赛区评阅编号（由赛区组委会填写）：

2021 年高教社杯全国大学生数学建模竞赛

承　诺　书

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》（以下简称 “竞赛章程和参赛规则”，可从 http://www.mcm.edu.cn 下载）。

我们完全清楚，在竞赛开始后参赛队员不能以任何方式，包括电话、电子邮件、“贴吧”、QQ 群、微信群等，与队外的任何人（包括指导教师）交流、讨论与赛题有关的问题；无论主动参与讨论还是被动接收讨论信息都是严重违反竞赛纪律的行为。

我们完全清楚，在竞赛中必须合法合规地使用文献资料和软件工具，不能有任何侵犯知识产权的行为。否则我们将失去评奖资格，并可能受到严肃处理。

我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。

我们授权全国大学生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号（从 A/B/C/D/E 中选择一项填写）：A

我们的报名参赛队号（12 位数字全国统一编号）：

参赛学校（完整的学校全称，不含院系名）：南京大学

参赛队员 (打印并签名) ：1. 郁博文

2. 游莫凡

3. 汪佳蓉

指导教师或指导教师组负责人 (打印并签名)：

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期：2022 年 7 月 31 日

赛区评阅编号：
（由赛区填写）

全国评阅编号：
（全国组委会填写）

2021 年高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅记录（可供赛区评阅时使用）：

评阅人						
备注						

送全国评阅统一编号:
（赛区组委会填写）

（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）

一、 摘要

本文针对数学建模比赛中的多阶段打分问题进行研究

针对问题一，我们将评委重复的问题转为了图论中三角形共边问题，通过程序计算出了当评委取不同数量时，满足题设条件的论文组数。对于问题一中的规划问题，我们采用贪心算法，设计了相对合理的分组方案

针对问题二，为了得到合理的打分结果，我们先检验原数据，去除异常值，再使用 Z-score 规约调整各个评委的打分，使得他们的打分能用一个统一的标准去评判。经检验，我们的调分方法效果良好。

针对问题三，由于数据集存在空缺，直接采用 Z-score 规约无法很好的评估分数，因此我们设计了掩盖矩阵，通过该矩阵和领域预测算法来扩充数据，再进行 Z-score 规约，成功针对各个打分步骤进行了合理的调分。

针对问题四，综合先前解题经验，我们认定题目的步骤三和四为合理，对步骤一和二进行了合理性检验。同时我们针对总体工作量进行了评估。最后，我们提出了一种可能的改进方案。

关键词： 贪心算法 Z-score 规约 领域预测 多阶段决策模型 圆桌模型

二、 问题重现

某学校举办数学建模竞赛，共有 200 个队比赛。组委会组织了 10 名评委对竞赛的论文进行评审，拟评出一等奖 10%，二等奖 15%，三等奖 25%。

评审的步骤如下：

一、初审

将 200 个队的论文随机的分为 20 个组。

步骤（1）

打分：每个组的 10 篇论文由 3 位评委评审，分别用百分制给出评分。

排序：每个组的 10 篇论文根据 3 位评委的给分进行平均，淘汰排名靠后的 40%，即 4 篇文章。

步骤（2）

打分：未淘汰的 120 篇论文，再由没有评审过的 2 位评委进行评审（评审时依然按原来的组进行，每组 6 篇），给出百分制得分。

排序：5 个评委的平均给分进行排序，淘汰排名靠后的 20 篇文章，剩下的 100 篇论文为获奖论文。

二、获奖论文评审

100 篇获奖论文中，排名 31 到 40 的论文，获二等奖；排名 61 到 100 的论文获三等奖；

步骤（3）

打分：排名 1 到 30 与排名 41 到 60 的 50 篇论文，再由 2 位评委进行打分，

排序：原排名 41 到 60 的 20 篇论文，7 位评委给分平均，新排名 41-50 的获二等奖，新排名 51-60 的获三等奖；

原排名 1 到 30 的 30 篇论文，7 位评委给分平均，新排名 26-30 名获二等

奖，新排名 1-15 名获一等奖。

步骤（4）

7 人平均分排名在 16 到 25 的论文，再由剩下的 3 位评委打分。最终排名 16-20 的获一等奖，最终排名 21-25 的获二等奖。

请你们解决如下问题：

问题一

若参加竞赛的一共有 m 个组（本问题中 m=20），n 个评委（本问题中 n=10），每组论文均要给 3 位评委评审，给出论文与评委的分配方法（即每个评委分别评审哪些组的论文），使得任意不同的两个组的评委尽量不同。当 m，n 满足什么条件时能够保证任意两个组不出现 3 位评委一样的情况？当 m，n 满足什么条件时能够保证任意两个组不出现有 2 位评委一样的情况？对 m=20，n=10，给出你认为最好的分配方案。

问题二

评审时会出现有的评委打分偏紧，有的评委打分偏松的情况。如果 10 个评委都评阅了所有的 200 篇论文，请你给出数学模型与算法，根据所有评委对每篇论文的打分，估计每位评委打分的偏差，对评委打分进行调整。

针对附录一中的数据，对评委的分数进行调整，然后根据平均分进行排序，给出获奖结果。

问题三

根据实际的评审步骤，评委不可能评阅所有的论文。在评委只评阅了部分论文后，就要对评委的分数进行调整，给出此时调整评委评分的数学模型与算法，注意，在评审的四个打分步骤之后都要利用你的算法调整评委的打分，然后对论文进行排序。

利用附录一的部分数据，计算出获奖结果。其中，评审步骤（1），（2）的分组方案见附录二，步骤（3）中论文采取随机的方法分配给评委。

提示：由于问题三只能利用附录一中的部分数据，因此计算的评委打分的偏差以及最终的结果与问题二不一致是很正常的。

问题四

对现行的评审步骤，在评审的公正性，评委的工作量安排是否合理？给出数学模型对现行的评审步骤进行评价。根据你的评价标准，改进现行的评审步骤。

三、 问题一的分析与解决

3.1 保证任意两组不出现 3 位相同评委的 m、n 取值

首先考虑组 m 的下限，在不考虑评委溢出的情况下，m 的下限只需要大于 0 就满足条件，若考虑评委过多组过少的问题，那么由于每三个评委打一个组的评分，只需要 m>=n/3 时，评委不会溢出。

接下来考虑上限，可以将题目转变为从评委中任意选出三个不重复的排列组合，即C_n^3，这就是 m 的上限，故最终结果为：

$$0 \leq \mathrm{m} \leq \mathrm{C}_{\mathrm{n}}^{3}(\text{考虑溢出})$$

$$\frac{\mathrm{n}}{3} \leq \mathrm{m} \leq \mathrm{C}_{\mathrm{n}}^{3}(\text{不考虑溢出})$$

3.2 保证任意两组不出现 2 位相同评委的 m、n 取值

首先考虑 m 的下限，此处同上.

接下来考虑上限，我们用图论的方法解决该问题，首先我们的总体思路为尝试将该问题转化为将评委看作点，在此基础上根据条件形成无向图问题。

随后将问题转化，首先每一组由三个评委打分，相当于从所有点之中随机选择三个点两两互连形成一个三角形的回路，接着讨论任意两个组不出现有 2 位评委一样的条件，因为任意两个点形成一条边，故该条件可以转变为任意两个三角形回路不共用边，所以该问题可以转变为：在一个平面上存在 n 个没有三点共线的点，能构造多少个没有公共边的三角形。

我们以此思路设计方案：

1. 将所有点 n 两两互连形成完全图
2. 以（vi，vj）代表点 i，j 互连形成的一条边，（vi，vj）= 0 时代表此边没有被使用，（vi，vj）= 1 时代表此边被使用，而且因为是无向图，所以还有（vi，vj）=（vj，vi），所以为了方便，我们设定 i < j 避免为（vj，vi）赋值(i=j 时不会形成边故而也不考虑)。综合为以下方程组：

$$1 \leq \mathrm{i} \leq \mathrm{n}$$

$$1 \leq \mathrm{j} \leq \mathrm{n}$$

$$\mathrm{i} < \mathrm{j}$$

$$(v_i, v_j) = \begin{cases} 0, \text{以 i，j 为顶点的边未被使用} \\ 1, \text{以 i，j 为顶点的边被使用} \end{cases}$$

3. 初始化所有(vi,vj)的值为 0，令 M 为最终结果，初始化为 0
4. 随机选取三个点 v_1,v_2,v_3，若(v_1,v_2)=(v_2,v_3)=(v_1,v_3)= 0,将(v_1,v_2)，(v_2,v_3)，(v_1,v_3)置为 1，并且 M = M + 1。

重复 4 步骤遍历所有点，最终得到的 M 就是在评委数为 n 的情况下组数的上限。

用顺序选取替代随机选取，将该过程转换为代码，获得结果，然而在我们选取其中一些样本进行手工验算时,发现在 n=10 时,m 的上限为 13，而程序获得的值是 10，这是点的选取问题，即顺序选取只能得到一个接近于且必然小于等于最小上界的值，而在对小样本进行归类时，我们发现对于每一个值 n，M 取最大时每一个点的使用次数趋于一致性（即每一个评委都尽可能评同样数量的小组），故而用五种方式优化选点：

三重循环顺序取点，即令 i，j，k，2 ≤ i < n，1 ≤ j < i，0 ≤ K < j 做三重循环,每一次取点 v_i,v_j,v_k

令 v_first，i， 0 ≤ v_first < n 顺序递增，v_first < i < n-1， i 从下限递增，此时取点 v_(v_first),v_i,v_(i+1)，此步骤之后在做一次三重循环顺序取点

令 v_first，i， 0 ≤ v_first < n 逆序递减，后同 2

令 v_first，i， 0 ≤ v_first < n 顺序递增 ，v_first < i < n-1，i 为奇数时从下限递增，此时取点 v_(v_first),v_i,v_(i+1)，i 为偶数时从上限递减，此时取点 v_(v_first),v_i,v_(i-1)，此步骤之后在做一次三重循环顺序取点

令 v_first，i， 0 ≤ v_first < n 逆序递减，后同 4

除了三重循环顺序取点，后四种考虑的因素是

v_first 是顺序还是逆序

i 是否考虑奇偶，根据奇偶选择顺序还是逆序

经过测试，i 单纯考虑顺序还是逆序这个因素对结果无影响故不考虑。

由于该代码用到三重循环，故时间复杂度为 0（$\sum_{i=3}^{n}\sum_{j=2}^{i-1}\sum_{k=1}^{j-1}1$），在 n 巨大时解题时间成本会变得巨大，而我校共有本科生 13350 人，由代码可知 M（104）=1595 此时可以满足所有本科生参加数学建模，故我们只列举 n<=104 的情况。

根据代码，我们可以得到以下图表

n	5	6	7	8	9	10	11	12	13	14
M(n)	2	4	7	8	10	13	17	19	23	28
n	15	16	17	18	19	20	21	22	23	24
M(n)	35	35	40	43	49	54	60	67	75	81
n	25	26	27	28	29	30	31	32	33	34
M(n)	89	96	104	113	126	140	155	155	158	165
n	35	36	37	38	39	40	41	42	43	44
M(n)	175	182	192	200	213	223	235	247	261	274
n	45	46	47	48	49	50	51	52	53	54
M(n)	289	303	319	333	349	364	380	394	408	423
n	55	56	57	58	59	60	61	62	63	64
M(n)	439	455	480	506	533	561	590	620	651	651
n	65	66	67	68	69	70	71	72	73	74

M(n)	652	654	662	678	696	713	731	748	766	783
n	75	76	77	78	79	80	81	82	83	84
M(n)	803	820	840	862	886	909	933	957	983	1008
n	85	86	87	88	89	90	91	92	93	94
M(n)	1035	1061	1089	1115	1143	1171	1200	1228	1257	1287
n	95	96	97	98	99	100	101	102	103	104
M(n)	1319	1348	1379	1410	1443	1474	1504	1534	1566	1595

故而 m 的最终取值为：

$$0 < m \leq M(n)\left(不考虑溢出\right)$$

$$\frac{n}{3} < m \leq M(n)\left(考虑溢出\right)$$

3.3 m=20，n=10 情况下最佳分配方案

定义一：设m_i代表第 i 组的评委编号（0<=i<20）

定义二：设(m_i, m_j)代表第 i 组和第 j 组中相同评委编号的个数，为避免重复运算，令 i>j

定义三：设 Q 为相似相似度，则需要得到 Q 最小的方案

在定义一，二下，因为题目说使得任意不同的两个组的评委尽量不同，那么我们认为任意两组有一位评委相同和任意两组有两位评委相同的相似程度不是线性关系的，即 Q（2（(m_i，m_j）＝1））≠Q（(m_i，m_j）＝2）。我们选取$(m_i, m_j)^2$作为判断标准，

我们需要得到 Q = $\sum_{i=2}^{20}\sum_{j=1}^{i-1}(m_i, m_j)^2$最小的方案，

这里我们采取贪心算法，即每一步都求局部最优解以求获得最终最优解，现在我们有两种初始化方式：

1. 什么也不设置
2. 由上题可得当 n=10 时，M(n)=13，我们先设置 13 组评委两两不相同的组，之后再采取贪心算法。

在初始化方式 1 下，我们可以获得如下结果：

组编号	评委 1 编号	评委 2 编号	评委 3 编号
0	0	1	2
1	3	4	5
2	6	7	8
3	0	3	9
4	1	4	6
5	2	5	7
6	1	8	9

7	2	3	6
8	0	4	7
9	0	5	8
10	2	4	9
11	1	3	7
12	5	6	9
13	2	3	8
14	0	1	5
15	4	6	9
16	0	7	9
17	0	2	6
18	1	3	4
19	5	7	8

（表一）

最终相似度 Q=181

在初始化方式 2 下，我们可以得到如下结果

组编号	评委 1 编号	评委 2 编号	评委 3 编号
0	0	8	9
1	0	6	7
2	0	4	5
3	0	2	3
4	1	3	4
5	1	5	6
6	1	7	8
7	2	4	6
8	2	5	7
9	3	5	8
10	1	2	9
11	3	6	9
12	4	7	9
13	2	4	8
14	0	1	3
15	5	6	8
16	3	7	9
17	0	5	9
18	1	2	6
19	4	7	8

（表二）

最终相似度 Q=180

因为 180<181，我们选取方式 2 的初始化方法得到的最终分配结果，并且在这个结果中，每位评委都审评 6 组，工作分配合理，故最佳分配方案为表二。

四、 问题二的分析与解决

4.1 问题分析

题目中描述为“有的评委打分偏紧，有的评委打分偏松”，也就是说每个评委的评分标准不一样，而题目中要求的最终得分是以平均分的形式呈现的，显然直接作平均是不合理的，为此我们需要针对评委，对他们的数据分别进行调整。

4.2 数据总体分析

首先我们需要对于数据总体进行分析，由于分数分布大多为正态分布，因此我们需要对于数据进行显著性检验。

取显著性水平为 0.05，假设数据服从正态分布，备择假设为数据不服从正态分布。我们使用 K-S 检验。

通过计算得知，每位评委的打分情况和总体的打分情况的 p 值都大于 0.05，因此接受原假设，即数据呈正态分布。

	D 值	P 值
1	0.04160948206793014	0.8646668230578755
2	0.03750966329942118	0.9309464316194455
3	0.038148354614719326	0.922109539580267
4	0.056971052038189196	0.516453874001993
5	0.03814835461471899	0.9221095395802718
6	0.0409681982110246	0.8764410590933746
7	0.042236924912615326	0.8527035610217388
8	0.047107227612820435	0.7482171297952694
9	0.04127894872161053	0.8707945465230698
10	0.04088679513953558	0.8779014514982229

4.3 异常值处理

由于对于每个评委的打分情况不确定，而异常值对于结果的影响又较为显著，所以我们需要对数据进行异常值处理。

同时，每个评委的打分标准不同，因此如果作总体的异常值检验是不合理的，所以我们对于每个评委的打分分别作异常件检验，得出结果后再对总体进行验证。

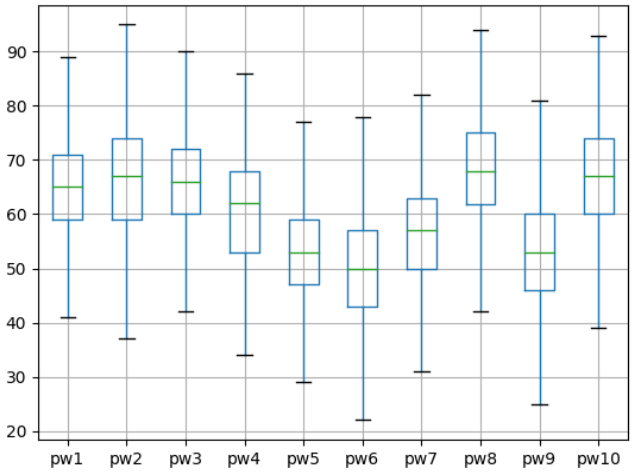
我们采用箱线图的方式来寻找异常值，结果如下：

	下限	上限	异常值
1	41.0	89.0	92,40,37
2	36.5	96.5	null
3	42.0	90.0	93,39
4	30.5	90.5	null
5	29.0	77.0	80,26
6	22.0	78.0	79

7	30.5	82.5	29
8	41.875	94.875	96,95
9	25.0	81.0	23,83
10	39.0	95.0	35

为了消除异常值，我们将超过上限的值规约为上限向下取整的整数值；将低于下限的值规约为下限向上取整的整数值.

调整异常值后的箱线图如图所示：



在这一节的最后，我们还对调整异常值后的数据进行了关于正态分布的检验，正态分布的检验结果如下，结果显示调分后所有评委的打分情况都符合正态分布。

	D 值	P 值
1	0.03986626230262025	0.8955226483836971
2	0.03750966329942118	0.9309464316194455
3	0.03721059679223626	0.9348820917816804
4	0.056971052038189196	0.516453874001993
5	0.03721059679223604	0.9348820917816831
6	0.04133371326770563	0.869787846748304
7	0.04217972833583972	0.8538115246386145
8	0.047228068946345125	0.7454327302335191
9	0.04082802726852086	0.8789508597256801
10	0.04257860367463401	0.8460146442996113

4.4 标准化处理

题目的评分要求对每位评委的打分结果平均处理。

由于各个评委的打分标准不同，简单的平均处理显然不能很好地反映该组的真实得分情况；同时，每位评委有自己的评分标准，在修正了异常值后，每位评委的打分都服从正态分布，因此评委的打分有其合理性。从现实视角

来说，如果简单地将十位评委的打分都按照一个统一的打分标准进行标准化处理，这无疑等价于一位评委重复十次打分，那么设置十位打分标准截然不同的评委就变得毫无意义。

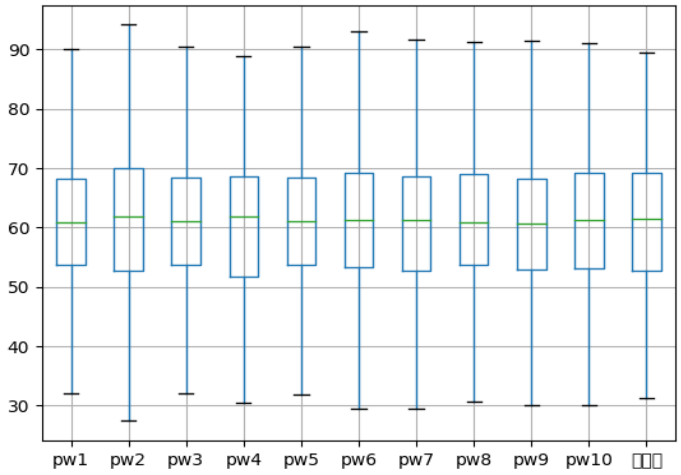
因此，为了保证结果的合理性，我们对于每位评委的全部打分作 Z-score 标准化处理，如此便可以直接将十位评委的分求平均。在最后给定一个具有现实意义的打分时，我们选择了总体样本的均值和标准差对每组的得分进行逆 Z-score 处理，而这个最终结果将用于评奖.

这样的标准化处理过程既充分考虑了每位评委的打分情况，又确保了结果的可靠性

$$Score_{m,n} = (Originalscore_{m,n} - N_n)/\sigma_n \times \sigma_a ll + N_a ll$$

$$Score_m = \sum Score_{m,i}/10$$

最后，为了确保最终调整结果的可靠性，我们对于调整后的数据再次进行异常值处理和正态分布检验
结果的箱线图和检验表如下：



	D 值	P 值
1	0.039839662704691525	0.8959643995136999
2	0.037509663299421514	0.930946431619441
3	0.03722622882793025	0.9346795994544034
4	0.05697105203818942	0.5164538740019882
5	0.03721059679223582	0.934882091781686
6	0.04133371326770563	0.869787846748304
7	0.04217972833583927	0.8538115246386233
8	0.04722806894634496	0.7454327302335232
9	0.040843201085901115	0.8786802984875202
10	0.0429062368090794	0.8394908800191188

4.5 最终获奖结果

1） 获一等奖的组为：

第 69 组，得分 89.45204275698605
第 151 组，得分 88.93927345327788
第 134 组，得分 86.04991061838611
第 15 组，得分 85.05440514858631
第 183 组，得分 84.9266639999702
第 153 组，得分 84.92441655607378
第 5 组，得分 84.81167891420571
第 61 组，得分 82.98460948941845
第 165 组，得分 82.75447539742956
第 83 组，得分 81.97797268546626
第 173 组，得分 81.86782311993224
第 148 组，得分 79.33017226527319
第 98 组，得分 78.53440376183383
第 10 组，得分 78.1396204196094
第 163 组，得分 77.4208591686517
第 147 组，得分 77.25033389957355
第 139 组，得分 77.22431005347678
第 89 组，得分 76.79239831255157
第 161 组，得分 75.82557127884458
第 101 组，得分 74.91813502140437

2） 获二等奖的组为：

第 65 组，得分 74.84871711919979
第 25 组，得分 74.14387310670755
第 8 组，得分 74.1209583705734
第 192 组，得分 73.74540884081318
第 149 组，得分 73.67666573241688
第 181 组，得分 73.54689792532858
第 7 组，得分 73.43732507850831
第 199 组，得分 73.25833699217354
第 172 组，得分 73.18482700524268
第 48 组，得分 72.75822388413002
第 166 组，得分 72.2861070027786
第 132 组，得分 72.09367040955381
第 177 组，得分 72.07499421567768
第 100 组，得分 71.88605116725782
第 168 组，得分 71.7173295642154
第 129 组，得分 71.41132925529902
第 122 组，得分 71.40768921259418
第 178 组，得分 71.1055312123787
第 156 组，得分 71.09330769429057
第 76 组，得分 70.98154244305127
第 102 组，得分 70.82369321503347

第 196 组，得分 70.7753948278049
第 81 组，得分 70.57291095433524
第 121 组，得分 70.1589485323026
第 90 组，得分 69.94405404633913
第 162 组，得分 69.91712685177188
第 180 组，得分 69.64604206980034
第 91 组，得分 69.49886975460124
第 115 组，得分 69.45562306425163
第 94 组，得分 69.22565258148249

3) 获三等奖的组为：
第 27 组，得分 68.32238268817862
第 87 组，得分 68.13482991396522
第 41 组，得分 68.01299989992371
第 128 组，得分 67.78370241971274
第 113 组，得分 67.73986757496517
第 97 组，得分 67.33253432330164
第 42 组，得分 67.2010977879061
第 198 组，得分 67.05855252806205
第 49 组，得分 66.73164781571225
第 176 组，得分 66.63794931448882
第 171 组，得分 66.43865526906842
第 62 组，得分 66.30723121843536
第 46 组，得分 66.29501152138089
第 2 组，得分 66.05392537824821
第 68 组，得分 66.00766628506396
第 21 组，得分 65.8763592928166
第 175 组，得分 65.74249183001453
第 103 组，得分 65.52078242708284
第 189 组，得分 65.39883531234669
第 70 组，得分 65.18141233899088
第 75 组，得分 65.09632133703548
第 35 组，得分 65.09007668954861
第 104 组，得分 64.99060993610053
第 123 组，得分 64.92576366087346
第 86 组，得分 64.88045198928906
第 14 组，得分 64.72974099851305
第 95 组，得分 64.64847282504584
第 24 组，得分 64.44552400524725
第 187 组，得分 64.05246853480101
第 17 组，得分 64.03494490039523
第 71 组，得分 63.83821130410786
第 19 组，得分 63.827645033804345
第 137 组，得分 63.51857837670784
第 54 组，得分 63.51489498883624

第 114 组，得分 63.44875635481168
第 28 组，得分 63.331047844461935
第 124 组，得分 63.21015552466213
第 88 组，得分 63.181659228071055
第 157 组，得分 63.087411761355
第 159 组，得分 62.8620196369332
第 136 组，得分 62.75022438398626
第 79 组，得分 62.71442200592664
第 119 组，得分 62.69594299504693
第 133 组，得分 62.526170680153314
第 141 组，得分 62.47541818820234
第 34 组，得分 62.32818972975366
第 110 组，得分 62.23126449633846
第 58 组，得分 61.64143116201667
第 55 组，得分 61.46693547976238

五、 问题三的分析与解决

5.1 问题分析

对于该问题，我们采用 Z-score 规约进行调分。如此我们便可以使用调分后的打分直接作平均值来评估小组的论文得分情况。

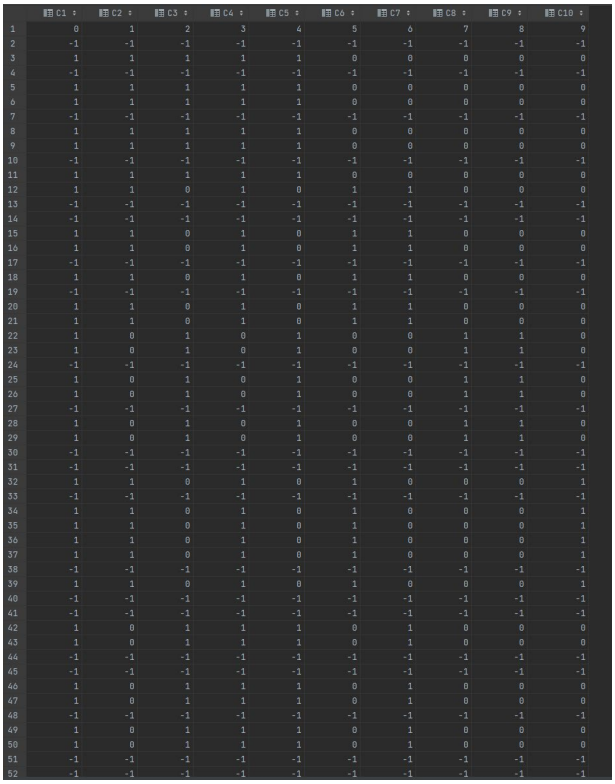
由于题目中存在四个打分步骤，而每个打分步骤结束以后都要进行调分，对调分后的结果进行排名，然后选出这一步骤中的淘汰小组和获奖小组。同时，由于一旦某个小组获得了奖项或者被淘汰，那么该小组的评委打分数据就不能再次被使用。这就造成了数据集过少和数据集状态问题。而对于缺失的数据集，直接使用 Z-score 规约会使得结果出现很大的误差，所以我们需要填充数据集。为此我们采用了掩蔽矩阵，领域预测和 Z-score 规约来设计我们的调分算法。

5.2 模型部分组件构成

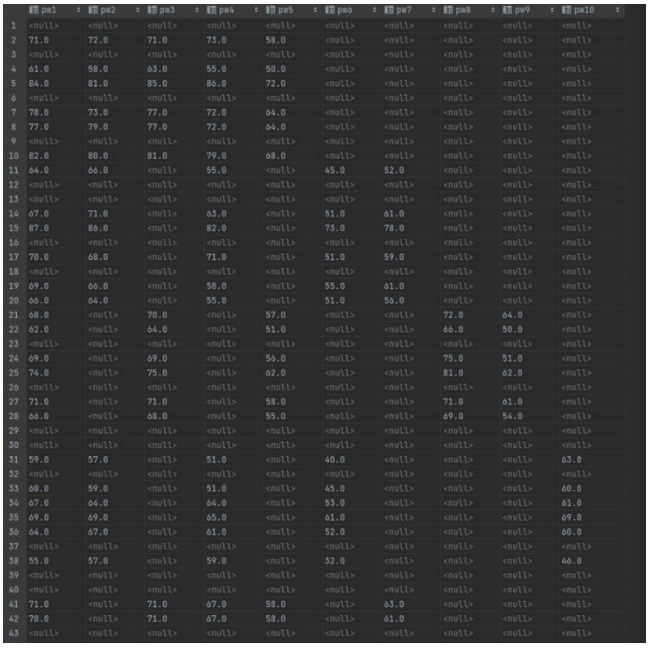
5.2.1 掩蔽矩阵

为了解决数据集状态问题，即每个步骤中可用的数据都是变化的，我们设计了掩蔽矩阵来描述每个步骤中可以被使用的数据。以下左图为矩阵数值的公式和部分矩阵展示。

在根据给定的分组情况获取对用的掩蔽矩阵后，由于掩蔽矩阵可以与原数据对应，我们使用掩蔽矩阵对原数据进行处理，保留本轮需要使用的数据，将不需要的数据置为 NaN，最终得到的处理完毕的数据如下右图所示。



	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	0	1	2	3	4	5	6	7	8	9
2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	1	1	1	1	1	0	0	0	0	0
4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	1	1	0	0	0	0	0
6	1	1	1	1	1	0	0	0	0	0
7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	1	1	1	1	1	0	0	0	0	0
9	1	1	1	1	1	0	0	0	0	0
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	1	1	1	1	1	0	0	0	0	0
12	1	1	0	1	0	1	1	0	0	0
13	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
14	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
15	1	1	0	1	0	1	1	0	0	0
16	1	1	0	1	0	1	1	0	0	0
17	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	1	1	0	1	0	1	1	0	0	0
19	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
20	1	1	0	1	0	1	1	0	0	0
21	1	1	0	1	0	1	1	0	0	0
22	1	0	1	0	1	0	0	1	1	0
23	1	0	1	0	1	0	0	1	1	0
24	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
25	1	0	1	0	1	0	0	1	1	0
26	1	0	1	0	1	0	0	1	1	0
27	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
28	1	0	1	0	1	0	0	1	1	0
29	1	0	1	0	1	0	0	1	1	0
30	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
31	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
32	1	1	0	1	0	1	0	0	0	1
33	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
34	1	1	0	1	0	1	0	0	0	1
35	1	1	0	1	0	1	0	0	0	1
36	1	1	0	1	0	1	0	0	0	1
37	1	1	0	1	0	1	0	0	0	1
38	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
39	1	1	0	1	0	1	0	0	0	1
40	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
41	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
42	1	0	1	1	1	0	1	0	0	0
43	1	0	1	1	1	0	1	0	0	0
44	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
45	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
46	1	0	1	1	1	0	1	0	0	0
47	1	0	1	1	1	0	1	0	0	0
48	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
49	1	0	1	1	1	0	1	0	0	0
50	1	0	1	1	1	0	1	0	0	0
51	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
52	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



	pw1	pw2	pw3	pw4	pw5	pw6	pw7	pw8	pw9	pw10
1	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
2	71.0	72.0	71.0	73.0	58.0	<null>	<null>	<null>	<null>	<null>
3	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
4	61.0	58.0	63.0	55.0	50.0	<null>	<null>	<null>	<null>	<null>
5	84.0	81.0	85.0	86.0	72.0	<null>	<null>	<null>	<null>	<null>
6	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
7	78.0	73.0	77.0	72.0	64.0	<null>	<null>	<null>	<null>	<null>
8	77.0	79.0	77.0	72.0	64.0	<null>	<null>	<null>	<null>	<null>
9	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
10	82.0	80.0	81.0	79.0	68.0	<null>	<null>	<null>	<null>	<null>
11	64.0	66.0	<null>	55.0	<null>	45.0	52.0	<null>	<null>	<null>
12	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
13	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
14	67.0	71.0	<null>	63.0	<null>	51.0	61.0	<null>	<null>	<null>
15	87.0	86.0	<null>	82.0	<null>	73.0	78.0	<null>	<null>	<null>
16	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
17	70.0	68.0	<null>	71.0	<null>	51.0	59.0	<null>	<null>	<null>
18	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
19	69.0	66.0	<null>	58.0	<null>	55.0	61.0	<null>	<null>	<null>
20	66.0	64.0	<null>	55.0	<null>	51.0	56.0	<null>	<null>	<null>
21	68.0	<null>	70.0	<null>	57.0	<null>	72.0	64.0	<null>	
22	62.0	<null>	64.0	<null>	51.0	<null>	<null>	66.0	58.0	<null>
23	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
24	69.0	<null>	69.0	<null>	56.0	<null>	<null>	75.0	51.0	<null>
25	74.0	<null>	75.0	<null>	62.0	<null>	<null>	81.0	62.0	<null>
26	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
27	71.0	<null>	71.0	<null>	58.0	<null>	<null>	71.0	61.0	<null>
28	66.0	<null>	68.0	<null>	55.0	<null>	<null>	69.0	54.0	<null>
29	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
30	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
31	59.0	57.0	<null>	51.0	<null>	40.0	<null>	<null>	63.0	
32	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
33	68.0	59.0	<null>	51.0	<null>	45.0	<null>	<null>	<null>	60.0
34	67.0	64.0	<null>	64.0	<null>	53.0	<null>	<null>	<null>	61.0
35	69.0	69.0	<null>	65.0	<null>	61.0	<null>	<null>	<null>	69.0
36	64.0	67.0	<null>	61.0	<null>	52.0	<null>	<null>	<null>	60.0
37	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
38	55.0	57.0	<null>	59.0	<null>	32.0	<null>	<null>	<null>	46.0
39	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
40	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
41	71.0	<null>	71.0	67.0	58.0	<null>	63.0	<null>	<null>	<null>
42	70.0	<null>	71.0	67.0	58.0	<null>	61.0	<null>	<null>	<null>
43	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>

5.2.2 邻域预测

由于每个评委不可能对所有论文打分，所以出现了数据集的缺失。同时，如果每个评委对每篇论文都打分了那么这样的结果一定是最好的。为了使现有的缺失数据集达到后者的完备状态以进行更好地调分，我们采用了领域预测的方式，通过计算每个评委在每篇论文的打分上的相关性，预测该评委在其未打分的小组上的可能的打分，使得每个小组都被全部十位评委打分，从而使数据集达到如问题二的数据的完备形式。

5.2.3 Z-score 规约

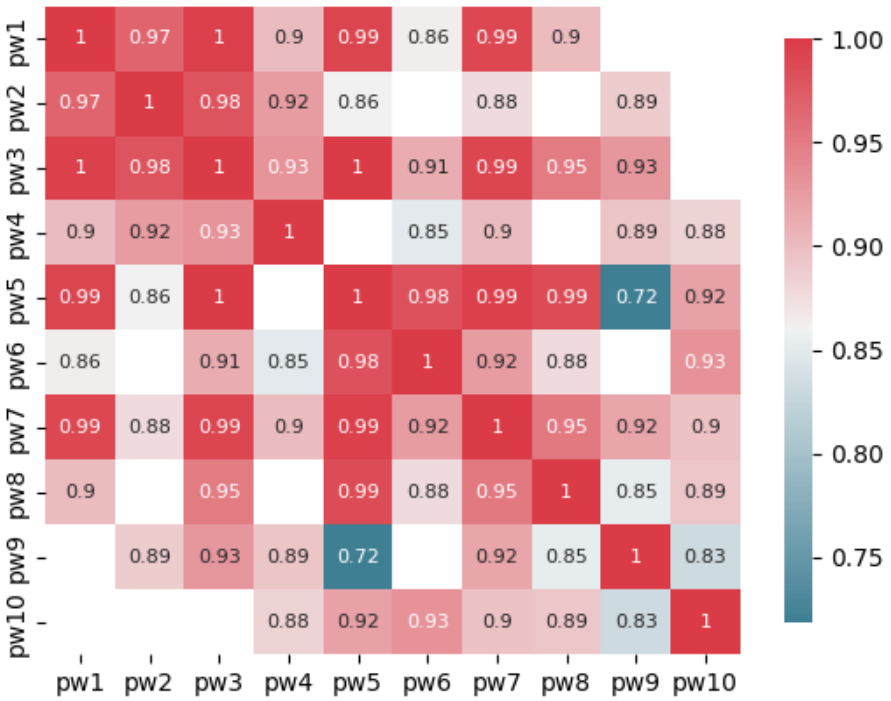
在获取了完备数据集的情况下，我们便可以参照问题二的解法，对于数据集作 Z-score 规约，从而使得每个评委的打分情况全被被规约到一个统一的标准，使直接的数值平均能反映小组得分的真实情况。

5.3 模型详细说明

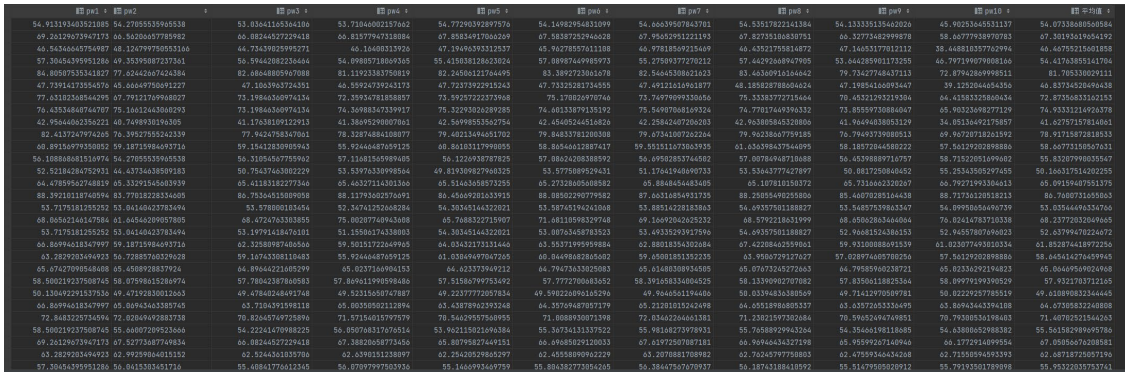
5.3.1 步骤一

在步骤一中，各个评委的分组情况是给定的，因此我们需要根据分组情况，获取在这一轮中的掩蔽矩阵。在获取了掩蔽矩阵后，我们根据该掩蔽矩阵对于原始的数据作掩蔽处理，获取在步骤一中可见的数据。在获取了可见数据后，为了扩充数据集，我们对于该可见数据作领域预测，填补表中的数据空缺。

领域预测计算的相关性矩阵如下：



在获取到填补结果后，便可以对该结果作 Z-score 规约，然后对于 Z-score 规约后的结果，我们可以直接根据掩蔽矩阵，计算该小组在步骤一中的对应的三个评委的打分的平均值，最终部分结果如下：



根据步骤一的排序方式，我们对每个 10 组大组进行排序，淘汰该大组中排名为后 40%的小组，更新掩蔽矩阵，将已经被淘汰的小组对应的掩蔽矩阵中的值置为-1.

5.3.2 步骤二

在步骤二中，各个评委的分组情况是给定的，因此我们需要根据分组情况，获取在这一轮中的掩蔽矩阵，步骤二的掩蔽矩阵中的部分数据如下图：

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	0	0	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	0	0	0	0	0
1	1	0	1	0	1	1	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	0	1	0	1	1	0	0	0
1	1	0	1	0	1	1	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	0	1	0	1	1	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	0	1	0	1	1	0	0	0
1	1	0	1	0	1	1	0	0	0
1	0	1	0	1	0	0	1	1	0
1	0	1	0	1	0	0	1	1	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	0	1	0	1	0	0	1	1	0
1	0	1	0	1	0	0	1	1	0
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	0	1	0	1	0	0	1	1	0
1	0	1	0	1	0	0	1	1	0

在获取了掩蔽矩阵后，我们根据该掩蔽矩阵对于原始的数据作掩蔽处理，获取在步骤二中可见的数据。步骤二的可见数据的部分数据如下：

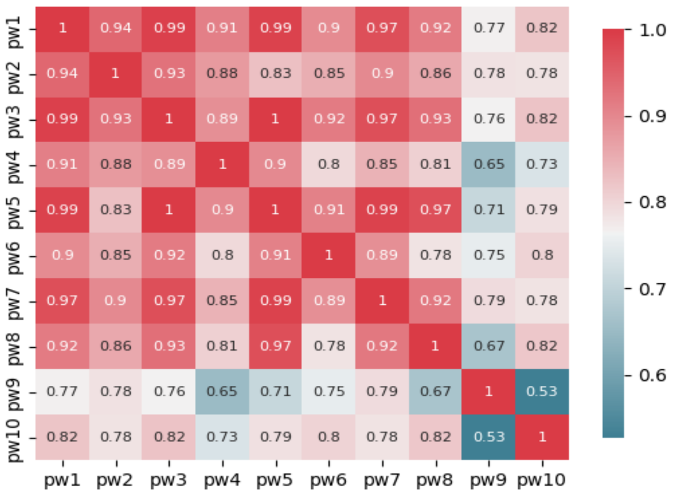
pw1	pw2	pw3	pw4	pw5	pw6	pw7	pw8	pw9	pw10
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
71.0	72.0	71.0	73.0	58.0	<null>	<null>	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
61.0	58.0	63.0	55.0	50.0	<null>	<null>	<null>	<null>	<null>
84.0	81.0	85.0	86.0	72.0	<null>	<null>	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
78.0	73.0	77.0	72.0	64.0	<null>	<null>	<null>	<null>	<null>
77.0	79.0	77.0	72.0	64.0	<null>	<null>	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
82.0	80.0	81.0	79.0	68.0	<null>	<null>	<null>	<null>	<null>
64.0	66.0	<null>	55.0	<null>	45.0	52.0	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
67.0	71.0	<null>	63.0	<null>	51.0	61.0	<null>	<null>	<null>
87.0	86.0	<null>	82.0	<null>	73.0	78.0	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
70.0	68.0	<null>	71.0	<null>	51.0	59.0	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
69.0	66.0	<null>	58.0	<null>	55.0	61.0	<null>	<null>	<null>
66.0	64.0	<null>	55.0	<null>	51.0	56.0	<null>	<null>	<null>
68.0	<null>	70.0	<null>	57.0	<null>	<null>	72.0	64.0	<null>
62.0	<null>	64.0	<null>	51.0	<null>	<null>	66.0	50.0	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
69.0	<null>	69.0	<null>	56.0	<null>	<null>	75.0	51.0	<null>
74.0	<null>	75.0	<null>	62.0	<null>	<null>	81.0	62.0	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
71.0	<null>	71.0	<null>	58.0	<null>	<null>	71.0	61.0	<null>
66.0	<null>	68.0	<null>	55.0	<null>	<null>	69.0	54.0	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>

如图所示，第一组在步骤一中已被淘汰，因此在步骤二中，第一组的数据便不再可用。

在获取了可见数据后，为了扩充数据集，我们对于该可见数据作领域预测，填补表中的数据空缺。填补的部分结果如下：

pw1	pw2	pw3	pw4	pw5	pw6	pw7	pw8	pw9	pw10
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
71.0	72.0	71.0	73.0	58.0	55.66444290195573	63.30394851769106	74.789928492653	61.07958563719563	74.38426666873583
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
61.0	58.0	63.0	55.0	50.0	44.263640423284656	51.9292065380131	63.435459803644264	49.609016081244796	62.92066687990149
84.0	81.0	85.0	86.0	72.0	68.30221291085235	75.95622042201066	87.44907465793855	73.62628001204965	86.99806494812515
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
78.0	73.0	77.0	72.0	64.0	59.619840178942546	67.28735769565282	78.77905502965041	64.97990611586181	78.30009446149782
77.0	79.0	77.0	72.0	64.0	60.58505567868299	68.2301032512347	79.72287576513901	66.0523673267404	79.2831881538181
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
82.0	80.0	81.0	79.0	68.0	64.75112618507632	72.40513622929366	83.8924367156798	70.15214910789061	83.45260017121417
64.0	66.0	62.905695874035665	55.0	50.30892799546431	45.0	52.0	65.1389546097199	51.445338299784645	64.66778135568498
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
67.0	71.0	69.07614014224183	63.0	56.518665646196816	51.0	61.0	71.31693885347102	57.604275496632944	70.81204107253124
87.0	86.0	87.66946248978736	82.0	75.17747137657277	73.0	78.0	89.84898821613794	76.15038612619523	89.43827889003094
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
70.0	68.0	70.21914439049812	71.0	57.75039013183119	51.0	59.0	72.43444630141646	58.557683015883974	71.93611705461596
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
69.0	66.0	68.34524191717352	58.0	55.866467935017795	55.0	61.0	70.53335860176227	56.899835219506826	70.11753322898883
66.0	64.0	64.93291649442224	55.0	52.41968408203033	51.0	56.0	67.1217143317179	53.48565752183457	66.7136120228118 7
68.0	71.11242126297671	70.0	64.65057911759652	57.0	53.65633746275084	61.270596287867804	72.0	64.0	72.12416480876381
62.0	63.666525440020045	64.0	57.39692363833073	51.0	46.2437391773877	53.90764035460672	66.0	50.0	65.00658156954441
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
69.0	69.1694115691399	69.0	62.98464660812828	56.0	51.69471129936263	59.436310099533664	75.0	51.0	70.7485038398912
74.0	75.85592000812665	75.0	69.54920089829245	62.0	58.36719403082404	66.07871319406784	81.0	62.0	77.22954991726563
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>
71.0	71.42808568228278	71.0	65.07341717637921	58.0	54.002576278412704	61.61237755672049	71.0	61.0	72.56734658558898
66.0	67.46865515505108	68.0	61.20277615316159	55.0	50.05961695966886	57.70982560859434	69.0	54.0	68.79012482062414
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>

领域预测计算的相关性矩阵如下：



在获取到填补结果后，便可以对该结果作 Z-score 规约，然后对于 Z-score 规约后的结果，我们可以直接根据掩蔽矩阵，计算该小组在步骤二中的对应的五个评委的打分的平均值，最终部分结果如下：

pw1	pw2	pw3	pw4	pw5	pw6	pw7	pw8	pw9	pw10	平均值
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
66.82797262180304	66.83508366206482	65.8227524933899	75.76401549401133	65.50839462339357	67.73700607704849	68.09265478306509	68.06564138426441	68.08049900928907	68.62896159425951	68.15164377893254
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
53.264746572367656	47.34942631574623	55.10593347546883	51.86923160143301	54.59255956983936	52.87951218129813	52.633938046399926	53.96626977802773	53.63102547688794	54.20210880144266	52.436379506971015
84.46016648606904	79.36157767041247	84.57718577475175	93.02135941642902	84.61110596711346	84.2065125976952	85.28758437299157	83.78509166332532	83.88556670050292	84.50333040398398	85.20627906295516
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
76.32223085640781	68.226916329659	73.86036675683069	74.4365274999792	73.69527091355924	72.8916687429479	73.50626277860103	73.01912524774728	72.99373171752293	73.55700096835061	73.3082624712871 8
74.96590825146427	76.57791233522411	73.86036675683069	74.4365274999792	73.69527091355924	74.14953488195927	74.78749065388183	74.19111133039041	74.34471083024516	74.79421687277059	74.707197151411 5
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
81.74752127618197	77.96974500281829	79.21877626579122	83.72894345820411	79.15318844033635	79.57874631104978	80.46152284637965	79.36864899302326	79.50920489014277	80.04139005707546	80.36363488866638
57.33371438719827	58.48408765649971	54.97960344414346	51.86923160143301	55.01408545007856	53.83913392166187	52.73014911453832	56.0815784690812	55.94424010536173	56.40083902742962 6	54.85126333626623
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
61.40268220202888 5	65.44325099447063	63.24554525446018	62.48913555369004	63.48714444023371	61.658317136288815	64.96149914239409	63.75306870045074	63.70265171513323	64.13334296062817	63.19097700577449
88.52913430089966	86.3207410083834	88.15320407202428	87.71140744030049	88.94670014637074	90.32865558992094	88.06516030612168	86.76517589330376	87.06518252615275	87.57432101525286	88.19101972912524
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
65.4716500168595	61.26775299168808	64.77671646261987	73.10903950594708	65.16780710480981	61.658317136288815	62.24342135842615	65.14072964812433	64.90365892299789	65.54798400010972	64.75003620184192
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
64.11532741191597	58.48408765649971	62.26643225463809	55.85169558352939	62.59723411054176	66.87110594604012	64.96149914239409	62.78006067561515	62.81526847842111	63.259310143118064	62.05674314807585
60.04635959708535	55.700422321311336	57.69527300640353	51.86923160143301	57.89416860998359	61.658317136288815	58.16630468247422	58.54366338842133	58.514429706920346	58.97550129908514	57.48812706771855
62.75900480697243	65.59972258081382	64.48315011614976	64.68025951549957	64.1439152416993	65.1200486864765	65.32925002163294	64.6012574795538	71.75934423429308	65.78464066268384	65.54933437573368
54.6210691773112	55.23628153491964	56.44553585270896	55.051118953928864	55.95703895153363	55.45997133819584	55.32270654836695	57.15077855760211	54.12354781922593	56.827216509334185	55.65959407167636
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
64.11532741191597	62.89537821547956	63.14354773890963	62.46875411031266	62.77943586000502	62.56366295812014	62.8363837526578	68.32649694052964	55.383247563159294	64.05338183855082	62.749611102903906
70.89694043663366	72.20187909300552	71.18116200235043	71.18312111667771	70.9663121501707	71.25922378707534	71.86366789436445	75.77697586248132	69.23994474642635	72.20972883461525	71.61226703961249
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0
66.82797262180304	66.03907463160115	65.8227524933899	65.2415719619534 4	65.50839462339357	65.57126614242243	65.79374405855536	63.35951099256185	67.98024500249296	66.34238154447415	65.89977514672826
60.04635959708535	60.52820987873041	61.8039453616695	60.10334247442381	61.41495647831074	60.4328125889093	60.4900241830641	60.87601801857795	59.1623467949594	61.58877679790505	60.660725250120585
<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	<null>	0.0

根据步骤二的排序方式，我们对每个 10 组大组的所有未被淘汰的论文进行排序，淘汰排名为后 20 名的小组，排名 31 到 40 的小组，获二等奖。排名 61 到 100 的小组，获三等奖。

5.3.3 步骤三

在步骤三中，我们需要对步骤二中排名 1 到 30 和 41 到 60 的论文，每次额外随机抽取两位评委进行评分，我们对两组论文分别随机选取评委，生成他们对应的掩蔽矩阵，如下图所示：

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
0	1	2	3	4	5	6	7	8	9	10
1	1	1	0	0	1	1	0	1	1	176
0	0	1	1	1	0	1	1	1	1	148
1	1	1	1	1	0	0	1	1	0	6
1	1	1	1	1	1	1	0	0	0	47
1	0	1	0	0	1	1	1	1	1	198
1	0	1	0	0	1	1	1	1	1	191
1	0	1	1	1	1	0	0	1	1	180
1	1	1	0	0	0	1	1	1	1	64
1	1	1	0	0	1	1	0	1	1	171
1	1	1	1	1	0	0	1	1	0	7
0	1	1	1	1	1	0	1	1	0	100
0	1	1	1	0	1	0	1	1	1	162
0	1	1	1	0	1	0	1	1	1	160
1	1	0	0	1	1	1	0	1	1	97
1	1	0	0	1	1	1	1	1	0	138
0	0	1	1	1	0	1	1	1	1	146
0	0	1	1	1	0	1	1	1	1	147
1	1	0	1	1	1	1	1	0	0	88
1	1	1	1	1	0	0	1	1	0	9
1	1	0	1	1	1	1	1	0	0	82
1	1	1	0	0	0	1	1	1	1	60
1	1	1	0	0	1	1	0	1	1	172
0	1	1	1	0	1	0	1	1	1	164
1	0	1	1	1	1	0	0	1	1	182
1	1	1	1	1	0	0	1	1	0	4
1	0	0	0	1	1	1	1	1	1	152
1	1	0	0	1	1	1	1	1	0	133
1	1	0	1	0	1	1	0	1	1	14
1	0	0	0	1	1	1	1	1	1	150
1	1	1	0	0	0	1	1	1	1	68

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
0	1	2	3	4	5	6	7	8	9	10
1	1	0	0	1	1	1	0	1	1	96
1	1	1	0	1	0	1	1	1	0	26
1	1	1	1	1	1	1	0	0	0	41
1	1	0	0	1	1	1	0	1	1	90
1	1	0	1	1	1	1	1	0	0	86
1	1	1	1	1	1	1	0	0	0	45
1	1	0	0	1	1	1	0	1	1	93
1	1	1	0	0	1	1	0	1	1	170
1	1	1	1	1	1	1	0	0	0	40
0	1	1	1	1	1	1	1	0	0	112
1	0	1	1	1	1	0	0	1	1	185
1	1	1	1	1	0	0	1	1	0	1
1	0	1	0	0	1	1	1	1	1	197
1	1	0	1	1	1	1	1	0	0	89
1	0	1	0	0	1	1	1	1	1	121
1	1	0	1	1	1	1	1	0	0	80
0	1	1	1	0	1	0	1	1	1	161
1	0	1	0	0	1	1	1	1	1	120
1	0	0	0	1	1	1	1	1	1	155
0	1	1	1	1	1	1	1	0	0	114

需要指出的是，由于先前的已经被淘汰的小组和已经获奖的小组的数据无法使用，于是我们缩小了掩蔽矩阵，添加了新的列来表征对应的小组号。

在获取了掩蔽矩阵后，我们根据该掩蔽矩阵对于原始的数据作掩蔽处理，获取在步骤三中可见的数据。步骤三的可见数据的部分数据如下：

0	1	2	3	4	5	6	7	8	9	index
74.0	74.0	76.0	<null>	<null>	60.0	68.0	<null>	52.0	80.0	176
<null>	<null>	77.0	69.0	64.0	<null>	68.0	80.0	62.0	77.0	148
78.0	73.0	77.0	72.0	64.0	<null>	<null>	80.0	68.0	<null>	6
76.0	81.0	77.0	71.0	64.0	61.0	67.0	<null>	<null>	<null>	47
75.0	<null>	76.0	<null>	<null>	64.0	66.0	75.0	67.0	78.0	198
75.0	<null>	77.0	<null>	<null>	63.0	68.0	76.0	67.0	76.0	191
75.0	<null>	78.0	68.0	65.0	58.0	<null>	<null>	67.0	78.0	180
77.0	78.0	78.0	<null>	<null>	<null>	69.0	82.0	61.0	80.0	64
76.0	75.0	77.0	<null>	<null>	63.0	66.0	<null>	65.0	83.0	171
77.0	79.0	77.0	72.0	64.0	<null>	<null>	78.0	62.0	<null>	7
<null>	78.0	78.0	77.0	65.0	58.0	<null>	79.0	69.0	<null>	100
<null>	81.0	80.0	69.0	<null>	59.0	<null>	85.0	68.0	86.0	162
<null>	75.0	78.0	69.0	<null>	59.0	<null>	84.0	77.0	79.0	160
83.0	76.0	<null>	<null>	70.0	66.0	73.0	<null>	62.0	83.0	97
78.0	78.0	<null>	<null>	66.0	65.0	69.0	84.0	70.0	<null>	138
<null>	<null>	79.0	75.0	66.0	<null>	70.0	84.0	70.0	81.0	146
<null>	<null>	81.0	77.0	68.0	<null>	72.0	83.0	77.0	79.0	147
78.0	79.0	<null>	77.0	66.0	66.0	70.0	85.0	<null>	<null>	88
82.0	80.0	81.0	79.0	68.0	<null>	<null>	84.0	60.0	<null>	9
80.0	88.0	<null>	77.0	69.0	67.0	72.0	86.0	<null>	<null>	82
86.0	88.0	87.0	<null>	<null>	<null>	78.0	84.0	66.0	83.0	60
83.0	81.0	84.0	<null>	<null>	64.0	76.0	<null>	66.0	93.0	172
<null>	86.0	84.0	80.0	<null>	73.0	<null>	84.0	69.0	88.0	164
88.0	<null>	87.0	80.0	74.0	69.0	<null>	<null>	73.0	83.0	182
84.0	81.0	85.0	86.0	72.0	<null>	<null>	90.0	79.0	<null>	4
87.0	<null>	<null>	<null>	74.0	69.0	79.0	89.0	72.0	83.0	152
86.0	85.0	<null>	<null>	74.0	72.0	79.0	92.0	74.0	<null>	133
87.0	86.0	<null>	82.0	<null>	73.0	78.0	<null>	67.0	91.0	14
87.0	<null>	<null>	<null>	76.0	71.0	80.0	95.0	83.0	87.0	150
92.0	90.0	93.0	<null>	<null>	<null>	82.0	96.0	76.0	87.0	68

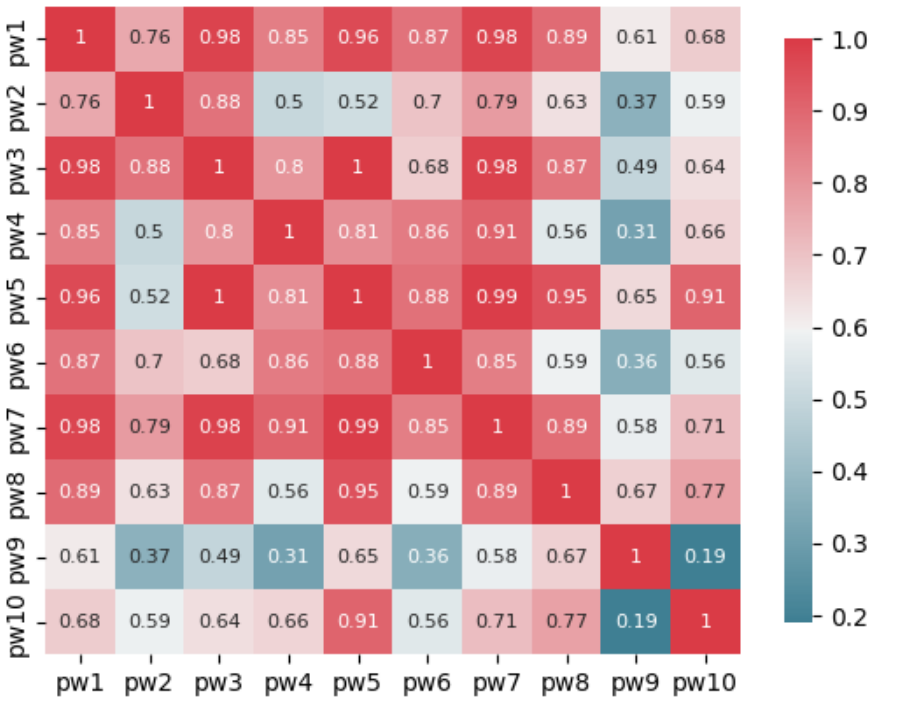
0	1	2	3	4	5	6	7	8	9	index
68.0	72.0	<null>	<null>	56.0	58.0	61.0	<null>	64.0	71.0	96
71.0	74.0	71.0	<null>	58.0	<null>	63.0	71.0	61.0	<null>	26
70.0	77.0	71.0	67.0	58.0	57.0	61.0	<null>	<null>	<null>	41
71.0	68.0	<null>	<null>	59.0	65.0	61.0	<null>	58.0	76.0	90
70.0	75.0	<null>	65.0	58.0	49.0	64.0	73.0	<null>	<null>	86
72.0	68.0	71.0	66.0	58.0	54.0	61.0	<null>	<null>	<null>	45
75.0	74.0	<null>	<null>	61.0	62.0	64.0	<null>	56.0	66.0	93
71.0	72.0	72.0	<null>	<null>	55.0	62.0	<null>	54.0	72.0	170
71.0	74.0	71.0	67.0	58.0	54.0	63.0	<null>	<null>	<null>	40
<null>	68.0	71.0	70.0	58.0	57.0	61.0	72.0	<null>	<null>	112
69.0	<null>	72.0	62.0	59.0	52.0	<null>	<null>	70.0	70.0	185
71.0	72.0	71.0	73.0	58.0	<null>	<null>	68.0	54.0	<null>	1
70.0	<null>	71.0	<null>	<null>	54.0	61.0	74.0	66.0	75.0	197
71.0	71.0	<null>	72.0	59.0	57.0	63.0	78.0	<null>	<null>	89
74.0	<null>	75.0	<null>	<null>	58.0	67.0	74.0	63.0	71.0	121
73.0	75.0	<null>	65.0	60.0	54.0	64.0	79.0	<null>	<null>	80
<null>	71.0	73.0	68.0	<null>	54.0	<null>	83.0	59.0	77.0	161
75.0	<null>	74.0	<null>	<null>	60.0	65.0	77.0	63.0	75.0	120
74.0	<null>	<null>	<null>	62.0	57.0	66.0	79.0	60.0	72.0	155
<null>	74.0	74.0	69.0	61.0	56.0	65.0	80.0	<null>	<null>	114

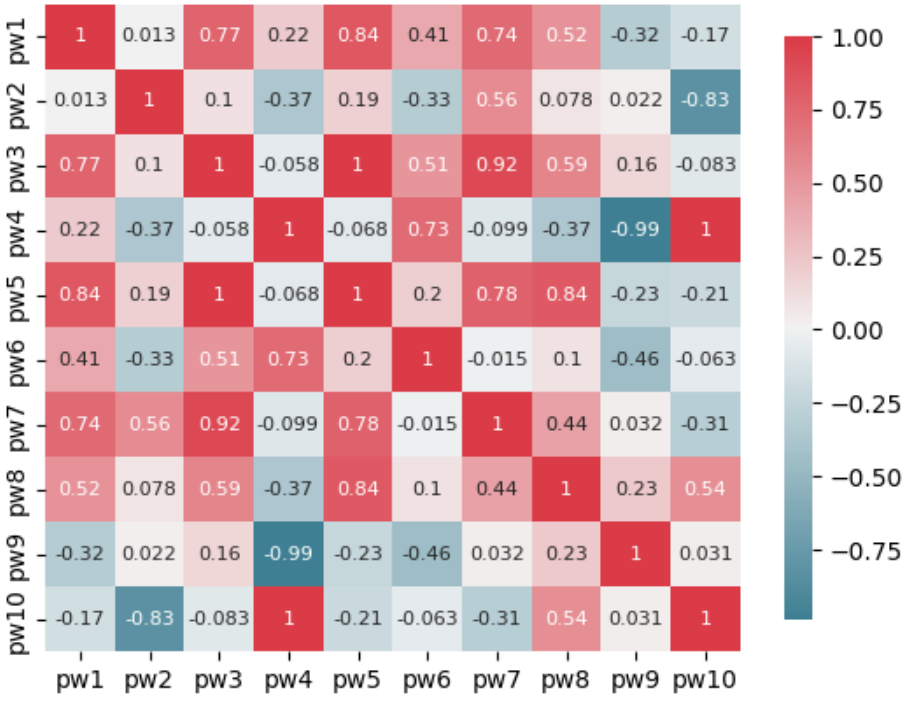
在获取了可见数据后，为了扩充数据集，我们对于该可见数据作领域预测，填补表中的数据空缺。填补的部分结果如下：

pm1	pm2	pm3	pm4	pm5	pm6	pm7	pm8	pm9	pm10	index
74.0	74.0	76.0	65.76392643022739	65.56820123819959	60.0	68.0	76.76480022631597	52.0	80.0	176
75.32567549953325	74.73675064247666	77.0	69.0	64.0	58.749548406377754	68.0	80.0	62.0	77.0	148
78.0	73.0	77.0	72.0	64.0	66.09958500833613	74.27996058730553	80.0	68.0	87.09153136760969	6
76.0	81.0	77.0	71.0	64.0	61.0	67.0	81.82560213767164	63.23684064239902	83.9282431176516	47
75.0	72.95360786056222	76.0	65.14787421660136	67.54158732750234	64.0	66.0	75.0	67.0	78.0	198
75.0	73.43505787165756	77.0	65.80164004248841	67.39414405836833	63.0	68.0	76.0	67.0	76.0	191
75.0	73.64978023519674	78.0	68.0	65.0	58.0	64.6400298123096	78.08565860414678	67.0	78.0	180
77.0	78.0	78.0	75.7643310683908	62.508141495792O7	64.21341642493263	69.0	82.0	61.0	80.0	64
76.0	75.0	77.0	67.9701336412414	67.7056259665867S	63.0	66.0	79.5040622997247	65.0	83.0	171
77.0	79.0	77.0	72.0	64.0	66.05376372311137	73.99364026130725	78.0	62.0	87.04930754504082	7
78.7631399637794	78.0	78.0	77.0	65.0	58.0	70.02105225423562	79.0	69.0	81.09166197321531	100
78.15094439811641	81.0	80.0	69.0	69.61350461867076	59.0	68.55635313989355	85.0	68.0	86.0	162
77.07425232370709	75.0	78.0	69.0	68.4188177288321	59.0	67.45094733404645	84.0	77.0	79.0	100
83.0	76.0	80.41085578305113	76.40733550478048	78.0	66.0	73.0	84.16644054336007	62.0	83.0	97
78.0	78.0	77.92749973180662	71.97766239696398	66.0	65.0	69.0	84.0	70.0	79.22706537978921	138
78.99084994841073	78.23716411540255	79.0	75.0	66.0	62.32196709939304	70.0	84.0	70.0	81.0	146
80.52120965951023	79.58305835996092	81.0	77.0	68.0	63.7350887964407B5	72.0	83.0	77.0	79.0	147
78.0	79.0	79.9690346559543O	77.0	66.0	66.0	70.0	85.0	66.8618888535266	83.05243097665316	88
82.0	80.0	81.0	79.0	68.0	69.85972812928324	77.63654206191890	84.0	60.0	90.86787978332315	9
80.0	88.0	82.78121461357539	77.0	69.0	67.0	72.0	86.0	69.14055391233035	86.38115053958643	82
86.0	88.0	87.0	82.65683242398944	68.59584840003755	71.27286035858488	78.0	84.0	66.0	83.0	60
83.0	81.0	84.0	73.89820706224012	73.81975899123731	64.0	76.0	85.75363302482359	66.0	93.0	172
83.29835088072812	86.0	84.0	80.0	74.58251198579083	73.0	73.56652486297597	84.0	69.0	88.0	164
88.0	82.79919044O1424	87.0	80.0	74.0	69.0	73.66952174O4634	87.71430358102031	73.0	83.0	182
84.0	81.0	85.0	86.0	72.0	74.92079910472489	83.06914969718347	90.0	79.0	95.02493336824045	4
87.0	83.58160937913165	83.82556850816478	77.08862966591427	74.0	69.0	79.0	89.0	72.0	83.0	152
86.0	85.0	85.58792738493956	79.02034232204206	74.0	72.0	79.0	92.0	74.0	86.89022567243411	133
87.0	86.0	88.821607126035	82.0	68.50121661022285	73.0	78.0	91.78037098533892	67.0	91.0	14
87.0	86.02980829018827	86.9807756655393	79.80840942416215	76.0	71.0	80.0	95.0	83.0	87.0	150
92.0	90.0	93.0	87.55900547938003	75.08018897712826	76.46571035307267	82.0	96.0	76.0	87.0	68

pm1	pm2	pm3	pm4	pm5	pm6	pm7	pm8	pm9	pm10	index
68.0	72.0	70.05908109330298	67.20361136182858	56.0	58.0	61.0	73.25678030595121	64.0	71.0	96
71.0	74.0	71.0	79.95819239760749	58.0	60.384260380O3407	63.0	71.0	61.0	63.711089198798209	26
70.0	77.0	71.0	67.0	58.0	57.0	61.0	65.07797882022218	58.7085956173O216	66.46553411693765	41
71.0	68.0	71.29384929948B9	72.46303316781754	59.0	65.0	61.0	73.94769381991426	58.0	70.0	90
70.0	75.0	69.36594380600868	65.0	58.0	49.0	64.0	73.0	60.3757958787683	69.75017792499469	86
72.0	68.0	71.0	66.0	58.0	54.0	61.0	65.8547692956223	51.9575273173778O4	68.96514128359034	45
75.0	74.0	73.47946042357058	69.063826664386O	61.0	62.0	64.0	74.29538216918911	56.0	66.0	93
71.0	72.0	72.0	61.637115777542004	67.0180479963479	55.0	62.0	87.75788043612704	54.0	72.0	170
71.0	74.0	71.0	67.0	58.0	54.0	63.0	65.10164962100346	50.92793468977554	67.03754187719471	40
70.39402439416048	68.0	71.0	70.0	58.0	57.0	61.0	72.0	61.19087230581014	74.75923880234596	112
69.0	66.11468163338495	72.0	62.0	59.0	52.0	67.62215432709947	88.91270873314679	70.0	70.0	185
71.0	72.0	71.0	73.0	58.0	63.230056382154714	56.85012794928312	68.0	54.0	65.46038675945408	1
70.0	61.21539959048111	71.0	56.40423228228759	66.9310568952182	54.0	61.0	74.0	66.0	75.0	197
71.0	71.0	71.26341912660891	72.0	59.0	57.0	63.0	78.0	57.192286938860B5	73.67324407761046	89
74.0	71.73308599307397	75.0	65.01360585059743	62.375115876438116	58.0	67.0	74.0	63.0	71.0	121
73.0	75.0	71.25028230131339	65.0	60.0	54.0	64.0	79.0	58.5179116945814	69.9286036733466	80
73.44358454163923	71.0	73.0	68.0	69.336717574O4327	54.0	67.78105379139987	83.0	59.0	77.0	161
75.0	69.54211826163507	74.0	66.36245314049454	62.347708863203904	60.0	65.0	77.0	63.0	75.0	120
74.0	68.18883809388716	78.95656661359462	61.76893984719B085	62.0	57.0	66.0	79.0	60.0	72.0	155
73.6359519a326501	74.0	74.0	69.0	61.0	56.0	65.0	80.0	62.829695367838B3	73.77459342288421	114

领域预测计算的相关性矩阵如下：


pw1 1 0.76 0.98 0.85 0.96 0.87 0.98 0.89 0.61 0.68
pw2 0.76 1 0.88 0.5 0.52 0.7 0.79 0.63 0.37 0.59
pw3 0.98 0.88 1 0.8 1 0.68 0.98 0.87 0.49 0.64
pw4 0.85 0.5 0.8 1 0.81 0.86 0.91 0.56 0.31 0.66
pw5 0.96 0.52 1 0.81 1 0.88 0.99 0.95 0.65 0.91
pw6 0.87 0.7 0.68 0.86 0.88 1 0.85 0.59 0.36 0.56
pw7 0.98 0.79 0.98 0.91 0.99 0.85 1 0.89 0.58 0.71
pw8 0.89 0.63 0.87 0.56 0.95 0.59 0.89 1 0.67 0.77
pw9 0.61 0.37 0.49 0.31 0.65 0.36 0.58 0.67 1 0.19
pw10 0.68 0.59 0.64 0.66 0.91 0.56 0.71 0.77 0.19 1
pw1 pw2 pw3 pw4 pw5 pw6 pw7 pw8 pw9 pw10
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2


pw1 1 0.013 0.77 0.22 0.84 0.41 0.74 0.52 -0.32 -0.17
pw2 0.013 1 0.1 -0.37 0.19 -0.33 0.56 0.078 0.022 -0.83
pw3 0.77 0.1 1 -0.058 1 0.51 0.92 0.59 0.16 -0.083
pw4 0.22 -0.37 -0.058 1 -0.068 0.73 -0.099 -0.37 -0.99 1
pw5 0.84 0.19 1 -0.068 1 0.2 0.78 0.84 -0.23 -0.21
pw6 0.41 -0.33 0.51 0.73 0.2 1 -0.015 0.1 -0.46 -0.063
pw7 0.74 0.56 0.92 -0.099 0.78 -0.015 1 0.44 0.032 -0.31
pw8 0.52 0.078 0.59 -0.37 0.84 0.1 0.44 1 0.23 0.54
pw9 -0.32 0.022 0.16 -0.99 -0.23 -0.46 0.032 0.23 1 0.031
pw10 -0.17 -0.83 -0.083 1 -0.21 -0.063 -0.31 0.54 0.031 1
pw1 pw2 pw3 pw4 pw5 pw6 pw7 pw8 pw9 pw10
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

在获取到填补结果后，便可以对该结果作 Z-score 规约，然后对于 Z-score 规约后的结果，我们可以直接根据掩蔽矩阵，计算该小组在步骤三中的对应的七个评委的打分的平均值，最终部分结果如下：

pw1	pw2	pw3	pw4	pw5	pw6	pw7	pw8	pw9	pw10	index	平均值
65.43226603317169	66.78519883860174	67.11002706563531	63.90837870754947	70.08347015837387	67.44617412831764	69.18374084519941	65.32426715821396	55.73120231133302	69.95004215959678	176	65.94837876883652
67.65404923287689	67.98034250175074	68.92768615126498	68.28884643205927	66.83563393234613	65.51476501256019	69.18374084519941	70.24467535900212	68.42434735558055	64.98908082071719	148	68.12771584230995
72.13611837256381	65.16301679457744	68.92768615126498	72.34975632492453	66.83563393234613	76.86740665236124	79.05714687975062	70.24467535900212	76.04023438212907	81.67697980898461	6	70.24244590240116
68.7841922028676	78.14047314677175	68.92768615126498	70.99611969396945	66.83563393234613	68.99074340434468	67.6115325449347	73.02122962394849	69.99428712263965	76.44599623839325	47	70.0409115823654 5
67.10822911801972	65.08776029899761	67.11002706563531	63.07446786468553	74.17046784250715	73.62445123242583	66.0393242447875 4	62.64018638615404	74.77091987770432	66.64273460034372	198	68.2765532178672
67.10822911801972	65.86875986209176	68.92768615126498	63.95942923448333	73.8651042162325	72.07988195639878	69.18374084519941	64.16108418072366	74.77091987770432	63.33542704109066	191	68.50956702434307
67.10822911801972	66.21707862467548	70.74534523689464	66.93520980110418	68.90669225503721	64.35703557626354	63.90116782766808	68.2456964287374	74.77091987770432	66.64273460034372	180	68.49516663790963
70.4601552871578	73.2739270146989	70.74534523689464	77.44529275014062	63.729339494347286	73.95408768537631	70.75594914540534	73.28647094814136	67.1550328511558	69.95004215959678	64	70.80384609194408
68.7841922028676	68.40738088262603	68.92768615126498	66.8947816038553	74.63446493058731	72.07988195639878	66.03932424478754	69.49040480440948	72.23229086885482	74.91100349847636	171	70.1973942578966
70.4601552871578	74.89610905872318	68.92768615126498	72.34975632492453	66.83563393234613	76.79663157936258	78.60699181247512	67.20287976986289	68.42434735558055	81.60715622520333	7	69.87093826863115
73.4151127804303	73.2739270146989	70.74534523689464	79.11793947969996	68.90669225503721	64.35703557626354	73.30458095458212	68.72377756443251	77.30954888655383	71.75527310767882	100	71.77632371622579
72.38909561167152	78.14047314677175	74.38066340815398	68.28884643205927	78.46152939230909	65.90160485229059	70.05844386958567	77.8491643318502	76.04023438212907	79.87196483735595	162	74.35327877008726
70.58459944121304	68.40738088262603	70.74534523689464	68.28884643205927	75.9872631660988	65.90160485229059	68.32051568653704	76.32826653728058	87.46406492195186	68.29638837997024	160	72.20455674901045
80.51593379680396	70.02956292665031	75.12745915510037	78.39690530696161	79.26198386849269	76.71358978447992	77.04478234622907	76.58140559260389	68.42434735558055	74.91100349847636	97	75.27160051095898
72.13611837256381	73.2739270146989	70.61356446570235	72.31951932720723	70.97775057772832	75.16902050845287	70.75594914540534	76.32826653728058	78.57886339097858	68.67187590348101	138	73.88855650672977
73.79674630872377	73.65865038419182	72.5630043225243	76.41066621778978	70.97775057772832	71.03261316998578	72.32815744561127	76.32826653728058	78.57886339097858	71.6036959392233	146	74.11291491873374
76.36157269107528	75.84193586087002	76.19832249378365	79.11793947969996	75.1198672231105	73.21527752653293	75.47257404602314	74.80736874271096	87.46406492195186	68.29638837997024	147	76.63950361246432
72.13611837256381	74.89610905872318	74.32437896920983	79.11793947969996	70.97775057772832	76.71358978447992	72.32815744561127	77.8491643318502	74.59561339626289	75.9898984489643	88	74.85983272152238
78.83997071195593	76.51829110274747	76.19832249378365	81.82521274161013	75.1198672231105	82.67520726678818	84.33439226023198	76.32826653728058	65.88571834673104	87.92175263981585	9	75.81652130817419
75.48804454225987	89.49574745494178	79.29055069275492	79.11793947969996	77.19092554580159	78.25815906050698	75.47257404602314	79.37006212641981	77.48795600612839	80.5022558677462	82	79.19906460795046
85.54382305134806	89.49574745494178	87.10427700756166	86.77523506398653	76.35390401107044	84.8578878911311	84.90582384725873	76.32826653728058	73.50160537327956	74.91100349847636	60	81.68436382430666
80.51593379680396	78.14047314677175	81.65129975067265	74.91923966847418	87.17292751796884	73.62445123242583	81.76140724684687	78.99536313721921	73.50160537327956	91.44754129474164	172	80.0918159773632
81.01595882571694	86.25138336689321	81.65129975067265	83.17884937256522	88.75263345549648	87.52557471666927	77.93547743807291	76.32826653728058	77.30954888655383	83.179272396609	164	82.2034564324634
88.89574922104411	81.05908757903964	87.10427700756166	83.17884937256522	87.54621715925705	81.34729761256108	78.09740998689678	81.97734266292895	82.38680690425284	74.91100349847636	182	83.62431439653119
82.191896881652	78.14047314677175	83.46895883630232	91.30066915829576	83.40410051387488	90.4923819992476	92.87558276171532	85.45365330469828	90.00269393080136	95.78827228040022	4	84.85177796748519
87.21978613619609	82.32831352628381	81.33424276471843	79.23791184207083	87.54621715925705	81.34729761256108	86.47803214746466	83.93275551012866	81.11749239982808	74.91100349847636	152	83.22179778055886
85.54382305134806	84.62920132286891	84.5376103892282	82.66493083245801	87.54621715925705	85.98100544064222	86.47803214746466	88.4954488938375	83.65612140867759	81.35401180797494	133	86.04712134629942
87.21978613619609	86.25138336689321	90.41533775173752	85.8861226344754	76.15791605518356	87.52557471666927	84.90582384725873	88.16141560981598	74.77091987770432	88.14023373548858	14	84.95712061638366
87.21978613619609	87.27304707614735	86.92392063278969	82.90867625791769	91.68833380463924	84.43643616461517	88.0502404476706	93.05814227754637	95.07995194850038	81.52561861698247	150	88.72264419945004
95.59960156043624	92.7401115429036	98.01023152133968	93.41099608304424	89.78335153041772	92.87860444763416	91.19465704808246	94.57904007211596	86.1947504175271	81.52561861698247	68	91.40628725421061

pw1	pw2	pw3	pw4	pw5	pw6	pw7	pw8	pw9	pw10	index	平均值
54.09268438899225	68.96254466681907	60.35197868975693	67.23956878341878	58.16647242765274	69.35919729625513	61.4718147244045 8	65.25569454456252	74.03869328113238	67.36303758072704	96	64.77920633799759
64.81733794231371	72.7786408774398	63.46986087064489 4	85.23101921219381	62.14845827130064	73.70964574921044	66.67263793191462	62.860923784240775	69.88015190274578	53.78048391126745	26	66.08971594008574
61.242453424539896	78.50278519337091	63.46986087064489 4	66.95235717585612	62.14845827130064	67.53454405014276	61.4718147244045 8	56.57680160985776	55.61440824702641	58.91326680818616	41	65.90318195860854
64.81733794231371	61.33035224557759	64.44357655243743	74.65844255442138	64.1394511931246	82.13177001904177	61.4718147244045 8	65.98864162081796	65.72161052435919	76.68030998992458	90	68.041806662678
61.242453424539896	74.68668898275017	58.05515971871037	64.1311825126813	62.14845827130064	52.93731808124375	69.27304953566964	64.98321344974933	69.01489234717863	65.03405103355644	86	64.20033775113353
68.39222246008754	61.33035224557759	63.46986087064489 4	65.54176984426871	62.14845827130064	62.06058431180563	61.4718147244045 8	57.40108880896175	57.34565296487314	63.57117098573445	45	63.48786610401279
79.11687601340901	72.7786408774398	71.68594136160226	69.86356492323173	68.12143703677249	76.65781028070464	69.27304953566964	66.35780154502638	62.9492496054348	58.045765171529496	93	69.56326121727999
64.81733794231371	68.96254466681907	66.78351750852975	59.38754063088696	80.08829600413819	63.885237557918	64.0722263281596	80.64346838381995	60.1768886865104	69.22649206256655	170	65.41774925040244
64.81733794231371	72.7786408774398	63.46986087064489 4	66.95235717585612	62.14845827130064	62.06058431180563	66.67263793191462	56.66558844839631	55.918451783285576	59.979177232684165	40	65.55712534018218
62.65104513084941	61.33035224557759	63.46986087064489 4	71.18411917061835	62.14845827130064	67.53454405014276	61.4718147244045 8	63.922068616995055	70.1447353633123	74.36820797526362	112	64.43731684995484
57.66756890676607	57.7330741082509	66.78351750852975	59.89942051791908	64.1394511931246	58.41127781958088	78.69214167845055	81.86890209705189	82.35577603790557	65.49958309888754	185	64.96522786895908
64.81733794231371	68.96254466681907	63.46986087064489 4	75.41588116538058	62.14845827130064	78.90223665130449	50.68043928962176	59.67748928597795	60.1768886865104	57.040220459974265	1	64.9526372698496
61.242453424539896	48.38500134361202	63.46986087064489 4	52.006101464695305	79.93132392984015	62.06058431180563	61.4718147244045 8	66.04435828250361	76.81105420005677	74.81685550808508	197	66.5595687602915
64.81733794231371	67.0544965615087	64.34274140805182	74.00529383379317	64.1394511931246	67.53454405014276	66.67263793191462	70.28893761352072	64.60197446216817	72.34450623821236	89	67.7875284466169
75.54199149563519	68.45325990162306	76.72448742218437	64.1503747531695	70.85928301344853	69.35919729625513	77.07428434693472	66.04435828250361	72.65251282167019	67.36303758072704	121	72.10855274941575
71.96710697786136	74.68668898275017	64.29921047973711	64.1311825126813	66.13044411494855	62.06058431180563	69.27304953566964	71.35008244627498	66.43952959511488	65.36653929399839	80	68.5141626974274
73.55287048809123	67.0544965615087	70.09717414641463	68.36294450744353	84.71978271913258	62.06058431180563	79.10534568924778	75.59466177729209	67.10779098382139	78.54376447176409	161	69.83163096572143
79.11687601340901	64.27277280847007	73.41083078429949	66.05304166100791	70.80471538384326	73.00850378847989	71.87346113942466	69.22779278076644	72.65251282167019	74.81685550808508	120	73.44383326230498
75.54199149563519	61.6906644128294	89.83519064455582	59.57348886506935	70.11242995859644	67.53454405014276	74.47387274317968	71.35008244627498	68.49397144328358	69.22649206256655	155	70.9619120285256
74.24056180538534	72.7786408774398	73.41083078429949	69.77353183903094	68.12143703677249	65.70989080403038	71.87346113942466	72.41122727902926	72.41643734556375	72.53336612988305	114	70.58271710857528

根据步骤三的排序方式，原排名 41 到 60 的 20 篇论文，7 位评委给分平均，新排名 41-50 的获二等奖，新排名 51-60 的获三等奖；

原排名 1 到 30 的 30 篇论文，7 位评委给分平均，新排名 26-30 名获二等奖，新排名 1-15 名获一等奖。

5.3.4 步骤四

在步骤四中，我们需要对步骤三中排名 16 到 25 的论文，让全部十位评委进行评分，我们生成对应的掩蔽矩阵，如下图所示：

0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	0	0	1	1	0	7
1	1	1	1	1	1	1	0	0	0	47
1	1	1	0	0	1	1	0	1	1	171
1	1	1	1	1	0	0	1	1	0	6
1	1	1	0	0	0	1	1	1	1	64
0	1	1	1	1	1	0	1	1	0	100
0	1	1	1	0	1	0	1	1	1	160
1	1	0	0	1	1	1	1	1	0	138
0	0	1	1	1	0	1	1	1	1	146
0	1	1	1	0	1	0	1	1	1	162

在获取了掩蔽矩阵后，我们根据该掩蔽矩阵对于原始的数据作掩蔽处理，获取在步骤四中可见的数据。步骤四的可见数据如下：

0	1	2	3	4	5	6	7	8	9	index
77	79	77	72	64	64	68	78	62	77	7
76	81	77	71	64	61	67	75	56	78	47
76	75	77	64	64	63	66	77	65	83	171
78	73	77	72	64	56	70	80	68	74	6
77	78	78	70	65	64	69	82	61	80	64
77	78	78	77	65	58	70	79	69	74	100
78	75	78	69	65	59	69	84	77	79	160
78	78	79	77	66	65	69	84	70	79	138
80	76	79	75	66	64	70	84	70	81	146
80	81	80	69	67	59	71	85	68	86	162

由于步骤四中全部十位评委都参与了打分，因此我们不需要使用领域预测填补数据。

在获取到填补结果后，便可以对该结果作 Z-score 规约，然后对于 Z-score 规约后的结果，我们可以直接根据掩蔽矩阵，计算该小组在步骤三中的对应的七个评委的打分的平均值，最终部分结果如下：

0	1	2	3	4	5	6	7	8	9	index	平均值
69.63026824217116	77.37134682152183	66.43136191228267	73.80196190397025	66.43136191228267	79.16614739405809	68.94848592239393	67.4598597583016	67.57620190994786	69.13744572215046	7	48.87023624604781
64.67350858812998	82.71053034842413	66.43136191228267	72.04705714404464	66.43136191228267	72.42598362288244	64.33569250283165	61.416852356481904	60.371247879445065	71.02437633065024	47	48.90554960308781
64.67350858812998	66.69297976771722	66.43136191228267	59.76272382456536	66.43136191228267	76.91942613699953	59.722899083269375	65.44552395769504	71.17867892519925	80.45902937314916	171	48.60778837867472
74.58702789621233	61.3537962408149	66.43136191228267	73.80196190397025	66.43136191228267	61.1923773375897	78.17407276151847	71.48853135951475	74.78115594045066	63.47665389665111	6	48.88751971655283
69.63026824217116	74.70175505807067	73.1	70.29215238411902	73.1	79.16614739405809	73.5612793419562	75.51720296072789	66.37537623819739	74.7982375476498	64	50.76841193887731
69.63026824217116	74.70175505807067	73.1	82.57648570359831	73.1	65.6858198517068	78.17407276151847	69.47419555890818	75.98198161220112	63.47665389665111	100	51.46202377844851
74.58702789621233	66.69297976771722	73.1	68.53724762419341	73.1	67.93254110876534	73.5612793419562	79.54587456194102	85.58858698620485	72.91130693915002	160	51.43085369879718
74.58702789621233	74.70175505807067	79.76863808771732	82.57648570359831	79.76863808771732	81.41286865111663	73.5612793419562	79.54587456194102	77.18280728395159	72.91130693915002	138	54.07602508809657
84.5005472042947	69.36257153116837	79.76863808771732	79.06667618374708	79.76863808771732	79.16614739405809	78.17407276151847	79.54587456194102	77.18280728395159	76.6851681561496	146	55.01918751227424
84.5005472042947	82.71053034842413	86.43727617543463	68.53724762419341	86.43727617543463	67.93254110876534	82.78686618108074	81.5602103625476	74.78115594045066	86.11982119864851	162	54.80787827584643

根据步骤四的排序方式，最终排名 16-20 的获一等奖，最终排名 21-25 的获二等奖。

5.4 获奖结果

第一步淘汰：

第 1 大组：

第 9 组：,得分：41.62757157814061

第 3 组：,得分：46.46755215601858

第 6 组：,得分：46.83734520496438

第 1 组：,得分：54.07338680560584

第 2 大组：

第 13 组：,得分：50.166317514202255

第 18 组：,得分：52.63799470224672

第 16 组：,得分：53.03544496334766

第 12 组：,得分：55.83207990035547

第 3 大组：

第 23 组：,得分：49.610890832344445

第 26 组：,得分：55.561582989695786

第 29 组：,得分：55.95322035753741

第 30 组：,得分：57.14197457737627

第 4 大组：

第 39 组：,得分：33.67957730702623

第 37 组：,得分：45.13321526030566
第 40 组：,得分：49.48776975396152
第 32 组：,得分：50.24582331941354
第 5 大组:
第 47 组：,得分：40.29612954344943
第 43 组：,得分：42.285224689203794
第 44 组：,得分：44.67736345794551
第 50 组：,得分：48.653614568461656
第 6 大组:
第 53 组：,得分：46.364672417773846
第 57 组：,得分：47.14357980980223
第 56 组：,得分：51.82391835484545
第 51 组：,得分：55.33346256730272
第 7 大组:
第 67 组：,得分：37.76898353529103
第 63 组：,得分：39.99722560293798
第 66 组：,得分：49.869159946684135
第 64 组：,得分：50.87550613831246
第 8 大组:
第 78 组：,得分：31.398463560754838
第 80 组：,得分：37.812845022042765
第 72 组：,得分：52.13017322194364
第 74 组：,得分：54.12969130695421
第 9 大组:
第 85 组：,得分：49.16192404340551
第 82 组：,得分：58.454126279952
第 86 组：,得分：60.79599442502964
第 84 组：,得分：61.2755719247744
第 10 大组:
第 92 组：,得分：49.91656177138068
第 99 组：,得分：55.06172563578984
第 93 组：,得分：59.00600433741463
第 96 组：,得分：61.44130056345612
第 11 大组:
第 109 组：,得分：52.80426415118512
第 108 组：,得分：53.97259856423148
第 106 组：,得分：57.16016236288374
第 105 组：,得分：57.53695546001668
第 12 大组:
第 116 组：,得分：44.69576453607479
第 120 组：,得分：52.2649122525901
第 117 组：,得分：52.66831398135178
第 111 组：,得分：55.05857075621725
第 13 大组:

第 126 组: ,得分: 40.770968172157474
第 125 组: ,得分: 56.13166982926713
第 127 组: ,得分: 58.5311522842757
第 130 组: ,得分: 58.92648663635611
第 14 大组:
第 131 组: ,得分: 37.51539544146138
第 140 组: ,得分: 53.919174032454855
第 138 组: ,得分: 57.85516380431371
第 135 组: ,得分: 58.60143347780309
第 15 大组:
第 145 组: ,得分: 49.427299740465145
第 146 组: ,得分: 50.60915253572554
第 142 组: ,得分: 51.36501083237072
第 150 组: ,得分: 54.51095222400619
第 16 大组:
第 155 组: ,得分: 36.03082398155854
第 154 组: ,得分: 48.88360395567121
第 158 组: ,得分: 49.67875744907388
第 152 组: ,得分: 52.81242375576321
第 17 大组:
第 167 组: ,得分: 34.049089147169944
第 169 组: ,得分: 39.07565947428069
第 170 组: ,得分: 43.35060102286068
第 164 组: ,得分: 46.02544394807003
第 18 大组:
第 179 组: ,得分: 51.49537313019601
第 174 组: ,得分: 52.69923058941942
第 175 组: ,得分: 64.48418525317908
第 180 组: ,得分: 66.08411212470793
第 19 大组:
第 182 组: ,得分: 30.614532915736746
第 184 组: ,得分: 51.66374619582359
第 188 组: ,得分: 53.12793938598188
第 190 组: ,得分: 57.85231737314734
第 20 大组:
第 195 组: ,得分: 46.58969111991704
第 194 组: ,得分: 46.96406395362399
第 200 组: ,得分: 54.29352742955749
第 197 组: ,得分: 54.67881400571556
第二步淘汰:
第 38 组: ,得分: 43.61374744112697
第 31 组: ,得分: 48.9388142945343
第 33 组: ,得分: 50.314913666880265
第 193 组: ,得分: 51.88974551725745

第 4 组：,得分：52.436379506971015
第 143 组：,得分：52.780130209241136
第 77 组：,得分：54.5519982632609
第 60 组：,得分：54.74446826697863
第 11 组：,得分：54.85126333626623
第 73 组：,得分：55.46705543057279
第 22 组：,得分：55.65959407167636
第 45 组：,得分：55.93246683912294
第 191 组：,得分：55.97418604632105
第 52 组：,得分：56.315461938448365
第 144 组：,得分：56.64885200722707
第 160 组：,得分：57.19066023675593
第 20 组：,得分：57.48812706771855
第 112 组：,得分：57.6032730225793
第 36 组：,得分：58.10635609046301
第 185 组：,得分：58.27281577838613
第二步获二等奖：
第 118 组：,得分：58.43681551454781
第 59 组：,得分：58.57581036283182
第 141 组：,得分：58.79474274121752
第 107 组：,得分：58.8251027381753
第 55 组：,得分：58.90850055225599
第 133 组：,得分：58.93877967501074
第 34 组：,得分：59.39388076248555
第 114 组：,得分：59.75620607445977
第 110 组：,得分：59.85040154458056
第 58 组：,得分：59.95232699304991
第 71 组：,得分：60.08399822141032
第 79 组：,得分：60.31727749646528
第 28 组：,得分：60.660725250120585
第 137 组：,得分：60.794596452037936
第 157 组：,得分：60.860741399545056
第 159 组：,得分：60.88088441247364
第 136 组：,得分：60.91702143616068
第 187 组：,得分：61.66185654285745
第 95 组：,得分：61.72453855337094
第 19 组：,得分：62.05674314807585
第 124 组：,得分：62.62173690785424
第 123 组：,得分：62.74006818467808
第 24 组：,得分：62.74961110290391
第 103 组：,得分：62.83057132176706
第 14 组：,得分：63.19097700577449
第 104 组：,得分：63.33278086941671
第 119 组：,得分：63.340011894208416

第 88 组：,得分：63.37173343873786
第 75 组：,得分：63.56895988357068
第 176 组：,得分：63.68087604799259
第 54 组：,得分：64.10933810731505
第 68 组：,得分：64.32443155074687
第 70 组：,得分：64.44763227815702
第 62 组：,得分：64.60312572938324
第 17 组：,得分：64.75003620184192
第 128 组：,得分：64.85158128589937
第 189 组：,得分：65.21395784118504
第 21 组：,得分：65.54933437573368
第 49 组：,得分：65.63790126559874
第 35 组：,得分：65.69244337641274
第二步获三等奖:
第 76 组：,得分：70.88876358460627
第 168 组：,得分：70.94210913179819
第 100 组：,得分：71.34028658163267
第 178 组：,得分：71.41491927586644
第 25 组：,得分：71.61226703961249
第 132 组：,得分：71.74036117133691
第 129 组：,得分：71.76405730756399
第 196 组：,得分：71.93407546343917
第 166 组：,得分：71.94227599338558
第 102 组：,得分：72.45888971091973
第三步的 1-30 名中，获二等奖为：
第 177 组：,得分：65.94837876883652
第 149 组：,得分：68.12771584230995
第 199 组：,得分：68.2765532178672
第 181 组：,得分：68.49516663790963
第 192 组：,得分：68.50956702434307
第三步的 1-30 名中，获一等奖为：
第 89 组：,得分：74.85983272152238
第 98 组：,得分：75.27160051095898
第 10 组：,得分：75.81652130817419
第 148 组：,得分：76.63950361246432
第 83 组：,得分：79.19906460795046
第 173 组：,得分：80.0918159773632
第 61 组：,得分：81.68436382430666
第 165 组：,得分：82.2034564324634
第 153 组：,得分：83.22179778055886
第 183 组：,得分：83.62431439653119
第 5 组：,得分：84.85177796748519
第 15 组：,得分：84.95712061638366
第 134 组：,得分：86.04712134629942

第 151 组：,得分：88.72264419945004
第 69 组：,得分：91.4062872542106
第四步中，获二等奖为:
第 172 组：,得分：48.60778837867472
第 8 组：,得分：48.87023624604781
第 7 组：,得分：48.88751971655283
第 48 组：,得分：48.90554960308781
第 65 组：,得分：50.76841193887731
第四步中，获一等奖为:
第 161 组：,得分：51.43085369879718
第 101 组：,得分：51.46202377844851
第 139 组：,得分：54.07602508809657
第 163 组：,得分：54.80787827584643
第 147 组：,得分：55.01918751227424

六、 问题四的分析与解决

6.1 合理性评估

经过我们解第二题和第三题的经验，分组越少，每个评委评的论文越多，结果越合理。而题目的评审过程的步骤三和步骤四中，都有超过一半的评委对每个小组的论文打分，同时对于每个打分区间都可以看作是在同一个组内打分，因此步骤三和步骤四设置得较为合理。而步骤一和步骤二由于分组较多，评委评判文章较少，因此其合理性尚待验证。为此，我们采用排名差值的方法来评估步骤一和步骤二的合理性。

6.1.1 排名差值

由于我们在问题三中的调分算法能很好地确保结果的合理性，因此在该模型中，我们以我们在问题三中提出的调分算法为基线，评估题目中原本的评分步骤。对于相同的排名，原步骤的小组排名应与基线的小组排名相差越少越好，同时在基线中出现的小组要尽可能在原步骤的小组排名中出现。因此我们采用排名差值的方法来量化合理性，具体公式如下

6.1.2 求解步骤

步骤一合理性验证
经过检验，按照原题的方式完成步骤一后，淘汰的小组排名与我们的方法淘汰的小组排名完全一致
步骤二合理性验证
步骤二中，首先需要淘汰 20 个小组，原题方法淘汰的小组及其对应打分如下：

小组编号
38
193
31
200
33

小组编号
60
11
52
191
77
73
160
4
45
59
143
20
55
22
107

将该排名与我们的调分算法调整后的排名进行对比，得到差值情况如下：

小组编号	排名之差
38	0
193	2
31	-1
200	未出现
33	-2
60	2
11	2
52	6
191	4
77	-3
73	-1
160	4
4	-8
45	-2
59	未出现
143	-10
20	0
55	未出现

小组编号	排名之差
22	-8
107	未出现

使用评估公式计算得分为-33.98，显然步骤二是不合理的。

6.1.3 工作量评估

经过计算，原题打分标准下，平均每个评委至少需要给 94 篇论文打分，相较于原本每个评委给 200 篇论文打分的方法减少了一半，说明评委的工作量被原题的打分流程显著降低了。

6.2 改进的评分步骤

步骤（1）

打分：每个组的 10 篇论文由 3 位评委评审，分别用百分制给出评分。评委的分配方式采取问题一中第三小问所得的最优分配方案。

排序：每个组的 10 篇论文根据 3 位评委的给分进行平均，总的 200 篇论文排名后，淘汰后 80 篇。

步骤（2）

打分：未淘汰的 120 篇论文，再由没有评审过的 2 位评委进行评审（评审时采取圆桌模式），给出百分制得分。

排序：对 120 篇论文根据 5 个评委的平均给分进行总体排序，淘汰排名靠后的 30 篇文章，剩下的 90 篇论文为获奖论文。

90 篇获奖论文中，排名 31 到 40 的论文，获二等奖；排名 61 到 90 的论文获三等奖；

步骤（3）

打分：排名 1 到 30 与排名 41 到 60 的 50 篇论文，再由 2 位评委进行打分（采取评委随机选论文的模式）

排序：原排名 41 到 60 的 20 篇论文，7 位评委给分平均，新排名 41-50 的获二等奖，新排名 51-60 的获三等奖；

原排名 1 到 30 的 30 篇论文，7 位评委给分平均，新排名 26-30 名获二等奖，新排名 1-15 名获一等奖。

步骤（4）

打分：7 人平均分排名在 16 到 25 的论文，再由剩下的 3 位评委打分。

排序：最终排名 16-20 的获一等奖，最终排名 21-25 的获二等奖。

附录一、问题二调分结果

pw1	pw2	pw3	pw4	pw5	pw
54.9131934	54.27055536	53.03641165	53.71046002	54.77290393	
69.26129674	66.56206658	66.08244527	66.81577947	67.85834917	
46.54346646	48.12479975	44.73439026	46.16400314	47.19496393	
57.30454396	49.35395087	56.59442082	54.09805718	55.41503813	
84.80507535	77.62442667	82.68648806	81.11923384	82.24506122	
47.73914174	45.66649751	47.10639637	46.55924739	47.72373923	
77.63102369	67.7912177	73.19846361	72.35934782	73.59257222	
76.43534841	75.16612443	73.19846361	74.36988347	75.32293026	
42.95644062	40.74989302	41.17638109	41.3869529	42.56998554	
82.4137248	76.39527555	77.94247583	78.32874884	79.40213495	

60.89156979	59.18715985	59.15412831	55.92446488	60.86103118
56.10886868	54.27055536	56.31054568	57.11681566	56.12269388
52.52184285	44.43734639	50.75437463	53.53976331	49.81930983
64.47859563	65.33291546	65.41183182	65.46327114	65.51463659
88.39210119	83.77018228	86.75364515	88.11793603	86.45669202
53.71751813	53.04140424	53.5780001	52.34741253	54.30345144
68.06562146	61.64546209	68.47247633	75.00207741	65.76883227
53.71751813	53.04140424	53.19791418	51.15506174	54.30345144
66.86994618	59.18715985	62.32580987	59.50151723	64.03432173
63.28292035	56.7288576	59.16743308	55.92446488	61.03049497
65.67427091	65.45089288	64.89644222	65.02371669	64.62337395
58.50021924	58.07598615	57.78042388	57.86961199	57.515868
50.13049229	49.4719283	49.47840248	49.52315651	49.22377772
66.86994618	65.06943463	63.71043916	65.00350502	63.43878962
72.84832257	72.02049493	70.8264575	71.57154016	70.54629558
58.50021924	55.6600721	54.22241471	56.05076832	53.96211502
69.26129674	67.52773688	66.08244527	67.38820659	65.80795827
63.28292035	62.99259064	62.5244361	62.63901512	62.2542053
57.30454396	56.04153035	55.40841777	56.07097998	55.14669935
58.50021924	57.27068147	56.59442082	57.26333076	56.33128367
54.9131934	53.62984958	52.83189595	51.15506174	53.64914533
50.13049229	49.91435297	50.77184512	48.77036018	51.85673322
56.10886868	54.25844677	55.15367202	51.15506174	57.1904698
64.47859563	66.46582826	66.1708773	66.65562193	66.07174513
66.86994618	68.32357656	70.44104219	67.84797271	71.976024
60.89156979	62.7783749	63.37525389	63.07856958	63.69636223
51.32616757	46.33907259	45.79631714	40.42390469	48.32783723
50.13049229	55.91989237	51.2310942	60.69386801	46.55406787
39.36941479	35.24866924	34.33309069	30.88509843	35.89280895
48.93481701	51.68797156	50.00606305	53.53976331	48.31540874
69.26129674	67.73745615	67.29319516	67.66908969	65.80795827
68.06562146	66.50065451	66.11109046	65.88055266	65.80795827
45.34779118	41.96681512	42.38922047	42.62970153	40.93168744
46.54346646	44.38358484	44.75913709	45.0144248	43.30085609
59.69589452	57.90424718	57.80517071	58.13028342	56.33128367
70.45697202	67.38056565	66.90194625	67.07288173	65.80795827
40.56509007	39.81622034	40.41027215	40.2450434	39.74710312
75.23967313	73.87556124	73.2271088	73.03465736	72.91546423
66.86994618	66.06834946	65.71176422	65.88057437	64.62337395
51.32616757	48.50334325	48.71647252	48.59145544	48.0391934
54.9131934	55.58164989	55.60377515	54.38392184	55.99769083
56.10886868	56.81080102	57.8797025	59.04882159	58.35554361
48.93481701	49.43589428	46.56682297	46.68589344	46.92274412
66.86994618	67.87316111	67.90421846	64.57115519	68.24358866

63.28292035	64.18570775	60.8759643	58.10031204	61.14672111
52.52184285	53.12334765	52.06474008	51.42046212	52.44628865
46.54346646	46.97759204	47.19380723	50.08877349	47.69304944
62.08724507	62.95655662	61.5373436	61.53802654	61.92117023
64.47859563	65.41485887	62.31732893	64.50148627	62.7098438
57.30454396	58.03995214	59.90026826	58.50489789	60.33485802
84.63442265	86.22848453	85.05849417	81.52012601	82.8332319
65.5036182	66.56206658	66.08244527	63.2178271	64.54039832
38.03683005	35.83328853	41.17638109	39.28846753	40.39632257
52.31744691	55.49970648	50.66440554	50.09617137	51.38092734
73.28394332	73.93697331	74.38446667	71.9386985	73.09810183
45.1940101	44.43734639	47.10639637	49.15327137	49.7511115
40.39443736	40.74989302	41.17638109	37.01549021	38.67335406
65.48674659	67.7912177	64.89644222	65.92852624	66.7876813
89.45086701	88.68678677	92.17451251	88.62126714	89.68215949
63.70166947	65.33291546	63.71043916	64.33527776	65.20819457
58.18518135	55.49970648	60.48820465	61.88621879	59.66419792
51.14081492	51.81225312	52.26757387	51.15506174	52.87349218
59.46731346	59.18715985	56.66755684	60.69386801	55.12556898
54.0651605	53.04140424	54.23207225	55.92446488	53.51858437
63.01111087	61.64546209	64.86401273	65.46327114	64.50813529
72.61974152	72.70782219	71.65460981	73.80972663	70.85345719
60.49007509	55.49970648	58.54996876	66.65562193	54.80876877
30.64142155	26.00007956	31.63731916	35.65450156	29.61569773
60.0867653	60.41631097	63.28771821	60.69386801	64.40253522
37.90193003	35.83328853	38.0169292	40.42390469	36.93440381
67.79582704	70.24951994	68.44715961	68.23932699	68.17712693
57.42708206	60.41631097	58.16422041	58.11229333	57.515868
79.72505748	86.22848453	80.29004281	80.79875022	78.83838585
60.21076514	64.10376433	60.92731481	61.10111829	59.88503665
48.30021241	45.66649751	49.09716976	47.92167568	50.40836205
59.87446574	55.49970648	60.57444099	58.63693673	63.43878962
66.99149584	70.24951994	67.6483251	68.23932699	65.80795827
61.38875116	67.7912177	62.10449947	62.32526107	61.06962097
74.18521367	75.16612443	74.78100134	74.1851849	75.28463288
65.42315419	65.33291546	66.08789165	65.23460603	66.9925426
63.40428888	61.64546209	64.70312296	63.13989191	66.9925426
49.01179085	50.58310199	49.68752044	49.47791084	49.22377772
56.79107927	57.95800873	58.72739379	58.97667476	57.515868
68.15739352	69.02036882	67.15595495	65.60467817	69.36171125
65.13120976	69.02036882	64.73823496	65.01851337	63.43878962
59.74327189	64.10376433	61.10260144	62.59376945	57.515868
63.95033271	66.56206658	67.35102652	69.08162465	63.43878962
74.78540136	71.47867107	73.89824501	70.33403896	80.02297018

54.41452694	54.27055536	54.8477733	53.641128	56.33128367
69.9287091	72.70782219	67.21657437	65.66474169	69.36171125
75.79143418	79.14055117	74.38446667	82.15618211	73.65638105
73.34633607	73.13574523	73.19846361	71.42502506	75.40582817
63.36542086	63.26729884	62.5244361	63.07856958	64.74456925
64.96701316	65.69036366	63.71043916	66.65562193	65.34234728
57.04447096	58.31545693	56.59442082	59.50151723	57.06122875
56.74316517	59.54460806	57.78042388	60.69386801	55.89858793
59.3945538	60.73852176	57.78042388	63.07856958	58.83261937
53.28613432	52.73142076	49.47840248	55.92446488	53.47456054
52.30303757	53.43408986	53.03641165	53.53976331	52.92066949
61.80174022	63.79378085	60.15242999	66.65562193	60.61498173
54.32354296	55.02742401	53.03641165	57.11681566	54.28557092
58.62209962	59.59916247	57.78042388	64.27092036	57.25191246
67.3648866	68.61199523	66.08244527	73.80972663	65.54400274
61.9495887	62.85936433	62.5244361	63.07856958	62.58091502
69.86976045	71.05182089	69.64045444	72.61737584	69.68842097
43.8639081	44.46187705	43.5483872	49.96271096	42.44623529
51.97165942	52.59325109	51.8504086	53.53976331	52.51032134
60.60092311	61.62456046	58.96642693	66.65562193	59.02716204
63.10579696	64.06438611	62.5244361	65.46327114	63.17158027
51.60103825	52.22990848	53.03641165	52.34741253	52.51357516
69.56012782	69.72649866	69.64045444	68.73818828	70.63849173
69.67133008	71.53619026	70.8264575	70.51661948	70.65067901
62.4585861	63.51267287	62.5244361	62.75624441	63.53765909
58.46172929	59.24467904	58.96642693	58.59311164	59.58865822
55.09838488	53.09892343	53.03641165	52.63135773	56.40911522
39.59329859	37.7685695	38.80437498	37.73706791	41.01503296
57.60337504	57.43498744	57.78042388	56.81468045	58.79264847
65.67447329	67.26819642	67.26844833	66.35348671	66.70167814
69.67376743	71.60426044	72.01246055	70.53680943	70.65285633
58.01564336	58.0835981	58.96642693	57.42095082	59.19007367
35.85997569	36.49114591	37.02023149	36.50791932	37.40382184
72.81524098	72.13652844	72.98445855	71.08609204	73.48767978
59.71814708	57.36938919	58.80251256	56.78613484	59.51460467
87.16334431	86.26743489	86.83228368	84.8001891	87.38724779
53.79510486	60.49857582	58.26818132	59.74019216	58.01158184
62.56899986	59.84501722	62.30203181	59.16258421	62.94002091
62.84567069	61.65842376	61.61722016	60.96555173	62.32755394
56.18645542	58.00562198	57.56649501	57.37199499	57.85719927
76.34693263	77.68936573	78.06259164	76.44135532	78.2995242
50.77824958	54.33549441	53.56586054	53.78669044	53.62878138
59.21468993	60.76457045	59.91534002	62.68111932	61.06962097
49.5924856	49.70221035	50.39673502	50.75761149	50.40836205

53.23631354	54.61881484	53.98532474	57.91171619	55.14669935
53.80575013	54.61881484	54.563036	66.25817167	55.14669935
46.68850057	48.47305923	47.4775982	50.75761149	49.22377772
51.92703405	50.93136147	52.73816057	43.60350679	51.59294637
76.97941277	75.51438391	77.6136442	75.79697794	75.28463288
77.66245358	77.97268615	78.25251657	73.41227637	77.65380153
73.44918905	73.05608167	74.08621559	71.0275748	72.91546423
57.33597385	57.07711708	58.09046461	44.79585757	57.515868
86.63080855	87.80589513	87.13569408	84.97730941	87.13047613
54.34757604	54.61881484	55.11361156	50.35238909	55.14669935
84.23945799	85.34759289	84.76368797	81.37688327	84.76130748
47.81270725	48.47305923	48.61770299	47.45332043	49.22377772
35.29946206	36.18154801	36.19177914	33.72959953	37.37793447
69.89135466	70.59777942	70.53165129	67.67653616	70.54629558
63.74759734	61.99372157	64.49320304	61.06017235	62.2542053
50.12136734	49.70221035	50.93549262	47.99106133	50.40836205
64.22139929	60.76457045	65.00487984	61.62128705	61.06962097
57.93460188	58.30626821	58.67162073	59.40020206	58.70045232
76.39413483	58.30626821	76.96622567	73.10054917	76.3467112
72.85905638	58.30626821	73.45554415	68.96238212	73.17415581
77.00495932	58.30626821	77.57105911	77.3555852	79.39381998
43.47410416	58.30626821	44.31564588	44.53087796	45.84426128
84.58204597	58.30626821	85.1026003	86.75414869	85.38098505
67.30968866	58.30626821	67.96471555	69.52349682	69.15291277
32.68705297	58.30626821	33.61795456	31.95276025	33.67107391
70.31186369	58.30626821	70.9415551	68.26102465	69.66139438
38.08057857	58.30626821	38.96680022	36.72216338	38.81386071
43.9810339	58.30626821	44.82582404	42.00593361	43.22117905
66.21294987	65.66068909	66.91692222	65.92615744	66.94698583
73.22454914	70.57729358	73.93618223	75.03384867	75.92610341
80.16628105	82.8688048	80.67690527	83.49593531	84.15894045
51.2263958	49.68172451	52.06769444	51.61016495	52.70169181
64.9360484	63.20238685	65.68254	63.51069524	64.62785037
62.62592403	61.97323573	63.35891305	65.54896644	66.40466113
72.82977699	73.03559582	73.4641286	73.49562512	74.40590371
70.75197239	66.88984021	71.51579696	72.6259837	73.55166924
49.31104358	45.99427114	50.2161211	50.38686766	51.48155984
66.93262681	68.11899134	67.58249249	65.1571386	66.22967641
71.35372848	74.88228572	74.33293469	71.42502506	74.87520975
33.09211958	33.13960087	34.01095482	33.26979999	27.57420453
85.70183181	86.01732577	85.01014824	85.73323446	81.31757719
55.80994986	55.28854772	55.36007183	55.92446488	48.37727656
64.17967681	62.6876811	62.47715219	64.27092036	56.29561368
64.17967681	72.92953912	72.54915105	64.27092036	71.11243921

67.76670264	66.37513446		66.03516136	67.84797271	59.84936666
49.83157347	49.14279211		49.43005655	49.96271096	55.09198644
70.1580532	69.43589894		68.99963799	70.23267428	67.39395064
59.3969757	55.9636899		55.9557284	59.50151723	57.31546241
59.59625491	57.7121785		59.02971702	56.31682625	58.55251392
70.35733241	77.37859645		73.81027598	73.02446339	72.73009017
46.44382686	56.48302738		51.8501569	58.0869447	54.99515575
42.85680102	44.19151616		44.18561628	46.75593071	46.67915294
40.46545047	51.56642289		46.50678897	47.3926031	45.65877729
67.96598186	82.29520094		74.95815397	72.46142192	71.06103314
48.83517741	54.02472514		51.87557361	55.70224313	54.4057183
67.96598186	76.14944533		72.02491721	71.83211261	71.10057304
69.16165714	77.37859645		73.21092027	74.20945109	73.11577903
61.98760547	46.6498184		54.94860227	48.54813843	54.86206998

pw1	pw2	pw3	pw4	pw5	pw
54.9131934	54.27055536	53.03641165	53.71046002	54.77290393	
69.26129674	66.56206658	66.08244527	66.81577947	67.85834917	
46.54346646	48.12479975	44.73439026	46.16400314	47.19496393	
57.30454396	49.35395087	56.59442082	54.09805718	55.41503813	
84.80507535	77.62442667	82.68648806	81.11923384	82.24506122	
47.73914174	45.66649751	47.10639637	46.55924739	47.72373923	
77.63102369	67.7912177	73.19846361	72.35934782	73.59257222	
76.43534841	75.16612443	73.19846361	74.36988347	75.32293026	
42.95644062	40.74989302	41.17638109	41.3869529	42.56998554	
82.4137248	76.39527555	77.94247583	78.32874884	79.40213495	
60.89156979	59.18715985	59.15412831	55.92446488	60.86103118	
56.10886868	54.27055536	56.31054568	57.11681566	56.12269388	
52.52184285	44.43734639	50.75437463	53.53976331	49.81930983	
64.47859563	65.33291546	65.41183182	65.46327114	65.51463659	
88.39210119	83.77018228	86.75364515	88.11793603	86.45669202	
53.71751813	53.04140424	53.5780001	52.34741253	54.30345144	
68.06562146	61.64546209	68.47247633	75.00207741	65.76883227	
53.71751813	53.04140424	53.19791418	51.15506174	54.30345144	
66.86994618	59.18715985	62.32580987	59.50151723	64.03432173	
63.28292035	56.7288576	59.16743308	55.92446488	61.03049497	
65.67427091	65.45089288	64.89644222	65.02371669	64.62337395	
58.50021924	58.07598615	57.78042388	57.86961199	57.515868	
50.13049229	49.4719283	49.47840248	49.52315651	49.22377772	
66.86994618	65.06943463	63.71043916	65.00350502	63.43878962	
72.84832257	72.02049493	70.8264575	71.57154016	70.54629558	
58.50021924	55.6600721	54.22241471	56.05076832	53.96211502	
69.26129674	67.52773688	66.08244527	67.38820659	65.80795827	
63.28292035	62.99259064	62.5244361	62.63901512	62.2542053	
57.30454396	56.04153035	55.40841777	56.07097998	55.14669935	

58.50021924	57.27068147	56.59442082	57.26333076	56.33128367
54.9131934	53.62984958	52.83189595	51.15506174	53.64914533
50.13049229	49.91435297	50.77184512	48.77036018	51.85673322
56.10886868	54.25844677	55.15367202	51.15506174	57.1904698
64.47859563	66.46582826	66.1708773	66.65562193	66.07174513
66.86994618	68.32357656	70.44104219	67.84797271	71.976024
60.89156979	62.7783749	63.37525389	63.07856958	63.69636223
51.32616757	46.33907259	45.79631714	40.42390469	48.32783723
50.13049229	55.91989237	51.2310942	60.69386801	46.55406787
39.36941479	35.24866924	34.33309069	30.88509843	35.89280895
48.93481701	51.68797156	50.00606305	53.53976331	48.31540874
69.26129674	67.73745615	67.29319516	67.66908969	65.80795827
68.06562146	66.50065451	66.11109046	65.88055266	65.80795827
45.34779118	41.96681512	42.38922047	42.62970153	40.93168744
46.54346646	44.38358484	44.75913709	45.0144248	43.30085609
59.69589452	57.90424718	57.80517071	58.13028342	56.33128367
70.45697202	67.38056565	66.90194625	67.07288173	65.80795827
40.56509007	39.81622034	40.41027215	40.2450434	39.74710312
75.23967313	73.87556124	73.2271088	73.03465736	72.91546423
66.86994618	66.06834946	65.71176422	65.88057437	64.62337395
51.32616757	48.50334325	48.71647252	48.59145544	48.0391934
54.9131934	55.58164989	55.60377515	54.38392184	55.99769083
56.10886868	56.81080102	57.8797025	59.04882159	58.35554361
48.93481701	49.43589428	46.56682297	46.68589344	46.92274412
66.86994618	67.87316111	67.90421846	64.57115519	68.24358866
63.28292035	64.18570775	60.8759643	58.10031204	61.14672111
52.52184285	53.12334765	52.06474008	51.42046212	52.44628865
46.54346646	46.97759204	47.19380723	50.08877349	47.69304944
62.08724507	62.95655662	61.5373436	61.53802654	61.92117023
64.47859563	65.41485887	62.31732893	64.50148627	62.7098438
57.30454396	58.03995214	59.90026826	58.50489789	60.33485802
84.63442265	86.22848453	85.05849417	81.52012601	82.8332319
65.5036182	66.56206658	66.08244527	63.2178271	64.54039832
38.03683005	35.83328853	41.17638109	39.28846753	40.39632257
52.31744691	55.49970648	50.66440554	50.09617137	51.38092734
73.28394332	73.93697331	74.38446667	71.9386985	73.09810183
45.1940101	44.43734639	47.10639637	49.15327137	49.7511115
40.39443736	40.74989302	41.17638109	37.01549021	38.67335406
65.48674659	67.7912177	64.89644222	65.92852624	66.7876813
89.45086701	88.68678677	92.17451251	88.62126714	89.68215949
63.70166947	65.33291546	63.71043916	64.33527776	65.20819457
58.18518135	55.49970648	60.48820465	61.88621879	59.66419792
51.14081492	51.81225312	52.26757387	51.15506174	52.87349218
59.46731346	59.18715985	56.66755684	60.69386801	55.12556898

54.0651605	53.04140424	54.23207225	55.92446488	53.51858437
63.01111087	61.64546209	64.86401273	65.46327114	64.50813529
72.61974152	72.70782219	71.65460981	73.80972663	70.85345719
60.49007509	55.49970648	58.54996876	66.65562193	54.80876877
30.64142155	26.00007956	31.63731916	35.65450156	29.61569773
60.0867653	60.41631097	63.28771821	60.69386801	64.40253522
37.90193003	35.83328853	38.0169292	40.42390469	36.93440381
67.79582704	70.24951994	68.44715961	68.23932699	68.17712693
57.42708206	60.41631097	58.16422041	58.11229333	57.515868
79.72505748	86.22848453	80.29004281	80.79875022	78.83838585
60.21076514	64.10376433	60.92731481	61.10111829	59.88503665
48.30021241	45.66649751	49.09716976	47.92167568	50.40836205
59.87446574	55.49970648	60.57444099	58.63693673	63.43878962
66.99149584	70.24951994	67.6483251	68.23932699	65.80795827
61.38875116	67.7912177	62.10449947	62.32526107	61.06962097
74.18521367	75.16612443	74.78100134	74.1851849	75.28463288
65.42315419	65.33291546	66.08789165	65.23460603	66.9925426
63.40428888	61.64546209	64.70312296	63.13989191	66.9925426
49.01179085	50.58310199	49.68752044	49.47791084	49.22377772
56.79107927	57.95800873	58.72739379	58.97667476	57.515868
68.15739352	69.02036882	67.15595495	65.60467817	69.36171125
65.13120976	69.02036882	64.73823496	65.01851337	63.43878962
59.74327189	64.10376433	61.10260144	62.59376945	57.515868
63.95033271	66.56206658	67.35102652	69.08162465	63.43878962
74.78540136	71.47867107	73.89824501	70.33403896	80.02297018
54.41452694	54.27055536	54.8477733	53.641128	56.33128367
69.9287091	72.70782219	67.21657437	65.66474169	69.36171125
75.79143418	79.14055117	74.38446667	82.15618211	73.65638105
73.34633607	73.13574523	73.19846361	71.42502506	75.40582817
63.36542086	63.26729884	62.5244361	63.07856958	64.74456925
64.96701316	65.69036366	63.71043916	66.65562193	65.34234728
57.04447096	58.31545693	56.59442082	59.50151723	57.06122875
56.74316517	59.54460806	57.78042388	60.69386801	55.89858793
59.3945538	60.73852176	57.78042388	63.07856958	58.83261937
53.28613432	52.73142076	49.47840248	55.92446488	53.47456054
52.30303757	53.43408986	53.03641165	53.53976331	52.92066949
61.80174022	63.79378085	60.15242999	66.65562193	60.61498173
54.32354296	55.02742401	53.03641165	57.11681566	54.28557092
58.62209962	59.59916247	57.78042388	64.27092036	57.25191246
67.3648866	68.61199523	66.08244527	73.80972663	65.54400274
61.9495887	62.85936433	62.5244361	63.07856958	62.58091502
69.86976045	71.05182089	69.64045444	72.61737584	69.68842097
43.8639081	44.46187705	43.5483872	49.96271096	42.44623529
51.97165942	52.59325109	51.8504086	53.53976331	52.51032134

60.60092311	61.62456046	58.96642693	66.65562193	59.02716204
63.10579696	64.06438611	62.5244361	65.46327114	63.17158027
51.60103825	52.22990848	53.03641165	52.34741253	52.51357516
69.56012782	69.72649866	69.64045444	68.73818828	70.63849173
69.67133008	71.53619026	70.8264575	70.51661948	70.65067901
62.4585861	63.51267287	62.5244361	62.75624441	63.53765909
58.46172929	59.24467904	58.96642693	58.59311164	59.58865822
55.09838488	53.09892343	53.03641165	52.63135773	56.40911522
39.59329859	37.7685695	38.80437498	37.73706791	41.01503296
57.60337504	57.43498744	57.78042388	56.81468045	58.79264847
65.67447329	67.26819642	67.26844833	66.35348671	66.70167814
69.67376743	71.60426044	72.01246055	70.53680943	70.65285633
58.01564336	58.0835981	58.96642693	57.42095082	59.19007367
35.85997569	36.49114591	37.02023149	36.50791932	37.40382184
72.81524098	72.13652844	72.98445855	71.08609204	73.48767978
59.71814708	57.36938919	58.80251256	56.78613484	59.51460467
87.16334431	86.26743489	86.83228368	84.8001891	87.38724779
53.79510486	60.49857582	58.26818132	59.74019216	58.01158184
62.56899986	59.84501722	62.30203181	59.16258421	62.94002091
62.84567069	61.65842376	61.61722016	60.96555173	62.32755394
56.18645542	58.00562198	57.56649501	57.37199499	57.85719927
76.34693263	77.68936573	78.06259164	76.44135532	78.2995242
50.77824958	54.33549441	53.56586054	53.78669044	53.62878138
59.21468993	60.76457045	59.91534002	62.68111932	61.06962097
49.5924856	49.70221035	50.39673502	50.75761149	50.40836205
53.23631354	54.61881484	53.98532474	57.91171619	55.14669935
53.80575013	54.61881484	54.563036	66.25817167	55.14669935
46.68850057	48.47305923	47.4775982	50.75761149	49.22377772
51.92703405	50.93136147	52.73816057	43.60350679	51.59294637
76.97941277	75.51438391	77.6136442	75.79697794	75.28463288
77.66245358	77.97268615	78.25251657	73.41227637	77.65380153
73.44918905	73.05608167	74.08621559	71.0275748	72.91546423
57.33597385	57.07711708	58.09046461	44.79585757	57.515868
86.63080855	87.80589513	87.13569408	84.97730941	87.13047613
54.34757604	54.61881484	55.11361156	50.35238909	55.14669935
84.23945799	85.34759289	84.76368797	81.37688327	84.76130748
47.81270725	48.47305923	48.61770299	47.45332043	49.22377772
35.29946206	36.18154801	36.19177914	33.72959953	37.37793447
69.89135466	70.59777942	70.53165129	67.67653616	70.54629558
63.74759734	61.99372157	64.49320304	61.06017235	62.2542053
50.12136734	49.70221035	50.93549262	47.99106133	50.40836205
64.22139929	60.76457045	65.00487984	61.62128705	61.06962097
57.93460188	58.30626821	58.67162073	59.40020206	58.70045232
76.39413483		76.96622567	73.10054917	76.3467112

72.85905638		73.45554415	68.96238212	73.17415581
77.00495932		77.57105911	77.3555852	79.39381998
43.47410416		44.31564588	44.53087796	45.84426128
84.58204597		85.1026003	86.75414869	85.38098505
67.30968866		67.96471555	69.52349682	69.15291277
32.68705297		33.61795456	31.95276025	33.67107391
70.31186369		70.9415551	68.26102465	69.66139438
38.08057857		38.96680022	36.72216338	38.81386071
43.9810339		44.82582404	42.00593361	43.22117905
66.21294987	65.66068909	66.91692222	65.92615744	66.94698583
73.22454914	70.57729358	73.93618223	75.03384867	75.92610341
80.16628105	82.8688048	80.67690527	83.49593531	84.15894045
51.2263958	49.68172451	52.06769444	51.61016495	52.70169181
64.9360484	63.20238685	65.68254	63.51069524	64.62785037
62.62592403	61.97323573	63.35891305	65.54896644	66.40466113
72.82977699	73.03559582	73.4641286	73.49562512	74.40590371
70.75197239	66.88984021	71.51579696	72.6259837	73.55166924
49.31104358	45.99427114	50.2161211	50.38686766	51.48155984
66.93262681	68.11899134	67.58249249	65.1571386	66.22967641
71.35372848	74.88228572	74.33293469	71.42502506	74.87520975
33.09211958	33.13960087	34.01095482	33.26979999	27.57420453
85.70183181	86.01732577	85.01014824	85.73323446	81.31757719
55.80994986	55.28854772	55.36007183	55.92446488	48.37727656
64.17967681	62.6876811	62.47715219	64.27092036	56.29561368
64.17967681	72.92953912	72.54915105	64.27092036	71.11243921
67.76670264	66.37513446	66.03516136	67.84797271	59.84936666
49.83157347	49.14279211	49.43005655	49.96271096	55.09198644
70.1580532	69.43589894	68.99963799	70.23267428	67.39395064
59.3969757	55.9636899	55.9557284	59.50151723	57.31546241
59.59625491	57.7121785	59.02971702	56.31682625	58.55251392
70.35733241	77.37859645	73.81027598	73.02446339	72.73009017
46.44382686	56.48302738	51.8501569	58.0869447	54.99515575
42.85680102	44.19151616	44.18561628	46.75593071	46.67915294
40.46545047	51.56642289	46.50678897	47.3926031	45.65877729
67.96598186	82.29520094	74.95815397	72.46142192	71.06103314
48.83517741	54.02472514	51.87557361	55.70224313	54.4057183
67.96598186	76.14944533	72.02491721	71.83211261	71.10057304
69.16165714	77.37859645	73.21092027	74.20945109	73.11577903
61.98760547	46.6498184	54.94860227	48.54813843	54.86206998

附录二、代码

问题一：

```
class Scratch {
        public static void solve(int n) {
            int[][] edge = new int[n][n];
```

```
            int m1 = 0;
            int m2 = 0;
            int m3 = 0;
            int m4 = 0;
            int m5 = 0;

            //v_first 顺序, i 交替
            for (int v_first = 0; v_first < n; v_first++) {
                if (v_first % 2 == 0) {
                    for (int i = n - 1; i > v_first + 1; i--) {
                        if (edge[v_first][i - 1] == 0 &&
edge[v_first][i] == 0 && edge[i - 1][i] == 0) {
                            m1++;
                            edge[v_first][i - 1] = edge[v_first][i] =
edge[i - 1][i] = 1;
                            System.out.println(v_first+"    "+(i-1)+"
"+i);
                        }
                    }
                } else {
                    for (int i = v_first + 1; i < n - 1; i++) {
                        if (edge[v_first][i] == 0 && edge[v_first][i
+ 1] == 0 && edge[i][i + 1] == 0) {
                            m1++;
                            edge[v_first][i] = edge[v_first][i + 1] =
edge[i][i + 1] = 1;
                          System.out.println(v_first+" "+i+" "+(i+1));
                        }
                    }
                }
            }

            for (int va = 2; va < n; va++) {
                for (int vb = 1; vb < va; vb++) {
                    for (int vc = 0; vc < vb; vc++) {
                        if (edge[vb][va] == 0 && edge[vc][vb] == 0 &&
edge[vc][va] == 0) {
                            m1++;
                            edge[vc][vb] = edge[vc][va] = edge[vb][va]
= 1;
                            System.out.println(vc+" "+vb+" "+va);
                            break;
                        }
                    }
```

```
                }
            }


            //v_first 逆序,i 交替
            edge = new int[n][n];
            for (int v_first = n - 1; v_first >= 0; v_first--) {
                if (v_first % 2 == 0) {
                    for (int i = 0; i < v_first - 1; i++) {
                        if (edge[i][i + 1] == 0 && edge[i][v_first] ==
0 && edge[i + 1][v_first] == 0) {
                            m2++;
                            edge[i][i + 1] = edge[i][v_first] = edge[i
+ 1][v_first] = 1;
                        }
                    }
                } else {
                    for (int i = v_first - 1; i >= 1; i--) {
                        if (edge[i][v_first] == 0 && edge[i-1][v_first]
== 0 && edge[i - 1][i] == 0) {
                            m2++;
                            edge[i][v_first] = edge[i-1][v_first] =
edge[i-1][i] = 1;
                        }
                    }
                }
            }

            for (int va = 2; va < n; va++) {
                for (int vb = 1; vb < va; vb++) {
                    for (int vc = 0; vc < vb; vc++) {
                        if (edge[vb][va] == 0 && edge[vc][vb] == 0 &&
edge[vc][va] == 0) {
                            m2++;
                            edge[vc][vb] = edge[vc][va] = edge[vb][va]
= 1;
                            break;
                        }
                    }

                }

            }
```

```
            //v_first 顺序, i 顺序
            edge = new int[n][n];
            for (int v_first = 0; v_first < n; v_first++) {
                for (int i = v_first + 1; i < n - 1; i++) {
                    if (edge[v_first][i] == 0 && edge[v_first][i + 1]
== 0 && edge[i][i + 1] == 0) {
                        m3++;
                        edge[v_first][i] = edge[v_first][i + 1] =
edge[i][i + 1] = 1;
                    }
                }
            }

            for (int va = 2; va < n; va++) {
                for (int vb = 1; vb < va; vb++) {
                    for (int vc = 0; vc < vb; vc++) {
                        if (edge[vb][va] == 0 && edge[vc][vb] == 0 &&
edge[vc][va] == 0) {
                            m3++;
                            edge[vc][vb] = edge[vc][va] = edge[vb][va]
= 1;
//                            System.out.println(vc+" "+va+" "+vb);
                            break;
                        }
                    }
                }
            }



            //最原始顺序遍历
            edge = new int[n][n];
            for (int va = 2; va < n; va++) {
                for (int vb = 1; vb < va; vb++) {
                    for (int vc = 0; vc < vb; vc++) {
                        if (edge[vb][va] == 0 && edge[vc][vb] == 0 &&
edge[vc][va] == 0) {
                            m4++;
                            edge[vc][vb] = edge[vc][va] = edge[vb][va]
= 1;
                            break;
                        }
                    }
                }
```

```
            }


            //v_first 逆序, i 顺序
            edge = new int[n][n];
            for (int v_first = n - 1; v_first >= 0; v_first--) {
                for (int i = 0; i < v_first - 1; i++) {
                    if (edge[i][i + 1] == 0 && edge[i][v_first] == 0
&& edge[i + 1][v_first] == 0) {
                        m5++;
                        edge[i][i + 1] = edge[i][v_first] = edge[i +
1][v_first] = 1;
                    }
                }
            }

            for (int va = 2; va < n; va++) {
                for (int vb = 1; vb < va; vb++) {
                    for (int vc = 0; vc < vb; vc++) {
                        if (edge[vb][va] == 0 && edge[vc][vb] == 0 &&
edge[vc][va] == 0) {
                            m5++;
                            edge[vc][vb] = edge[vc][va] = edge[vb][va]
= 1;
                            break;
                        }
                    }
                }
            }

//            System.out.println(m1+" "+m2+" "+m3+" "+m4+" "+m5);
            int                          res                          =
Math.max(Math.max(Math.max(m1,m2),Math.max(m3,m4)),m5);
            System.out.printf("when n = %d, m need to less than or
equal to %d.\n", n, res);

        }


//        public static void main(String[] args) {
//               solve(10);
//    }
}
```

```
class Solution{
    public static void solve(){
        int[][] m = new int[20][10];
        int final_similarity = 0;
        //初始化
        m[0][0] = m[0][8] = m[0][9] = m[1][0] = m[1][6] = m[1][7] =
m[2][0] = m[2][4] = m[2][5] = m[3][0] = m[3][2] = m[3][3] = 1;
        m[4][1] = m[4][3] = m[4][4] = m[5][1] = m[5][5] = m[5][6] =
m[6][1] = m[6][7] = m[6][8] = m[7][2] = m[7][4] = m[7][6] = 1;
        m[8][2] = m[8][5] = m[8][7] = m[9][3] = m[9][5] = m[9][8] =
m[10][1] = m[10][2] = m[10][9] = m[11][3] = m[11][6] = m[11][9] = 1;
        m[12][4] = m[12][7] = m[12][9] = 1;
        for(int i = 1;i < 13;i++){
            int tmp_similarity = 0;
            for(int j = 0;j < i;j++){
                int tmp = 0;
                for(int k = 0;k < 10;k++){
                    tmp += m[i][k]*m[j][k];
                    tmp = (int)Math.pow(tmp,2);
                }
                tmp_similarity += tmp;
            }
            final_similarity += tmp_similarity;
        }
        for(int i = 13;i < 20;i++){
            int similarity = 100000;
            int p1 = 0,p2 = 0,p3 = 0;//评委编号
            for(int j = 2;j < 10;j++){
                for(int k = 1;k < j;k++){
                    for(int l = 0;l < k;l++){
                        //开始计算最小相似度
                        int tmp = 0;
                        for(int line = 0;line < i ;line++){
                            tmp                                          +=
(int)Math.pow(m[line][j]+m[line][k]+m[line][l],2);
                        }
                        if(tmp < similarity){
                            similarity = tmp;
                            p1 = j;
                            p2 = k;
                            p3 = l;
                        }
```

```
                    }
                }
            }
            final_similarity+=similarity;
            m[i][p1] = m[i][p2] = m[i][p3] = 1;
            System.out.println(i+":"+p1+"                    "+p2+"
"+p3+":tmp_similarity:"+similarity);
        }
        System.out.println(final_similarity);




    }
    public static void main(String[] args) {
        solve();
    }
}
```

问题二:

```
import math

import numpy
import pandas as pd
import numpy as np
import scipy.stats as stats
import csv
import matplotlib.pyplot as plt
def readCol(col,path):
    df = pd.read_csv(path)
    to_check = np.array(df.iloc[:, col-1:col])
    res = []
    for line in to_check:
        res.append(line[0])
    return np.array(res)
def readRow(row,path):
    df = pd.read_csv(path)
    to_check = np.array(df.iloc[row-1:row, :])
    res = to_check[0]
    return np.array(res)
def generateBox(col,path):
    to_check = readCol(col,path)
    plt.boxplot(to_check)
```

```
    plt.show()
    Q1 = np.quantile(a=to_check, q=0.25)
    Q3 = np.quantile(a=to_check, q=0.75)
    # 计算 四分位差
    QR = Q3 - Q1
    # 下限 与 上线
    low_limit = Q1 - 1.5 * QR
    up_limit = Q3 + 1.5 * QR
    print('下限为：', low_limit)
    print('上限为：', up_limit)
    print('异常值有：')
    print(to_check[(to_check < low_limit) + (to_check > up_limit)])
    return low_limit,up_limit,to_check[(to_check < low_limit) +
(to_check > up_limit)]
def generateBoxAll(path):
    dt = pd.read_csv(path)
    dt.boxplot()
    plt.show()
def gaussCheck(col,path):
    to_check = readCol(col,path)
    df = pd.DataFrame(to_check, columns=['value'])
    u = df['value'].mean()  # 计算均值
    std = df['value'].std()  # 计算标准差
    print(stats.kstest(df['value'], 'norm', (u, std)))
    # .kstest 方法：KS 检验，参数分别是：待检验的数据，检验方法（这里设置成 norm
正态分布），均值与标准差
    # 结果返回两个值：statistic → D 值，pvalue → P 值
    # p 值大于 0.05，为正态分布
def checkAll(path):
    to_check = readCol(1,path)
    for i in range(1, 11):
        to_check = np.append(to_check, readCol(i,path))
    df = pd.DataFrame(to_check, columns=['value'])
    u = df['value'].mean()  # 计算均值
    std = df['value'].std()  # 计算标准差
    print(stats.kstest(df['value'], 'norm', (u, std)))
    plt.boxplot(to_check)
    plt.show()
    Q1 = np.quantile(a=to_check, q=0.25)
    Q3 = np.quantile(a=to_check, q=0.75)
    # 计算 四分位差
    QR = Q3 - Q1
    # 下限 与 上线
    low_limit = Q1 - 1.5 * QR
```

```
    up_limit = Q3 + 1.5 * QR
    print('下限为：', low_limit)
    print('上限为：', up_limit)
    print('异常值有：')
    print(to_check[(to_check < low_limit) + (to_check > up_limit)])
    return low_limit,up_limit,to_check[(to_check < low_limit) +
(to_check > up_limit)]
def checkAllSep(path):
    for i in range(1,11):
        gaussCheck(i,path)
        generateBox(i,path)
        print("*"*20)
def Zscore(path):
    all_data = pd.read_csv(path)
    to_check = readCol(1, path)
    for i in range(1, 11):
        to_check = np.append(to_check, readCol(i, path))
    df = pd.DataFrame(to_check, columns=['value'])
    u_all = df['value'].mean()  # 计算均值
    std_all = df['value'].std()  # 计算标准差
    for i in range(1,11):
        to_check = readCol(i,path)
        df = pd.DataFrame(to_check, columns=['value'])
        u = df['value'].mean()  # 计算均值
        std = df['value'].std()  # 计算标准差
        for j in range(200):
            all_data.iloc[j-1,i-1]=(all_data.iloc[j-1,i-1]-
u)/std*std_all+u_all
    all_data.to_csv(r"data_after_normalization.csv", mode='w',
index=False)
def calAve(path):
    ave = []
    for i in range(1,201):
        row = readRow(i,path)
        row_ave = sum(row)/len(row)
        ave.append(row_ave)
    df = pd.read_csv("data_after_normalization.csv")
    df['平均值'] = ave
    df.to_csv(r"data_after_normalization.csv", mode='w', index=False)
def setPrize(path):
    res = open("prize.txt",'w',encoding="utf-8")
    ave = readCol(11,path)
    to_sort = []
    idx=1
```

```
    for i in ave:
        to_sort.append([idx,i])
        idx+=1
    to_sort.sort(key=lambda x: (-x[1]))
    res.write("获一等奖的组为：\n")
    for i in range(0,20):
        res.write("第{}组".format(str(to_sort[i][0]))+", 得分
{}".format(str(to_sort[i][1]))+'\n')
    res.write("获二等奖的组为：\n")
    for i in range(20,50):
        res.write("第{}组".format(str(to_sort[i][0]))+", 得分
{}".format(str(to_sort[i][1])+'\n'))
    res.write("获三等奖的组为：\n")
    for i in range(51, 100):
        res.write("第{}组".format(str(to_sort[i][0]))+", 得分
{}".format(str(to_sort[i][1]))+'\n')
def modifyData(path,path2):
    all_data = pd.read_csv(path)
    for i in range(1,11):
        low,up,invalid = generateBox(i,path)
        low=math.ceil(low)
        up =math.floor(up)
        for j in range(200):
            if all_data.iloc[j - 1, i - 1] in invalid:
                if all_data.iloc[j - 1, i - 1]>up:
                    all_data.iloc[j - 1, i - 1] = up
                else:
                    all_data.iloc[j - 1, i - 1] = low
    all_data.to_csv(path2, mode='w', index=False)
    # low_all, up_all, invalid_all =
checkAll(r"data_after_modification.csv")
    # low_all = math.ceil(low_all)
    # up_all = math.floor(up_all)
    # for i in range(1, 11):
    #     for j in range(200):
    #         if all_data.iloc[j - 1, i - 1] in invalid_all:
    #             if all_data.iloc[j - 1, i - 1] > up_all:
    #                 all_data.iloc[j - 1, i - 1] = up_all
    #             else:
    #                 all_data.iloc[j - 1, i - 1] = low_all
    # all_data.to_csv(r"data_after_modification.csv", mode='w',
index=False)
if __name__=="__main__":
    # checkAllSep("data.csv")  # 检查每个评委的打分情况是否符合正态分布
```

```
    # modifyData("data.csv","data_after_modification.csv")
    # checkAllSep("data_after_modification.csv")
    # generateBoxAll("data_after_modification.csv")
    # Zscore("data_after_modification.csv")
    #
modifyData("data_after_normalization.csv","data_after_normalization.c
sv")
    checkAllSep("data_after_normalization.csv")
    # generateBoxAll("data_after_normalization.csv")
    # calAve("data_after_normalization.csv")
    # setPrize("data_after_normalization.csv")
```

问题三：

```
import math
import random

import pandas as pd
import numpy as np
from numpy import nan as NaN
def reacCsv(path):
    df = pd.read_csv(path)
    return df
def readCol(col,path):
    df = pd.read_csv(path)
    to_check = np.array(df.iloc[:, col-1:col])
    res = []
    for line in to_check:
        res.append(line[0])
    return np.array(res)

def generateMask(toselect,step,matrix=None): #mask_matrix 是遮蔽矩阵，0
代表本轮未被使用，1 代表本轮被使用，-1 代表淘汰，2 代表已获奖
    if step!=1:
        mask_matrix = np.array(matrix)
    else:
        mask_matrix = np.full((200, 10), 0, dtype=int)
    if step==1 or step ==2:
        for i in range(1, 11):
            group_col = readCol(i, "group.csv")
            col_visible = []
            for idx in range(len(group_col)):
                if group_col[idx] in toselect:
```

```
                    for j in range(idx * 10, idx * 10 + 10):
                        col_visible.append(j)
            for j in col_visible:
                if mask_matrix[j][i - 1] != -1:
                    mask_matrix[j][i - 1] = 1

    return pd.DataFrame(mask_matrix)

def dealData(data_path,step,mask_matrix,step3=0):
    data = pd.read_csv(data_path)
    if step==1 or step==2:
        toselect = []
        toselect.append(step)
        for i in range(200):
            for j in range(10):
                if mask_matrix.iloc[i, j] != 1:
                    data.iloc[i, j] = NaN
        data.to_csv(r"data_step{}_raw.csv".format(step), mode='w',
index=False)
    if step==3 and step3==1:
        for i in range(30):
            for j in range(10):
                if mask_matrix.iloc[i, j] != 1:
                    data.iloc[i, j] = NaN
        data.to_csv(r"data_step{}.{}_raw.csv".format(step,step3),
mode='w', index=False)
    if step==3 and step3==2:
        for i in range(20):
            for j in range(10):
                if mask_matrix.iloc[i, j] != 1:
                    data.iloc[i, j] = NaN
        data.to_csv(r"data_step{}.{}_raw.csv".format(step,step3),
mode='w', index=False)
    if step==4 :
        data.to_csv(r"data_step{}_raw.csv".format(step), mode='w',
index=False)



def Zscore(filled_path,step,step3=0):
    if step==1 or step==2:
        all_data = pd.read_csv(filled_path)
        to_check = readCol(1, filled_path)
        for i in range(1, 11):
```

```
            to_check = np.append(to_check, readCol(i, filled_path))
        df = pd.DataFrame(to_check, columns=['value'])
        u_all = df['value'].mean()  # 计算均值
        std_all = df['value'].std()  # 计算标准差
        for i in range(1, 11):
            to_check = readCol(i, filled_path)
            df = pd.DataFrame(to_check, columns=['value'])
            u = df['value'].mean()  # 计算均值
            std = df['value'].std()  # 计算标准差
            for j in range(200):
                all_data.iloc[j - 1, i - 1] = (all_data.iloc[j - 1, i -
1] - u) / std * std_all + u_all
        all_data.to_csv(r"data_step{}_norm.csv".format(step),
mode='w', index=False)
    if step==3 and step3 ==1:
        all_data = pd.read_csv(filled_path)
        to_check = readCol(1, filled_path)
        for i in range(1, 11):
            to_check = np.append(to_check, readCol(i, filled_path))
        df = pd.DataFrame(to_check, columns=['value'])
        u_all = df['value'].mean()  # 计算均值
        std_all = df['value'].std()  # 计算标准差
        for i in range(1, 11):
            to_check = readCol(i, filled_path)
            df = pd.DataFrame(to_check, columns=['value'])
            u = df['value'].mean()  # 计算均值
            std = df['value'].std()  # 计算标准差
            for j in range(30):
                all_data.iloc[j - 1, i - 1] = (all_data.iloc[j - 1, i -
1] - u) / std * std_all + u_all
        all_data.to_csv(r"data_step{}.{}_norm.csv".format(step,step3),
mode='w', index=False)
    if step==3 and step3 ==2:
        all_data = pd.read_csv(filled_path)
        to_check = readCol(1, filled_path)
        for i in range(1, 11):
            to_check = np.append(to_check, readCol(i, filled_path))
        df = pd.DataFrame(to_check, columns=['value'])
        u_all = df['value'].mean()  # 计算均值
        std_all = df['value'].std()  # 计算标准差
        for i in range(1, 11):
            to_check = readCol(i, filled_path)
            df = pd.DataFrame(to_check, columns=['value'])
            u = df['value'].mean()  # 计算均值
```

```
            std = df['value'].std()  # 计算标准差
            for j in range(20):
                all_data.iloc[j - 1, i - 1] = (all_data.iloc[j - 1, i -
1] - u) / std * std_all + u_all
        all_data.to_csv(r"data_step{}.{}_norm.csv".format(step,step3),
mode='w', index=False)
    if step==4:
        all_data = pd.read_csv(filled_path)
        to_check = readCol(1, filled_path)
        for i in range(1, 11):
            to_check = np.append(to_check, readCol(i, filled_path))
        df = pd.DataFrame(to_check, columns=['value'])
        u_all = df['value'].mean()  # 计算均值
        std_all = df['value'].std()  # 计算标准差
        for i in range(1, 11):
            to_check = readCol(i, filled_path)
            df = pd.DataFrame(to_check, columns=['value'])
            u = df['value'].mean()  # 计算均值
            std = df['value'].std()  # 计算标准差
            for j in range(10):
                all_data.iloc[j - 1, i - 1] = (all_data.iloc[j - 1, i -
1] - u) / std * std_all + u_all
        all_data.to_csv(r"data_step4_norm.csv", mode='w', index=False)

def calAve(norm_path,step,mask_matrix,step3=0):
    if step ==1 or step ==2:
        data = pd.read_csv(norm_path)
        ave = []
        judge = [3,5,7,10]
        for i in range(200):
            tem_ave = 0
            for j in range(10):
                if mask_matrix.iloc[i,j]==1:
                    tem_ave += data.iloc[i, j]
            tem_ave = tem_ave/judge[step-1]
            ave.append(tem_ave)
        data['平均值'] = ave
        data.to_csv(norm_path, mode='w', index=False)
    if step ==3 and step3 ==1:
        data = pd.read_csv(norm_path)
        ave = []
        judge = [3,5,7,10]
        for i in range(30):
            tem_ave = 0
```

```
            for j in range(10):
                if mask_matrix.iloc[i,j]==1:
                    tem_ave += data.iloc[i, j]
            tem_ave = tem_ave/judge[step-1]
            ave.append(tem_ave)
        data['平均值'] = ave
        data.to_csv(norm_path, mode='w', index=False)
    if step ==3 and step3 ==2:
        data = pd.read_csv(norm_path)
        ave = []
        judge = [3,5,7,10]
        for i in range(20):
            tem_ave = 0
            for j in range(10):
                if mask_matrix.iloc[i,j]==1:
                    tem_ave += data.iloc[i, j]
            tem_ave = tem_ave/judge[step-1]
            ave.append(tem_ave)
        data['平均值'] = ave
        data.to_csv(norm_path, mode='w', index=False)
    if step ==4:
        data = pd.read_csv(norm_path)
        ave = []
        judge = [3,5,7,10]
        for i in range(10):
            tem_ave = 0
            for j in range(10):
                if mask_matrix.iloc[i,j]==1:
                    tem_ave += data.iloc[i, j]
            tem_ave = tem_ave/judge[step-1]
            ave.append(tem_ave)
        data['平均值'] = ave
        data.to_csv(norm_path, mode='w', index=False)

def ranking(norm_path,mask_matrix,step,step3=0):
    data = pd.read_csv(norm_path)
    f = open("prize3.txt",'a+',encoding='utf-8')
    if step ==1:
        f.write("第一步淘汰:\n")
        for group_index in range(20):
            to_sort = []
            f.write("第{}大组: \n".format(group_index+1))
            for sub_group_index in
range(10*group_index,10*group_index+10):
```

```
                
to_sort.append([sub_group_index,data.iloc[sub_group_index,10]])
            to_sort.sort(key=lambda x: (x[1]))
            for ele in to_sort[:4]:
                mask_matrix.iloc[ele[0],:] = -1
                f.write("第{}组: ,得分: {}\n".format(ele[0] + 1,ele[1]))
        for i in range(200):
            for j in range(10):
                if mask_matrix.iloc[i,j]!=1:
                    data.iloc[i, j] = NaN
        data.to_csv(r"data_step{}_final.csv".format(step), mode='w',
index=False)
        return mask_matrix
    if step ==2:
        original_data = np.array(pd.read_csv("data.csv"))
        f.write("第二步淘汰:\n")
        to_sort = []
        for group_index in range(200):
            to_sort.append([group_index,data.iloc[group_index,10]])
        to_sort.sort(key=lambda x: (x[1]))
        for ele in to_sort[:100]:
            mask_matrix.iloc[ele[0],:] = -1
            f.write("第{}组: ,得分: {}\n".format(ele[0] + 1, ele[1]))
        f.write("第二步获二等奖:\n")
        for ele in to_sort[100:140]:
            mask_matrix.iloc[ele[0],:] = -1
            f.write("第{}组: ,得分: {}\n".format(ele[0] + 1, ele[1]))
        f.write("第二步获三等奖:\n")
        for ele in to_sort[160:170]:
            mask_matrix.iloc[ele[0],:] = -1
            f.write("第{}组: ,得分: {}\n".format(ele[0] + 1, ele[1]))
                
        for i in range(1, 11):
            group_col = readCol(i, "group_new.csv")
            col_visible = []
            for idx in range(len(group_col)):
                if group_col[idx] in [3]:
                    for j in range(idx * 10, idx * 10 + 10):
                        col_visible.append(j)
            for j in col_visible:
                if mask_matrix.iloc[j,i - 1] != -1:
                    mask_matrix.iloc[j,i - 1] = 1
        group_1 = []
        mask_matrix_1=[]
```

```
        idx_1 = []
        for ele in to_sort[170:200]:
            group_1.append(original_data[ele[0]])
            idx_1.append(ele[0])
            mask_matrix_1.append(np.array(mask_matrix.iloc[ele[0],:]))

        res1 = pd.DataFrame(group_1)
        res1["index"]=idx_1
        res1.to_csv(r"data_step3.1.csv", mode='w', index=False)
        mask_matrix_1 =pd.DataFrame(mask_matrix_1)
        mask_matrix_1['index']=idx_1
        group_2 = []
        mask_matrix_2 = []
        idx_2 = []
        for ele in to_sort[140:160]:
            group_2.append(original_data[ele[0]])
            idx_2.append(ele[0])
            mask_matrix_2.append(np.array(mask_matrix.iloc[ele[0], :]))
        res2 = pd.DataFrame(group_2)
        res2["index"] = idx_2
        res2.to_csv(r"data_step3.2.csv", mode='w', index=False)
        mask_matrix_2 = pd.DataFrame(mask_matrix_2)
        mask_matrix_2['index'] = idx_2
        for i in range(200):
            for j in range(10):
                if mask_matrix.iloc[i,j]!=1:
                    data.iloc[i, j] = NaN
        data.to_csv(r"data_step{}_final.csv".format(step), mode='w',
index=False)
        return mask_matrix_1,mask_matrix_2
    if step ==3 and step3 ==2:
        to_sort=[]
        for index in range(20):
            to_sort.append([data.iloc[index, 10], data.iloc[index,
11]])
        to_sort.sort(key=lambda x: (x[1]))
        f.write("第三步的 41-60 名中，获三等奖为：\n")
        for ele in to_sort[0:10]:
            mask_matrix.iloc[ele[0], :] = -1
            f.write("第{}组：,得分：{}\n".format(ele[0] + 1, ele[1]))
        f.write("第三步的 41-60 名中，获二等奖为：\n")
        for ele in to_sort[10:20]:
            mask_matrix.iloc[ele[0], :] = -1
            f.write("第{}组：,得分：{}\n".format(ele[0] + 1, ele[1]))
```

```
        for i in range(20):
            for j in range(10):
                if mask_matrix.iloc[i,j]!=1:
                    data.iloc[i, j] = NaN
        data.to_csv(r"data_step{}.2_final.csv".format(step), mode='w',
index=False)
    if step ==3 and step3 ==1:
        original_data = np.array(pd.read_csv("data.csv"))
        to_sort=[]
        for index in range(30):
            to_sort.append([data.iloc[index, 10], data.iloc[index,
11]])
        to_sort.sort(key=lambda x: (x[1]))
        f.write("第三步的 1-30 名中，获二等奖为：\n")
        for ele in to_sort[0:5]:
            f.write("第{}组：,得分：{}\n".format(ele[0] + 1, ele[1]))
        f.write("第三步的 1-30 名中，获一等奖为：\n")
        for ele in to_sort[15:30]:
            f.write("第{}组：,得分：{}\n".format(ele[0] + 1, ele[1]))
        group_1 = []
        mask_matrix_1 = []
        idx_1 = []
        for ele in to_sort[5:15]:
            group_1.append(original_data[ele[0]])
            idx_1.append(ele[0])
            for i in range(30):
                if mask_matrix.iloc[i, 10]==ele[0]:

mask_matrix_1.append(np.array(mask_matrix.iloc[i, :]))
                    break
        mask_matrix_1 = pd.DataFrame(mask_matrix_1)
        res1 = pd.DataFrame(group_1)
        res1["index"] = idx_1
        res1.to_csv(r"data_step4.csv", mode='w', index=False)
        for i in range(30):
            for j in range(10):
                mask_matrix.iloc[i,j]=1

        data.to_csv(r"data_step{}.1_final.csv".format(step), mode='w',
index=False)
        return mask_matrix_1
    if step == 4:
        original_data = np.array(pd.read_csv("data.csv"))
        to_sort = []
```

```
        for index in range(10):
            to_sort.append([data.iloc[index, 10], data.iloc[index,
11]])
        to_sort.sort(key=lambda x: (x[1]))
        f.write("第四步中，获二等奖为：\n")
        for ele in to_sort[0:5]:
            f.write("第{}组：,得分：{}\n".format(ele[0] + 1, ele[1]))
        f.write("第四步中，获一等奖为：\n")
        for ele in to_sort[5:10]:
            f.write("第{}组：,得分：{}\n".format(ele[0] + 1, ele[1]))
        data.to_csv(r"data_step4_final.csv", mode='w', index=False)


if __name__=="__main__":
    step1_matrix_raw = generateMask([1],1)
    step1_matrix_raw.to_csv("step1_mask_matrix.csv", mode='w',
index=False)
    # dealData("data.csv",1,step1_matrix_raw)
    # Zscore("data_step1_new.csv",1)
    # calAve("data_step1_norm.csv",1,step1_matrix_raw)
    # step1_matrix_ranked =
ranking("data_step1_norm.csv",step1_matrix_raw,1)
    # step2_matrix_raw =generateMask([2],2,step1_matrix_ranked)
    # dealData("data.csv",2,step2_matrix_raw)
    # step2_matrix_raw.to_csv("step2_mask_matrix.csv", mode='w',
index=False)
    # Zscore("data_step2_new.csv",2)
    # calAve("data_step2_norm.csv", 2, step2_matrix_raw)
    # step2_matrix_ranked_1,step2_matrix_ranked_2 =
ranking("data_step2_norm.csv",step2_matrix_raw,2)
    # step3_matrix_raw_1 = generateMask([3],3,step2_matrix_ranked_1)
    # step3_matrix_raw_2 = generateMask([3], 3,
step2_matrix_ranked_2)
    # step3_matrix_raw_1.to_csv("step3.1_mask_matrix.csv", mode='w',
index=False)
    # step3_matrix_raw_2.to_csv("step3.2_mask_matrix.csv", mode='w',
index=False)
    # dealData("data_step3.1.csv",3,step3_matrix_raw_1,1)
    # dealData("data_step3.2.csv", 3, step3_matrix_raw_2, 2)


    # step3_matrix_raw_1 = pd.read_csv("step3.1_mask_matrix.csv")
    # step3_matrix_raw_2 = pd.read_csv("step3.2_mask_matrix.csv")
    # Zscore("data_step3.1_new.csv", 3,1)
```

```
    # Zscore("data_step3.2_new.csv",3,2)
    # calAve("data_step3.1_norm.csv",3,step3_matrix_raw_1,1)
    # calAve("data_step3.2_norm.csv", 3, step3_matrix_raw_2,2)
    # step4_matrix_raw=ranking("data_step3.1_norm.csv",
step3_matrix_raw_1, 3,1)
    # step4_matrix_raw.to_csv("step4_mask_matrix.csv", mode='w',
index=False)
    # ranking("data_step3.2_norm.csv", step3_matrix_raw_2, 3)
    # step4_matrix_raw = pd.read_csv("step4_mask_matrix.csv")
    # dealData("data_step4.csv",4,step4_matrix_raw)
    # Zscore("data_step4_raw.csv",4)
    # calAve("data_step4_norm.csv",4,step4_matrix_raw)
    # ranking("data_step4_norm.csv", step4_matrix_raw, 4)
```