

作业一：牌九游戏程序设计

1. 需求分析

牌九作为中国传统的牌类游戏，使用32张牌进行游戏，每位玩家每轮获得2张牌后进行比牌。游戏的核心在于牌型的复杂性，既包含至尊宝、双天等特殊对牌，也有通过点数计算胜负的普通牌型，其中丁三牌和二四牌具有特殊的对调规则，使得游戏策略更加丰富。基于这些特点，本程序需要实现三个核心功能：首先设计一套完整的牌编码体系并将其保存为Excel文件以供后续使用，然后开发一个能够随机洗牌并向4位玩家各发2张牌的发牌系统，最后构建一个能够准确识别各种牌型并判断胜负的比较算法。

2. 设计思路

在数据结构设计上，选择字典作为基础数据类型来存储每张牌的信息，其中包含牌名、点数以及是否为特殊牌（如二四牌）的标识，这种设计既保证了数据访问的高效性，又为后续的牌型判断提供了便利。

程序主要包含四个核心模块：牌组生成模块负责根据牌九规则创建包含32张牌的完整牌组并进行编码，发牌模块实现牌组的随机洗牌和按规则分发，胜负判断模块处理复杂的牌型识别和分数计算逻辑，游戏运行模块则将前述功能整合为完整的游戏流程。

3. 核心代码实现

3.1 牌组编码与生成

牌组编码与生成模块是整个牌九游戏系统的基础，其核心任务是根据牌九游戏的传统规则构建一个包含32张牌的完整数据结构。在实现过程中，首先需要根据牌九的规则将所有牌分为两类：成对出现的牌（如天、地、人等，每种有2张）和单独出现的牌（如红九、黑九等，每种只有1张）。程序通过遍历这两个分类列表，为每张牌创建包含名称和点数信息的字典对象，并将所有牌组织成一个统一的列表结构。这种设计不仅保证了牌组数据的完整性和准确性，还为后续的数据持久化提供了标准化的数据格式。生成的牌组数据随即被转换为 `DataFrame` 格式，并同时保存为Excel和CSV两种文件格式

```
def create_full_deck():  
    """  
    根据定义的牌种和标准数量，生成一个包含32张牌的完整列表作为编码。  
    """  
    # 成对的牌（每种2张）  
    double_tile_names = [  
        '天', '地', '人', '和', '梅花', '长三', '板凳',  
        '斧头', '红头十', '高脚七', '零霖六'  
    ]  
    # 只有一张的牌  
    single_tile_names = [  
        '红九', '黑九', '平八', '斜八', '红七', '黑七',  
        '红五', '黑五', '丁三', '二四'  
    ]  
  
    deck = []  
    # 添加成对的牌  
    for name in double_tile_names:  
        for _ in range(2):  
            card_info = CARD[name].copy()  
            card_info['name'] = name  
            deck.append(card_info)
```

```

# 添加单张牌
for name in single_tile_names:
    card_info = CARD[name].copy()
    card_info['name'] = name
    deck.append(card_info)

return deck

# 保存编码到文件
full_deck_encoding = create_full_deck()
df = pd.DataFrame(full_deck_encoding)
df.to_excel("paijiu_encoding.xlsx", index=False, engine='openpyxl')
df.to_csv("paijiu_encoding.csv", index=False, encoding='utf-8-sig')

```

3.2 发牌程序

首先创建原始牌组的副本以避免修改原始数据，然后使用Python内置的 `random.shuffle()` 函数对牌组进行随机打乱，这种方法能够保证每张牌出现在任意位置的概率相等。在发牌阶段，依次为每位玩家分配指定数量的牌，通过维护一个牌索引变量来跟踪当前发牌的位置，确保不会出现重复发牌或遗漏的情况。

```

def distro_cards(deck, num_players=4, cards_per_hand=2):
    """随机发牌给指定数量的玩家"""
    shuffled_deck = deck[:]
    random.shuffle(shuffled_deck)    # 打乱牌组顺序

    # 按顺序分发给玩家
    player = []
    card_index = 0
    for i in range(num_players):
        hand = shuffled_deck[card_index : card_index + cards_per_hand]
        player.append(hand)
        card_index += cards_per_hand

    return player

```

3.3 胜负判断核心逻辑

胜负判断模块首先检查玩家手牌是否构成特殊对牌（如至尊宝、双天等），这些对牌具有固定的高分值和明确的优先级关系。为了提高匹配效率，程序使用 `frozenset` 数据结构来表示牌的组合，利用集合的哈希特性实现快速查找，避免了逐一比较的性能开销。

当手牌不构成特殊对牌时，程序转入点数计算模式，根据两张牌的点数和计算最终得分。在这个过程中，需要特别处理二四牌的特殊规则：当手牌中包含二四牌且不与丁三构成至尊宝时，二四牌可以灵活地按3点或6点计算，程序会自动选择能够产生更高得分的计算方式。这种多层次的判断逻辑既保证了规则的准确实现，又通过算法优化提升了执行效率。

```

def evaluate_cards(hand):
    """评估手牌的分数和牌型"""
    card1, card2 = hand[0], hand[1]
    card_names = frozenset([card1['name'], card2['name']])

    # 优先检查特殊对牌
    for definition in HAND_DEFINITIONS:
        if card_names == definition['cards']:
            hand_name = definition['name']

```

```

        score = PAIR_RANKING.get(hand_name, 0)
        return (score, hand_name)

# 计算点数（处理二四牌的特殊规则）
points = (card1['p'] + card2['p']) % 10

# 二四牌特殊规则：可以当3点或6点使用
is_gee_joon = card_names == frozenset(['丁三', '二四'])
if not is_gee_joon:
    if card1.get('is_wild'): # 二四牌
        wild_points = (3 + card2['p']) % 10
        if wild_points > points:
            points = wild_points
    elif card2.get('is_wild'):
        wild_points = (card1['p'] + 3) % 10
        if wild_points > points:
            points = wild_points

return (points, f"点数 {points}")

```

4. 运行结果示例

⑧29701 on D:/Temp/大作业

python .\hw1.py

玩家 1 的手牌：[高脚七，黑七]

玩家 2 的手牌：[和，长三]

玩家 3 的手牌：[天，红头十]

玩家 4 的手牌：[高脚七，红九]

→ 牌型：点数 4，分数：4

→ 牌型：点数 0，分数：0

→ 牌型：点数 2，分数：2

→ 牌型：点数 6，分数：6

获胜者是：玩家 4

手牌：[高脚七，红九]（点数 6）

⑧29701 on D:/Temp/大作业

python .\hw1.py

玩家 1 的手牌：[红头十，人]

玩家 2 的手牌：[红九，零霖六]

玩家 3 的手牌：[地，长三]

玩家 4 的手牌：[板凳，梅花]

→ 牌型：点数 8，分数：8

→ 牌型：点数 5，分数：5

→ 牌型：点数 8，分数：8

→ 牌型：点数 4，分数：4

本轮为平局，共同获胜者：玩家 1，玩家 3

二者都以牌型“点数 8”获得最高分 8

⑧29701 on D:/Temp/大作业

python .\hw1.py

玩家 1 的手牌：[平八，地]

玩家 2 的手牌：[丁三，板凳]

玩家 3 的手牌：[黑五，天]

玩家 4 的手牌：[零霖六，斧头]

→ 牌型：地牌八，分数：81

→ 牌型：点数 7，分数：7

→ 牌型：点数 7，分数：7

→ 牌型：点数 7，分数：7

获胜者是：玩家 1

手牌：[平八，地]（地牌八）

作业二：学生成绩数据处理

1. 需求分析

在本题中需要从 ds_1_2.xlsx 和 ds_3_4.xlsx 两个文件中提取1-4班学生的work1至work5平时成绩，同时从 数据结构成绩记录表(1).xlsx 文件的4个sheet中获取标准的录分名单作为数据整合的基准。在数据处理过程中，程序需要根据可配置的计算规则（如work1和work2求和作为平时作业1成绩，work3、work4、work5求和作为平时作业2成绩）对原始成绩进行处理，并确保最终输出的成绩表严格

按照录分名单的顺序排列，同时提供按分组统计的详细分析报告。

2. 设计思路

通过 `GradeProcessor` 类来封装所有的数据处理逻辑，这种设计不仅提高了代码的可维护性，还使得计算规则的修改变得灵活便捷。

首先进行数据预处理阶段，将原始的Excel文件转换为更易处理的CSV格式，并从录分名单模板中提取各班级的学生信息作为数据整合的基准。

接下来进入数据读取和整合阶段，程序会合并来自不同班级的成绩数据，然后根据预设的计算规则对原始成绩进行处理，生成新的综合成绩列。

在数据匹配阶段，系统通过学生姓名作为关键字，将处理后的成绩数据与学生名单进行精确匹配，确保每个学生的成绩都能正确对应到其基本信息。

最后，程序将整合后的数据按照不同的格式和分组方式进行输出，既生成完整的CSV文件供后续分析使用，又创建按分组划分的Excel文件便于教学管理。

3. 核心代码实现

3.1 数据预处理模块

数据预处理模块承担着整个成绩处理系统的基础数据准备工作，其主要目标是将复杂的Excel文件转换为程序更易处理的标准化格式。由于不同的Excel文件可能采用不同的工作表结构和数据组织方式，该模块需要针对每个输入文件的特点进行定制化处理。通过 `pandas` 库的 `read_excel` 功能，程序能够准确读取指定工作表的内容，并利用 `to_csv` 方法将数据转换为UTF-8编码的CSV格式，这种格式不仅处理速度更快，还能有效避免Excel文件可能存在的格式兼容性问题

```
def convert_xlsx_to_csv(self):
    """前置操作：将Excel文件转换为CSV文件"""
    try:
        # 转换ds_1_2.xlsx到CSV
        df_12 = pd.read_excel('data/ds_1_2.xlsx', sheet_name='c1_2')
        df_12.to_csv('data/ds_1_2.csv', index=False, encoding='utf-8-sig')

        # 转换ds_3_4.xlsx到CSV
        df_34 = pd.read_excel('data/ds_3_4.xlsx', sheet_name='01-绪论')
        df_34.to_csv('data/ds_3_4.csv', index=False, encoding='utf-8-sig')

        print("✓ Excel文件转换完成")
        return True
    except Exception as e:
        print(f"✗ Excel转换失败: {e}")
        return False
```

3.2 学生名单提取模块

学生名单提取模块的设计目标是从标准化的录分名单模板中提取各班级的学生基本信息，为后续的数据匹配建立准确的基准数据。该模块需要处理包含多个工作表的复杂Excel文件，每个工作表对应不同的班级分组。

程序首先建立工作表名称与班级标识之间的映射关系，然后逐一访问每个工作表并提取姓名班级等信息。

在数据提取过程中，程序采用 `iloc` 切片操作来精确选择所需的列范围，避免包含不相关的数据字段。提取的数据随即被保存为独立的CSV文件

```

def extract_student_lists_from_xlsx(self):
    """前置操作：从原始xlsx文件中提取学生名单"""
    try:
        excel_file = pd.ExcelFile('data/数据结构成绩记录表(1).xlsx')

        # 定义sheet名称映射
        sheet_mapping = {
            '20201-1小1': '1小1',
            '20201-1小2': '1小2',
            '20201-3小1': '3小1',
            '20201-3小2': '3小2'
        }

        for sheet_name, group_name in sheet_mapping.items():
            if sheet_name in excel_file.sheet_names:
                # 读取sheet数据，只保留前两列（姓名和班级）
                df = pd.read_excel('data/数据结构成绩记录表(1).xlsx',
sheet_name=sheet_name)
                df_output = df.iloc[:, :2]

                # 保存为CSV文件
                output_filename = f'data/{sheet_name}.csv'
                df_output.to_csv(output_filename, index=False, encoding='utf-8-
sig', header=False)

                print("✓ 学生名单提取完成")
                return True
    except Exception as e:
        print(f"✗ 学生名单提取失败: {e}")
        return False

```

3.3 成绩计算模块

成绩计算模块实现了灵活可配置的成绩处理规则，能够根据预设的计算公式对原始成绩数据进行加工处理。该模块的核心是基于规则引擎的设计思想，通过在类初始化时定义的 `scoring_rules` 字典来指定不同成绩项目的计算方式。

在执行计算时，程序首先验证所需的原始成绩列是否在数据中存在，对于缺失的数据列会给出明确的警告信息并跳过相应的计算规则。对于可以执行的计算规则，程序利用 `pandas` 的向量化操作特性，通过 `fillna` 方法处理缺失值（将其替换为0），然后使用 `sum` 函数沿行方向计算各work列的总和。

```

def calculate_scores(self, df_grades):
    """根据规则计算成绩"""
    if df_grades is None:
        return None

    result_df = df_grades.copy()

    # 根据规则计算新的成绩列
    calculated_count = 0
    for rule_name, work_columns in self.scoring_rules.items():
        # 检查所需列是否存在
        missing_cols = [col for col in work_columns if col not in
df_grades.columns]
        if missing_cols:
            print(f"⚠ 缺少列 {missing_cols}，跳过 {rule_name}")
            continue

```

```

# 计算成绩（处理缺失值）
scores = df_grades[work_columns].fillna(0).sum(axis=1)
result_df[rule_name] = scores.round(1) # 保留一位小数
calculated_count += 1

print(f"✓ 成绩计算完成: {calculated_count}项")
return result_df

```

3.4 数据整合模块

数据整合模块负责将来自不同数据源的信息按照统一的结构进行合并和匹配。

该模块首先需要处理多个学生名单文件的合并工作，通过为每个班级的学生数据添加分组标识，然后使用 `pandas` 的 `concat` 函数将所有名单数据纵向连接成一个统一的 `DataFrame`。

在数据匹配阶段，程序采用基于学生姓名的精确匹配策略，遍历合并后的学生名单，在计算后的成绩数据中查找对应的记录。当找到匹配的学生记录时，程序会将该学生的所有成绩信息（包括原始的work成绩和计算后的综合成绩）复制到最终的成绩表中相应的位置。

```

def create_final_grade_table(self, df_calculated, student_lists):
    """创建最终的成绩记录表"""
    # 合并所有学生名单
    all_students = []
    for group_name, student_df in student_lists.items():
        students_with_group = student_df.copy()
        students_with_group['分组'] = group_name
        all_students.append(students_with_group)

    df_all_students = pd.concat(all_students, ignore_index=True)

    # 创建最终表格并匹配成绩数据
    final_df = pd.DataFrame(columns=['姓名', '班级', '分组', 'work1', 'work2',
                                     'work3', 'work4', 'work5', '平时作业1', '平时作业2'])

    # 填充学生基本信息并匹配成绩
    matched_count = 0
    for idx, student_name in enumerate(df_all_students['姓名']):
        student_grade = df_calculated[df_calculated['name'] == student_name]
        if not student_grade.empty:
            matched_count += 1
            # 数据匹配和填充逻辑...

    print(f"✓ 成绩表创建完成: {len(df_all_students)}人，匹配{matched_count}人")
    return final_df

```

4. 可视化分析

在数据处理阶段，程序首先从主数据框中提取各个分组的数据，然后对指定的成绩列进行数值化处理，通过 `pd.to_numeric` 函数确保数据类型的正确性，并使用 `dropna` 方法剔除无效数据。图表绘制采用直方图的形式来展示成绩分布的频率特征，同时叠加统计线（如均值线、标准差线等）来突出重要的统计特征。这种组合式的可视化方法不仅能够直观展示每个班级的成绩分布形态，还便于进行跨班级的比较分析，为教学质量评估和教学改进提供了有价值的数据支撑。

```

class Gradevisualizer:
    def plot_group_score_distribution(self, score_column, save_path='assets/'):
        """绘制各分组的的成绩分布图"""

```

```

groups = self.df['分组'].unique()

fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle(f'{score_column} Score Distribution by Group', fontsize=16)

for i, group in enumerate(groups):
    group_data = self.df[self.df['分组'] == group]
    scores = pd.to_numeric(group_data[score_column],
errors='coerce').dropna()

    # 绘制直方图和统计信息
    axes.flatten()[i].hist(scores, bins=15, alpha=0.7, color='skyblue')
    axes.flatten()[i].axvline(scores.mean(), color='red', linestyle='--',
                                label=f'Mean: {scores.mean():.1f}')

    # 添加其他统计线...

```

5. 运行结果

② 29701 on D:/Temp/大作业

python .\hw2.py

数据结构成绩处理开始 ...

✓ Excel文件转换完成

✓ 学生名单提取完成

计算规则：平时作业1(work1+work2) | 平时作业2(work3+work4+work5)

✓ 成绩数据读取完成：145条记录

✓ 学生名单读取完成：136人

✓ 成绩计算完成：2项

✓ 成绩表创建完成：136人，匹配136人

✓ 结果保存完成：136条记录，4个分组

✓ 处理完成！输出文件：

- 数据结构成绩记录表_完整.csv
- 数据结构成绩记录表_最终.xlsx

是否生成可视化图表？(y/n): y

✓ 数据加载成功：136 条记录

正在生成图表 ...

正在生成分组成绩分布图 ...

✓ 已生成 平时作业1 分组分布图

✓ 已生成 平时作业2 分组分布图

✓ 图表已保存到 assets/ 目录

✓ 分组成绩统计汇总：

sum 1:

1小1组：均值 168.1 最高 197.5 最低 0.0 标准差 38.5

1小2组：均值 172.7 最高 197.5 最低 100.0 标准差 24.7

3小1组：均值 169.6 最高 197.5 最低 65.6 标准差 31.9

3小2组：均值 166.6 最高 197.5 最低 85.0 标准差 31.7

sum 2:

1小1组：均值 248.2 最高 285.6 最低 97.3 标准差 48.7

1小2组：均值 234.4 最高 285.0 最低 85.3 标准差 53.7

3小1组：均值 242.9 最高 293.6 最低 87.3 标准差 55.9

3小2组：均值 205.6 最高 293.3 最低 0.0 标准差 71.3

✓ 图表生成完成！请查看 assets/ 目录

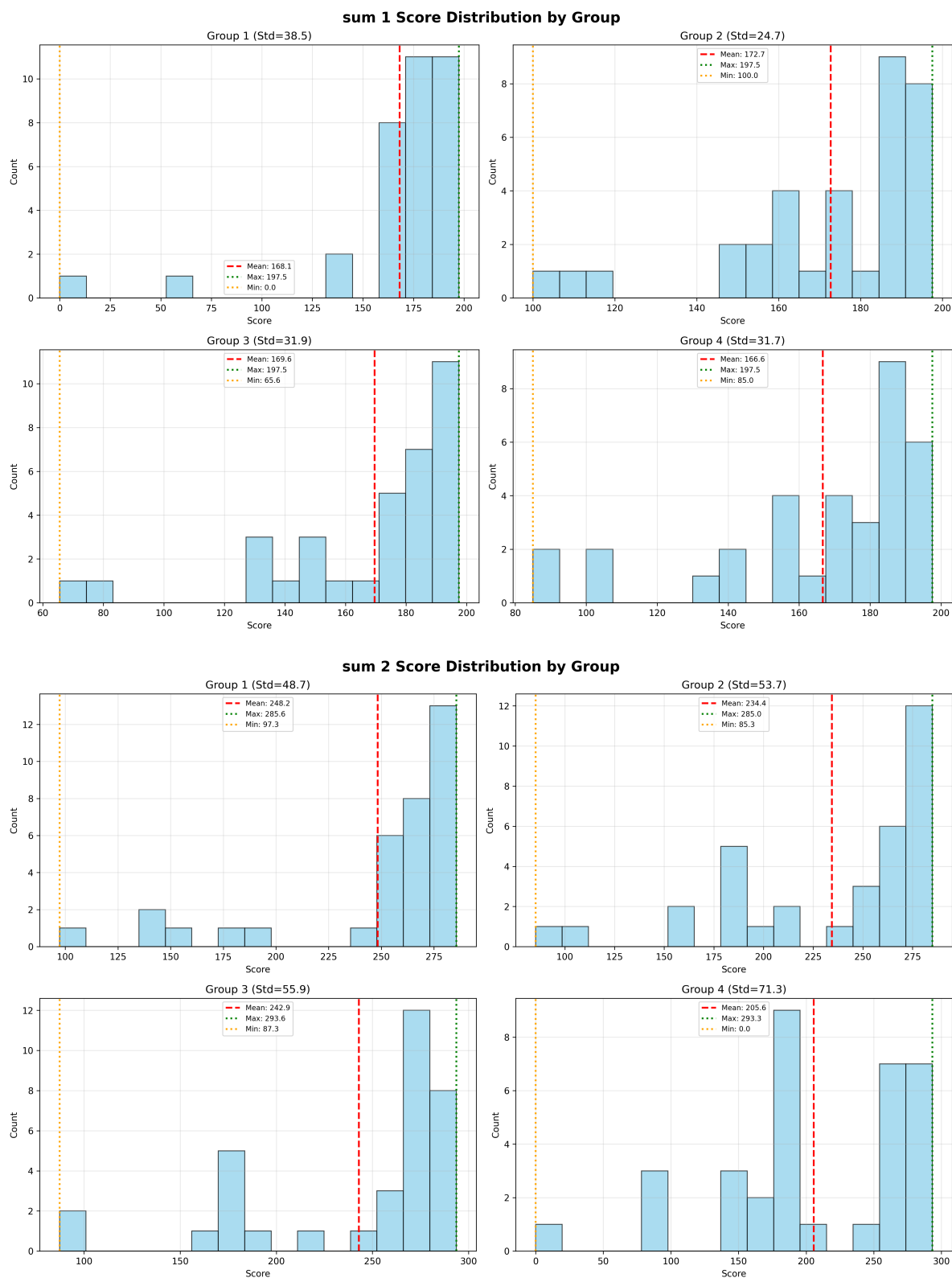
② 29701 on D:/Temp/大作业

█

#	A	B	C	D	E	F	G	H	I	J	K	L	M
1	姓名	班级	work1	work2	work3	work4	work5	平时作业	平时作业	选题 (总分)	选题 (总分)	备注	
2	闫麒卉	通信工程2	92.4	77.5	96.3	80.2	99.3	169.9	275.8	83.6	8.8		
3	叶琼	通信工程2	45.6	95	88.9	86.8	99.3	140.6	275	45.6	0		
4	黄可	通信工程2	92.4	85	0	86.8	99.3	177.4	186.1	83.6	8.8		
5	王爽	通信工程2	92.4	87.5	85.2	86.8	93.3	179.9	265.3	83.6	8.8		
6	王莉娜	通信工程2	91.2	82.5	92.6	73.6	99.3	173.7	265.5	91.2	0		
7	刘鑫	通信工程2	92.4	80	88.8	80.2	99.3	172.4	268.3	83.6	8.8		
8	牛鹤清	通信工程2	92.4	85	88.9	93.4	95.3	177.4	277.6	83.6	8.8		
9	王杰	通信工程2	77.2	87.5	96.3	100	89.3	164.7	285.6	68.4	8.8		
10	丁帅宇	通信工程2	100	95	96.3	93.4	83.3	195	273	91.2	8.8		
11	林召芬	通信工程2	91.2	75	85.2	100	93.3	166.2	278.5	91.2	0		
12	席奥宇	通信工程2	84.8	82.5	81.5	80.2	93.3	167.3	255	76	8.8		
13	徐佳田	通信工程2	76	85	92.6	93.4	79.3	161	265.3	76	0		
14	放涛	通信工程2	0	0	88.9	93.4	91.3	0	273.6	0	0		
15	汪洋瑞	通信工程2	100	85	85.2	93.4	95.3	185	273.9	91.2	8.8		
16	王龙	通信工程2	92.4	82.5	0	53.8	89.3	174.9	143.1	83.6	8.8		
17	郭佳宁	通信工程2	100	87.5	0	73.6	85	187.5	158.6	91.2	8.8		
18	张东场	通信工程2	100	85	81.5	80.2	89.3	185	251	91.2	8.8		
19	吴涛	通信工程2	92.4	90	92.5	67	95.3	182.4	254.8	83.6	8.8		
20	王孚鹏	通信工程2	100	92.5	88.9	100	95.3	192.5	284.2	91.2	8.8		
21	曾以亮	通信工程2	84.8	90	88.9	86.8	93.3	174.8	269	76	8.8		
22	张新浩	通信工程2	92.4	95	77.8	86.8	90.7	187.4	255.3	83.6	8.8		
23	岗晓政	通信工程2	0	60	92.6	60.4	91.3	60	244.3	0	0		
24	肖明豪	通信工程2	53.2	82.5	83.4	93.4	93.3	135.7	270.1	53.2	0		
25	吕玉米	通信工程2	92.4	75	0	67	70.3	167.4	137.3	83.6	8.8		
26	庞毅	通信工程2	100	90	81.5	100	91.3	190	272.8	91.2	8.8		
27	王婉婷	通信工程2	100	90	74.1	93.4	87.3	190	254.8	91.2	8.8		
28	孙聪	通信工程2	92.4	87.5	85.2	80.2	91.3	179.9	256.7	83.6	8.8		
29	张怡	通信工程2	100	80	92.6	93.4	95.3	180	281.3	91.2	8.8		
30	蒋婧	通信工程2	84.8	75	96.3	93.4	89.3	159.8	279	76	8.8		
31	刘婷	通信工程2	100	92.5	88.9	93.4	95.3	192.5	277.6	91.2	8.8		
32	樊泽珍	通信工程2	100	95	96.3	93.4	91.3	195	281	91.2	8.8		
33	常奇	通信工程2	76	95	0	86.8	93.3	171	180.1	76	0		
34	薛劲茁	通信工程2	92.4	90	0	0	97.3	182.4	97.3	83.6	8.8		
35	祁静滢	通信工程2	100	97.5	88.9	93.4	91.3	197.5	273.6	91.2	8.8		
36													
37													
38													
39	<	>	>	20201-1小1	20201-1小2	20201-3小1	20201-3小2	+					

#	A	B	C	D	E	F	G	H	I	J	K	L
1	姓名	班级	work1	work2	work3	work4	work5	平时作业	平时作业	选题 (总分)	选题 (总分)	备注
2	黄展	通信工程2	69.6	85	96.3	60.4	93.3	154.6	250	60.8	8.8	
3	曾浩	通信工程2	100	90	0	100	95.3	190	195.3	91.2	8.8	
4	杨剑峰	通信工程2	92.4	85	92.5	80.2	91.3	177.4	264	83.6	8.8	
5	于嘉豪	通信工程2	100	90	83.4	86.8	93.3	190	263.5	91.2	8.8	
6	朱阿蒙	通信工程2	69.6	95	96.3	60.4	0	164.6	156.7	60.8	8.8	
7	黄仁盛	通信工程2	65.2	90	85.1	100	95.3	155.2	280.4	60.8	4.4	
8	邱明珠	通信工程2	100	85	88.9	93.4	91.3	185	273.6	91.2	8.8	
9	颜康	通信工程2	100	90	96.3	86.8	0	190	183.1	91.2	8.8	
10	肖文奇	通信工程2	100	92.5	88.9	100	93.3	192.5	282.2	91.2	8.8	
11	刘宏宇	通信工程2	53.2	65	0	93.4	95.3	118.2	188.7	53.2	0	
12	周远阳	通信工程2	92.4	72.5	88.8	100	83.3	164.9	272.1	83.6	8.8	
13	陈俊	通信工程2	80.4	90	90	53.8	91.3	170.4	235.1	76	4.4	
14	陈小虹	通信工程2	84.8	75	61.2	40.6	85.3	159.8	187.1	76	8.8	
15	龚凌辉	通信工程2	100	95	88.9	93.4	95.3	195	277.6	91.2	8.8	
16	王杜俊	通信工程2	62	87.5	96.3	40.6	75.3	149.5	212.2	53.2	8.8	
17	杨俊杰	通信工程2	100	92.5	88.9	93.4	95.3	192.5	277.6	91.2	8.8	
18	郑燕	通信工程2	76	97.5	92.6	86.8	91.3	173.5	270.7	76	0	
19	孙琦	通信工程2	72.8	35	0	80.2	24	107.8	104.2	68.4	4.4	
20	吴佳怡	通信工程2	91.2	82.5	96.3	93.4	95.3	173.7	285	91.2	0	
21	罗艳	通信工程2	100	92.5	85.2	86.8	95.3	192.5	267.3	91.2	8.8	
22	李兆妍	通信工程2	69.6	80	70.4	86.8	91.3	149.6	248.5	60.8	8.8	
23	赵睿扬	通信工程2	100	95	0	0	85.3	195	85.3	91.2	8.8	
24	聂煜森	通信工程2	83.6	92.5	88.9	100	95.3	176.1	284.2	83.6	0	
25	张洪宁	通信工程2	92.4	92.5	79.7	93.4	87.3	184.9	260.4	83.6	8.8	
26	李星翰	通信工程2	100	92.5	88.9	100	0	192.5	188.9	91.2	8.8	
27	何勇	通信工程2	100	85	92.6	86.8	99.3	185	278.7	91.2	8.8	
28	曾佑民	通信工程2	100	92.5	88.9	100	93.3	192.5	282.2	91.2	8.8	
29	刘宁	通信工程2	92.4	92.5	85.2	80.2	95.3	184.9	260.7	83.6	8.8	
30	李端	通信工程2	92.4	92.5	81.4	100	0	184.9	181.4	83.6	8.8	
31	曾炳晖	通信工程2	100	90	88.9	93.4	97.3	190	279.6	91.2	8.8	
32	万星辰	通信工程2	0	100	77.8	86.8	83.3	100	247.9	0	0	
33	陈文翀	通信工程2	84.8	77.5	0	80.2	83.3	162.3	163.5	76	8.8	
34	龙斐	通信工程2	100	97.5	88.8	93.4	93.3	197.5	275.5	91.2	8.8	
35	曹青正	通信工程2	92.4	87.5	81.5	80.2	46	179.9	207.7	83.6	8.8	
36												
37												
38												
39	<	>	>	20201-1小1	20201-1小2	20201-3小1	20201-3小2	+				

#	A	B	C	D	E	F	G	H	I	J	K	L
1	姓名	班级	work1	work2	work3	work4	work5	平时作业	平时作业	作业总题	总成绩	备注
2	范彤	通信工程	72.8	82.5	81.5	0	91.3	155.3	172.8	68.4	4.4	
3	叶林正	通信工程	77.2	80	96.2	0	95.3	157.2	191.5	68.4	8.8	
4	吴仁杰	通信工程	77.2	77.5	88.9	80.2	93.3	154.7	262.4	68.4	8.8	
5	徐曾	通信工程	92.4	85	85.2	80.2	97.3	177.4	262.7	83.6	8.8	
6	段景涛	通信工程	92.4	82.5	81.5	93.4	95.3	174.9	270.2	83.6	8.8	
7	丁昊莹	通信工程	100	72.5	63	0	85.3	172.5	148.3	91.2	8.8	
8	颜嘉豪	通信工程	77.2	95	92.6	0	87.3	172.2	179.9	68.4	8.8	
9	吴志军	通信工程	65.2	67.5	0	0	0	132.7	0	60.8	4.4	
10	何沛伦	通信工程	100	90	88.9	93.4	95.3	190	277.6	91.2	8.8	
11	荆嘉诚	通信工程	0	90	92.5	0	93.3	90	185.8	0	0	
12	吴卓俊	通信工程	100	87.5	92.6	86.8	93.3	187.5	272.7	91.2	8.8	
13	廖瑜彦	通信工程	100	0	0	93.4	93.3	100	186.7	91.2	8.8	
14	申蕊鸣	通信工程	92.4	90	81.5	100	97.3	182.4	278.8	83.6	8.8	
15	陈博扬	通信工程	100	87.5	92.6	93.4	97.3	187.5	283.3	91.2	8.8	
16	郑博轩	通信工程	100	82.5	100	80.2	0	182.5	180.2	91.2	8.8	
17	袁文铮	通信工程	100	87.5	100	80.2	0	187.5	180.2	91.2	8.8	
18	王艺杰	通信工程	80.4	90	92.6	0	0	170.4	92.6	76	4.4	
19	徐吉民	通信工程	92.4	92.5	70.3	73.6	0	184.9	143.9	83.6	8.8	
20	吴泰洲	通信工程	100	97.5	77.8	0	93.3	197.5	171.1	91.2	8.8	
21	邹道胜	通信工程	100	92.5	81.5	93.4	97.3	192.5	272.2	91.2	8.8	
22	肖梁坤	通信工程	100	87.5	100	100	0	187.5	200	91.2	8.8	
23	张海平	通信工程	100	97.5	88.9	93.4	93.3	197.5	275.6	91.2	8.8	
24	蒋佳鹏	通信工程	69	90	0	80.2	100	159.6	180.2	60.8	8.8	
25	叶新	通信工程	59.6	90	0	0	97.3	140	97.3	45.6	4.4	
26	游安懿	通信工程	100	97.5	100	100	93.3	197.5	293.3	91.2	8.8	
27	李宇航	通信工程	95.6	92.5	88.9	93.4	88.7	188.1	271	91.2	4.4	
28	许睿轩	通信工程	100	87.5	100	80.2	91.3	187.5	271.5	91.2	8.8	
29	何泓春	通信工程	92.4	97.5	77.7	0	100	189.9	177.7	83.6	8.8	
30	谭尧	通信工程	76	87.5	100	86.8	92	163.5	278.8	76	0	
31	杨虹	通信工程	100	92.5	85.2	93.4	97.3	192.5	275.9	91.2	8.8	
32	黄耀涛	通信工程	0	85	83.4	80.2	81.3	85	244.9	0	0	
33	杨梦甜	通信工程	77.2	65	85.2	0	93.3	142.2	178.5	68.4	8.8	
34	熊章瑶	通信工程	38	65	72.3	0	71.3	103	143.6	38	0	
35	张佳丽	通信工程	92.4	90	88.9	0	0	182.4	88.9	83.6	8.8	
36												
37												
38	<	>	20201-1小1	20201-1小2	20201-3小1	20201-3小2	+					



总结

本次大作业通过两个不同领域的应用案例，全面展示了Python在数据处理和算法实现方面的强大能力。牌九游戏程序的开发过程体现了从传统游戏规则到计算机算法的转化思维，通过合理的数据结构设计和高效的算法实现，成功构建了一个功能完整的游戏系统，不仅准确实现了复杂的牌型判断逻辑，还通过优化的数据结构提升了程序的执行效率。成绩数据处理系统则展现了现代数据处理的完整流程，从多源数据的整合、清洗，到规则化计算和统计分析，能够有效解决教学管理中的实际问题。

通过这两个题目的练习，初步掌握了Python编程的核心技能，包括面向对象程序设计、数据结构选择与优化、文件处理与数据格式转换、异常处理与程序健壮性设计等多个方面。更重要的是，在项目开发过程中培养了将实际问题抽象为计算问题的能力，学会了如何通过合理的系统设计来处理复杂的业务逻辑，这些经验和技能为后续在数据科学、软件开发等领域的深入学习和实践奠定了坚实的基础。

代码实现

作业一：牌九游戏程序设计

```
import pandas as pd
import random

# 牌组编码与生成
CARD = {
    '天': {'p': 12}, '地': {'p': 2}, '人': {'p': 8}, '和': {'p': 4},
    '梅花': {'p': 10}, '长三': {'p': 6}, '板凳': {'p': 4}, '斧头': {'p': 11},
    '红头十': {'p': 10}, '高脚七': {'p': 7}, '零霖六': {'p': 6},

    '红九': {'p': 9}, '黑九': {'p': 9},
    '平八': {'p': 8}, '斜八': {'p': 8},
    '红七': {'p': 7}, '黑七': {'p': 7},
    '红五': {'p': 5}, '黑五': {'p': 5},

    '丁三': {'p': 3}, '二四': {'p': 6, 'is_wild': True}, # 丁三牌及二四牌可以对调用
}

def create_full_deck():
    """
    根据定义的牌种和标准数量，生成一个包含32张牌的完整列表作为编码。
    """
    # 成对的牌
    double_tile_names = [
        '天', '地', '人', '和', '梅花', '长三', '板凳',
        '斧头', '红头十', '高脚七', '零霖六'
    ]
    # 只有一张的牌
    single_tile_names = [
        '红九', '黑九', '平八', '斜八', '红七', '黑七', '红五', '黑五', '丁三', '二四'
    ]

    deck = []
    for name in double_tile_names:
        for _ in range(2):
            card_info = CARD[name].copy()
            card_info['name'] = name
            deck.append(card_info)

    for name in single_tile_names:
        card_info = CARD[name].copy()
        card_info['name'] = name
        deck.append(card_info)

    return deck

full_deck_encoding = create_full_deck()
```

```

df = pd.DataFrame(full_deck_encoding)
df.to_excel("paijiu_encoding.xlsx", index=False, engine='openpyxl')
df.to_csv("paijiu_encoding.csv", index=False, encoding='utf-8-sig')

# 发牌程序
def distro_cards(deck, num_players=4, cards_per_hand=2):
    shuffled_deck = deck[:]
    random.shuffle(shuffled_deck)    # 打乱牌组顺序

    # 打乱牌组之后，按照顺序分发给玩家
    player = []
    card_index = 0
    for i in range(num_players):
        hand = shuffled_deck[card_index : card_index + cards_per_hand]
        player.append(hand)
        card_index += cards_per_hand

    return player

# 对牌的种类
HAND_DEFINITIONS = [
    {'name': "至尊宝", 'cards': frozenset(['丁三', '二四'])},
    {'name': "双天", 'cards': frozenset(['天', '天'])},
    {'name': "双地", 'cards': frozenset(['地', '地'])},
    {'name': "孖人", 'cards': frozenset(['人', '人'])},
    {'name': "孖和", 'cards': frozenset(['和', '和'])},
    {'name': "孖梅", 'cards': frozenset(['梅花', '梅花'])},
    {'name': "孖长", 'cards': frozenset(['长三', '长三'])},
    {'name': "孖板凳", 'cards': frozenset(['板凳', '板凳'])},
    {'name': "孖斧头", 'cards': frozenset(['斧头', '斧头'])},
    {'name': "孖红头", 'cards': frozenset(['红头十', '红头十'])},
    {'name': "孖高脚", 'cards': frozenset(['高脚七', '高脚七'])},
    {'name': "孖零霖", 'cards': frozenset(['零霖六', '零霖六'])},
    {'name': "杂九对", 'cards': frozenset(['红九', '黑九'])},
    {'name': "杂八对", 'cards': frozenset(['平八', '斜八'])},
    {'name': "杂七对", 'cards': frozenset(['红七', '黑七'])},
    {'name': "杂五对", 'cards': frozenset(['红五', '黑五'])},
    {'name': "天王", 'cards': frozenset(['天', '红九'])},
    {'name': "天王", 'cards': frozenset(['天', '黑九'])},
    {'name': "地王", 'cards': frozenset(['地', '红九'])},
    {'name': "地王", 'cards': frozenset(['地', '黑九'])},
    {'name': "天牌八", 'cards': frozenset(['天', '平八'])},
    {'name': "天牌八", 'cards': frozenset(['天', '斜八'])},
    {'name': "地牌八", 'cards': frozenset(['地', '平八'])},
    {'name': "地牌八", 'cards': frozenset(['地', '斜八'])},
    {'name': "天高九", 'cards': frozenset(['天', '黑七'])},
    {'name': "地高九", 'cards': frozenset(['地', '高脚七'])}
]

# 定义对牌分数
PAIR_RANKING = {
    "至尊宝": 100, "双天": 99, "双地": 98, "孖人": 97, "孖和": 96,
    "孖梅": 95, "孖长": 94, "孖板凳": 93, "孖斧头": 92, "孖红头": 91,
    "孖高脚": 90, "孖零霖": 89, "杂九对": 88, "杂八对": 87,
    "杂七对": 86, "杂五对": 85, "天王": 84, "地王": 83, "天牌八": 82,
    "地牌八": 81, "天高九": 80, "地高九": 79
}

```

```

def evaluate_cards(hand):
    card1, card2 = hand[0], hand[1]
    card_names = frozenset([card1['name'], card2['name']])

    # 优先: 查阅规则书, 判断是否为特殊对牌
    for definition in HAND_DEFINITIONS:
        if card_names == definition['cards']:
            hand_name = definition['name']
            score = PAIR_RANKING.get(hand_name, 0)
            return (score, hand_name)

    # 其次: 如果不是特殊对牌, 则计算点数
    points = (card1['p'] + card2['p']) % 10

    # 仅当手中包含二四, 且不与丁三成对时, 此规则生效
    is_gee_joon = card_names == frozenset(['丁三', '二四'])
    if not is_gee_joon:
        # 检查是否有二四牌
        if card1.get('is_wild'): # '二四'
            # 计算二四当3点用时的分数
            wild_points = (3 + card2['p']) % 10
            # 取更优的结果
            if wild_points > points: points = wild_points
        elif card2.get('is_wild'): # '二四'
            wild_points = (card1['p'] + 3) % 10
            if wild_points > points: points = wild_points

    # 点数牌的分数是0-9, 远小于特殊对牌的79-100分
    return (points, f"点数 {points}")

def run_game_round():
    # 创建并洗牌
    deck = create_full_deck()

    # 发牌给4位玩家
    player_hands = distro_cards(deck, num_players=4)

    # 评估每位玩家的手牌并储存结果
    evaluations = []
    for i, hand in enumerate(player_hands):
        hand_str = f"[{hand[0]['name']}, {hand[1]['name']}]"
        rank_score, rank_name = evaluate_cards(hand)
        evaluations.append({
            'player': i + 1,
            'hand_str': hand_str,
            'score': rank_score,
            'name': rank_name
        })
        print(f"玩家 {i+1} 的手牌: {hand_str}<20 -> 牌型: {rank_name}, 分数: {rank_score}")
        print("-" * 30)

    # 找出获胜者
    winner_info = max(evaluations, key=lambda x: x['score'])

    # 检查是否有平局

```

```

max_score = winner_info['score']
winners = [p for p in evaluations if p['score'] == max_score]

if len(winners) == 1:
    winner = winners[0]
    print(f"获胜者是: 玩家 {winner['player']}")
    print(f"手牌: {winner['hand_str']} ({winner['name']})")
else:
    winner_players_str = ", ".join([f"玩家 {w['player']}" for w in winners])
    print(f"本轮为平局, 共同获胜者: {winner_players_str}")
    print(f"二者都以牌型“{winners[0]['name']}”获得最高分 {max_score}")

if __name__ == "__main__":
    run_game_round()

```

作业二：学生成绩数据处理

```

import pandas as pd
import numpy as np
from pathlib import Path
import openpyxl

class GradeProcessor:
    def __init__(self):
        """初始化成绩处理器"""
        # 定义成绩计算规则
        self.scoring_rules = {
            '平时作业1': ['work1', 'work2'], # work1和work2求和
            '平时作业2': ['work3', 'work4', 'work5'] # work3、work4、work5求和
        }

    def convert_xlsx_to_csv(self):
        """前置操作：将Excel文件转换为CSV文件"""
        try:
            # 转换ds_1_2.xlsx到CSV
            df_12 = pd.read_excel('data/ds_1_2.xlsx', sheet_name='c1_2')
            df_12.to_csv('data/ds_1_2.csv', index=False, encoding='utf-8-sig')

            # 转换ds_3_4.xlsx到CSV
            df_34 = pd.read_excel('data/ds_3_4.xlsx', sheet_name='01-绪论')
            df_34.to_csv('data/ds_3_4.csv', index=False, encoding='utf-8-sig')

            print("✓ Excel文件转换完成")
            return True

        except Exception as e:
            print(f"✗ Excel转换失败: {e}")
            return False

    def extract_student_lists_from_xlsx(self):
        """前置操作：从原始xlsx文件中提取学生名单"""
        try:
            # 读取原始的成绩记录表
            excel_file = pd.ExcelFile('data/数据结构成绩记录表(1).xlsx')

            # 定义sheet名称映射
            sheet_mapping = {

```

```

        '20201-1小1': '1小1',
        '20201-1小2': '1小2',
        '20201-3小1': '3小1',
        '20201-3小2': '3小2'
    }

    for sheet_name, group_name in sheet_mapping.items():
        if sheet_name in excel_file.sheet_names:
            # 读取sheet数据
            df = pd.read_excel('data/数据结构成绩记录表(1).xlsx',
sheet_name=sheet_name)

            # 只保留前两列（姓名和班级）
            df_output = df.iloc[:, :2]

            # 保存为CSV文件
            output_filename = f'data/{sheet_name}.csv'
            df_output.to_csv(output_filename, index=False, header=False,
encoding='utf-8-sig')
        else:
            print(f"X 未找到sheet '{sheet_name}'")

    print("✓ 学生名单提取完成")
    return True

except Exception as e:
    print(f"X 学生名单提取失败: {e}")
    return False

def read_grade_files(self):
    """读取平时成绩文件"""
    try:
        # 读取1、2班成绩
        df_12 = pd.read_csv('data/ds_1_2.csv')

        # 读取3、4班成绩
        df_34 = pd.read_csv('data/ds_3_4.csv')

        # 合并两个班级的数据
        df_combined = pd.concat([df_12, df_34], ignore_index=True)
        print(f"✓ 成绩数据读取完成: {len(df_combined)}条记录")

        return df_combined

    except FileNotFoundError as e:
        print(f"X 文件未找到: {e}")
        return None
    except Exception as e:
        print(f"X 读取文件错误: {e}")
        return None

def read_student_lists(self):
    """读取学生名单文件（添加表头）"""
    student_lists = {}

    # 定义表头
    headers = ['姓名', '班级']

```

```

try:
    # 读取四个班级的学生名单
    files = [
        ('data/20201-1小1.csv', '1小1'),
        ('data/20201-1小2.csv', '1小2'),
        ('data/20201-3小1.csv', '3小1'),
        ('data/20201-3小2.csv', '3小2')
    ]

    total_students = 0
    for file_name, group_name in files:
        df = pd.read_csv(file_name, header=None, names=headers)
        student_lists[group_name] = df
        total_students += len(df)

    print(f"✓ 学生名单读取完成: {total_students}人")
    return student_lists

except Exception as e:
    print(f"✗ 读取学生名单错误: {e}")
    return None

def calculate_scores(self, df_grades):
    """根据规则计算成绩"""
    if df_grades is None:
        return None

    # 创建结果DataFrame
    result_df = df_grades.copy()

    # 根据规则计算新的成绩列
    calculated_count = 0
    for rule_name, work_columns in self.scoring_rules.items():
        # 检查所需列是否存在
        missing_cols = [col for col in work_columns if col not in
df_grades.columns]
        if missing_cols:
            print(f"⚠ 缺少列 {missing_cols}, 跳过 {rule_name}")
            continue

        # 计算成绩（处理缺失值）
        scores = df_grades[work_columns].fillna(0).sum(axis=1)
        # 保留一位小数
        result_df[rule_name] = scores.round(1)
        calculated_count += 1

    print(f"✓ 成绩计算完成: {calculated_count}项")
    return result_df

def create_final_grade_table(self, df_calculated, student_lists):
    """创建最终的成绩记录表"""
    if df_calculated is None or student_lists is None:
        return None

    # 合并所有学生名单
    all_students = []
    for group_name, student_df in student_lists.items():
        students_with_group = student_df.copy()

```

```

        students_with_group['分组'] = group_name
        all_students.append(students_with_group)

df_all_students = pd.concat(all_students, ignore_index=True)

# 创建最终表格的列
final_columns = ['姓名', '班级', '分组', 'work1', 'work2', 'work3',
'work4', 'work5']

# 添加计算出的成绩列
for rule_name in self.scoring_rules.keys():
    final_columns.append(rule_name)

# 添加其他可能需要的列
final_columns.extend(['单选题（总分）', '多选题（总分）', '备注'])

# 创建最终DataFrame
final_df = pd.DataFrame(columns=final_columns)

# 填充学生基本信息
final_df['姓名'] = df_all_students['姓名']
final_df['班级'] = df_all_students['班级']
final_df['分组'] = df_all_students['分组']

# 匹配成绩数据
matched_count = 0
for idx, student_name in enumerate(df_all_students['姓名']):
    # 在成绩数据中查找该学生
    student_grade = df_calculated[df_calculated['name'] == student_name]

    if not student_grade.empty:
        matched_count += 1
        # 填充work成绩
        for work_col in ['work1', 'work2', 'work3', 'work4', 'work5']:
            if work_col in student_grade.columns:
                # 保留一位小数
                final_df.loc[idx, work_col] =
round(float(student_grade.iloc[0][work_col]), 1) if
pd.notna(student_grade.iloc[0][work_col]) else 0.0

        # 填充计算后的成绩
        for rule_name in self.scoring_rules.keys():
            if rule_name in student_grade.columns:
                # 保留一位小数
                final_df.loc[idx, rule_name] =
round(float(student_grade.iloc[0][rule_name]), 1) if
pd.notna(student_grade.iloc[0][rule_name]) else 0.0

        # 填充其他成绩
        if '单选题（总分）' in student_grade.columns:
            final_df.loc[idx, '单选题（总分）'] =
round(float(student_grade.iloc[0]['单选题（总分）']), 1) if
pd.notna(student_grade.iloc[0]['单选题（总分）']) else 0.0
            if '多选题（总分）' in student_grade.columns:
                final_df.loc[idx, '多选题（总分）'] =
round(float(student_grade.iloc[0]['多选题（总分）']), 1) if
pd.notna(student_grade.iloc[0]['多选题（总分）']) else 0.0

```

```

print(f"✓ 成绩表创建完成: {len(df_all_students)}人, 匹配{matched_count}人")
return final_df

def save_results(self, final_df, csv_filename='data/数据结构成绩记录表_完整.csv', xlsx_filename='data/数据结构成绩记录表_最终.xlsx'):
    """保存最终结果: 先保存完整CSV, 再按分组保存到Excel的不同sheet"""
    if final_df is None:
        print("X 没有数据可保存")
        return False

    try:
        numeric_columns = ['work1', 'work2', 'work3', 'work4', 'work5', '平时作业1', '平时作业2', '单选题(总分)', '多选题(总分)']
        for col in numeric_columns:
            if col in final_df.columns:
                final_df[col] = pd.to_numeric(final_df[col],
errors='coerce').round(1)

        # 第一步: 保存完整的CSV文件
        final_df.to_csv(csv_filename, index=False, encoding='utf-8-sig')

        # 第二步: 按分组保存到Excel的不同sheet, 并添加总结sheet
        with pd.ExcelWriter(xlsx_filename, engine='openpyxl') as writer:
            # 按分组分别保存到不同的sheet
            groups = final_df['分组'].unique()

            for group in groups:
                # 筛选当前分组的数据
                group_df = final_df[final_df['分组'] == group].copy()

                # 删除分组列(因为每个sheet代表一个分组)
                group_df_output = group_df.drop('分组', axis=1)

                # 重新排序列, 确保逻辑顺序
                output_columns = ['姓名', '班级', 'work1', 'work2', 'work3',
'work4', 'work5']

                # 添加计算出的成绩列
                for rule_name in self.scoring_rules.keys():
                    if rule_name in group_df_output.columns:
                        output_columns.append(rule_name)

                # 添加其他列
                remaining_columns = [col for col in group_df_output.columns
if col not in output_columns]
                output_columns.extend(remaining_columns)

                # 重新排序DataFrame
                group_df_output = group_df_output[output_columns]

                # 保存到对应的sheet, 使用分组名作为sheet名
                sheet_name = f"20201-{group}"
                group_df_output.to_excel(writer, sheet_name=sheet_name,
index=False)

            print(f"✓ 结果保存完成: {len(final_df)}条记录, {len(final_df['分
组'].unique())}个分组")
            return True
    
```

```
except Exception as e:
    print(f"X 保存失败: {e}")
    return False

def display_rules(self):
    """显示当前的计算规则"""
    print("计算规则: ", end="")
    rules = []
    for rule_name, work_columns in self.scoring_rules.items():
        rules.append(f"{rule_name}({'+'.join(work_columns)})")
    print(" | ".join(rules))

def process_all(self):
    """执行完整的处理流程"""
    print("数据结构成绩处理开始...")

    # 前置操作1: 转换Excel文件为CSV
    if not self.convert_xlsx_to_csv():
        return False

    # 前置操作2: 提取学生名单
    if not self.extract_student_lists_from_xlsx():
        return False

    # 显示计算规则
    self.display_rules()

    # 读取成绩数据
    df_grades = self.read_grade_files()
    if df_grades is None:
        return False

    # 读取学生名单
    student_lists = self.read_student_lists()
    if student_lists is None:
        return False

    # 计算成绩
    df_calculated = self.calculate_scores(df_grades)
    if df_calculated is None:
        return False

    # 创建最终表格
    final_df = self.create_final_grade_table(df_calculated, student_lists)
    if final_df is None:
        return False

    # 保存结果
    success = self.save_results(final_df)

    if success:
        print("✓ 处理完成! 输出文件: ")
        print(" - 数据结构成绩记录表_完整.csv")
        print(" - 数据结构成绩记录表_最终.xlsx")

    return success
```

```
# 使用示例
if __name__ == "__main__":
    # 创建处理器实例
    processor = GradeProcessor()

    # 执行完整处理流程
    success = processor.process_all()

    # 如果处理成功，询问是否生成图表
    if success:
        try:
            user_input = input("\n是否生成可视化图表? (y/n): ").strip().lower()
            if user_input in ['y', 'yes', '是']:
                from grade_visualizer import GradeVisualizer

                visualizer = GradeVisualizer()
                if visualizer.load_data():
                    print("正在生成图表...")
                    visualizer.generate_all_charts()
                    visualizer.create_summary_report()
                    print("✓ 图表生成完成! 请查看 charts/ 目录")
        except Exception as e:
            print(f"生成图表时出错: {e}")
```