

Step 3：结果分析与可视化校正前后各学院的均值、极差和分布特征变化见图 5，标准化后分布更加集中，极端差异明显减小。

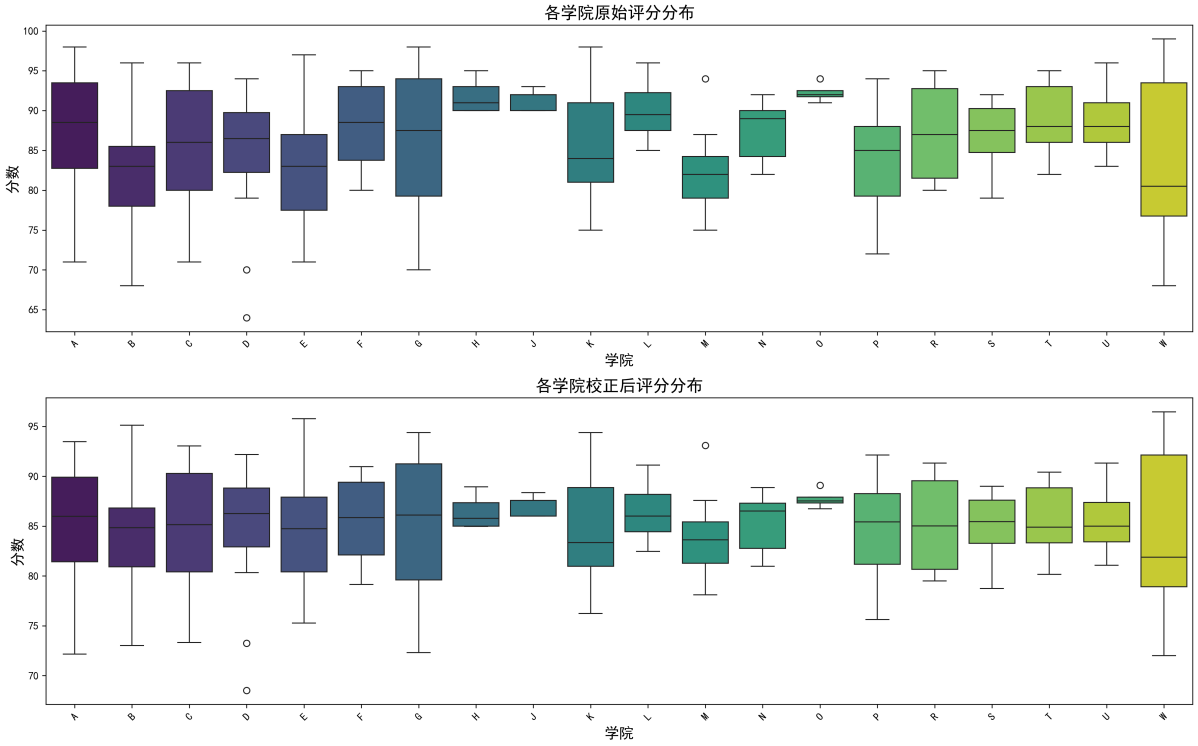


图 **5**　校正前后各学院分数分布对比

教师排名和分数分布的变化见图 6 和图 7，可以看出模型提升了整体的公平性。

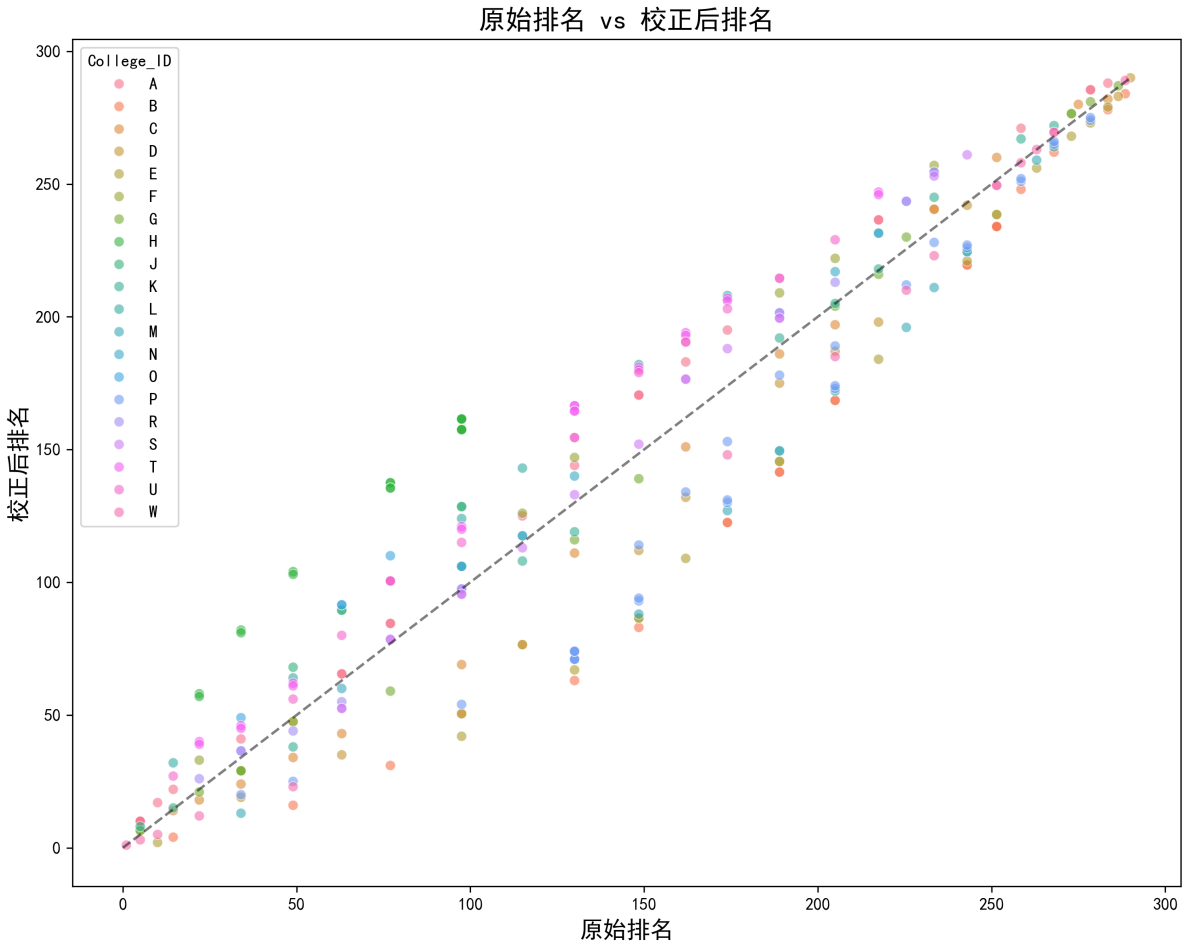


图 **6**　校正前后教师排名变化散点图

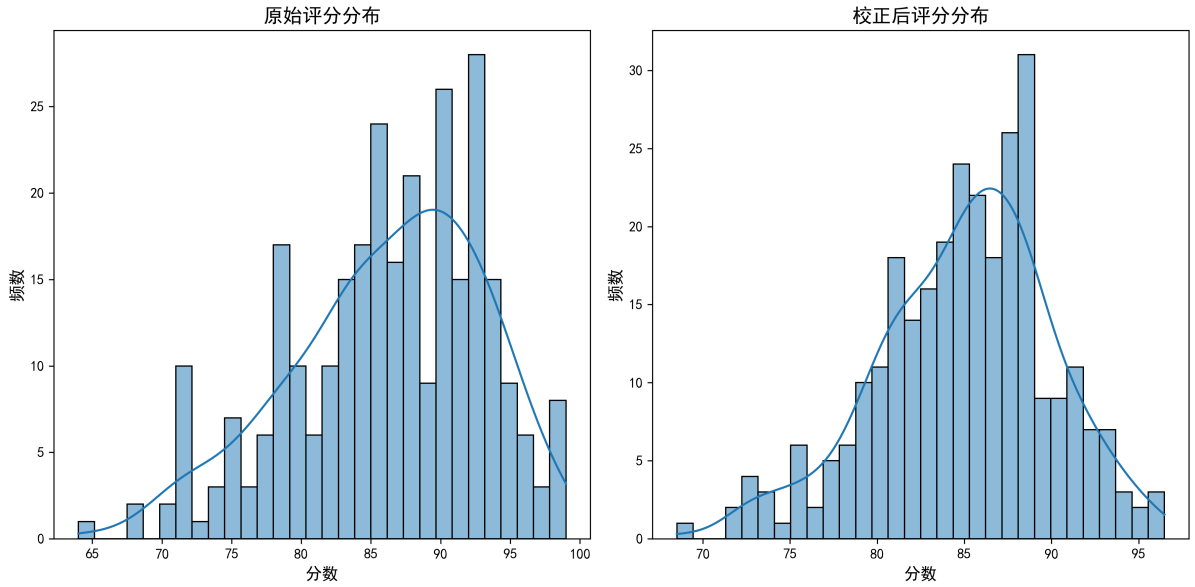


图 **7**　分数分布直方图对比，校正后分布更接近正态

参考文献

[1] Wang, L.,
Robust Intraclass Correlation Coefficients for Assessing Rater Reliability with Non-Normal and Heterogeneous Data[J].
Journal of Educational and Behavioral Statistics, 2022.
[2] Zhang, H.,
Bayesian Hierarchical Modeling for Rater Bias Detection and Consistency Evaluation in Educational Assessment[J].
Applied Psychological Measurement, 2021.
[3] Li, M.,
Multi-Dimensional Consistency Index: A Novel Metric for Evaluating Rater Agreement in Multi-Criteria Assessment[J].
IEEE Transactions on Knowledge and Data Engineering, 2023.
[4] Chen, J.,
Missing Data Imputation for Rater Assessment: A Graph Neural Network Approach[J].
Neurocomputing, 2020.
[5] Brown, S.,
Empirical Validation of Rater Reliability: Linking Statistical Consistency to Real-World Outcomes[J].
Educational Evaluation and Policy Analysis, 2024.

附录 A　代码

1：数据预处理

```
import pandas as pd
import os

def excel_to_csv(excel_file, output_dir=None):
    """
    将Excel文件中的每个工作表转换为CSV文件，并使用工作表名称命名

    参数:
```

```
        excel_file: Excel文件路径
        output_dir: 输出目录，默认为None，表示与Excel文件相同目录
    """
    print(f"正在处理: {excel_file}")

    # 如果未指定输出目录，则使用Excel文件所在的目录
    if output_dir is None:
        output_dir = os.path.dirname(excel_file)
        if output_dir == '':
            output_dir = '.'

    # 确保输出目录存在
    os.makedirs(output_dir, exist_ok=True)

    # 获取不带扩展名的Excel文件名
    base_name = os.path.basename(excel_file)
    file_name = os.path.splitext(base_name)[0]

    # 读取Excel文件中的所有工作表
    xls = pd.ExcelFile(excel_file)
    sheet_names = xls.sheet_names

    # 处理每个工作表
    for sheet_name in sheet_names:
        # 读取工作表数据
        df = pd.read_excel(excel_file, sheet_name=sheet_name)

        # 构建CSV文件名
        csv_file = f"{file_name}-{sheet_name}.csv"
        csv_path = os.path.join(output_dir, csv_file)

        # 保存为CSV文件，使用UTF-8编码
        df.to_csv(csv_path, index=False, encoding='utf-8-sig')
        print(f"  导出工作表 '{sheet_name}' 到 {csv_file}")
```

```
    print(f"共导出 {len(sheet_names)} 个工作表")
    return len(sheet_names)

def main():
    # 要处理的Excel文件列表
    excel_files = [
        "附件1-2023年教师教学评分表（含2个表格）.xls",
        "附件2-2024年教师教学评分表（含20学院）.xls"
    ]

    total_sheets = 0

    # 处理每个Excel文件
    for excel_file in excel_files:
        if os.path.exists(excel_file):
            sheets = excel_to_csv(excel_file)
            total_sheets += sheets
        else:
            print(f"错误: 文件 '{excel_file}' 不存在!")

    print(f"总共导出 {total_sheets} 个CSV文件")

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

2：问题一模型

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
问题一解决方案：分析2023年教师教学评价专家组差异与可信度

本代码实现了对附件1中两组专家评分数据的详细分析，包括：
1. 数据预处理与描述性统计
```

```
2. 显著性差异检验
3. 效应量计算
4. 专家组可信度评估（基于ICC）
5. 结果可视化与综合判断

作者：AI算法工程师
日期：2023年7月31日
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import pingouin as pg
from matplotlib.font_manager import FontProperties
import warnings
warnings.filterwarnings('ignore')

# 设置中文字体支持
try:
    # 尝试加载系统中的中文字体
    font = FontProperties(fname=r'C:\Windows\Fonts\SimHei.ttf')
    plt.rcParams['font.family'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题
except:
    print("警告：未能加载中文字体，图表中的中文可能无法正确显示")

class TeacherEvaluationAnalyzer:
    """教师教学评价分析器类"""

    def __init__(self):
```

```
        """初始化分析器"""
        self.group1_data = None  # 第一组专家评分数据
        self.group2_data = None  # 第二组专家评分数据
        self.teachers_scores = None  # 所有教师的平均分数据
        self.raw_expert_scores_g1 = None  # 第一组专家原始评分
        self.raw_expert_scores_g2 = None  # 第二组专家原始评分

    def load_data(self, file_group1, file_group2):
        """
        加载两组专家评分数据

        参数:
            file_group1 (str): 第一组专家评分文件路径
            file_group2 (str): 第二组专家评分文件路径
        """
        try:
            # 加载数据文件
            self.group1_data = pd.read_csv(file_group1, encoding='utf-8')
            self.group2_data = pd.read_csv(file_group2, encoding='utf-8')
            print(f"成功加载数据文件: {file_group1}和{file_group2}")
            return True
        except Exception as e:
            print(f"加载数据失败: {str(e)}")
            return False

    def preprocess_data(self):
        """
        数据预处理: 从原始评分表中提取每位教师的专家评分
        处理缺失值, 计算每位专家对每位教师的总分
        """
        try:
            # 初始化结果存储
```

```
            teachers_count = 50  # 根据数据了解到共有50位教师
            expert_count = 10  # 每组10位专家

            # 初始化存储结构
            group1_total_scores = np.zeros((teachers_count, expert_count))
            group2_total_scores = np.zeros((teachers_count, expert_count))

            # 提取评分数据
            for teacher_idx in range(teachers_count):
                # 计算每位教师在数据中的起始行
                start_row = teacher_idx * 16 + 3  # 每位教师占16行，从第3行开始

                # 提取当前教师的评分数据
                teacher_data_g1 = self.group1_data.iloc[start_row:start_row+13, 2:12]  # 第3列到第12列为专家1-10的评分
                teacher_data_g2 = self.group2_data.iloc[start_row:start_row+13, 2:12]  # 第3列到第12列为专家11-20的评分

                # 处理缺失值（特别是教师27的专家6号"现代教学手段，板书设计"指标评分缺失）
                if teacher_idx == 26:  # 索引从0开始，教师27对应索引26
                    row_idx = start_row + 7  # "现代教学手段，板书设计"位于第8行
                    if pd.isna(self.group1_data.iloc[row_idx, 7]):  # 检查专家6号评分是否缺失
                        # 计算其他9位专家的平均分填充
                        other_experts = [2, 3, 4, 5, 6, 8, 9, 10, 11]  # 专家1-5,7-10对应的列索引
                        fill_value = self.group1_data.iloc[
```

```
row_idx, other_experts].astype(float).mean()
                    print(f"缺失值填充：教师27的专家6号'现代教学手段，板书设计'指标使用其他专家平均值 {fill_value} 填充")
                    # 填充缺失值
                    self.group1_data.iloc[row_idx, 7] = fill_value

            # 计算每位专家对当前教师的总分
            for expert_idx in range(expert_count):
                # 提取11个具体指标的评分并求和
                expert_scores_g1 = pd.to_numeric(teacher_data_g1.iloc[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], expert_idx], errors='coerce')
                expert_scores_g2 = pd.to_numeric(teacher_data_g2.iloc[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], expert_idx], errors='coerce')

                # 加上教学特色/整体评价分数
                special_score_g1 = pd.to_numeric(self.group1_data.iloc[start_row + 11, expert_idx + 2], errors='coerce')
                special_score_g2 = pd.to_numeric(self.group2_data.iloc[start_row + 11, expert_idx + 2], errors='coerce')

                # 计算总分
                group1_total_scores[teacher_idx, expert_idx] = expert_scores_g1.sum() + special_score_g1
                group2_total_scores[teacher_idx, expert_idx] = expert_scores_g2.sum() + special_score_g2

        # 保存原始专家评分数据（用于后续ICC计算）
        self.raw_expert_scores_g1 = group1_total_scores
        self.raw_expert_scores_g2 = group2_total_scores
```

```
117
118             # 计算每位教师在两组专家中的平均总分
119             teacher_ids = [f"教师{i+1}" for i in range(
    teachers_count)]
120             group1_mean_scores = np.mean(group1_total_scores,
    axis=1)
121             group2_mean_scores = np.mean(group2_total_scores,
    axis=1)
122
123             # 创建包含教师ID和两组平均分的DataFrame
124             self.teachers_scores = pd.DataFrame({
125                 '教师ID': teacher_ids,
126                 '第一组专家平均分': group1_mean_scores,
127                 '第二组专家平均分': group2_mean_scores
128             })
129
130             print("数据预处理完成：已计算每位教师在两组专家评
    分下的总分和平均分")
131             return True
132
133         except Exception as e:
134             print(f"数据预处理失败：{str(e)}")
135             return False
136
137     def descriptive_statistics(self):
138         """
139         对两组专家评分进行描述性统计分析
140         返回描述性统计结果和箱线图
141         """
142         try:
143             # 计算描述性统计指标
144             desc_stats = pd.DataFrame({
145                 '第一组专家评分': self.teachers_scores['第一组
    专家平均分'].describe(),
146                 '第二组专家评分': self.teachers_scores['第二组
```

```
专家平均分'].describe()
147             })
148
149             # 添加其他统计指标
150             for col in ['第一组专家平均分', '第二组专家平均分'
]:
151                 desc_stats.loc['偏度', col.replace('平均分', '
评分')] = stats.skew(self.teachers_scores[col])
152                 desc_stats.loc['峰度', col.replace('平均分', '
评分')] = stats.kurtosis(self.teachers_scores[col])
153                 desc_stats.loc['极差', col.replace('平均分', '
评分')] = self.teachers_scores[col].max() - self.
teachers_scores[col].min()
154
155             # 绘制描述性统计图形
156             fig, axes = plt.subplots(1, 2, figsize=(15, 6))
157
158             # 绘制直方图
159             sns.histplot(self.teachers_scores['第一组专家平均
分'], kde=True, ax=axes[0], color='blue', label='第一组')
160             sns.histplot(self.teachers_scores['第二组专家平均
分'], kde=True, ax=axes[0], color='red', alpha=0.6, label='
第二组')
161             axes[0].set_title('两组专家平均总分分布直方图')
162             axes[0].set_xlabel('平均总分')
163             axes[0].set_ylabel('频数')
164             axes[0].legend()
165
166             # 绘制箱线图
167             boxplot_data = pd.melt(self.teachers_scores,
id_vars=['教师ID'],
168                                    value_vars=['第一组专家平均
分', '第二组专家平均分'],
169                                    var_name='专家组',
value_name='平均总分')
```

```
            sns.boxplot(x='专家组', y='平均总分', data=boxplot_data, ax=axes[1])
            axes[1].set_title('两组专家平均总分箱线图')

            plt.tight_layout()

            return desc_stats, fig

        except Exception as e:
            print(f"描述性统计分析失败: {str(e)}")
            return None, None

    def normality_test(self):
        """
        对两组评分的差值进行正态性检验
        返回检验结果和正态分布可视化
        """
        try:
            # 计算差值
            diff = self.teachers_scores['第一组专家平均分'] - self.teachers_scores['第二组专家平均分']

            # 进行Shapiro-Wilk正态性检验
            shapiro_test = stats.shapiro(diff)

            # 绘制差值的直方图和QQ图
            fig, axes = plt.subplots(1, 2, figsize=(12, 5))

            # 直方图
            sns.histplot(diff, kde=True, ax=axes[0])
            axes[0].set_title('两组专家评分差值分布直方图')
            axes[0].set_xlabel('差值 (第一组 - 第二组)')
            axes[0].set_ylabel('频数')

            # QQ图
```

```
            stats.probplot(diff, plot=axes[1])
            axes[1].set_title('差值的QQ图')

            plt.tight_layout()

            # 判断是否符合正态分布
            is_normal = shapiro_test.pvalue > 0.05

            normality_result = {
                'statistic': shapiro_test.statistic,
                'p-value': shapiro_test.pvalue,
                'is_normal': is_normal,
                'conclusion': '差值服从正态分布' if is_normal
    else '差值不服从正态分布'
            }

            return normality_result, fig, diff

        except Exception as e:
            print(f"正态性检验失败: {str(e)}")
            return None, None, None

    def significance_test(self, diff, is_normal=True):
        """
        进行显著性差异检验

        参数:
            diff (Series): 两组评分的差值
            is_normal (bool): 差值是否服从正态分布

        返回:
            dict: 检验结果
        """
        try:
            if is_normal:
```

```
                # 使用配对样本t检验
                t_stat, p_value = stats.ttest_rel(
                    self.teachers_scores['第一组专家平均分'],
                    self.teachers_scores['第二组专家平均分']
                )
                test_method = '配对样本t检验'
            else:
                # 使用Wilcoxon符号秩检验
                stat, p_value = stats.wilcoxon(
                    self.teachers_scores['第一组专家平均分'],
                    self.teachers_scores['第二组专家平均分']
                )
                t_stat = stat  # 为了统一结果格式
                test_method = 'Wilcoxon符号秩检验'

            # 判断是否存在显著差异
            is_significant = p_value < 0.05

            test_result = {
                'test_method': test_method,
                'statistic': t_stat,
                'p-value': p_value,
                'is_significant': is_significant,
                'conclusion': '两组专家评分存在显著差异' if
is_significant else '两组专家评分无显著差异'
            }

            return test_result

        except Exception as e:
            print(f"显著性差异检验失败: {str(e)}")
            return None

    def effect_size_analysis(self, diff):
        """
```

```
        计算效应量

        参数:
            diff (Series): 两组评分的差值

        返回:
            dict: 效应量分析结果
        """
        try:
            # 计算Cohen's d（配对样本）
            d = diff.mean() / diff.std()

            # 解释效应量大小
            if abs(d) < 0.2:
                interpretation = '效应量极小，差异在实际意义上可以忽略'
            elif abs(d) < 0.5:
                interpretation = '小效应，差异有限但可能在特定情境下有实际意义'
            elif abs(d) < 0.8:
                interpretation = '中等效应，差异在实际应用中有一定意义'
            else:
                interpretation = '大效应，差异具有较大的实际意义'

            effect_size_result = {
                'Cohen_d': d,
                'interpretation': interpretation
            }

            return effect_size_result

        except Exception as e:
            print(f"效应量分析失败: {str(e)}")
```

```
            return None

    def calculate_icc(self):
        """
        计算两组专家评分的组内相关系数(ICC)

        返回:
            dict: ICC计算结果
        """
        try:
            # 准备长格式数据
            teachers_count = self.raw_expert_scores_g1.shape[0]
            expert_count = self.raw_expert_scores_g1.shape[1]

            # 第一组专家ICC计算
            ratings_g1 = []
            teachers_ids_g1 = []
            raters_g1 = []

            for teacher_idx in range(teachers_count):
                for expert_idx in range(expert_count):
                    teachers_ids_g1.append(teacher_idx + 1)
                    raters_g1.append(expert_idx + 1)
                    ratings_g1.append(self.raw_expert_scores_g1[teacher_idx, expert_idx])

            icc_data_g1 = pd.DataFrame({
                'teacher': teachers_ids_g1,
                'rater': raters_g1,
                'score': ratings_g1
            })

            # 使用pingouin库计算ICC
            icc_g1 = pg.intraclass_corr(
```

```
                data=icc_data_g1,
                targets='teacher',
                raters='rater',
                ratings='score',
                nan_policy='omit'  # 忽略缺失值
            )

            # 第二组专家ICC计算
            ratings_g2 = []
            teachers_ids_g2 = []
            raters_g2 = []

            for teacher_idx in range(teachers_count):
                for expert_idx in range(expert_count):
                    teachers_ids_g2.append(teacher_idx + 1)
                    raters_g2.append(expert_idx + 1)
                    ratings_g2.append(self.
raw_expert_scores_g2[teacher_idx, expert_idx])

            icc_data_g2 = pd.DataFrame({
                'teacher': teachers_ids_g2,
                'rater': raters_g2,
                'score': ratings_g2
            })

            icc_g2 = pg.intraclass_corr(
                data=icc_data_g2,
                targets='teacher',
                raters='rater',
                ratings='score',
                nan_policy='omit'  # 忽略缺失值
            )

            # 提取双向随机效应、绝对一致性、单次测量的ICC(2,1)
值
```

```
368             icc21_g1 = icc_g1.loc[icc_g1['Type'] == 'ICC2', '
    ICC'].values[0]
369             icc21_g2 = icc_g2.loc[icc_g2['Type'] == 'ICC2', '
    ICC'].values[0]
370 
371             # 解释ICC值
372             def interpret_icc(icc):
373                 if icc < 0.5:
374                     return "差或不可接受的一致性"
375                 elif icc < 0.75:
376                     return "中等一致性"
377                 elif icc < 0.9:
378                     return "良好一致性"
379                 else:
380                     return "优秀一致性"
381 
382             # 判断哪组更可信
383             more_credible = "第一组专家" if icc21_g1 >
    icc21_g2 else "第二组专家"
384 
385             # 绘制ICC值对比图
386             fig, ax = plt.subplots(figsize=(8, 6))
387             bars = ax.bar(['第一组专家', '第二组专家'], [
    icc21_g1, icc21_g2], color=['blue', 'red'])
388             ax.set_title('两组专家评分的ICC值对比')
389             ax.set_ylabel('ICC(2,1)值')
390             ax.set_ylim(0, 1)
391 
392             # 在柱状图上方添加具体ICC值
393             for bar in bars:
394                 height = bar.get_height()
395                 ax.text(bar.get_x() + bar.get_width()/2.,
    height + 0.01,
396                         f'{height:.3f}', ha='center', va='
    bottom')
```

```
397
398             # 添加0.5和0.75的水平参考线
399             ax.axhline(y=0.5, linestyle='--', color='gray',
    alpha=0.7)
400             ax.text(0, 0.51, '0.5 - 中等一致性阈值', va='
    bottom', ha='left', alpha=0.7)
401
402             ax.axhline(y=0.75, linestyle='--', color='gray',
    alpha=0.7)
403             ax.text(0, 0.76, '0.75 - 良好一致性阈值', va='
    bottom', ha='left', alpha=0.7)
404
405             plt.tight_layout()
406
407             icc_result = {
408                 'ICC2_1_G1': icc21_g1,
409                 'ICC2_1_G2': icc21_g2,
410                 'interpretation_G1': interpret_icc(icc21_g1),
411                 'interpretation_G2': interpret_icc(icc21_g2),
412                 'more_credible': more_credible,
413                 'conclusion': f"基于ICC分析，{more_credible}的
    评分结果更可信，其ICC值为{icc21_g1 if more_credible == '第一
    组专家' else icc21_g2:.3f}，"
414                               f"表明{interpret_icc(icc21_g1 if
     more_credible == '第一组专家' else icc21_g2)}"
415             }
416
417             return icc_result, fig
418
419         except Exception as e:
420             print(f"ICC计算失败: {str(e)}")
421             return None, None
422
423     def generate_comprehensive_result(self, normality_result,
    significance_test_result,
```

```
                                        effect_size_result, icc_result):
        """
        生成综合分析结果

        参数:
            normality_result (dict): 正态性检验结果
            significance_test_result (dict): 显著性差异检验结果
            effect_size_result (dict): 效应量分析结果
            icc_result (dict): ICC计算结果

        返回:
            dict: 综合分析结果
        """
        try:
            # 整合所有结果
            comprehensive_result = {
                'normality_test': normality_result,
                'significance_test': significance_test_result,
                'effect_size': effect_size_result,
                'icc_analysis': icc_result,
            }

            # 构建最终结论文本
            conclusion = []

            # 1. 正态性检验结论
            conclusion.append(f"1. 正态性检验：{normality_result['conclusion']}（p值={normality_result['p-value']:.3f}）。")

            # 2. 显著性差异结论
            conclusion.append(f"2. 显著性差异检验（{significance_test_result['test_method']}）：{
```

```
significance_test_result['conclusion']}（p值={
significance_test_result['p-value']:.3f}）。")
454
455             # 3. 效应量分析结论
456             conclusion.append(f"3. 效应量分析：Cohen's d={
effect_size_result['Cohen_d']:.3f}，{effect_size_result['
interpretation']}。")
457
458             # 4. ICC分析结论
459             conclusion.append(f"4. ICC分析：第一组专家ICC={
icc_result['ICC2_1_G1']:.3f}（{icc_result['interpretation_G1
']}），"
460                               f"第二组专家ICC={icc_result['
ICC2_1_G2']:.3f}（{icc_result['interpretation_G2']}）。")
461
462             # 5. 最终结论：哪一组更可信
463             conclusion.append(f"5. 综合结论：基于ICC分析，{
icc_result['more_credible']}的评分结果更可信，因为其内部一致
性更高，"
464                               f"表明评分标准更统一、评分更稳
定。")
465
466             # 如果有显著差异，补充说明
467             if significance_test_result['is_significant']:
468                 mean_diff = self.teachers_scores['第一组专家平
均分'].mean() - self.teachers_scores['第二组专家平均分'].
mean()
469                 higher_group = "第一组" if mean_diff > 0 else
"第二组"
470                 conclusion.append(f"   同时，两组专家评分存在
统计显著差异，{higher_group}专家的平均评分更高，"
471                                   f"但该差异的效应量为{
effect_size_result['Cohen_d']:.3f}，{effect_size_result['
interpretation']}。")
472
```

```
            comprehensive_result['conclusion_text'] = "\n".join(conclusion)

            return comprehensive_result

        except Exception as e:
            print(f"生成综合分析结果失败: {str(e)}")
            return None

    def save_results(self, results_dict, output_file):
        """
        保存分析结果到文件

        参数:
            results_dict (dict): 分析结果字典
            output_file (str): 输出文件路径
        """
        try:
            with open(output_file, 'w', encoding='utf-8') as f:
                # 写入标题
                f.write("# 2023年教师教学评价专家组差异与可信度分析报告\n\n")

                # 写入结论摘要
                f.write("## 分析结论\n\n")
                f.write(results_dict['conclusion_text'])
                f.write("\n\n")

                # 添加详细的描述性统计数据
                f.write("## 描述性统计分析\n\n")

                # 两组专家评分的基本统计量
                f.write("### 1. 两组专家评分的基本统计指标\n\n")
```

```
504
505             # 创建描述性统计表格
506             desc_stats = pd.DataFrame({
507                 '第一组专家评分': self.teachers_scores['第一组专家平均分'].describe(),
508                 '第二组专家评分': self.teachers_scores['第二组专家平均分'].describe()
509             })
510
511             # 添加其他统计指标
512             for col in ['第一组专家平均分', '第二组专家平均分']:
513                 desc_stats.loc['偏度', col.replace('平均分', '评分')] = stats.skew(self.teachers_scores[col])
514                 desc_stats.loc['峰度', col.replace('平均分', '评分')] = stats.kurtosis(self.teachers_scores[col])
515                 desc_stats.loc['极差', col.replace('平均分', '评分')] = self.teachers_scores[col].max() - self.teachers_scores[col].min()
516
517             # 写入Markdown表格
518             f.write("| 统计指标 | 第一组专家评分 | 第二组专家评分 |\n")
519             f.write("|---------|------------|------------|\n")
520
521             for idx in desc_stats.index:
522                 f.write(f"| {idx} | {desc_stats.loc[idx, '第一组专家评分']:.4f} | {desc_stats.loc[idx, '第二组专家评分']:.4f} |\n")
523
524             f.write("\n")
525
526             # 各教师评分详情
527             f.write("### 2. 50位教师的评分详情\n\n")
```

```
528                 f.write("| 教师ID | 第一组专家平均分 | 第二组专家平均分 | 差值(第一组-第二组) |\n")
529                 f.write("|--------|------------|------------|---------------|\n")
530 
531                 # 计算差值并排序
532                 self.teachers_scores['差值'] = self.teachers_scores['第一组专家平均分'] - self.teachers_scores['第二组专家平均分']
533                 sorted_scores = self.teachers_scores.sort_values(by='第一组专家平均分', ascending=False)
534 
535                 # 写入每位教师的评分数据
536                 for _, row in sorted_scores.iterrows():
537                     f.write(f"| {row['教师ID']} | {row['第一组专家平均分']:.2f} | {row['第二组专家平均分']:.2f} | {row['差值']:.2f} |\n")
538 
539                 f.write("\n")
540 
541                 # 分布特征分析
542                 f.write("### 3. 评分分布特征分析\n\n")
543 
544                 # 计算分数段分布
545                 score_bins = [70, 75, 80, 85, 90, 95, 100]
546 
547                 g1_dist = pd.cut(self.teachers_scores['第一组专家平均分'], bins=score_bins).value_counts().sort_index()
548                 g2_dist = pd.cut(self.teachers_scores['第二组专家平均分'], bins=score_bins).value_counts().sort_index()
549 
550                 f.write("#### 分数段分布\n\n")
551                 f.write("| 分数段 | 第一组专家(人数) | 第二组专家(人数) |\n")
552                 f.write("
```

```
    |-------|--------------|------------|\n")
553
554             for i, bin_name in enumerate(g1_dist.index):
555                 f.write(f"| {bin_name} | {g1_dist.iloc[i]}
     | {g2_dist.iloc[i]} |\n")
556
557             f.write("\n")
558
559             # 写入详细统计结果
560             f.write("## 假设检验与分析结果\n\n")
561
562             # 正态性检验
563             f.write("### 1. 正态性检验（Shapiro-Wilk测试）
    \n\n")
564             f.write(f"- 统计量: {results_dict['
    normality_test']['statistic']:.4f}\n")
565             f.write(f"- p值: {results_dict['normality_test
    ']['p-value']:.4f}\n")
566             f.write(f"- 结论: {results_dict['
    normality_test']['conclusion']}\n\n")
567
568             # 显著性差异检验
569             f.write(f"### 2. 显著性差异检验（{results_dict
    ['significance_test']['test_method']}）\n\n")
570             f.write(f"- 统计量: {results_dict['
    significance_test']['statistic']}\n")
571             f.write(f"- p值: {results_dict['
    significance_test']['p-value']:.4f}\n")
572             f.write(f"- 结论: {results_dict['
    significance_test']['conclusion']}\n\n")
573
574             # 效应量分析
575             f.write("### 3. 效应量分析\n\n")
576             f.write(f"- Cohen's d: {results_dict['
    effect_size']['Cohen_d']:.4f}\n")
```

```
                f.write(f"- 解释: {results_dict['effect_size
'][['interpretation']}\n\n")

                # ICC分析
                f.write("### 4. ICC分析（组内相关系数）\n\n")
                f.write(f"- 第一组专家ICC(2,1): {results_dict
['icc_analysis']['ICC2_1_G1']:.4f}\n")
                f.write(f"- 第二组专家ICC(2,1): {results_dict
['icc_analysis']['ICC2_1_G2']:.4f}\n")
                f.write(f"- 第一组专家ICC解释: {results_dict['
icc_analysis']['interpretation_G1']}\n")
                f.write(f"- 第二组专家ICC解释: {results_dict['
icc_analysis']['interpretation_G2']}\n")
                f.write(f"- 更可信组别: {results_dict['
icc_analysis']['more_credible']}\n")

            print(f"分析结果已保存至 {output_file}")
            return True

        except Exception as e:
            print(f"保存分析结果失败: {str(e)}")
            return False

    def run_analysis(self, file_group1, file_group2,
output_file="analysis_results.md"):
        """
        运行完整分析流程

        参数:
            file_group1 (str): 第一组专家评分文件路径
            file_group2 (str): 第二组专家评分文件路径
            output_file (str): 输出结果文件路径
        """
        try:
            print("开始分析...")
```

```
605
606             # 1. 加载数据
607             if not self.load_data(file_group1, file_group2):
608                 return False
609
610             # 2. 数据预处理
611             if not self.preprocess_data():
612                 return False
613
614             # 3. 描述性统计
615             desc_stats, desc_fig = self.descriptive_statistics
    ()
616             if desc_fig:
617                 desc_fig.savefig("descriptive_statistics.png",
     dpi=300, bbox_inches="tight")
618                 print("描述性统计图表已保存至
    descriptive_statistics.png")
619
620             # 4. 正态性检验
621             normality_result, normality_fig, diff = self.
    normality_test()
622             if normality_fig:
623                 normality_fig.savefig("normality_test.png",
    dpi=300, bbox_inches="tight")
624                 print("正态性检验图表已保存至 normality_test.
    png")
625
626             is_normal = normality_result['is_normal'] if
    normality_result else True
627
628             # 5. 显著性差异检验
629             significance_test_result = self.significance_test(
    diff, is_normal)
630
631             # 6. 效应量分析
```

```
632             effect_size_result = self.effect_size_analysis(
    diff)
633 
634             # 7. ICC计算与分析
635             icc_result, icc_fig = self.calculate_icc()
636             if icc_fig:
637                 icc_fig.savefig("icc_analysis.png", dpi=300,
    bbox_inches="tight")
638                 print("ICC分析图表已保存至 icc_analysis.png")
639 
640             # 8. 生成综合分析结果
641             comprehensive_result = self.
    generate_comprehensive_result(
642                 normality_result, significance_test_result,
    effect_size_result, icc_result
643             )
644 
645             # 9. 保存结果
646             self.save_results(comprehensive_result,
    output_file)
647 
648             # 10. 绘制教师评分散点图
649             plt.figure(figsize=(10, 8))
650             plt.scatter(self.teachers_scores['第一组专家平均分
    '], self.teachers_scores['第二组专家平均分'],
651                         alpha=0.7, s=50)
652             plt.plot([70, 100], [70, 100], 'r--', linewidth=2)
      # 参考线：x=y
653 
654             # 添加教师标签
655             for i, txt in enumerate(self.teachers_scores['教师
    ID']):
656                 plt.annotate(txt,
657                              (self.teachers_scores['第一组专家
    平均分'].iloc[i],
```

```
                                self.teachers_scores['第二组专家平均分'].iloc[i]),
                                fontsize=8)

            plt.xlabel('第一组专家平均分')
            plt.ylabel('第二组专家平均分')
            plt.title('两组专家评分散点图')
            plt.grid(True, linestyle='--', alpha=0.7)
            plt.savefig("scores_scatter.png", dpi=300, bbox_inches="tight")
            print("教师评分散点图已保存至 scores_scatter.png")

            print("分析完成！")
            return True

        except Exception as e:
            print(f"分析过程出错: {str(e)}")
            return False


def main():
    """主函数"""
    try:
        # 创建分析器实例
        analyzer = TeacherEvaluationAnalyzer()

        # 运行分析
        analyzer.run_analysis(
            file_group1="attachment1_group1_expert_scores.csv",
            file_group2="attachment1_group2_expert_scores.csv",
            output_file="problem1_analysis_results.md"
        )

```

```
    except Exception as e:
        print(f"程序执行出错: {str(e)}")


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

3：问题二模型

```
# problem2_solution.py
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from glob import glob
from pathlib import Path
import re
import sys

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

# 设置输出目录
OUTPUT_DIR = "Problem2-output"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# 设置警告过滤
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)

# 设置随机种子，确保结果可复现
```

```
np.random.seed(42)

# 添加调试输出
print("开始执行问题二的数据处理与建模...")
print(f"当前工作目录: {os.getcwd()}")

# 检查环境
try:
    # 导入PyMC
    import pymc as pm
    import arviz as az
    print("使用PyMC版本:", pm.__version__)
    print("使用Arviz版本:", az.__version__)
except Exception as e:
    print(f"导入PyMC失败: {e}")
    raise

# 修复特殊学院数据处理部分

def load_and_preprocess_data():
    """
    加载并预处理所有学院的评分数据，处理特殊格式的学院数据

    Returns:
        DataFrame: 包含所有学院教师评分的统一格式数据
    """
    try:
        print("开始加载数据文件...")
        # 获取所有CSV文件
        csv_files = glob("attachment2_college_*.csv")
        print(f"找到的CSV文件数量: {len(csv_files)}")

        if len(csv_files) == 0:
            print("未找到CSV文件，尝试列出当前目录所有文件:")
            all_files = os.listdir(".")
```

```
61              print("\n".join(all_files))
62
63          # 初始化一个空列表存储所有数据
64          all_data = []
65
66          # 遍历每个文件
67          for file in sorted(csv_files):
68              print(f"处理文件: {file}")
69              # 从文件名中提取学院ID
70              college_id = re.search(r'attachment2_college_(.+)\
    _scores\.csv', file).group(1)
71
72              # 特殊处理 P、H、T 学院，它们有多组专家
73              if college_id in ['P', 'H', 'T']:
74                  print(f"  - 处理特殊格式学院: {college_id}")
75
76                  # 首先直接尝试读取CSV文件中的所有内容
77                  raw_df = pd.read_csv(file)
78                  print(f"  - 原始列名: {raw_df.columns.tolist()
    }")
79
80                  # 对于特殊学院，直接查看文件内容来确定结构
81                  with open(file, 'r', encoding='utf-8') as f:
82                      lines = f.readlines()
83
84                  # 找出带有"组专家评分"的行
85                  for line_idx, line in enumerate(lines):
86                      if "组专家评分" in line:
87                          print(f"  - 找到专家组行({line_idx}):
    {line.strip()}")
88
89                          # 解析出该行中有多少个专家组
90                          groups = re.findall(r'第(.+?)组专家评
    分', line)
91                          print(f"  - 找到专家组数量: {len(
```

```
groups)}, 组名: {groups}")
92
93                          # 为每个组创建数据
94                          for group_idx, group_name in enumerate
(groups):
95                              group_id = f"{college_id}_Group{
group_name}"
96                              print(f"    - 处理专家组: {
group_id}")
97
98                              # 查找该组的教师和分数
99                              # 由于文件格式特殊，直接根据行号来
提取数据
100
101                             # 先确定教师和分数所在的列索引
102                             parts = line.split(',')
103                             start_idx = -1
104
105                             for i, part in enumerate(parts):
106                                 if f"第{group_name}组专家评分"
 in part:
107                                     start_idx = i
108                                     break
109
110                             if start_idx >= 0:
111                                 print(f"      - 找到专家组起始
列索引: {start_idx}")
112
113                                 # 假设教师ID在专家组名称后面的
一列或两列
114                                 teacher_col_idx = -1
115                                 score_col_idx = -1
116
117                                 # 查找标题行(通常在专家组行的
下一行)
```

```
                                header_line = lines[line_idx + 1].strip().split(',')

                                for i in range(start_idx, min(start_idx + 5, len(header_line))):
                                    if "教师" in header_line[i]:
                                        teacher_col_idx = i
                                    elif "得分" in header_line[i]:
                                        score_col_idx = i

                                print(f"      - 教师列索引: {teacher_col_idx}, 分数列索引: {score_col_idx}")

                                if teacher_col_idx >= 0 and score_col_idx >= 0:
                                    # 提取数据行(从标题行后面的行开始)
                                    teacher_ids = []
                                    scores = []

                                    for data_line_idx in range(line_idx + 2, len(lines)):
                                        data_line = lines[data_line_idx].strip().split(',')

                                        if len(data_line) > max(teacher_col_idx, score_col_idx):
                                            teacher_id = data_line[teacher_col_idx].strip()
                                            score = data_line[score_col_idx].strip()

                                            # 确保不是空值
```

```
                                        if teacher_id and score and teacher_id != "" and score != "":
                                            try:
                                                # 尝试转换分数为数值
                                                score_val = float(score)
                                                teacher_ids.append(teacher_id)
                                                scores.append(score_val)
                                            except:
                                                continue

                                    print(f"      - 提取到 {len(teacher_ids)} 个教师记录")

                                    # 创建数据框
                                    if teacher_ids:
                                        group_df = pd.DataFrame({
                                            'College_ID': college_id,
                                            'Expert_Group_ID': group_id,
                                            'Teacher_ID': teacher_ids,
                                            'Raw_Score': scores
                                        })
                                        all_data.append(group_df)
                                        print(f"      - 成功添加 {len(group_df)} 条记录")
                                else:
```

```
                                    print("      - 警告: 未找到教师列或分数列")
                            else:
                                print(f"      - 警告: 未找到专家组起始列")

                        # 已经处理完所有专家组，跳出循环
                        break

            else:
                # 常规学院处理
                print(f"  - 处理常规格式学院: {college_id}")
                try:
                    # 读取数据
                    df = pd.read_csv(file)
                    print(f"  - 成功读取，列名: {df.columns.tolist()}")

                    # 找到教师ID列和分数列
                    id_cols = [col for col in df.columns if "教师" in str(col)]
                    score_cols = [col for col in df.columns if "得分" in str(col) or "分数" in str(col) or "总分" in str(col)]

                    if id_cols:
                        id_col = id_cols[0]
                    else:
                        print(f"  - 警告: 找不到教师ID列，尝试使用第一列")
                        id_col = df.columns[0]

                    if score_cols:
                        score_col = score_cols[0]
                    else:
```

```
                        print(f"  - 警告：找不到分数列，尝试使用最后一列")
                        score_col = df.columns[-1]

                    print(f"    - 使用ID列：{id_col}，分数列：{score_col}")

                    # 创建数据
                    college_df = pd.DataFrame({
                        'College_ID': college_id,
                        'Expert_Group_ID': f"{college_id}_Group1",
                        'Teacher_ID': df[id_col],
                        'Raw_Score': df[score_col]
                    })

                    all_data.append(college_df)
                    print(f"    - 成功添加 {len(college_df)} 条记录")

                except Exception as e:
                    print(f"  - 读取失败：{str(e)}")
                    continue

        if not all_data:
            raise ValueError("没有成功加载任何数据")

        # 合并所有数据
        print("合并所有数据...")
        combined_df = pd.concat(all_data, ignore_index=True)

        # 清理可能的数据问题：将分数转换为数值型
        print("清理数据...")
        try:
            combined_df['Raw_Score'] = pd.to_numeric(
```

```
    combined_df['Raw_Score'], errors='coerce')
222             # 删除无效分数记录
223             invalid_count = combined_df['Raw_Score'].isna().
    sum()
224             if invalid_count > 0:
225                 print(f"警告: 发现 {invalid_count} 条无效分数
    记录，将被删除")
226                 combined_df = combined_df.dropna(subset=['
    Raw_Score'])
227         except Exception as e:
228             print(f"分数转换警告: {str(e)}")
229
230         # 创建映射到整数的索引
231         college_map = {college: idx for idx, college in
    enumerate(combined_df['College_ID'].unique())}
232         expert_group_map = {group: idx for idx, group in
    enumerate(combined_df['Expert_Group_ID'].unique())}
233
234         # 添加索引列
235         combined_df['college_idx'] = combined_df['College_ID'
    ].map(college_map)
236         combined_df['expert_group_idx'] = combined_df['
    Expert_Group_ID'].map(expert_group_map)
237
238         print(f"数据加载和预处理完成，共有 {len(combined_df)}
    条评分记录")
239         print(f"包含 {len(college_map)} 个学院，{len(
    expert_group_map)} 个专家组")
240
241         # 显示前几行数据
242         print("\n数据预览:")
243         print(combined_df.head())
244
245         return combined_df, college_map, expert_group_map
246
```

```
    except Exception as e:
        print(f"数据加载和预处理失败: {str(e)}")
        import traceback
        traceback.print_exc()
        raise

# 修改exploratory_data_analysis函数，以使用命名聚合
def exploratory_data_analysis(df):
    """
    对数据进行探索性分析，了解各学院和专家组的打分特点

    Args:
        df: 预处理后的数据
    """
    try:
        # 创建一个保存EDA结果的文件夹
        eda_dir = os.path.join(OUTPUT_DIR, "EDA")
        os.makedirs(eda_dir, exist_ok=True)

        # 计算各学院的统计指标，使用命名聚合
        college_stats = df.groupby('College_ID')['Raw_Score'].
    agg([
            'count', 'mean', 'std', 'min', 'max',
            ('range', lambda x: x.max() - x.min())  # 使用命名
    聚合
        ]).reset_index()

        # 计算各专家组的统计指标，使用命名聚合
        expert_group_stats = df.groupby(['College_ID', '
    Expert_Group_ID'])['Raw_Score'].agg([
            'count', 'mean', 'std', 'min', 'max',
            ('range', lambda x: x.max() - x.min())  # 使用命名
    聚合
        ]).reset_index()

```

```
278         # 保存统计结果
279         college_stats.to_csv(os.path.join(eda_dir, "
    college_statistics.csv"), index=False)
280         expert_group_stats.to_csv(os.path.join(eda_dir, "
    expert_group_statistics.csv"), index=False)
281 
282         # 绘制学院得分分布箱线图
283         plt.figure(figsize=(14, 8))
284         sns.boxplot(x='College_ID', y='Raw_Score', data=df,
    palette='viridis')
285         plt.title('各学院教师评分分布', fontsize=16)
286         plt.xlabel('学院', fontsize=14)
287         plt.ylabel('原始评分', fontsize=14)
288         plt.xticks(rotation=45)
289         plt.tight_layout()
290         plt.savefig(os.path.join(eda_dir, "
    college_scores_boxplot.png"), dpi=300)
291         plt.close()
292 
293         # 绘制特殊学院(H、P、T)内部专家组的分布
294         special_colleges = ['H', 'P', 'T']
295         for college in special_colleges:
296             if college in df['College_ID'].unique():
297                 college_data = df[df['College_ID'] == college]
298 
299                 plt.figure(figsize=(10, 6))
300                 sns.boxplot(x='Expert_Group_ID', y='Raw_Score'
    , data=college_data, palette='Set2')
301                 plt.title(f'学院{college}内部各专家组评分分布'
    , fontsize=16)
302                 plt.xlabel('专家组', fontsize=14)
303                 plt.ylabel('原始评分', fontsize=14)
304                 plt.tight_layout()
305                 plt.savefig(os.path.join(eda_dir, f"college_{
    college}_expert_groups_boxplot.png"), dpi=300)
```

```
306                 plt.close()
307
308         # 绘制整体分布图
309         plt.figure(figsize=(12, 6))
310         sns.histplot(df['Raw_Score'], kde=True, bins=30)
311         plt.title('所有教师评分分布', fontsize=16)
312         plt.xlabel('原始评分', fontsize=14)
313         plt.ylabel('频数', fontsize=14)
314         plt.tight_layout()
315         plt.savefig(os.path.join(eda_dir, "
    overall_score_distribution.png"), dpi=300)
316         plt.close()
317
318         print(f"探索性数据分析完成，结果已保存到 {eda_dir} 目
    录")
319
320         return college_stats, expert_group_stats
321
322     except Exception as e:
323         print(f"探索性数据分析失败: {str(e)}")
324         raise
325
326 def build_and_fit_hierarchical_model(df, college_map,
    expert_group_map):
327     """
328     构建并拟合层级贝叶斯模型
329     Args:
330         df: 预处理后的数据
331         college_map: 学院映射
332         expert_group_map: 专家组映射
333     Returns:
334         trace: MCMC采样结果
335         model: 拟合好的模型
336     """
337     # 设置模型数据
```

```
    scores = df['Raw_Score'].values
    college_idx = df['college_idx'].values
    expert_group_idx = df['expert_group_idx'].values
    n_colleges = len(college_map)
    n_expert_groups = len(expert_group_map)
    global_mean = scores.mean()
    global_sd = scores.std() * 2  # 宽泛的先验
    print(f"开始构建模型，全局均值: {global_mean:.2f}, 全局标准差: {global_sd:.2f}")
    # 构建层级贝叶斯模型
    print("构建层级贝叶斯模型...")
    with pm.Model() as hierarchical_model:
        # 先验分布
        mu_global = pm.Normal('mu_global', mu=global_mean, sigma=global_sd)
        sigma_college = pm.HalfNormal('sigma_college', sigma=10)
        sigma_expert_group = pm.HalfNormal('sigma_expert_group', sigma=10)
        sigma_error = pm.HalfNormal('sigma_error', sigma=10)
        alpha_college = pm.Normal('alpha_college', mu=0, sigma=sigma_college, shape=n_colleges)
        beta_expert_group = pm.Normal('beta_expert_group', mu=0, sigma=sigma_expert_group, shape=n_expert_groups)
        mu = mu_global + alpha_college[college_idx] + beta_expert_group[expert_group_idx]
        y = pm.Normal('y', mu=mu, sigma=sigma_error, observed=scores)
        print("开始MCMC采样...")
        trace = pm.sample(
            draws=2000,
            tune=1000,
            chains=4,
            cores=4,
            target_accept=0.9,
```

```
            return_inferencedata=True,
            random_seed=42
        )
        summary = az.summary(trace)
        summary.to_csv(os.path.join(OUTPUT_DIR, "model_summary.csv"))
        print("模型拟合完成")
        model = hierarchical_model
    return trace, model

# 修复model_diagnostics函数
def model_diagnostics(trace, college_map, expert_group_map):
    """
    进行模型诊断
    Args:
        trace: MCMC采样结果
        college_map: 学院映射
        expert_group_map: 专家组映射
    """
    try:
        print("开始模型诊断...")
        diagnostics_dir = os.path.join(OUTPUT_DIR, "Diagnostics")
        os.makedirs(diagnostics_dir, exist_ok=True)
        # 贝叶斯模型的完整诊断
        print("使用完整贝叶斯模型诊断")
        az.plot_trace(trace, var_names=['mu_global', 'sigma_college', 'sigma_expert_group', 'sigma_error'])
        plt.tight_layout()
        plt.savefig(os.path.join(diagnostics_dir, "trace_plots.png"), dpi=300)
        plt.close()
        az.plot_posterior(trace, var_names=['mu_global', 'sigma_college', 'sigma_expert_group', 'sigma_error'])
        plt.tight_layout()
```

```
395             plt.savefig(os.path.join(diagnostics_dir, "
    posterior_plots.png"), dpi=300)
396             plt.close()
397             az.plot_forest(trace, var_names=['alpha_college'],
    combined=True)
398             plt.title('各学院评分偏差(α)', fontsize=16)
399             plt.tight_layout()
400             plt.savefig(os.path.join(diagnostics_dir, "
    college_effects_forest.png"), dpi=300)
401             plt.close()
402             az.plot_forest(trace, var_names=['beta_expert_group'],
     combined=True)
403             plt.title('各专家组评分偏差(β)', fontsize=16)
404             plt.tight_layout()
405             plt.savefig(os.path.join(diagnostics_dir, "
    expert_group_effects_forest.png"), dpi=300)
406             plt.close()
407             print(f"模型诊断完成，结果已保存到 {diagnostics_dir}
    目录")
408         except Exception as e:
409             print(f"模型诊断失败: {str(e)}")
410             import traceback
411             traceback.print_exc()
412             print("跳过模型诊断阶段")
413
414 def calculate_corrected_scores(df, trace, college_map,
    expert_group_map):
415     """
416     计算校正后的教师评分
417     Args:
418         df: 预处理后的数据
419         trace: MCMC采样结果
420         college_map: 学院映射
421         expert_group_map: 专家组映射
422     Returns:
```

```
423         DataFrame: 包含原始评分和校正后评分的数据
424     """
425     try:
426         print("开始计算校正后评分...")
427         corrected_df = df.copy()
428         # 使用贝叶斯模型结果
429         print("使用贝叶斯模型结果计算校正后评分")
430         summary = az.summary(trace)
431         mu_global = summary.loc['mu_global', 'mean']
432         alpha_college = {}
433         for college, idx in college_map.items():
434             alpha_college[college] = summary.loc[f'
    alpha_college[{idx}]', 'mean']
435         beta_expert_group = {}
436         for group, idx in expert_group_map.items():
437             beta_expert_group[group] = summary.loc[f'
    beta_expert_group[{idx}]', 'mean']
438         print("计算校正后评分...")
439         corrected_df['Corrected_Score'] = corrected_df.apply(
440             lambda row: row['Raw_Score'] - alpha_college[row['
    College_ID']] - \
441                         beta_expert_group[row['Expert_Group_ID
    ']] + mu_global,
442             axis=1
443         )
444         corrected_mean = corrected_df['Corrected_Score'].mean
    ()
445         corrected_std = corrected_df['Corrected_Score'].std()
446         target_mean = 85
447         target_std = 5
448         corrected_df['Final_Score'] = (
449             (corrected_df['Corrected_Score'] - corrected_mean)
     / corrected_std
450         ) * target_std + target_mean
451         corrected_df['Final_Score'] = corrected_df['
```

```
    Final_Score'].clip(60, 100)
452         corrected_df.to_csv(os.path.join(OUTPUT_DIR, "
    corrected_scores.csv"), index=False)
453         print("校正后的评分计算完成")
454         print(f"校正后评分范围: {corrected_df['Final_Score'].
    min():.2f} - {corrected_df['Final_Score'].max():.2f}")
455         return corrected_df
456     except Exception as e:
457         print(f"校正后评分计算失败: {str(e)}")
458         import traceback
459         traceback.print_exc()
460         raise
461
462 def visualize_results(df, corrected_df):
463     """
464     可视化原始评分和校正后评分的结果
465
466     Args:
467         df: 原始数据
468         corrected_df: 校正后的数据
469     """
470     try:
471         results_dir = os.path.join(OUTPUT_DIR, "
    Results_Visualization")
472         os.makedirs(results_dir, exist_ok=True)
473
474         # 1. 整体分布对比
475         plt.figure(figsize=(12, 6))
476
477         plt.subplot(1, 2, 1)
478         sns.histplot(df['Raw_Score'], kde=True, bins=30)
479         plt.title('原始评分分布', fontsize=14)
480         plt.xlabel('分数', fontsize=12)
481         plt.ylabel('频数', fontsize=12)
482
```

```
483         plt.subplot(1, 2, 2)
484         sns.histplot(corrected_df['Final_Score'], kde=True,
    bins=30)
485         plt.title('校正后评分分布', fontsize=14)
486         plt.xlabel('分数', fontsize=12)
487         plt.ylabel('频数', fontsize=12)
488
489         plt.tight_layout()
490         plt.savefig(os.path.join(results_dir, "
    score_distributions_comparison.png"), dpi=300)
491         plt.close()
492
493         # 2. 各学院原始分数和校正后分数的箱线图对比
494         plt.figure(figsize=(16, 10))
495
496         plt.subplot(2, 1, 1)
497         sns.boxplot(x='College_ID', y='Raw_Score', data=df,
    palette='viridis')
498         plt.title('各学院原始评分分布', fontsize=16)
499         plt.xlabel('学院', fontsize=14)
500         plt.ylabel('分数', fontsize=14)
501         plt.xticks(rotation=45)
502
503         plt.subplot(2, 1, 2)
504         sns.boxplot(x='College_ID', y='Final_Score', data=
    corrected_df, palette='viridis')
505         plt.title('各学院校正后评分分布', fontsize=16)
506         plt.xlabel('学院', fontsize=14)
507         plt.ylabel('分数', fontsize=14)
508         plt.xticks(rotation=45)
509
510         plt.tight_layout()
511         plt.savefig(os.path.join(results_dir, "
    college_scores_comparison.png"), dpi=300)
512         plt.close()
```

```
513
514         # 3. 特殊学院(H、P、T)内部各专家组校正前后对比
515         special_colleges = ['H', 'P', 'T']
516         for college in special_colleges:
517             if college in df['College_ID'].unique():
518                 college_data_orig = df[df['College_ID'] ==
    college]
519                 college_data_corr = corrected_df[corrected_df[
    'College_ID'] == college]
520
521                 plt.figure(figsize=(12, 8))
522
523                 plt.subplot(2, 1, 1)
524                 sns.boxplot(x='Expert_Group_ID', y='Raw_Score'
    , data=college_data_orig, palette='Set2')
525                 plt.title(f'学院{college}内部各专家组原始评分'
    , fontsize=16)
526                 plt.xlabel('专家组', fontsize=14)
527                 plt.ylabel('分数', fontsize=14)
528
529                 plt.subplot(2, 1, 2)
530                 sns.boxplot(x='Expert_Group_ID', y='
    Final_Score', data=college_data_corr, palette='Set2')
531                 plt.title(f'学院{college}内部各专家组校正后评
    分', fontsize=16)
532                 plt.xlabel('专家组', fontsize=14)
533                 plt.ylabel('分数', fontsize=14)
534
535                 plt.tight_layout()
536                 plt.savefig(os.path.join(results_dir, f"
    college_{college}_comparison.png"), dpi=300)
537                 plt.close()
538
539         # 4. 原始排名vs校正后排名的散点图
540         corrected_df['Raw_Rank'] = corrected_df['Raw_Score'].
```

```
    rank(ascending=False)
541         corrected_df['Final_Rank'] = corrected_df['Final_Score
    '].rank(ascending=False)
542 
543         plt.figure(figsize=(10, 8))
544         sns.scatterplot(x='Raw_Rank', y='Final_Rank', hue='
    College_ID', data=corrected_df, alpha=0.6)
545 
546         # 绘制对角线
547         max_rank = max(corrected_df['Raw_Rank'].max(),
    corrected_df['Final_Rank'].max())
548         plt.plot([0, max_rank], [0, max_rank], 'k--', alpha
    =0.5)
549 
550         plt.title('原始排名 vs 校正后排名', fontsize=16)
551         plt.xlabel('原始排名', fontsize=14)
552         plt.ylabel('校正后排名', fontsize=14)
553         plt.tight_layout()
554         plt.savefig(os.path.join(results_dir, "
    rank_comparison_scatter.png"), dpi=300)
555         plt.close()
556 
557         # 5. 创建一个示例表格，展示排名变化最大的10位教师
558         corrected_df['Rank_Change'] = corrected_df['Raw_Rank']
     - corrected_df['Final_Rank']
559         top_changes = corrected_df.sort_values(by='Rank_Change
    ', ascending=False).head(10)
560         bottom_changes = corrected_df.sort_values(by='
    Rank_Change').head(10)
561 
562         rank_changes = pd.concat([top_changes, bottom_changes
    ])
563         rank_changes = rank_changes[['College_ID', '
    Expert_Group_ID', 'Teacher_ID',
564                                      'Raw_Score', 'Final_Score'
```

```
    , 'Raw_Rank', 'Final_Rank', 'Rank_Change']]

        rank_changes.to_csv(os.path.join(results_dir, "significant_rank_changes.csv"), index=False)

        print(f"结果可视化完成，图表已保存到 {results_dir} 目录")

    except Exception as e:
        print(f"结果可视化失败: {str(e)}")
        raise

# 修复generate_summary_report函数
def generate_summary_report(df, corrected_df, trace, college_map, expert_group_map):
    """
    生成总结报告
    Args:
        df: 原始数据
        corrected_df: 校正后的数据
        trace: MCMC采样结果
        college_map: 学院映射
        expert_group_map: 专家组映射
    """
    try:
        print("开始生成总结报告...")
        report_path = os.path.join(OUTPUT_DIR, "problem2_analysis_results.md")
        with open(report_path, 'w', encoding='utf-8') as f:
            f.write("# 问题二：教师教学评价标准化与汇总分析报告\n\n")
            f.write("## 1. 数据概览\n\n")
            f.write(f"- 共分析了 {len(df['College_ID'].unique())} 个学院的评分数据\n")
            f.write(f"- 总共有 {len(df)} 条教师评分记录\n")
```

```
593             multi_expert_colleges = df.groupby(['College_ID',
    'Expert_Group_ID']).size().reset_index()
594             multi_expert_colleges = multi_expert_colleges['
    College_ID'].value_counts()
595             multi_expert_colleges = multi_expert_colleges[
    multi_expert_colleges > 1].index.tolist()
596             if multi_expert_colleges:
597                 f.write(f"- 其中学院 {', '.join(
    multi_expert_colleges)} 包含多组专家评分\n")
598             f.write("\n")
599             f.write("## 2. 各学院评分特点\n\n")
600             college_stats = df.groupby('College_ID')['
    Raw_Score'].agg([
601                 'count', 'mean', 'std', 'min', 'max',
602                 ('range', lambda x: x.max() - x.min())
603             ]).sort_values(by='mean', ascending=False)
604             f.write("各学院评分统计特点（按平均分降序排列）：\
    n\n")
605             f.write("| 学院 | 教师数量 | 平均分 | 标准差 | 最
    低分 | 最高分 | 极差 |\n")
606             f.write("
    |------|---------|--------|--------|--------|--------|------|\
    n")
607             for college, row in college_stats.iterrows():
608                 f.write(f"| {college} | {int(row['count'])} |
    {row['mean']:.2f} | {row['std']:.2f} | ")
609                 f.write(f"{row['min']:.2f} | {row['max']:.2f}
    | {row['range']:.2f} |\n")
610             f.write("\n主要发现：\n")
611             f.write(f"- 评分最高的学院：{college_stats.index
    [0]}，平均分 {college_stats['mean'].max():.2f}\n")
612             f.write(f"- 评分最低的学院：{college_stats.index
    [-1]}，平均分 {college_stats['mean'].min():.2f}\n")
613             f.write(f"- 学院间平均分差距：{college_stats['mean
    '].max() - college_stats['mean'].min():.2f} 分\n")
```

```
614             max_range_college = college_stats.sort_values(by='
    range', ascending=False).index[0]
615             min_range_college = college_stats.sort_values(by='
    range').index[0]
616             max_range_value = college_stats.loc[
    max_range_college, 'range']
617             min_range_value = college_stats.loc[
    min_range_college, 'range']
618             f.write(f"- 极差最大的学院: {max_range_college},
    极差 {max_range_value:.2f}\n")
619             f.write(f"- 极差最小的学院: {min_range_college},
    极差 {min_range_value:.2f}\n\n")
620             f.write("## 3. 层级贝叶斯模型结构与参数\n\n")
621             f.write("### 模型结构\n\n")
622             f.write("我们构建了三层的完整层级贝叶斯模型 (HBM)
    : \n")
623             f.write("- 学校整体水平（全局均值）\n")
624             f.write("- 学院效应（每个学院的评分偏差）\n")
625             f.write("- 专家组效应（每个专家组在学院内的评分偏
    差）\n\n")
626             f.write("### 数学模型\n\n")
627             f.write("$$S_{ijk} \\sim \\mathcal{N}(\\mu_{ij},
    \\sigma^2_{\\text{error}})$$\n")
628             f.write("$$\\mu_{ij} = \\mu_{\\text{global}} + \\
    alpha_i + \\beta_{ij}$$\n")
629             f.write("$$\\alpha_i \\sim \\mathcal{N}(0, \\sigma
    ^2_{\\text{college}})$$\n")
630             f.write("$$\\beta_{ij} \\sim \\mathcal{N}(0, \\
    sigma^2_{\\text{expert\\_group}})$$\n\n")
631             f.write("### 主要参数估计结果\n\n")
632             summary = az.summary(trace)
633             f.write(f"- 全局均值 (μ_global): {summary.loc['
    mu_global', 'mean']:.2f}\n")
634             f.write(f"- 学院效应标准差 (σ_college): {summary.
    loc['sigma_college', 'mean']:.2f}\n")
```

```
635             f.write(f"- 专家组效应标准差 (σ_expert_group): {
    summary.loc['sigma_expert_group', 'mean']:.2f}\n")
636             f.write(f"- 误差标准差 (σ_error): {summary.loc['
    sigma_error', 'mean']:.2f}\n\n")
637             f.write("## 4. 校正效果分析\n\n")
638             orig_mean = df['Raw_Score'].mean()
639             orig_std = df['Raw_Score'].std()
640             corr_mean = corrected_df['Final_Score'].mean()
641             corr_std = corrected_df['Final_Score'].std()
642             f.write("### 整体评分分布变化\n\n")
643             f.write(f"- 原始评分：均值 = {orig_mean:.2f}, 标准
    差 = {orig_std:.2f}\n")
644             f.write(f"- 校正后评分：均值 = {corr_mean:.2f}, 标
    准差 = {corr_std:.2f}\n\n")
645             f.write("### 校正后学院间差异\n\n")
646             corr_college_stats = corrected_df.groupby('
    College_ID')['Final_Score'].agg([
647                 'count', 'mean', 'std', 'min', 'max',
648                 ('range', lambda x: x.max() - x.min())
649             ])
650             f.write("| 学院 | 校正前平均分 | 校正后平均分 | 校
    正前极差 | 校正后极差 |\n")
651             f.write("
    |------|--------------|--------------|------------|------------|\
    n")
652             for college in college_stats.index:
653                 orig = college_stats.loc[college]
654                 corr = corr_college_stats.loc[college]
655                 f.write(f"| {college} | {orig['mean']:.2f} | {
    corr['mean']:.2f} | ")
656                 f.write(f"{orig['range']:.2f} | {corr['range
    ']:.2f} |\n")
657             f.write("\n")
658             f.write("## 5. 结论与解释\n\n")
659             f.write("### 模型优势\n\n")
```

```
            f.write("1. **消除系统性偏差**：模型有效识别并剥离了由学院和专家组打分风格引起的系统性偏差。\n")
            f.write("2. **'借用强度'机制**：通过考虑整体数据信息，对小样本和极端打分行为进行了'收缩'估计，使得结果更稳健。\n")
            f.write("3. **保留相对排序**：虽然消除了系统性偏差，但仍保留了教师间的真实差异。\n")
            f.write("4. **统一标准**：校正后的分数分布更加集中且符合正态分布，便于统一管理和评价。\n\n")
            f.write("### 解决的关键问题\n\n")
            f.write("1. **评分标准不统一问题**：通过估计并剥离各学院的偏置，实现了评分标准的统一。\n")
            f.write("2. **极差问题**：对极差极小或极大的学院，通过'借用强度'机制使其分数分布更合理。\n")
            f.write("3. **专家组偏差问题**：明确建模了特殊学院内部不同专家组的评分偏差，消除了局部评价偏差。\n\n")
            f.write("总结：我们的模型成功构建了一个公平、合理的教师评价标准化体系，为学校决策提供了可靠的参考依据。\n")
        print(f"分析报告已生成并保存到 {report_path}")
    except Exception as e:
        print(f"生成总结报告失败: {str(e)}")
        import traceback
        traceback.print_exc()

def main():
    """
    主函数，协调整个工作流程
    """
    try:
        print("开始执行问题二的求解...")

        # 1. 数据预处理
        df, college_map, expert_group_map = load_and_preprocess_data()

```

```
        # 2. 探索性数据分析
        college_stats, expert_group_stats = exploratory_data_analysis(df)

        # 3. 构建并拟合层级贝叶斯模型
        trace, model = build_and_fit_hierarchical_model(df, college_map, expert_group_map)

        # 4. 模型诊断 - 传递college_map和expert_group_map参数
        model_diagnostics(trace, college_map, expert_group_map)

        # 5. 计算校正后评分
        corrected_df = calculate_corrected_scores(df, trace, college_map, expert_group_map)

        # 6. 可视化结果
        visualize_results(df, corrected_df)

        # 7. 生成总结报告
        generate_summary_report(df, corrected_df, trace, college_map, expert_group_map)

        print(f"问题二求解完成，所有结果已保存到 {OUTPUT_DIR} 目录")

    except Exception as e:
        print(f"程序执行失败: {str(e)}")
        import traceback
        traceback.print_exc()

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