		第一题	第二题	第三题	第四题	sum
何宇迪	1	11	21	19	29	80
刘昱君	1	11	21	24	26	82
吴天行	1	7	21	19	24	71
和丽兴	1	7	21	22	28	78
塔巴江村						未交
崔杰	1	7	16	22	20	65
庄嘉帆	1	7	23	21	23	74
张浩祖	1	9	21	19	25	74
张露丹	1	9	21	24	22	76
施習						未交
李上卫						未交
李俊杰	1	0	23	30	18	71
杨晨晨	1	13	22	19	26	80
杨翔	1	3	24	22	18	67
王丽媛	1	11	23	19	26	79
王乐						未交
王康宇	1	14	21	20	25	80
王立弘	1	10	21	30	26	87
白天琪	1	10	23	24	27	84
罗昊然	1	9	21	19	24	73
谢皓椿	1	10	21	17	21	69
贺馨姜艾	1	9	22	19	24	74
达尔汗	1	9	23	20	26	78
郭嘉						未交
郭欣遥	1	10	23	21	28	82
依夏·克热木江						未交
陈琰	1	10	22	26	24	82
韦淑荣	1	5	21	24	28	78
马宵	1	14	20	26	18	78
马月璐	1	11	21	19	25	76
黎小源	1	10	22	19	19	70
龚卓能	1	9	22	20	26	77
龚新宇	1	9	21	19	26	75
	平均分	9.04	21.52	21.59	24.15	76.30

第一题

> 评分标准：运行两个算例： [5, 7, 11, 16, 6, 9, 10, 20, 15, 13] 和 [5,6,7,8]
> 正确答案为 77 和 19
> 满分 15 分，每个算例运行报错 -6； 死循环或时间复杂度过高 -5
> 结果小于标准答案 -4
> 结果在集合（78,79,80,81） 和 （19,20,21）中： -1
> 结果在区间 （82，99）和 （22，30）： -2
> 结果在其他范围 -3
> 有详细注释或方案输出：+2 上限 15

1|1 何宇迪 第一题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 21:35:52 2024
@author: Stu
"""
def ferry_time(times):
    times.sort()
    n = len(times)
    total_time = 0
    # 5要用4次，先完成这部分
    total_time += 4 * times[0]
    # 选择五个加和最小的较大数（除了5之外），数字不能重复选
    remaining_times = times[1:]
    selected_times = []
    for _ in range(5):
        min_sum = float('inf')
        selected_pair = None
        for i in range(len(remaining_times)):
            for j in range(i + 1, len(remaining_times)):
                pair_sum = remaining_times[i] + remaining_times[j]
                if pair_sum < min_sum:
                    min_sum = pair_sum
                    selected_pair = (i, j)
        if selected_pair is not None:
            selected_times.append(max(remaining_times[selected_pair
[0]], remaining_times[selected_pair[1]]))
            # 移除已选的两个数，避免重复选择
            remaining_times.pop(max(selected_pair[0], selected_pair
[1]))
            remaining_times.pop(min(selected_pair[0], selected_pair
```

```
[1]))
    total_time += sum(selected_times)
    return total_time
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
print(ferry_time(times))
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 66
[5,6,7,8] -> 27

2|2 刘昱君 第一题

```
#第一题
def minimum_crossing_time_with_plan(times):
    # Step 1: 将所有渡河时间排序
    times.sort()
    total_time = 0  # 初始化总时间
    plans = []  # 记录渡河方案
    # Step 2: 开始循环处理，当人数大于3时按照策略操作
    while len(times) > 3:
        # 方案一：最快的两人来回帮助最慢的人
        option1 = times[1] + times[0] + times[-1] + times[1]
        # 方案二：最快的人多次协助最慢的两人
        option2 = times[-1] + times[0] + times[-2] + times[0]
        # 选出最优方案，并移除已到对岸的人
        if option1 < option2:
            # 选择方案一
            total_time += option1
            plans.append(f"{times[0]} 和 {times[1]} 过河 -> 用时 {times[1]}")
            plans.append(f"{times[0]} 返回 -> 用时 {times[0]}")
            plans.append(f"{times[-2]} 和 {times[-1]} 过河 -> 用时 {times[-1]}")
            plans.append(f"{times[1]} 返回 -> 用时 {times[1]}")
            # 移除最慢的两人
            times = times[:-2]
        else:
            # 选择方案二
            total_time += option2
            plans.append(f"{times[0]} 和 {times[-1]} 过河 -> 用时 {times[-1]}")
            plans.append(f"{times[0]} 返回 -> 用时 {times[0]}")
            plans.append(f"{times[0]} 和 {times[-2]} 过河 -> 用时 {times[-2]}")
            plans.append(f"{times[0]} 返回 -> 用时 {times[0]}")
            # 移除最慢的两人
            times = times[:-2]
```

```
    # Step 3: 当剩余人数小于等于3时直接处理
    if len(times) == 3:
        # 三人同时过河
        total_time += times[2]
        plans.append(f"{times[0]}、{times[1]} 和 {times[2]} 过河 -> 用时 {times[2]}")
    elif len(times) == 2:
        # 两人一起过河
        total_time += times[1]
        plans.append(f"{times[0]} 和 {times[1]} 过河 -> 用时 {times[1]}")
    elif len(times) == 1:
        # 一人过河
        total_time += times[0]
        plans.append(f"{times[0]} 过河 -> 用时 {times[0]}")
    return total_time, plans
# 测试用例：村民的渡河时间
villagers_times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 调用函数计算最小渡河时间和方案
min_time, crossing_plan = minimum_crossing_time_with_plan(villagers_times)
# 输出结果
print("最少需要的渡河时间为：", min_time)
print("详细渡河方案为：")
for step in crossing_plan:
    print(step)
```

输出：
详细渡河方案为：
5 和 6 过河 -> 用时 6
5 返回 -> 用时 5
16 和 20 过河 -> 用时 20
6 返回 -> 用时 6
5 和 6 过河 -> 用时 6
5 返回 -> 用时 5
13 和 15 过河 -> 用时 15
6 返回 -> 用时 6
5 和 6 过河 -> 用时 6
5 返回 -> 用时 5
10 和 11 过河 -> 用时 11
6 返回 -> 用时 6
5 和 9 过河 -> 用时 9
5 返回 -> 用时 5
5 和 7 过河 -> 用时 7

5 返回 -> 用时 5
5 和 6 过河 -> 用时 6

[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 129
[5,6,7,8] -> 31

3|3 吴天行 第一题

```python
from itertools import combinations
# 定义村民的渡河时间列表，对应1 - 10号村民
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
n = len(times)
# 创建一个字典用于存储动态规划的中间结果，键为状态，值为对应状态下的最少渡河时间
dp = {}
# 定义一个函数来获取当前状态下的最少渡河时间
def min_time(state):
    if state in dp:
        return dp[state]
    # 将状态转换为集合，方便后续操作
    left_people = set([i for i in range(n) if (state >> i) & 1])
    if not left_people:
        return 0
    min_time_value = float('inf')
    # 枚举从左边（牛市）送到右边（河对岸）的人员组合，人数从1到3人
    for r in range(1, min(4, len(left_people) + 1)):
        for combination in combinations(left_people, r):
            right_people = left_people - set(combination)
            # 计算本次送过去花费的时间，以组合中最慢的人时间为准
            current_time = max([times[i] for i in combination])
            # 递归计算将剩下的人送过去的最少时间（假设已经把这组人送过去了）
            remaining_time = min_time(sum(1 << i for i in right_people))
            # 从对岸选最快的人送回（这里简单模拟，通过找剩余人员中最快的）
            if right_people:
                back_time = min([times[i] for i in right_people])
                # 计算总时间
                total_time = current_time + remaining_time + back_time
            else:
                total_time = current_time + remaining_time
            # 更新最少时间
            min_time_value = min(min_time_value, total_time)
    dp[state] = min_time_value
    return min_time_value
# 初始状态下所有人都在左边（牛市），对应状态为所有位都为1的二进制数
initial_state = (1 << n) - 1
print(min_time(initial_state))
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 62
[5,6,7,8] -> 18

4|4 和丽兴 第一题

```
def find_min_time(times):
    times.sort()  # 将渡河时间从小到大排序
    memo = {}
    def helper(people):
        # 如果只有一个人，直接返回其时间
        if len(people) == 1:
            return people[0]
        # 如果没有人，返回 0
        elif len(people) == 0:
            return 0
        people_tuple = tuple(people)
        if people_tuple in memo:
            return memo[people_tuple]
        # 用于存储可能的最小时间
        min_time = float('inf')
        # 考虑两种主要策略：
        # 1. 让最轻的两个人过河
        if len(people) >= 2:
            first = people[0]
            second = people[1]
            # 只有两个人时
            if len(people) == 2:
                time = max(first, second) + helper([])  # 直接返回
            else:
                # 两人过，那个人回来
                time = max(first, second) + helper(people[2:])  # 去掉前两个
            min_time = min(min_time, time)
        # 2. 让最轻和最重的两人组合过河
        if len(people) >= 3:
            first = people[0]
            second = people[1]
            third = people[2]
            # 让最重的两人过，返回最轻的
            time = max(third, second) + first + helper(people[3:])
            min_time = min(min_time, time)
        memo[people_tuple] = min_time
        return min_time
    return helper(times)
# 村民的渡河时间
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 计算最小时间
```

```
min_time = find_min_time(times)
print(f"最少花费时间: {min_time}")
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 61
[5,6,7,8] -> 14

5|5 塔巴江村 第一题

6|6 崔杰 第一题

```
#整体思路:
#通过循环来逐步安排村民过河，每次根据当前剩余村民的人数情况，选择合适的策略让一部分村民过河，同时记录累计花费的最少时间，直到所有村民都过河为止。
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
n = len(times)
min_time = 0
while n > 0:
    if n == 1:
        min_time += times[0]
        break
    elif n == 2:
        min_time += max(times[0], times[1])
        break
    elif n == 3:
        min_time += max(times[0], times[1], times[2])
        break
    else:
        # 每次选择最快的3个人组合过河（这里假设取前3个元素模拟每次选择相对快的3人）
        group = times[:3]
        min_time += max(group)
        times = times[3:]
        n -= 3
print(min_time)
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 60
[5,6,7,8] -> 15

7|7 庄嘉帆 第一题

```
def shortest_time_to_cross(people):
    people.sort()  # 将渡船时间从小到大排序
    total_time = 0  # 总过河时间
```

```
    left = 0  # 左岸人数
    right = len(people) - 1  # 右岸人数
    while left < right:
        # 当左岸人数小于等于2时，直接让剩下的人都过河
        if left + 2 >= right:
            total_time += people[right]
            break
        # 贪心策略：让渡船时间最长的两个人先过河
        total_time += people[right]  # 两个人过河
        right -= 2
        # 如果还有人没过河，让渡船时间最短的人把船划回来
        if left < right:
            total_time += people[left]
            left += 1
    return total_time
# 每个人渡船时间列表
people_times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 计算所需的最短时间
print(shortest_time_to_cross(people_times))

```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] ->64
[5,6,7,8] -> 13

8|8 张浩祖 第一题

```
def river_crossing(times):
    times.sort()
    n = len(times)
    if n == 1:
        return times[0]
    elif n == 2:
        return times[1]
    elif n == 3:
        return times[2]
    else:
        total_time = 0
        while n > 3:
            option1 = times[0] + times[1] + times[n - 1] + times[1]
            option2 = times[0] + times[n - 2] + times[0] + times[n -
1]
            total_time += min(option1, option2)
            n -= 2
        if n == 3:
            total_time += times[2]
        elif n == 2:

```

```
            total_time += times[1]
        else:
            total_time += times[0]
        return total_time
if __name__ == "__main__":
    times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
    print(river_crossing(times))
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 129
[5,6,7,8] -> 31

9|9 张露丹 第一题

```
import sys
# 初始化输入数据
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
N = len(times)
# 对渡河时间进行排序，以便每次让最慢的人过河，最快的人返回
times.sort()
# 初始化变量
total_time = 0
current_time = 0
# 每次最多载3人，但需要保证有人能回来划船
for i in range(N):
    if i % 3 == 2:  # 每3个人过河后，最快的返回
        total_time += max(times[i - 2:i + 1])  # 加上这3人中最慢的
        current_time = times[i]  # 最快的返回
    elif i == N - 1 or (i + 1) % 3 == 0:  # 最后一个人过河或者3的倍数人过河
        total_time += max(times[i - (i % 3):i + 1])  # 加上最后这组人中最慢的
# 如果最后一组人数不足3人，但有人需要返回，则加上返回时间
if N % 3 != 0:
    total_time += current_time  # 最后一个最快的返回的人的时间不再增加，因为已经在计算内了
print("最少需要花费的时间为:", total_time)
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 70
[5,6,7,8] -> 22

10|10 施习 第一题

11|11 李上卫 第一题

12|12 李俊杰 第一题

13|13 杨晨晨 第一题

```
def river_crossing(times):
    times.sort()
    n = len(times)
    plan = []
    if n == 1:
        return times[0], plan
    elif n == 2:
        return max(times), plan
    elif n == 3:
        return max(times), plan
    else:
        min_total_time = float('inf')
        best_plan = []
        # 尝试所有可能的第一次送两人过河的组合情况
        for left in range(1, n - 1):
            for right in range(left + 1, n):
                # 方案一：先送两人过去（left和right位置对应的两人），然后让最快的回来
                plan_1_time = max(times[left], times[right]) + times[0]
                rest_people = times[1:left] + times[left + 1:right] + times[right + 1:]
                # 对剩下的人继续安排过河，递归调用函数
                sub_time_1, sub_plan_1 = river_crossing(rest_people)
                plan_1_time += sub_time_1
                this_plan_1 = [("方案一", (left, right), (times[left], times[right]), (0,), (times[0],), tuple([x for x in sub_plan_1]))]
                # 方案二：先送最快的和次快的过去，然后让最快的回来
                plan_2_time = max(times[0], times[1]) + times[0]
                rest_people_2 = times[2:]
                sub_time_2, sub_plan_2 = river_crossing(rest_people_2)
                plan_2_time += sub_time_2
                this_plan_2 = [("方案二", (0, 1), (times[0], times[1]), (0,), (times[0],), tuple([x for x in sub_plan_2]))]
                # 比较当前两种方案时间，选择更优的
                if plan_1_time < plan_2_time:
                    cur_plan = this_plan_1
                    cur_time = plan_1_time
                else:
```

```
                    cur_plan = this_plan_2
                    cur_time = plan_2_time
                if cur_time < min_total_time:
                    min_total_time = cur_time
                    best_plan = cur_plan
        return min_total_time, best_plan
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
result, plan_list = river_crossing(times)
print("最少花费时间:", result)
for idx, (scheme, first_go_indexes, first_go_times, first_back_index, first_back_time, sub_plan) in enumerate(plan_list):
    print(f"第{idx + 1}步:")
    print(f"选择 {scheme}")
    print(f"第一次送过去的人员索引为 {first_go_indexes}，渡河时间为 {first_go_times}")
    print(f"第一次返回的人员索引为 {first_back_index}，渡河时间为 {first_back_time}")
    if sub_plan:
        for sub_idx, (sub_scheme, sub_first_go_indexes, sub_first_go_times, sub_first_back_index, sub_first_back_time, _) in enumerate(sub_plan):
            print(f"  子步骤{sub_idx + 1}:")
            print(f"    选择 {sub_scheme}")
            print(f"    第一次送过去的人员索引为 {sub_first_go_indexes}，渡河时间为 {sub_first_go_times}")
            print(f"    第一次返回的人员索引为 {sub_first_back_index}，渡河时间为 {sub_first_back_time}")
```

输出:
第1步:
选择 方案二
第一次送过去的人员索引为 (0, 1)，渡河时间为 (5, 6)
第一次返回的人员索引为 (0,)，渡河时间为 (5,)
子步骤1:
选择 方案二
第一次送过去的人员索引为 (0, 1)，渡河时间为 (7, 9)
第一次返回的人员索引为 (0,)，渡河时间为 (7,)
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 70
[5,6,7,8] -> 19

14|14 杨翔 第一题

```
def min_crossing_time(times):
    times.sort()
    n = len(times)
```

```
    total_time = 0
    across = 0
    while across < n:
        if across < 2 and (n - across) > 0:
            slowest = times[across + (n - across - 1 if across + 2 <
= n else 1)]
            total_time += slowest
            across += 2
            across -= 1
        elif across == 0 or (across == 1 and (n - across) > 0):
            slowest = times[across + (n - across if across + 1 <= n
 else 0)]
            total_time += slowest * 2
            if across == 0:
                across += 1
    return total_time
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
print(min_crossing_time(times))
```

输出：死循环

15|15 王丽媛 第一题

```
# 1
def min_time_to_cross(T):
    N = len(T)
    T.sort()
    time = 0
    i = 0
    while i < N:
        if N - i <= 3:
            time += T[-1]
            break
        if N - i == 4:
            time += T[2] + T[3]
            time += T[0] + T[3]
            break
        if i + 3 < N and T[i] + T[i+1] + T[i+2] > T[i+2] + T[i+3] +
 T[0]:
            time += T[i+2] + T[i+3]
            time += T[0]
            i += 3
        else:
            time += T[i] + T[i+1] + T[i+2]
            time += T[i]
            i += 3
    return time

```

```
T = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
print(min_time_to_cross(T))
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 82
[5,6,7,8] -> 28

16|16 王乐 第一题

17|17 王康宇 第一题

```
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
timelist = times
list.sort(times)
daoan = []
timetotal = 0
a = 0
def lunci_one(times,daoan):
    if len(times)>2:
        num = times[:3]
        times = times[3:]
        daoan+=num
        time = max(num) + min(daoan)
        if len(times) == 0:
            pass
        else:
            times.append(min(daoan))
            daoan.remove(min(daoan))
    else:
        num = times
        time = max(num)
        times = []
    list.sort(times)
    list.sort(daoan)
    print(time, times, daoan)
    return time,times,daoan
def lunci_two(times,daoan):
    if len(times)>2:
        num = times[-3:]
        times = times[:-3]
        daoan+=num
        time = max(num) + min(daoan)
        if len(times) == 0:
            pass
        else:
            times.append(min(daoan))
            daoan.remove(min(daoan))
```

```
    else:
        num = times
        daoan += num
        time = max(num)
        times = []
        pass
    list.sort(times)
    list.sort(daoan)
    print(time, times, daoan)
    return time, times,daoan
ttt = 0
for i in range(1000):
    if i%2==0:
        a,times,daoan = lunci_one(times,daoan)
        ttt += a
        if len(times) == 0:
            break
    if i%2!=0:
        a, times,daoan = lunci_two(times, daoan)
        ttt += a
        if len(times) == 0:
            break
    print(a)
print(f"最终花费时间{ttt}")
```

输出:

12 [5, 9, 10, 11, 13, 15, 16, 20] [6, 7]

12

26 [5, 6, 9, 10, 11, 13] [7, 15, 16, 20]

26

14 [5, 10, 11, 13] [6, 7, 9, 15, 16, 20]

14

19 [5, 6] [7, 9, 10, 11, 13, 15, 16, 20]

19

6 [] [7, 9, 10, 11, 13, 15, 16, 20]

最终花费时间77

[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 77

[5,6,7,8] -> 20

18|18 王立弘 第一题

```
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
times.sort()
total_time = 0
n = len(times)
while n > 3:
```

```
    # 每次选择最快的和两个最慢的过去，花费时间是两个最慢的人中较慢的那个时间
    go_time = max(times[-2:])
    # 最快的回来，花费最快的那个人的时间
    back_time = times[0]
    total_time += go_time + back_time
    n -= 3
if n == 3:
    total_time += max(times[:3])
elif n == 2:
    total_time += max(times[:2])
elif n == 1:
    total_time += times[0]
print(total_time)
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 80
[5,6,7,8] -> 18

19|19 白天琪 第一题

```
import itertools
# 村民的渡河时间（10个村民）
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 计算从一组村民中渡河的时间（最慢的人的时间）
def calculate_time(group):
    return max(group)
# 递归函数来计算最小总时间，并列举所有情况
def min_travel_time(times):
    # 记录最小时间，初始设置为一个很大的值
    min_time = float('inf')
    all_scenarios = []  # 用于记录所有的过河情况
    # 使用递归来枚举所有的可能性
    def recurse(left_times, right_times, total_time, scenario):
        nonlocal min_time, all_scenarios
        # 如果所有村民都过河了，更新最小时间并记录情况
        if not left_times:
            min_time = min(min_time, total_time)
            all_scenarios.append((list(scenario), total_time))
            return
        # 枚举所有可能的组合，最多3个村民一起过河
        for num_people in range(2, 4):  # 2到3个人一起过河
            for group in itertools.combinations(left_times, num_peop
le):
                # 计算这组人的渡河时间
                group_time = calculate_time(group)
                # 计算过河后的时间
                new_total_time = total_time + group_time
```

```
                # 过河后，有人必须返回，返回的人是最慢的那个
                if right_times:  # 如果已经有过河的人
                    # 从已过河的村民中选择最慢的人带船回来
                    for return_person in sorted(right_times):  # 排序保证选择时间最短的返回
                        return_time = return_person
                        new_total_time_with_return = new_total_time + return_time
                        # 递归，继续考虑剩下的村民
                        new_scenario = scenario + [group]
                        recurse(
                            [time for time in left_times if time not in group],  # 移除已过河的村民
                            right_times + list(group),  # 记录已过河的村民
                            new_total_time_with_return,  # 累加总时间
                            new_scenario  # 更新当前过河情况
                        )
                else:  # 如果没有过河的人，就直接过去
                    new_scenario = scenario + [group]
                    recurse(
                        [time for time in left_times if time not in group],  # 移除已过河的村民
                        right_times + list(group),  # 记录已过河的村民
                        new_total_time,  # 不需要返回的情况
                        new_scenario  # 更新当前过河情况
                    )
    # 开始递归计算最小总时间
    recurse(times, [], 0, [])
    # 输出所有的过河情况及其时间
    print("所有过河情况和对应的时间：")
    for scenario, total_time in all_scenarios:
        print(f"情况: {scenario} => 总时间: {total_time}")
    return min_time
# 最短的时间
result = min_travel_time(times)
print(f"最少的总时间是: {result}")
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 时间复杂度太高，无法运行
[5,6,7,8] -> 19

20|20 罗昊然 第一题

```
def calculate_time(crossing_times):
    crossing_times.sort()
```

```
    total_time = 0
    # 第一次过河
    first_crossing_time = max(crossing_times[0], crossing_times[1], crossing_times[2])
    total_time += first_crossing_time
    total_time += crossing_times[0]  # 5回来
    # 第二次过河
    second_crossing_time = max(crossing_times[7], crossing_times[8], crossing_times[9])
    total_time += second_crossing_time
    total_time += crossing_times[1]  # 6回来
    # 第三次过河
    third_crossing_time = max(crossing_times[0], crossing_times[1], crossing_times[3])
    total_time += third_crossing_time
    total_time += crossing_times[0]  # 5回来
    # 第四次过河
    fourth_crossing_time = max(crossing_times[4], crossing_times[5], crossing_times[6])
    total_time += fourth_crossing_time
    total_time += crossing_times[1]  # 6回来
    # 第五次过河
    fifth_crossing_time = max(crossing_times[0], crossing_times[1])
    total_time += fifth_crossing_time
    return total_time
# 示例渡河时间
crossing_times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
total_time = calculate_time(crossing_times)
print("最少花费的时间为:", total_time)
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 77
[5,6,7,8] -> 报错，无结果

21|21 谢皓椿 第一题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 21:21:37 2024
@author: Stu
"""
def min_crossing_time(people_times):
    """
    计算多人过河的最短时间，采用新策略：最快三个过去，最快一个回来，最慢三个过去，最快一个又回来，重复。
    :param people_times: list of int, 每个人的过河时间（单位：分钟或其他时间单位）。
```

```
    :return: int, 最短过河时间（单位与输入相同）。
    """
    # 对过河时间进行排序，并同时记录每个人的索引，以便后续按时间顺序选择人
    sorted_times_with_indices = sorted((time, index) for index, time
in enumerate(people_times))
    n = len(people_times)
    total_time = 0
    # 初始化两个列表来跟踪对岸和起始岸的人
    across_river = []  # 对岸的人（按过河时间排序）
    start_side = sorted_times_with_indices  # 起始岸的人（按过河时间排
序）
    while start_side:
        # 每次选择最快的三个人过去
        fastest_three = start_side[:3]
        start_side = start_side[3:]
        # 更新对岸的人列表
        across_river.extend(fastest_three)
        # 计算这次过河的时间（由最慢的那个人决定）
        slowest_in_group = max(time for time, _ in fastest_three)
        total_time += slowest_in_group
        # 如果还有人在起始岸，让对岸最快的那个人返回
        if start_side:
            # 找到对岸最快的人（即之前过河且现在在对岸的人中时间最短的那个）
            fastest_across = min(time for time, _ in across_river)
            fastest_return_time, _ = next((time_index for time_index
in across_river if time_index[0] == fastest_across))
            total_time += fastest_return_time
    return total_time
# 示例输入：每个人的过河时间（分钟）
people_times = [5,6,7,9,10,11,13,15,16,20]
print("最短过河时间:", min_crossing_time(people_times))
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 69
[5,6,7,8] -> 20

22|22 贺馨姜艾 第一题

```
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
times.sort()
n = len(times)
if n > 3:
    sum_time = 0
    while n > 3:
        option1 = times[0] + times[-1] + times[0] + times[-2]
        option2 = times[1] + times[0] + times[-1] + times[1]
        if option1 < option2:
```

```
            sum_time += option1
        else:
            sum_time += option2
        n -= 2
    if n == 3:
        sum_time += times[2]
    elif n == 2:
        sum_time += times[1]
    else:
        sum_time += times[0]
else:
    sum_time = times[-1]
print("最少花费", sum_time, "，才能使所有人都过河")
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 154
[5,6,7,8] -> 31

23|23 达尔汗 第一题

```
# 定义村民的渡河时间列表
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 初始化总时间
total_time = 0
# 第一次渡河
# 5、6、7过河，花费时间取三人中最大的，然后5回来
total_time += max(times[0], times[1], times[4])
total_time += times[0]
# 第二次渡河
# 15、16、20过河，花费时间取三人中最大的，然后6回来
total_time += max(times[8], times[5], times[7])
total_time += times[4]
# 第三次渡河
# 5、6、9过河，花费时间取三人中最大的，然后5回来
total_time += max(times[0], times[4], times[5])
total_time += times[0]
# 第四次渡河
# 10、11、13过河，花费时间取三人中最大的，然后6回来
total_time += max(times[6], times[2], times[9])
total_time += times[4]
# 第五次渡河
# 5、6过河，花费时间取两人中最大的
total_time += max(times[0], times[4])
print("最少花费的总时间为:", total_time)
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 77
[5,6,7,8] -> 报错，无结果

24|24 郭嘉 第一题

25|25 郭欣遥 第一题

```
(1)
# 定义渡河时间列表
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 对渡河时间列表进行排序，方便后续处理
times.sort()
n = len(times)
# 初始化总时间
total_time = 0
# 当还有超过3个人没过河时
while n > 3:
    # 选择最快的两个人先过去，然后最快的回来
    option1 = times[0] + times[1] + times[n - 1]
    # 选择最快的和最慢的过去，然后最快的回来，再选两个快的过去，最慢的回来
    option2 = 2 * times[0] + times[n - 2] + times[n - 1]
    # 选择时间花费较少的方案
    total_time += min(option1, option2)
    n -= 2
# 处理剩下3个人或者2个人或者1个人的情况
if n == 3:
    total_time += times[2]
elif n == 2:
    total_time += times[1]
else:
    total_time += times[0]
print(total_time)
#解题思路
# 整体思路
#本题采用贪心算法来求解最少渡河时间，贪心算法的核心是在每一步选择中都选择当前状态下看似最优的选项，希望通过一系列的局部最优选择能够得到全局最优解.
# 具体步骤
#(1)数据预处理**：首先将所有人的渡河时间进行排序，这样可以方便后续按照时间长短来安排渡河策略，排序后的时间列表记为 `times`，人数记为 `n` 。
#(2)多人渡河策略**：当还有超过3个人没过河时，有两种主要的渡河方案可供选择，并从中选取时间花费较少的方案。
# 一、方案一**：选择最快的两个人先过去，然后最快的人回来。此方案花费的时间为 `times[0] + times[1] + times[n - 1]` 。这样做的目的是先利用最快的两人将船送到对岸，然后让最快的人把船带回来，以便后续人员继续渡河，其中 `times[0]` 和 `times[1]` 分别是最快的两个人渡河时间，`times[n - 1]` 是最后一个人渡河时间 。
##二、方案二**：选择最快的和最慢的过去，然后最快的回来，再选两个快的过去，最慢的
```

```
回来。该方案花费的时间为 `2 * times[0] + times[n - 2] + times[n - 1]` 。这种方案是先让最快的人和最慢的人一起过河，然后最快的人回来，接着让最快的人和次慢的人过去，再让最快的人回来，以便继续运送剩余人员 。
#更新人数和总时间：每次选择完方案后，将人数 `n` 减去2，表示已经有两人成功渡河，同时将所选方案的时间加到总时间 `total_time` 中。
#处理剩余人员：当剩下3个人、2个人或者1个人时，分别进行处理。
   # - 如果剩下3个人，那么总时间加上三人中最慢的人的时间，即 `times[2]` ，因为此时可以让最快的人先送一个人过去，然后再和剩下的那个人一起过去，总时间取决于最慢的那个人的渡河时间 。
   # - 如果剩下2个人，总时间加上较慢的那个人的时间，即 `times[1]` ，两人直接一起过河即可 。
   # - 如果剩下1个人，总时间加上这个人的渡河时间，即 `times[0]` 。
### 代码实现思路
#- 按照上述解题思路，使用Python语言实现了相应的代码。首先定义了渡河时间列表 `times` 并进行排序，然后通过循环和条件判断来逐步计算最少渡河时间，直到所有人都过河，最后输出总时间 `total_time` 。
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 105
[5,6,7,8] -> 25

26|26 阿依夏·克热木江 第一题

27|27 陈琰 第一题

```python
# (1)定义人员渡河时间列表
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
times.sort()
n = len(times)
def ferry_optimized_again(times):
    left_num = n
    total_time = 0
    while left_num > 3:
          option1 = times[0] + times[1] + times[left_num - 1] + times[0]
          option2 = 2 * times[0] + times[left_num - 1] + times[left_num - 2]
          if option1 < option2:
             total_time += option1
          else:
             total_time += option2
          left_num -= 2
# 简化剩余人数不同情况的处理
    remain_times = times[-left_num:]
    if left_num == 3:
       total_time += sum(remain_times)
```

```
    elif left_num == 2:
        total_time += remain_times[1]
    elif left_num == 1:
        total_time += remain_times[0]
    return total_time
print(ferry_optimized_again(times))

```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 139
[5,6,7,8] -> 32

28|28 韦淑荣 第一题

```
def min_crossing_time(times):
    # 对渡河时间进行排序
    times.sort()
    # 初始化总时间为0
    total_time = 0
    # 当还有人未过河时继续循环
    while len(times) > 3:
        # 最快的两个人先过河
        first = times.pop(0)
        second = times.pop(0)
        # 最慢的一个人返回
        third = times.pop(-1)
        # 更新总时间（最快两人过河的时间 + 最慢一人返回的时间）
        total_time += second
        # 剩下的人中再选择最快的两个过河
        fourth = times.pop(0)
        fifth = times.pop(0)
        # 更新总时间（剩下最快的两人过河的时间）
        total_time += fifth
        # 把之前返回的最慢的人放回列表
        times.append(third)
        # 重新排序剩余的人
        times.sort()
    # 最后三人一起过河
    if len(times) == 3:
        total_time += max(times)
    elif len(times) == 2:
        total_time += times[1]
    elif len(times) == 1:
        total_time += times[0]
    return total_time
# 示例数据

```

```
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
print("最少需要花费的时间是:", min_crossing_time(times))
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 61
[5,6,7,8] -> 错误，无输出

29|29 马宵 第一题

```
def min_crossing_time(times):
    times.sort()
    total_time = 0
    steps = []
    while len(times) > 3:
        fastest = times[0]
        slowest = times[-1]
        second_slowest = times[-2]
        steps.append((fastest, second_slowest, slowest))
        total_time += slowest
        steps.append((fastest,))
        total_time += fastest
        times.pop(-1)
        times.pop(-1)
    if len(times) == 3:
        steps.append((times[0], times[1]))
        total_time += times[1]
        steps.append((times[0],))
        total_time += times[0]
        steps.append((times[0], times[2]))
        total_time += times[2]
    elif len(times) == 2:
        steps.append((times[0], times[1]))
        total_time += times[1]
    elif len(times) == 1:
        steps.append((times[0],))
        total_time += times[0]
    return steps, total_time
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
steps, total_time = min_crossing_time(times)
print("最短时间过河的方案:")
for step in steps:
    print(f"村民 {step} 过河")
print("总时间:", total_time)
```

输出:
最短时间过河的方案:

村民 (5, 16, 20) 过河
村民 (5,) 过河
村民 (5, 13, 15) 过河
村民 (5,) 过河
村民 (5, 10, 11) 过河
村民 (5,) 过河
村民 (5, 7, 9) 过河
村民 (5,) 过河
村民 (5, 6) 过河
总时间: 81
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 81
[5,6,7,8] -> 19

30|30 马月璐 第一题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 20:50:02 2024
@author: Stu
"""
def river_crossing(times):
    times.sort()
    n = len(times)
    plan = []
    if n == 1:
        return times[0], plan
    elif n == 2:
        return max(times), plan
    elif n == 3:
        return max(times), plan
    else:
        min_total_time = float('inf')
        best_plan = []
        # 尝试所有可能的第一次送两人过河的组合情况
        for left in range(1, n - 1):
            for right in range(left + 1, n):
                # 方案一：先送两人过去（left和right位置对应的两人），然后让最快的回来
                plan_1_time = max(times[left], times[right]) + times[0]
                rest_people = times[1:left] + times[left + 1:right] + times[right + 1:]
                # 对剩下的人继续安排过河，递归调用函数
                sub_time_1, sub_plan_1 = river_crossing(rest_people)
                plan_1_time += sub_time_1
                this_plan_1 = [("方案一", (left, right), (times[left], times[right]), (0,), (times[0],), tuple([x for x in sub_plan_
```

```
1]))]
                # 方案二：先送最快的和次快的过去，然后让最快的回来
                plan_2_time = max(times[0], times[1]) + times[0]
                rest_people_2 = times[2:]
                sub_time_2, sub_plan_2 = river_crossing(rest_people_
2)
                plan_2_time += sub_time_2
                this_plan_2 = [("方案二", (0, 1), (times[0], times
[1]), (0,), (times[0],), tuple([x for x in sub_plan_2]))]
                # 比较当前两种方案时间，选择更优的
                if plan_1_time < plan_2_time:
                    cur_plan = this_plan_1
                    cur_time = plan_1_time
                else:
                    cur_plan = this_plan_2
                    cur_time = plan_2_time
                if cur_time < min_total_time:
                    min_total_time = cur_time
                    best_plan = cur_plan
        return min_total_time, best_plan
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
result, plan_list = river_crossing(times)
print("最少花费时间:", result)
```

输出:
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 70
[5,6,7,8] -> 19

31|31 黎小源 第一题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 19:29:31 2024
@author: Stu
"""
def min_time_to_cross(T):
    T.sort()
    N = len(T)
    total_time = 0
    i = 0
    j = N - 1
    while i < j - 1:
        total_time += T[0] + T[1] + T[0]
        i += 2
        j -= 1
    if i + 1 == j:
        total_time += T[0] + T[j] + T[j]
```

```
        else:
            total_time += T[j] * 3
        return total_time
T = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
print(min_time_to_cross(T))
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 87
[5,6,7,8] -> 37

32|32 龚卓能 第一题

```
# 定义村民的渡河时间列表
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
# 初始化总时间
total_time = 0
# 第一次渡河
# 5、6、7过河，花费时间取三人中最大的，然后5回来
total_time += max(times[0], times[1], times[4])
total_time += times[0]
# 第二次渡河
# 15、16、20过河，花费时间取三人中最大的，然后6回来
total_time += max(times[8], times[5], times[7])
total_time += times[4]
# 第三次渡河
# 5、6、9过河，花费时间取三人中最大的，然后5回来
total_time += max(times[0], times[4], times[5])
total_time += times[0]
# 第四次渡河
# 10、11、13过河，花费时间取三人中最大的，然后6回来
total_time += max(times[6], times[2], times[9])
total_time += times[4]
# 第五次渡河
# 5、6过河，花费时间取两人中最大的
total_time += max(times[0], times[4])
print("最少花费的总时间为:", total_time)
```

输出：
[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 77
[5,6,7,8] -> 报错，无输出

33|33 龚新宇 第一题

```
# -*- coding: utf-8 -*-
"""
```

```
Spyder 编辑器
这是一个临时脚本文件。
"""
def min_crossing_time(times):
    times.sort()
    n = len(times)
    total_time = 0
    crossed = 0
    left = 0
    right = n - 1
    current_boat = 2
    while crossed < n:
        if current_boat == 0 or (current_boat == 1 and left < n -
 1):
            if current_boat == 0:
                slowest = max(times[left], times[left + 1])
                left += 2
            else:
                slowest = times[left]
                left += 1
            current_boat = 2 if current_boat == 0 else 3
            total_time += slowest
        else:
            if current_boat == 2:
                fastest_return = times[left]
                left += 1
                total_time += fastest_return
                current_boat = 1
            elif current_boat == 3:
                if left < n:
                    fastest_return = times[left]
                    left += 1
                else:
                    fastest_return = float('inf')
                total_time += fastest_return
                current_boat = 1
        if current_boat == 1:
            crossed += 1
        elif current_boat == 0 and left < n:
            crossed += 1
    return total_time
times = [5, 7, 11, 16, 6, 9, 10, 20, 15, 13]
print(min_crossing_time(times))
```

输出:

[5, 7, 11, 16, 6, 9, 10, 20, 15, 13] -> 92

[5,6,7,8] -> 18

第二题

34|1 何宇迪 第二题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 19:24:50 2024
@author: Stu
"""
import mysql.connector
import csv
import os
# 连接数据库
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Qwe_1234",
    database="online_store"
)
mycursor = mydb.cursor()
# 创建orders表格
mycursor.execute('''
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
''')
# 获取当前脚本所在的目录
script_dir = os.path.dirname(os.path.abspath(__file__))
# 构建文件的绝对路径
file_path = os.path.join(script_dir, 'C://Users/Stu/Desktop/Demo_1.csv')
# 从Demo_1.csv文件中读取数据并插入到orders表格
with open(file_path, 'r', encoding='gbk') as file:
    reader = csv.reader(file)
    next(reader)  # 跳过标题行
    for row in reader:
        sql = "INSERT INTO orders (id, sku, kinds, stime, num, unit_
```

```
price, total_price, longitude, latitude) VALUES (%s, %s, %s, %s, %s,
%s, %s, %s, %s, %s)"
        mycursor.execute(sql, row)
mydb.commit()
# 输出创建表格和插入数据后的结果（展示前10条数据）
mycursor.execute("SELECT * FROM orders LIMIT 10")
results = mycursor.fetchall()
print("插入数据后的前10条订单数据：")
for row in results:
    print(row)
# 查询三同订单（同一时间、同一地点、相同商品）并统计数量
mycursor.execute('''
SELECT ANY_VALUE(kinds) AS kinds, COUNT(*) AS count
FROM orders
GROUP BY stime, longitude, latitude, sku
HAVING COUNT(*) > 1
''')
results = mycursor.fetchall()
# 统计每个种类的三同订单数量
kind_counts = {}
for row in results:
    kind = row[0]
    count = row[1]
    if kind in kind_counts:
        kind_counts[kind] += count
    else:
        kind_counts[kind] = count
# 找到数量最多的种类
max_count = 0
max_kind = ""
for kind, count in kind_counts.items():
    if count > max_count:
        max_count = count
        max_kind = kind
print("三同订单最多的商品种类kinds：", max_kind)
```

35|2 刘昱君 第二题

```
#第二题
import pymysql
import csv
import os
# 数据库连接信息
DB_HOST = "localhost"
DB_USER = "root"
DB_PASSWORD = "your_password"
DB_NAME = "online_store"
CSV_FILE_PATH = "C:/Users/Stu/Desktop/Demo_2.csv"
```

```
def create_database_and_table():
   # """创建数据库和表"""
    try:
        connection = pymysql.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD)
        cursor = connection.cursor()
        # 创建数据库
        cursor.execute("CREATE DATABASE IF NOT EXISTS online_store;")
        cursor.execute("USE online_store;")
        # 创建 orders 表
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS orders (
                id VARCHAR(20),
                sku VARCHAR(15),
                kinds VARCHAR(15),
                stime VARCHAR(15),
                num INT,
                unit_price FLOAT,
                total_price FLOAT,
                longitude FLOAT,
                latitude FLOAT
            );
        """)
        connection.commit()
        print("数据库和表创建成功！")
        cursor.close()
        connection.close()
    except Exception as e:
        print("创建数据库或表失败：", e)
def import_csv_to_database():
    #"""从 CSV 文件中读取数据并插入到数据库"""
    if not os.path.exists(CSV_FILE_PATH):  # 检查 CSV 文件是否存在
        print(f"错误：无法找到 CSV 文件 {CSV_FILE_PATH}")
        return
    try:
        connection = pymysql.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD, database=DB_NAME)
        cursor = connection.cursor()
        # 打开 CSV 文件并读取数据
        with open(CSV_FILE_PATH, 'r') as file:
            csv_reader = csv.reader(file)
            next(csv_reader)  # 跳过标题行
            for row in csv_reader:
                print("正在写入行：", row)  # 输出插入数据
                cursor.execute("""
                    INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

```
                """, row)
        connection.commit()
        print("CSV 数据成功导入到表 orders 中！")
        cursor.close()
        connection.close()
    except Exception as e:
        print("导入 CSV 数据失败：", e)
def find_max_three_same_kinds():
    #"""查找三同订单中数量最多的商品种类"""
    try:
        connection = pymysql.connect(host=DB_HOST, user=DB_USER, pas
sword=DB_PASSWORD, database=DB_NAME)
        cursor = connection.cursor()
        # 执行 SQL 查询
        query = """
            SELECT kinds, COUNT(*) AS order_count
            FROM orders
            GROUP BY kinds, stime, longitude, latitude
            ORDER BY order_count DESC
            LIMIT 1;
        """
        cursor.execute(query)
        result = cursor.fetchone()
        if result:
            kinds, order_count = result
            print(f"三同订单最多的商品种类是：{kinds}，订单数量是：{order_
count}")
        else:
            print("查询为空：没有找到符合条件的三同订单。")
        cursor.close()
        connection.close()
    except Exception as e:
        print("查询三同订单失败：", e)
if __name__ == "__main__":
    create_database_and_table()
    import_csv_to_database()
    find_max_three_same_kinds()
```

36|3 吴天行 第二题

```
import sqlite3
import csv
# 创建数据库连接
conn = sqlite3.connect('online_store.db')
cursor = conn.cursor()
# 创建orders表
cursor.execute('''
CREATE TABLE IF NOT EXISTS orders (
```

```
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
''')
# 读取Demo_1.csv文件并插入数据到orders表
with open('/mnt/Demo_1.csv', 'r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    for row in csv_reader:
        cursor.execute('''
        INSERT INTO orders (id, sku, kinds, stime, num, unit_price,
 total_price, longitude, latitude)
        VALUES (?,?,?,?,?,?,?,?,?)
        ''', row)
# 提交事务
conn.commit()
# 查找三同订单最多的商品种类
cursor.execute('''
SELECT kinds, COUNT(*) AS order_count
FROM (
    SELECT kinds, stime, longitude, latitude
    FROM orders
    GROUP BY stime, longitude, latitude, kinds
    HAVING COUNT(*) > 1
) AS subquery
GROUP BY kinds
ORDER BY order_count DESC
LIMIT 1
''')
# 获取结果
result = cursor.fetchone()
if result:
    print(f"三同订单最多的商品种类是: {result[0]}，订单数量为: {result
[1]}")
else:
    print("没有找到符合条件的三同订单")
# 关闭数据库连接
conn.close()
```

37|4 和丽兴 第二题

```sql
CREATE DATABASE online_store;
USE online_store;
CREATE TABLE orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
LOAD DATA INFILE 'C:/Users/Stu/Desktop/Demo_2.csv'
INTO TABLE orders
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES
(id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude);
SELECT * FROM orders LIMIT 10;
WITH ThreeSameOrders AS (
    -- 按照时间、地点和商品进行分组，计算每个组的订单数量
    SELECT
        order_date,
        location,
        product,
        COUNT(*) AS order_count
    FROM
        orders
    GROUP BY
        order_date,
        location,
        product
    HAVING
        COUNT(*) >= 3  -- 只选择订单数量大于等于3的组
)
-- 统计三同订单中每个商品的出现频率
SELECT
    product,
    COUNT(*) AS three_same_order_count
FROM
    ThreeSameOrders
GROUP BY
    product
ORDER BY
    three_same_order_count DESC
LIMIT 1;  -- 选出三同订单最多的商品种类
```

38|5 塔巴江村 第二题

39|6 崔杰 第二题

```
在query中的代码：
CREATE TABLE `online_store`.`orders` (
  `id` VARCHAR(20) NOT NULL,
  `sku` VARCHAR(15) NULL,
  `kinds` VARCHAR(15) NULL,
  `stime` VARCHAR(15) NULL,
  `num` INT NULL,
  `unit_price` FLOAT NULL,
  `total_price` FLOAT NULL,
  `longitude` FLOAT NULL,
  `latitude` FLOAT NULL,
  PRIMARY KEY (`id`));
在orders中的代码：
SELECT * FROM online_store.orders;
SELECT kinds, COUNT(*) AS order_count
FROM (
    SELECT kinds, stime, longitude, latitude
    FROM orders
    GROUP BY kinds, stime, longitude, latitude
    HAVING COUNT(*) > 1
) AS subquery
GROUP BY kinds
ORDER BY order_count DESC
LIMIT 1;
```

40|7 庄嘉帆 第二题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 19:04:00 2024
@author: Stu
"""
import mysql.connector
# 连接到MySQL数据库
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Qwe_1234"
)
# 创建游标对象
```

```
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE IF NOT EXISTS online_store")
mydb.commit()
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Qwe_1234",
    database="online_store"
)
mycursor = mydb.cursor()
# 创建orders表（如果不存在）
mycursor.execute("""
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
""")
# 从Demo_1.csv文件中读取数据并插入到orders表中
try:
    with open(r"C:\Users\Stu\Documents\WeChat Files\wxid_do87lzskr1zt12\FileStorage\File\2024-12\Demo_2.csv") as file:
        # 跳过标题行
        next(file)
        for line in file:
            values = line.strip().split(',')
            sql = "INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
            mycursor.execute(sql, values)
    mydb.commit()
except FileNotFoundError:
    print("文件未找到，请检查文件路径。")
# 查询三同订单最多的商品种类kinds
# 查询三同订单最多的商品种类kinds
mycursor.execute("""
SELECT subquery.kinds, COUNT(*) AS count
FROM (
    -- 子查询，先找出符合三同条件的订单组
    SELECT o1.id, o1.sku, o1.kinds, o1.stime, o1.longitude, o1.latitude,
           GROUP_CONCAT(o2.id) AS related_orders
    FROM orders o1
```

```
    JOIN orders o2 ON o1.sku = o2.sku AND o1.stime = o2.stime AND o1.longitude = o2.longitude AND o1.latitude = o2.latitude AND o1.id!= o2.id
    GROUP BY o1.id, o1.sku, o1.kinds, o1.stime, o1.longitude, o1.latitude
    HAVING COUNT(*) >= 1  -- 至少有两个订单符合三同条件（包含自身和至少一个其他订单）
) AS subquery
GROUP BY subquery.kinds
ORDER BY count DESC
LIMIT 1
""")
result = mycursor.fetchone()
if result:
    print(f"三同订单最多的商品种类是: {result[0]}，数量为: {result[1]}")
else:
    print("未找到三同订单。")
# 关闭游标和数据库连接
mycursor.close()
mydb.close()
```

41|8 张浩祖 第二题

```
#(1)
import sqlite3
import pandas as pd
# 创建SQLite数据库连接
conn = sqlite3.connect('online_store.db')
# 创建游标对象
cursor = conn.cursor()
# 创建orders表格
cursor.execute('''
    CREATE TABLE IF NOT EXISTS orders (
        id VARCHAR(20) PRIMARY KEY,
        sku VARCHAR(15),
        kinds VARCHAR(15),
        stime VARCHAR(15),
        num INT,
        unit_price FLOAT,
        total_price FLOAT,
        longitude FLOAT,
        latitude FLOAT
    )
''')
# 读取Demo_1.csv文件
df = pd.read_csv('Demo_2(1).csv')
# 将数据插入orders表格
df.to_sql('orders', conn, if_exists='replace', index=False)
```

```
# 提交事务
conn.commit()
# 查看orders表格的基本信息和前几行
print('orders表格基本信息：')
cursor.execute('PRAGMA table_info(orders)')
for column in cursor.fetchall():
    print(column)
# 查看orders表格的前几行内容
print('orders表格前几行内容：')
cursor.execute('SELECT * FROM orders LIMIT 5')
for row in cursor.fetchall():
    print(row)
# 关闭游标和连接
cursor.close()
conn.close()
#(2)
import sqlite3
# 创建SQLite数据库连接
conn = sqlite3.connect('online_store.db')
# 创建游标对象
cursor = conn.cursor()
# 查询三同订单中最多的商品种类kinds
cursor.execute('''
    SELECT kinds, COUNT(*) AS count
    FROM orders
    GROUP BY stime, longitude, latitude, kinds
    ORDER BY count DESC
    LIMIT 1
''')
# 获取查询结果
result = cursor.fetchone()
# 输出结果
if result:
    print('三同订单中最多的商品种类为：', result[0])
else:
    print('没有找到三同订单。')
# 关闭游标和连接
cursor.close()
conn.close()
```

42|9 张露丹 第二题

```
-- 创建数据库
CREATE DATABASE online_store;
-- 使用数据库
USE online_store;
-- 创建orders表
CREATE TABLE orders (
```

```
    id VARCHAR(20) PRIMARY KEY,
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
LOAD DATA LOCAL INFILE 'C:/Users/Stu/Desktop/12.20/2_Demo_2.csv'
INTO TABLE orders
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude);
WITH OrderGroups AS (
    SELECT
        kinds,
        DATE_FORMAT(STR_TO_DATE(stime, '%Y-%m-%d %H:%i:%s'), '%Y-%m-%d %H:%i') AS stime_rounded, -- 假设时间格式，并四舍五入到分钟
        ROUND(longitude, 6) AS longitude_rounded, -- 四舍五入经纬度到小数点后6位
        ROUND(latitude, 6) AS latitude_rounded,
        COUNT(*) AS order_count
    FROM
        orders
    GROUP BY
        kinds,
        stime_rounded,
        longitude_rounded,
        latitude_rounded
    HAVING
        COUNT(*) >= 3 -- 只考虑至少3个订单的组
)
SELECT
    kinds,
    MAX(order_count) AS max_order_count
FROM
    OrderGroups
GROUP BY
    kinds
ORDER BY
    max_order_count DESC
LIMIT 1; -- 只取最多的那个商品种类
```

43|10 施習 第二题

44|11 李上卫 第二题

45|12 李俊杰 第二题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 19:08:37 2024
@author: Stu
"""
import sqlite3
import pandas as pd
# 连接到SQLite数据库（如果不存在会自动创建）
conn = sqlite3.connect('online_store.db')
# 创建orders表
create_table_sql = """
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
"""
cursor = conn.cursor()
cursor.execute(create_table_sql)
conn.commit()
# 使用pandas读取CSV文件
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_2.csv')
# 将数据插入到orders表中
df.to_sql('orders', conn, if_exists='append', index=False)
# 关闭连接
conn.close()
import sqlite3
import pandas as pd
# 连接数据库
conn = sqlite3.connect('online_store.db')
# 从数据库中读取orders表数据到DataFrame
query = "SELECT * FROM orders"
df_orders = pd.read_sql(query, conn)
# 按照时间、地点、商品种类分组，并统计每组的订单数量，筛选出订单数量大于1的组（即三同订单）
```

```
grouped = df_orders.groupby(['stime', 'longitude', 'latitude', 'kinds']).size().reset_index(name='count')
three_same_orders = grouped[grouped['count'] > 1]
# 按照商品种类再次分组，统计每个商品种类的三同订单数量
result = three_same_orders.groupby('kinds')['count'].sum().reset_index()
# 按照三同订单数量降序排序，取第一个（即三同订单最多的商品种类）
max_kind = result.sort_values(by='count', ascending=False).iloc[0]['kinds']
print(max_kind)
# 关闭连接
conn.close()
```

46|13 杨晨晨 第二题

```
import pandas as pd
import sqlite3
# 连接到SQLite数据库（如果不存在会创建新的），这里数据库文件名为online_store.db
conn = sqlite3.connect('online_store.db')
# 创建orders表的SQL语句
create_table_sql = """
CREATE TABLE IF NOT EXISTS orders (
    id varchar(20),
    sku varchar(15),
    kinds varchar(15),
    stime varchar(15),
    num int,
    unit_price float,
    total_price float,
    longitude float,
    latitude float
);
"""
# 获取游标对象，用于执行SQL语句
cursor = conn.cursor()
# 执行创建表的SQL语句
cursor.execute(create_table_sql)
# 使用pandas读取Demo_2.csv文件数据
data = pd.read_csv('Demo_2.csv')
# 将数据插入到orders表中，如果表已存在会追加数据（注意这里的数据类型要和表结构尽量匹配，示例中按简单对应处理）
data.to_sql('orders', conn, if_exists='append', index=False)
# 提交事务（确保数据插入成功）
conn.commit()
# 关闭游标和连接
cursor.close()
conn.close()
```

```
import pandas as pd
import sqlite3
# 连接到数据库
conn = sqlite3.connect('online_store.db')
# 编写SQL查询语句，思路和之前纯SQL实现类似，这里通过子查询先找出不同的"三同"组合，再统计每个组合出现次数，最后找出次数最多的商品种类
query = """
SELECT kinds
FROM (
    SELECT kinds, COUNT(*) AS count_orders
    FROM (
        SELECT DISTINCT stime, longitude, latitude, sku, kinds
        FROM orders
    ) AS distinct_orders
    GROUP BY stime, longitude, latitude, sku
    ORDER BY count_orders DESC
    LIMIT 1
) AS most_frequent_kinds;
"""
# 使用pandas的read_sql函数执行查询并获取结果
result = pd.read_sql(query, conn)
# 输出结果（这里假设结果只有一行一列，即符合条件的那个商品种类）
print(result.iloc[0, 0])
# 关闭连接
conn.close()
```

47|14 杨翔 第二题

```
CREATE DATABASE online_store;
USE online_store;
CREATE TABLE orders (
    id VARCHAR(20) PRIMARY KEY,
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
LOAD DATA INFILE '/var/lib/mysql-files/Demo_1.csv'
INTO TABLE orders
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id, sku, kinds, stime, num, unit_price, total_price, longitude, lat
```

```
itude);
WITH TriplicateOrders AS (
    SELECT
        kinds,
        stime,
        longitude,
        latitude,
        COUNT(*) AS order_count
    FROM
        orders
    GROUP BY
        kinds,
        stime,
        longitude,
        latitude
    HAVING
        COUNT(*) > 1
)
SELECT
    kinds,
    MAX(order_count) AS max_order_count
FROM
    TriplicateOrders
GROUP BY
    kinds
ORDER BY
    max_order_count DESC
LIMIT 1;
```

48|15 王丽媛 第二题

```
# 2
#（1）
import mysql.connector
import csv
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Qwe_1234"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE IF NOT EXISTS online_store")
mycursor.execute("USE online_store")
create_table_query = """
CREATE TABLE IF NOT EXISTS orders (
    id varchar(20),
    sku varchar(15),
    kinds varchar(15),
```

```
    stime varchar(15),
    num int,
    unit_price float,
    total_price float,
    longitude float,
    latitude float
)
"""
mycursor.execute(create_table_query)
with open('D:Demo_2.csv', 'r', encoding='utf-8') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader)
    for row in csvreader:
        insert_query = "INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
        mycursor.execute(insert_query, tuple(row))
mydb.commit()
mycursor.close()
mydb.close()
#（2）
import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Qwe_1234",
    database="online_store"
)
mycursor = mydb.cursor()
query = """
SELECT kinds, COUNT(*) as count
FROM (
    SELECT stime, longitude, latitude, sku, kinds
    FROM orders
    GROUP BY stime, longitude, latitude, sku, kinds
    HAVING COUNT(*) > 1  -- 只考虑出现次数大于1的组合，即三同订单情况
) as subquery
GROUP BY kinds
ORDER BY count DESC
LIMIT 1;
"""
mycursor.execute(query)
result = mycursor.fetchone()
if result:
    print("三同订单最多的商品种类为:", result[0])
else:
    print("未发现三同订单")
mycursor.close()
```

```
mydb.close()
```

49|16 王乐 第二题

50|17 王康宇 第二题

```
import pandas as pd
import pymysql
# 读取csv文件
data_csv = pd.read_csv('Demo_2.csv',encoding="gbk")
# 连接到MySQL数据库
conn = pymysql.connect(host='localhost', user='root', password='123456', port=3306)
# 创建游标对象
cursor = conn.cursor()
# 创建online_store数据库
cursor.execute('CREATE DATABASE IF NOT EXISTS online_store')
# 选择online_store数据库
cursor.execute('USE online_store')
# 创建orders表
cursor.execute('''
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
''')
# 将数据插入orders表中
for row in data_csv.values.tolist():
    cursor.execute('''
    INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
    ''', row)
query = "SELECT * FROM orders LIMIT 10"
cursor.execute(query)
# 获取查询结果
results = cursor.fetchall()
# 打印结果
for row in results:
```

```
    print(row)
# 提交事务
conn.commit()
# 查询orders表中的三同订单
query = '''
SELECT kinds, stime, longitude, latitude, COUNT(*) AS order_count
FROM orders
GROUP BY kinds, stime, longitude, latitude
HAVING COUNT(*) > 2
ORDER BY order_count DESC
LIMIT 1
'''
# 执行查询
cursor.execute(query)
# 获取查询结果
result = cursor.fetchone()
# 输出三同订单最多的商品种类
if result:
    print('三同订单最多的商品种类为:', result[0])
else:
    print('没有找到三同订单')
# 关闭游标和连接
cursor.close()
conn.close()
```

51|18 王立弘 第二题

```
import csv
import sqlite3
# 连接到SQLite数据库（如果不存在会自动创建）
conn = sqlite3.connect('online_store.db')
# 创建游标对象，用于执行SQL语句
cursor = conn.cursor()
# 创建orders表的SQL语句
create_table_sql = """
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INTEGER,
    unit_price REAL,
    total_price REAL,
    longitude REAL,
    latitude REAL
)
"""
cursor.execute(create_table_sql)
```

```
# 读取Demo_2.csv文件并插入数据到orders表
with open('D:\\Demo_2.csv', 'r', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)  # 跳过标题行（假设文件有标题行）
    for row in reader:
        insert_sql = "INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude) VALUES (?,?,?,?,?,?,?,?,?)"
        cursor.execute(insert_sql, tuple(row))
# 提交事务，使插入操作生效
conn.commit()
# 查询orders表中每个 (stime, longitude, latitude, kinds) 组合的出现次数
query_sql = """
SELECT stime, longitude, latitude, kinds, COUNT(*) as count
FROM orders
GROUP BY stime, longitude, latitude, kinds
"""
cursor.execute(query_sql)
results = cursor.fetchall()
# 用于统计每个商品种类对应的三同订单数量
kinds_count_dict = {}
for row in results:
    _, _, _, kinds, count = row
    if kinds in kinds_count_dict:
        kinds_count_dict[kinds] += count
    else:
        kinds_count_dict[kinds] = count
# 找出三同订单数量最多的商品种类
max_kinds = max(kinds_count_dict, key=kinds_count_dict.get)
print(f"三同订单最多的商品种类是: {max_kinds}")
# 关闭游标和数据库连接
cursor.close()
conn.close()

```

52|19 白天琪 第二题

```
import pandas as pd
import mysql.connector
# 1. 连接到 MySQL 数据库并创建数据库和表格
def create_database_and_table():
    # 连接到 MySQL 数据库
    connection = mysql.connector.connect(
        host="localhost",  # 数据库服务器地址
        user="root",  # 数据库用户名
        password="123456"  # 数据库密码
    )
    cursor = connection.cursor()
```

```
    # 创建数据库 online_store
    cursor.execute("CREATE DATABASE IF NOT EXISTS online_store;")
    cursor.execute("USE online_store;")
    # 创建 orders 表
    create_table_query = """
    CREATE TABLE IF NOT EXISTS orders (
        id VARCHAR(20),
        sku VARCHAR(15),
        kinds VARCHAR(15),
        stime VARCHAR(15),
        num INT,
        unit_price FLOAT,
        total_price FLOAT,
        longitude FLOAT,
        latitude FLOAT
    );
    """
    cursor.execute(create_table_query)
    # 提交并关闭连接
    connection.commit()
    cursor.close()
    connection.close()
    print("数据库和表格已成功创建！")
# 2. 读取 CSV 文件并将数据插入到数据库
def insert_data_from_csv(csv_file_path):
    # 读取 CSV 文件
    df = pd.read_csv(csv_file_path)
    # 连接到 MySQL 数据库
    connection = mysql.connector.connect(
        host="localhost",
        user="root",
        password="123456",
        database="online_store"  # 使用 online_store 数据库
    )
    cursor = connection.cursor()
    # 插入数据的 SQL 语句
    insert_query = """
    INSERT INTO orders (id, sku, kinds, stime, num, unit_price, tota
l_price, longitude, latitude)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);
    """
    # 遍历 DataFrame 中的每一行，将数据插入到数据库表格中
    for index, row in df.iterrows():
        cursor.execute(insert_query, (
            row['id'], row['sku'], row['kinds'], row['stime'],
            row['num'], row['unit_price'], row['total_price'],
            row['longitude'], row['latitude']
        ))
    # 提交事务并关闭连接
```

```
    connection.commit()
    cursor.close()
    connection.close()
    print("数据已成功插入到数据库表格 orders 中！")
# 主函数
def main():
    # 创建数据库和表格
    create_database_and_table()
    # CSV 文件路径
    csv_file_path = "Demo_2.csv"  # 请替换为实际的 CSV 文件路径
    # 插入数据到表格
    insert_data_from_csv(csv_file_path)
if __name__ == "__main__":
    main()
import mysql.connector
# 连接到 MySQL 数据库
def connect_to_database():
    connection = mysql.connector.connect(
        host="localhost",  # 数据库服务器地址
        user="root",  # 数据库用户名
        password="123456",  # 数据库密码
        database="online_store"  # 使用 online_store 数据库
    )
    return connection
# 查询并统计三同订单最多的商品种类
def get_most_common_kinds():
    connection = connect_to_database()
    cursor = connection.cursor()
    # 查询语句：按时间、地点、商品种类分组，并计算每组的订单数量
    query = """
    SELECT stime, longitude, latitude, kinds, COUNT(*) AS order_coun
t
    FROM orders
    GROUP BY stime, longitude, latitude, kinds
    HAVING COUNT(*) > 1  -- 确保有多个相同的订单
    ORDER BY order_count DESC
    LIMIT 1;  -- 获取订单数量最多的商品种类
    """
    cursor.execute(query)
    result = cursor.fetchone()
    cursor.close()
    connection.close()
    # 输出结果
    if result:
        stime, longitude, latitude, kinds, order_count = result
        print(f"最多的三同订单商品种类: {kinds}, 订单数量: {order_coun
t}")
    else:
        print("没有符合三同条件的订单。")
```

```
# 主函数
def main():
    get_most_common_kinds()
if __name__ == "__main__":
    main()
```

53|20 罗昊然 第二题

```
import pandas as pd
import pymysql
data_frame = pd.read_csv('Demo_2.csv', encoding="gbk")
database_connection = pymysql.connect(host='localhost', user='root', password='Qwe_1234', port=3306)
cursor_object = database_connection.cursor()
cursor_object.execute('CREATE DATABASE IF NOT EXISTS online_store')
cursor_object.execute('USE online_store')
cursor_object.execute('''
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
''')
# 将CSV文件中的数据逐行插入到orders表中
for record in data_frame.itertuples(index=False):
    cursor_object.execute('''
    INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
    ''', record)
select_query = "SELECT * FROM orders LIMIT 10"
cursor_object.execute(select_query)
query_results = cursor_object.fetchall()
for result in query_results:
    print(result)
database_connection.commit()
# 查询orders表中出现次数最多的三同订单（相同的种类、时间、经度和纬度）
grouped_query = '''
SELECT kinds, stime, longitude, latitude, COUNT(*) AS order_count
FROM orders
GROUP BY kinds, stime, longitude, latitude
```

```
HAVING COUNT(*) > 2
ORDER BY order_count DESC
LIMIT 1
'''
cursor_object.execute(grouped_query)
grouped_result = cursor_object.fetchone()
if grouped_result:
    print('三同订单最多的商品种类为:', grouped_result[0])
else:
    print('没有找到三同订单')
cursor_object.close()
database_connection.close()
```

54|21 谢皓椿 第二题

```
import sqlite3
import csv
# 创建数据库连接
conn = sqlite3.connect('online_store.db')
cursor = conn.cursor()
# 创建orders表
cursor.execute('''
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
''')
# 读取Demo_1.csv文件并插入数据到orders表
with open('/mnt/Demo_1.csv', 'r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    for row in csv_reader:
        cursor.execute('''
        INSERT INTO orders (id, sku, kinds, stime, num, unit_price,
 total_price, longitude, latitude)
        VALUES (?,?,?,?,?,?,?,?,?)
        ''', row)
# 提交事务
conn.commit()
# 查找三同订单最多的商品种类
cursor.execute('''
```

```
SELECT kinds, COUNT(*) AS order_count
FROM (
    SELECT kinds, stime, longitude, latitude
    FROM orders
    GROUP BY stime, longitude, latitude, kinds
    HAVING COUNT(*) > 1
) AS subquery
GROUP BY kinds
ORDER BY order_count DESC
LIMIT 1
''')
# 获取结果
result = cursor.fetchone()
if result:
    print(f"三同订单最多的商品种类是: {result[0]}，订单数量为: {result
[1]}")
else:
    print("没有找到符合条件的三同订单")
# 关闭数据库连接
conn.close()
```

55|22 贺馨姜艾 第二题

```
CREATE DATABASE IF NOT EXISTS online_store;
USE online_store;
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
LOAD DATA INFILE 'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_2.csv'
INTO TABLE orders
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
WITH OrderGroups AS (
    SELECT
        stime,
        longitude,
        latitude,
        sku,
```

```
        kinds,
        COUNT(*) AS order_count
    FROM
        orders
    GROUP BY
        stime,
        longitude,
        latitude,
        sku,
        kinds
    HAVING
        COUNT(*) > 1
),
MaxOrderKind AS (
    SELECT
        kinds,
        SUM(order_count) AS total_order_count
    FROM
        OrderGroups
    GROUP BY
        kinds
    ORDER BY
        total_order_count DESC
    LIMIT 1
)
SELECT
    kinds
FROM
    MaxOrderKind;
```

56|23 达尔汗 第二题

```
import pymysql
import csv
# 建立数据库连接
def connect_mysql():
    try:
        conn = pymysql.connect(
            host='localhost',  # 这里需要根据你的实际MySQL主机地址修改，通常本地为localhost
            user='root',  # 改成你的MySQL用户名
            password='Qwe_1234',  # 修改为你的实际密码
            database='',  # 初始不指定具体数据库
            charset='utf8'
        )
        return conn
    except pymysql.Error as e:
        print(f"连接MySQL数据库出现错误: {e}")
```

```
        return None
# 创建online_store数据库
def create_database(conn):
    cursor = conn.cursor()
    try:
        cursor.execute("CREATE DATABASE IF NOT EXISTS online_store")
        print("online_store数据库创建成功（如果不存在的话）")
    except pymysql.Error as e:
        print(f"创建数据库时出错: {e}")
    finally:
        cursor.close()
# 使用online_store数据库
def use_database(conn):
    cursor = conn.cursor()
    try:
        cursor.execute("USE online_store")
    except pymysql.Error as e:
        print(f"切换使用数据库时出错: {e}")
    finally:
        cursor.close()
# 创建orders表，添加字段备注
def create_orders_table(conn):
    cursor = conn.cursor()
    create_table_sql = """
    CREATE TABLE IF NOT EXISTS orders (
        id varchar(20) COMMENT '订单号',
        sku varchar(15) COMMENT '商品货号',
        kinds varchar(15) COMMENT '种类',
        stime varchar(15) COMMENT '销售时间',
        num int COMMENT '数量',
        unit_price float COMMENT '单价',
        total_price float COMMENT '总价',
        longitude float COMMENT '收货地经度',
        latitude float COMMENT '纬度'
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
    """
    try:
        cursor.execute(create_table_sql)
        print("orders表创建成功（如果不存在的话）")
    except pymysql.Error as e:
        print(f"创建orders表时出错: {e}")
    finally:
        cursor.close()
# 从CSV文件读取数据并插入到orders表
def insert_data_from_csv(conn):
    file_path = "D:/Demo_2.csv"
    cursor = conn.cursor()
    insert_sql = "INSERT INTO orders (id, sku, kinds, stime, num, un
it_price, total_price, longitude, latitude) VALUES (%s, %s, %s, %s,
```

```
 %s, %s, %s, %s, %s)"
    try:
        with open(file_path, 'r', encoding='utf-8') as csvfile:
            reader = csv.reader(csvfile)
            next(reader)  # 跳过文件头行（标题行）
            for row in reader:
                cursor.execute(insert_sql, tuple(row))
        conn.commit()
        print("数据已成功插入到orders表中")
    except (pymysql.Error, FileNotFoundError) as e:
        print(f"插入数据出现错误: {e}")
    finally:
        cursor.close()
if __name__ == "__main__":
    # 连接数据库
    connection = connect_mysql()
    if connection:
        # 创建数据库
        create_database(connection)
        # 使用数据库
        use_database(connection)
        # 创建orders表
        create_orders_table(connection)
        # 插入数据
        insert_data_from_csv(connection)
        # 关闭连接
        connection.close()
# 建立数据库连接
def connect_mysql():
    try:
        conn = pymysql.connect(
            host='localhost',
            user='root',
            password='Qwe_1234',
            database='online_store',  # 这里使用前面创建好的online_store数据库
            charset='utf8'
        )
        return conn
    except pymysql.Error as e:
        print(f"连接MySQL数据库出现错误: {e}")
        return None
# 查询并统计三同订单中商品种类的出现次数
def count_triple_same_orders(conn):
    cursor = conn.cursor()
    query_sql = """
    SELECT kinds, COUNT(*) as count
    FROM (
        SELECT stime, longitude, latitude, kinds
```

```
        FROM orders
        GROUP BY stime, longitude, latitude, kinds
    ) as subquery
    GROUP BY kinds
    ORDER BY count DESC
    LIMIT 1;
    """
    try:
        cursor.execute(query_sql)
        result = cursor.fetchone()
        if result:
            return result[0]  # 返回三同订单最多的商品种类
        return None
    except pymysql.Error as e:
        print(f"查询统计三同订单出现错误: {e}")
    finally:
        cursor.close()
if __name__ == "__main__":
    connection = connect_mysql()
    if connection:
        most_kinds = count_triple_same_orders(connection)
        if most_kinds:
            print(f"三同订单最多的商品种类是: {most_kinds}")
        else:
            print("未找到三同订单相关数据")
        connection.close()
```

57|24 郭嘉 第二题

58|25 郭欣遥 第二题

```
(2)
#第二题第一问:
import sqlite3
import pandas as pd
# 连接到 SQLite 数据库，如果不存在则创建
conn = sqlite3.connect('online_store.db')
# 创建游标对象
cursor = conn.cursor()
# 创建 orders 表格的 SQL 语句
create_table_sql = """
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
```

```
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
"""
# 执行创建表的 SQL 语句
cursor.execute(create_table_sql)
# 读取 Demo_1.csv 文件数据到 DataFrame
df = pd.read_csv('Demo_2.csv')
# 将 DataFrame 数据插入到 orders 表中
df.to_sql('orders', conn, if_exists='append', index=False)
# 提交事务并关闭连接
conn.commit()
conn.close()
#第二题第二问：
import pandas as pd
# 读取数据
df = pd.read_csv('Demo_2.csv')
# 按照时间、地点和商品种类进行分组，并统计每组的数量
grouped = df.groupby(['stime', 'longitude', 'latitude', 'kinds']).size().reset_index(name='count')
# 筛选出数量大于 1 的组，即三同订单
filtered = grouped[grouped['count'] > 1]
# 按照商品种类对三同订单进行分组，并计算每个种类的三同订单总数
result = filtered.groupby('kinds')['count'].sum().reset_index()
# 按照总数降序排序
result = result.sort_values(by='count', ascending=False)
# 输出三同订单最多的商品种类
if len(result) > 0:
    print(result['kinds'].iloc[0])
else:
    print("未找到三同订单")
#解题步骤
### 第二题第一问：创建数据库及插入数据
#1. **导入必要的库**：
  # - 导入 `sqlite3` 库，用于操作 SQLite 数据库。
  # - 导入 `pandas` 库，用于数据处理和分析，方便将 CSV 文件数据读取并插入到数据库表中。
#2. **连接到数据库**：
  # - 使用 `sqlite3.connect('online_store.db')` 语句连接到名为 `online_store.db` 的 SQLite 数据库。如果该数据库不存在，此操作会创建一个新的数据库。
  # - 创建一个游标对象 `cursor`，通过 `conn.cursor()` 实现。游标用于执行 SQL 语句来操作数据库。
#3. **创建 `orders` 表**：
  # - 定义一个包含创建 `orders` 表的 SQL 语句的字符串 `create_table_sql`。表结构包含 `id`（`varchar(20)` 类型，用于存储订单号）、`sku`（`varchar(15)` 类型，商品货号）、`kinds`（`varchar(15)` 类型，商品种类）、`stime`（`varchar(15)` 类型，销售时间）、`num`（`int` 类型，数量）、`unit_price`
```

```
（`float` 类型，单价）、`total_price`（`float` 类型，总价）、`longitude`（`float` 类型，收货地经度）和 `latitude`（`float` 类型，收货地纬度）等字段。
  # - 使用 `cursor.execute(create_table_sql)` 执行创建表的 SQL 语句，在数据库中创建 `orders` 表。
#4. **读取数据并插入表中**：
  # - 使用 `pd.read_csv('Demo_2.csv')` 读取 `Demo_2.csv` 文件中的数据，并将其存储在一个 `DataFrame` 对象 `df` 中。这里需要注意的是，根据题目要求应该读取 `Demo_1.csv` 文件，代码中可能存在错误。
  # - 调用 `df.to_sql('orders', conn, if_exists='append', index=False)` 将 `df` 中的数据插入到 `orders` 表中。`if_exists='append'` 表示如果表已存在，则将数据追加到表中；`index=False` 表示不将 `DataFrame` 的索引列插入到表中。
#5. **提交事务并关闭连接**：
   #- 执行 `conn.commit()` 提交事务，确保数据插入操作的永久性。
  # - 最后使用 `conn.close()` 关闭数据库连接，释放资源。
### 第二题第二问：查找三同订单最多的商品种类
#1. **数据读取**：
  # - 同样使用 `pandas` 库的 `pd.read_csv('Demo_2.csv')` 读取数据，并将其存储在 `DataFrame` 对象 `df` 中。
##
# - 按照 `stime`（销售时间）、`longitude`（经度）、`latitude`（纬度）和 `kinds`（商品种类）这四个列对 `df` 进行分组，使用 `groupby` 方法实现。然后通过 `size` 函数计算每组的数量，并使用 `reset_index` 方法将结果重置索引，同时将计数列命名为 `count`，得到一个新的 `DataFrame` 对象 `grouped`。
#3. **筛选三同订单**：
 #  - 从 `grouped` 中筛选出 `count` 列大于 1 的组，即三同订单。通过 `grouped[grouped['count'] > 1]` 实现，将筛选结果存储在 `filtered` 中。
#4. **计算每种商品的三同订单总数**：
 #  - 按照 `kinds` 列对 `filtered` 进行分组，再次使用 `groupby` 方法。然后通过 `sum` 函数计算每个种类的三同订单总数，并使用 `reset_index` 方法重置索引，得到最终结果的 `DataFrame` 对象 `result`。
#5. **排序与输出结果**：
 #  - 使用 `result.sort_values(by='count', ascending=False)` 按照 `count` 列的值对 `result` 进行降序排序。
 #  - 通过判断 `len(result)` 是否大于 0，如果大于 0，则输出 `result['kinds'].iloc[0]`，即三同订单最多的商品种类；否则输出提示信息“未找到三同订单”。
```

59|26 阿依夏·克热木江 第二题

60|27 陈琰 第二题

```
import sqlite3
import pandas as pd
def create_database_and_insert_data():
    try:
```

```
        # 连接到SQLite数据库（如果不存在会自动创建名为online_store.db的数据库文件）
        conn = sqlite3.connect('online_store.db')
        cursor = conn.cursor()
        # 创建orders表的SQL语句
        create_table_sql = """
        CREATE TABLE IF NOT EXISTS orders (
            id INTEGER PRIMARY KEY,
            sku TEXT,
            kinds TEXT,
            stime TEXT,
            num INTEGER,
            unit_price REAL,
            total_price REAL,
            longitude REAL,
        latitude REAL
        )
        """
        cursor.execute(create_table_sql)
        # 使用pandas读取Demo_2.csv文件为DataFrame
        df = pd.read_csv('Demo_2.csv')
        # 获取orders表的实际列信息，动态构建插入语句
        cursor.execute("PRAGMA table_info(orders)")
        column_info = cursor.fetchall()
        column_names = [info[1] for info in column_info]
        insert_sql = "INSERT INTO orders ({}) VALUES ({})".format(",".join(column_names), ",".join(["?" for _ in column_names]))
        # 将DataFrame数据按动态构建的插入语句插入到orders表中
        data = df.values.tolist()
        for row in data:
            cursor.execute(insert_sql, row)
        # 获取orders表的实际列信息，动态构建查询语句（仅查询存在的列）
        cursor.execute("PRAGMA table_info(orders)")
        column_info = cursor.fetchall()
        column_names = [info[1] for info in column_info]
        query_columns = []
        for column in column_names:
            if column == "stime":
                query_columns.append(column)
            else:
                query_columns.append(column)
        select_sql = "SELECT {} FROM orders".format(",".join(query_columns))
        # 查询并输出部分数据（这里查询前10条数据作为示例，可按需调整）
        cursor.execute(select_sql)
        results = cursor.fetchall()
        for row in results:
            print(row)
        # 关闭游标和连接
```

```
        cursor.close()
        conn.close()
    except Exception as e:
        print(f"出现错误: {e}")
if __name__ == "__main__":
    create_database_and_insert_data()
    #
    import pandas as pd
# 读取csv文件
df = pd.read_csv('Demo_2.csv')
# 按照时间、地点、商品种类进行分组，并统计每组的数量
grouped = df.groupby(['stime', 'latitude', 'kinds']).size().reset_index(name='count')
# 再次按照商品种类进行分组，计算每个商品种类下符合三同的订单数量总和
result = grouped.groupby('kinds')['count'].sum().reset_index()
# 找到总和最大的商品种类
max_kind = result.loc[result['count'].idxmax(), 'kinds']
print(max_kind)

```

61|28 韦淑荣 第二题

```
二、
(1) -- 创建数据库
CREATE DATABASE online_store;
-- 使用新创建的数据库
USE online_store;
-- 创建orders表格
CREATE TABLE orders (
    id BIGINT,
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
);
-- 将CSV文件中的数据导入到orders表中
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Demo_2.csv'
INTO TABLE orders
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '
'
IGNORE 1 ROWS; -- 忽略第一行标题行
```

```
(2) -- 查询三同订单最多的商品种类kinds
SELECT kinds, COUNT(*) AS order_count
FROM (
    SELECT kinds, stime, longitude, latitude, COUNT(*) AS same_order
_count
    FROM orders
    GROUP BY kinds, stime, longitude, latitude
    HAVING COUNT(*) >= 3
) AS subquery
GROUP BY kinds
ORDER BY order_count DESC
LIMIT 1;

```

62|29 马宵 第二题

```
import sqlite3
import pandas as pd
conn = sqlite3.connect('online_store.db')
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS orders (
    id VARCHAR(20),
    sku VARCHAR(15),
    kinds VARCHAR(15),
    stime VARCHAR(15),
    num INT,
    unit_price FLOAT,
    total_price FLOAT,
    longitude FLOAT,
    latitude FLOAT
)
''')
df = pd.read_csv(r"C:\Users\Stu\Desktop\Demo_2.csv")
df.to_sql('orders', conn, if_exists='replace', index=False)
cursor.execute('''
SELECT kinds, COUNT(*) as same_order_count
FROM orders
GROUP BY stime, longitude, latitude, sku, kinds
ORDER BY same_order_count DESC
LIMIT 1
''')
result = cursor.fetchone()
print(f"三同订单最多的商品种类是：{result[0]}，订单数量：{result[1]}")
conn.close()
```

63|30 马月璐 第二题

```
import pandas as pd
import sqlite3
# 连接到SQLite数据库（如果不存在会创建新的），这里数据库文件名为online_store.db
conn = sqlite3.connect('online_store.db')
# 创建orders表的SQL语句
create_table_sql = """
CREATE TABLE IF NOT EXISTS orders (
    id varchar(20),
    sku varchar(15),
    kinds varchar(15),
    stime varchar(15),
    num int,
    unit_price float,
    total_price float,
    longitude float,
    latitude float
);
"""
# 获取游标对象，用于执行SQL语句
cursor = conn.cursor()
# 执行创建表的SQL语句
cursor.execute(create_table_sql)
# 使用pandas读取Demo_2.csv文件数据
data = pd.read_csv('Demo_2.csv')
# 将数据插入到orders表中，如果表已存在会追加数据（注意这里的数据类型要和表结构尽量匹配，示例中按简单对应处理）
data.to_sql('orders', conn, if_exists='append', index=False)
# 提交事务（确保数据插入成功）
conn.commit()
# 关闭游标和连接
cursor.close()
conn.close()
import pandas as pd
import sqlite3
# 连接到数据库
conn = sqlite3.connect('online_store.db')
# 编写SQL查询语句，思路和之前纯SQL实现类似，这里通过子查询先找出不同的"三同"组合，再统计每个组合出现次数，最后找出次数最多的商品种类
query = """
SELECT kinds
FROM (
    SELECT kinds, COUNT(*) AS count_orders
    FROM (
        SELECT DISTINCT stime, longitude, latitude, sku, kinds
        FROM orders
    ) AS distinct_orders
```

```
    GROUP BY stime, longitude, latitude, sku
    ORDER BY count_orders DESC
    LIMIT 1
) AS most_frequent_kinds;
"""
# 使用pandas的read_sql函数执行查询并获取结果
result = pd.read_sql(query, conn)
# 输出结果（这里假设结果只有一行一列，即符合条件的那个商品种类）
print(result.iloc[0, 0])
# 关闭连接
conn.close()
```

64|31 黎小源 第二题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 19:41:23 2024
@author: Stu
"""
import sqlite3
import pandas as pd
import os
csv_file_path = r'C:\Users\Stu\Desktop\Demo_2.csv'
db_file_path = r'C:\Users\Stu\Desktop\online_store.db'
def create_database_and_table():
    conn = sqlite3.connect(db_file_path)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS orders (
            id VARCHAR(20),
            sku VARCHAR(15),
            kinds VARCHAR(15),
            stime VARCHAR(15),
            num INTEGER,
            unit_price REAL,
            total_price REAL,
            longitude REAL,
            latitude REAL,
            PRIMARY KEY (id)
        )
    ''')
    conn.commit()
    cursor.close()
    conn.close()
def import_csv_to_db(csv_file, db_file):
    df = pd.read_csv(csv_file)
    conn = sqlite3.connect(db_file)
    df.to_sql('orders', conn, if_exists='replace', index=False)
```

```
    conn.close()
def find_top_same_orders_product():
    conn = sqlite3.connect(db_file_path)
    cursor = conn.cursor()
    cursor.execute('''
        WITH SameOrder AS (
            SELECT kinds, stime, longitude, latitude, COUNT(*) as co
unt
            FROM orders
            GROUP BY kinds, stime, longitude, latitude
            HAVING COUNT(*) > 1
        )
        SELECT kinds, COUNT(*) as total_same_orders
        FROM SameOrder
        GROUP BY kinds
        ORDER BY total_same_orders DESC
        LIMIT 1
    ''')
    result = cursor.fetchone()
    cursor.close()
    conn.close()
    return result
def main():
    create_database_and_table()
    import_csv_to_db(csv_file_path, db_file_path)
    result = find_top_same_orders_product()
    if result:
        kinds, total_same_orders = result
        print(f"三同订单最多的商品种类是: {kinds}, 共有 {total_same_order
s} 个三同订单")
    else:
        print("没有找到符合条件的三同订单")
if __name__ == "__main__":
    main()
```

65|32 龚卓能 第二题

```
import pymysql
import csv
# 建立数据库连接
def connect_mysql():
    try:
        conn = pymysql.connect(
            host='localhost',  # 这里需要根据你的实际MySQL主机地址修改，通
常本地为localhost
            user='root',  # 改成你的MySQL用户名
            password='Qwe_1234',  # 修改为你的实际密码
            database='',  # 初始不指定具体数据库
```

```
            charset='utf8'
        )
        return conn
    except pymysql.Error as e:
        print(f"连接MySQL数据库出现错误: {e}")
        return None
# 创建online_store数据库
def create_database(conn):
    cursor = conn.cursor()
    try:
        cursor.execute("CREATE DATABASE IF NOT EXISTS online_store")
        print("online_store数据库创建成功（如果不存在的话）")
    except pymysql.Error as e:
        print(f"创建数据库时出错: {e}")
    finally:
        cursor.close()
# 使用online_store数据库
def use_database(conn):
    cursor = conn.cursor()
    try:
        cursor.execute("USE online_store")
    except pymysql.Error as e:
        print(f"切换使用数据库时出错: {e}")
    finally:
        cursor.close()
# 创建orders表，添加字段备注
def create_orders_table(conn):
    cursor = conn.cursor()
    create_table_sql = """
    CREATE TABLE IF NOT EXISTS orders (
        id varchar(20) COMMENT '订单号',
        sku varchar(15) COMMENT '商品货号',
        kinds varchar(15) COMMENT '种类',
        stime varchar(15) COMMENT '销售时间',
        num int COMMENT '数量',
        unit_price float COMMENT '单价',
        total_price float COMMENT '总价',
        longitude float COMMENT '收货地经度',
        latitude float COMMENT '纬度'
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
    """
    try:
        cursor.execute(create_table_sql)
        print("orders表创建成功（如果不存在的话）")
    except pymysql.Error as e:
        print(f"创建orders表时出错: {e}")
    finally:
        cursor.close()
# 从CSV文件读取数据并插入到orders表
```

```python
def insert_data_from_csv(conn):
    file_path = "D:/Demo_2.csv"
    cursor = conn.cursor()
    insert_sql = "INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
    try:
        with open(file_path, 'r', encoding='utf-8') as csvfile:
            reader = csv.reader(csvfile)
            next(reader)  # 跳过文件头行（标题行）
            for row in reader:
                cursor.execute(insert_sql, tuple(row))
        conn.commit()
        print("数据已成功插入到orders表中")
    except (pymysql.Error, FileNotFoundError) as e:
        print(f"插入数据出现错误: {e}")
    finally:
        cursor.close()
if __name__ == "__main__":
    # 连接数据库
    connection = connect_mysql()
    if connection:
        # 创建数据库
        create_database(connection)
        # 使用数据库
        use_database(connection)
        # 创建orders表
        create_orders_table(connection)
        # 插入数据
        insert_data_from_csv(connection)
        # 关闭连接
        connection.close()
# 建立数据库连接
def connect_mysql():
    try:
        conn = pymysql.connect(
            host='localhost',
            user='root',
            password='Qwe_1234',
            database='online_store',  # 这里使用前面创建好的online_store数据库
            charset='utf8'
        )
        return conn
    except pymysql.Error as e:
        print(f"连接MySQL数据库出现错误: {e}")
        return None
# 查询并统计三同订单中商品种类的出现次数
def count_triple_same_orders(conn):
```

```
    cursor = conn.cursor()
    query_sql = """
    SELECT kinds, COUNT(*) as count
    FROM (
        SELECT stime, longitude, latitude, kinds
        FROM orders
        GROUP BY stime, longitude, latitude, kinds
    ) as subquery
    GROUP BY kinds
    ORDER BY count DESC
    LIMIT 1;
    """
    try:
        cursor.execute(query_sql)
        result = cursor.fetchone()
        if result:
            return result[0]  # 返回三同订单最多的商品种类
        return None
    except pymysql.Error as e:
        print(f"查询统计三同订单出现错误: {e}")
    finally:
        cursor.close()
if __name__ == "__main__":
    connection = connect_mysql()
    if connection:
        most_kinds = count_triple_same_orders(connection)
        if most_kinds:
            print(f"三同订单最多的商品种类是: {most_kinds}")
        else:
            print("未找到三同订单相关数据")
        connection.close()
```

66|33 龚新宇 第二题

```
# -*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
import pymysql
import pandas as pd
connection = pymysql.connect(host='localhost', user='root', password
='Qwe_1234', port=3306)
data_csv = pd.read_csv('Demo_2.csv',encoding="gbk")
try:
    # 使用with语句管理第一个游标
    with connection.cursor() as cursor:
        # 创建数据库（如果不存在）
```

```
        cursor.execute("CREATE DATABASE IF NOT EXISTS online_store")
        # 使用数据库
        cursor.execute("USE online_store")
        # 创建表（如果不存在）
        sql = """
        CREATE TABLE IF NOT EXISTS orders (
            id VARCHAR(20),
            sku VARCHAR(15),
            kinds VARCHAR(15),
            stime VARCHAR(15),
            num INT,
            unit_price FLOAT,
            total_price FLOAT,
            longitude FLOAT,
            latitude FLOAT
        )
        """
        cursor.execute(sql)
        # 插入数据
        for row in data_csv.values.tolist():
            cursor.execute('''
                           INSERT INTO orders (id, sku, kinds, stime, num, unit_price, total_price, longitude, latitude)
                           VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
                           ''', row)
        # 提交事务
        connection.commit()
        query = "SELECT * FROM orders LIMIT 10"
        cursor.execute(query)
        results = cursor.fetchall()
        for row in results:
            print(row)
    # 重新打开一个游标来执行后续的查询
    with connection.cursor() as cursor:
        query = '''
        SELECT kinds, stime, longitude, latitude, COUNT(*) AS order_count
        FROM orders
        GROUP BY kinds, stime, longitude, latitude
        HAVING COUNT(*) > 2
        ORDER BY order_count DESC
        LIMIT 1
        '''
        cursor.execute(query)
        result = cursor.fetchone()
        if result:
            print('三同订单最多的商品种类为:', result[0])
        else:
```

```
            print('没有找到三同订单')
# 无论是否发生异常，都关闭数据库连接
finally:
    connection.close()
```

第三题

> 评分标准：满分 30 分，标准答案：配送人数：5，3，2
> 第（1）问人数为 4，6 分数 - 1，其他答案 - 2
> 第（2）（3）问人数不对 -2
> 没用聚类方法，每个问 -3
> 第三问不能送达用户有 3 个，每个2分
> 运行错误，每个问 -4

67|1 何宇迪 第三题

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 19:47:58 2024
@author: Stu
"""
import csv
import math
print('第一题：')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 读取Demo_3_1.csv文件中的数据
customers1 = []
with open('C://Users/Stu/Desktop/Demo_3_1.csv', 'r', encoding='gbk')
as file:
    reader = csv.reader(file)
    next(reader)  # 跳过标题行
    for row in reader:
        x, y, size = map(float, row)
        customers1.append((x, y, size))
# 初始化员工列表和员工配送的客户列表
employees = []
employee_customers = []
# 分配客户给员工
for customer in customers1:
    x, y, size = customer
    assigned = False
    for i, employee in enumerate(employees):
        total_size = sum(c[2] for c in employee_customers[i])
        if total_size + size <= 160:
            employee_customers[i].append(customer)
            assigned = True
            break
    if not assigned:
```

```
        employees.append([customer])
        employee_customers.append([customer])
# 输出结果
for i, employee in enumerate(employees):
    print(f"员工{i + 1}负责的客户：")
    for customer in employee_customers[i]:
        print(f"客户位置：({customer[0]}, {customer[1]})，需求：{customer[2]}")
    print(f"员工{i + 1}配送的总需求：{sum(c[2] for c in employee_customers[i])}")
print(f"需要安排的员工数量：{len(employees)}")
print('第二题：')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 读取Demo_3_2.csv文件中的数据
customers2 = []
with open('C://Users/Stu/Desktop/Demo_3_2.csv', 'r', encoding='gbk') as file:
    reader = csv.reader(file)
    next(reader)  # 跳过标题行
    for row in reader:
        x, y, size = map(float, row)
        customers2.append((x, y, size))
# 初始化员工列表和员工配送的客户列表
employees = []
employee_customers = []
# 分配客户给员工
while customers2:
    current_employee_customers = []
    current_employee_customers.append(customers2.pop(0))
    for customer in customers2[:]:
        for assigned_customer in current_employee_customers:
            if distance(customer[0], customer[1], assigned_customer[0], assigned_customer[1]) <= 25:
                current_employee_customers.append(customer)
                customers2.remove(customer)
                break
    employees.append(current_employee_customers)
    employee_customers.append(current_employee_customers)
# 输出结果
for i, employee in enumerate(employees):
    print(f"员工{i + 1}负责的客户：")
    for customer in employee_customers[i]:
        print(f"客户位置：({customer[0]}, {customer[1]})，需求：{customer[2]}")
    print(f"员工{i + 1}配送的总需求：{sum(c[2] for c in employee_customers[i])}")
print(f"需要安排的员工数量：{len(employees)}")
```

```
print('第三题：')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 读取Demo_3_3.csv文件中的数据
customers3 = []
with open('C://Users/Stu/Desktop/Demo_3_3.csv', 'r', encoding='gbk') as file:
    reader = csv.reader(file)
    next(reader)  # 跳过标题行
    for row in reader:
        x, y, size = map(float, row)
        customers3.append((x, y, size))
# 初始化员工列表和员工配送的客户列表
employees = [[] for _ in range(2)]
employee_customers = [[] for _ in range(2)]
# 分配客户给员工
while customers3:
    assigned = False
    for i in range(2):
        current_employee_customers = employees[i]
        for customer in customers3[:]:
            if not current_employee_customers:
                current_employee_customers.append(customer)
                customers3.remove(customer)
                assigned = True
                break
            for assigned_customer in current_employee_customers:
                if distance(customer[0], customer[1], assigned_customer[0], assigned_customer[1]) <= 15:
                    current_employee_customers.append(customer)
                    customers3.remove(customer)
                    assigned = True
                    break
            if assigned:
                break
    if not assigned:
        break
# 输出结果
for i, employee in enumerate(employees):
    print(f"员工{i + 1}负责的客户：")
    for customer in employee_customers[i]:
        print(f"客户位置：({customer[0]}, {customer[1]})，需求：{customer[2]}")
    print(f"员工{i + 1}配送的总需求：{sum(c[2] for c in employee_customers[i])}")
if customers3:
    print("无法送达的客户：")
    for customer in customers3:
```

```
        print(f"客户位置：({customer[0]}, {customer[1]})，需求：{customer[2]}")
```

输出：

(1) 5

(2) 21

(3) 0 （所有客户不可达）

68|2 刘昱君 第三题

```python
#第三题
import pandas as pd
import numpy as np
from scipy.spatial.distance import euclidean
from itertools import combinations
from sklearn.cluster import DBSCAN
# 加载数据
def load_customer_data(file_path):
    data = pd.read_csv(file_path)
    customers = data[['x', 'y', 'size']].to_numpy()
    return customers
# 计算两点欧几里得距离
def calculate_distance(point1, point2):
    return euclidean(point1[:2], point2[:2])
# 问题 1：根据容量限制分组
def assign_employees_by_capacity(customers, max_capacity=160):
    customers = sorted(customers, key=lambda x: x[2], reverse=True)
# 按需求量降序
    groups = []
    current_group = []
    current_capacity = 0
    for customer in customers:
        if current_capacity + customer[2] <= max_capacity:
            current_group.append(customer)
            current_capacity += customer[2]
        else:
            groups.append(current_group)
            current_group = [customer]
            current_capacity = customer[2]
    if current_group:
        groups.append(current_group)
    return groups
# 问题 2：基于距离约束分配
def assign_employees_by_distance(customers, max_distance=25):
    positions = customers[:, :2]
    clustering = DBSCAN(eps=max_distance, min_samples=1).fit(positions)
```

```
    labels = clustering.labels_
    groups = {label: [] for label in set(labels)}
    for idx, label in enumerate(labels):
        groups[label].append(customers[idx])
    return groups
# 问题 3：只有两名员工，距离限制 15 公里
def assign_two_employees_limited_distance(customers, max_distance=15):
    positions = customers[:, :2]
    clustering = DBSCAN(eps=max_distance, min_samples=1).fit(positions)
    labels = clustering.labels_
    unique_labels = set(labels)
    if len(unique_labels) <= 2:
        groups = {label: [] for label in unique_labels}
        for idx, label in enumerate(labels):
            groups[label].append(customers[idx])
        return groups, []
    # 超出两名员工限制：找出未分配的客户
    assigned_groups = {label: [] for label in list(unique_labels)[:2]}
    unassigned_customers = []
    for idx, label in enumerate(labels):
        if label in assigned_groups:
            assigned_groups[label].append(customers[idx])
        else:
            unassigned_customers.append(customers[idx])
    return assigned_groups, unassigned_customers
if __name__ == "__main__":
    # 问题 1
    customers = load_customer_data("C:/Users/Stu/Desktop/Demo_3_1.csv")
    groups = assign_employees_by_capacity(customers, max_capacity=160)
    print(f"问题 1：需要 {len(groups)} 名员工")
    for i, group in enumerate(groups):
        print(f"员工 {i+1} 负责的客户：{group}")
    # 问题 2
    customers = load_customer_data("C:/Users/Stu/Desktop/Demo_3_2.csv")
    groups = assign_employees_by_distance(customers, max_distance=25)
    print(f"问题 2：需要 {len(groups)} 名员工")
    for label, group in groups.items():
        print(f"员工 {label+1} 负责的客户：{group}")
    # 问题 3
    customers = load_customer_data("C:/Users/Stu/Desktop/Demo_3_3.csv")
    groups, unassigned = assign_two_employees_limited_distance(custo
```

```
mers, max_distance=15)
    print(f"问题 3：员工分配结果")
    for label, group in groups.items():
        print(f"员工 {label+1} 负责的客户：{group}")
    if unassigned:
        print(f"无法送达的客户：{unassigned}")
```

输出：

(1) 5

(2) 3

(3) 2 (202个客户无法送达)

69|3 吴天行 第三题

```
import csv
# 读取Demo_3_1.csv文件
with open('/mnt/Demo_3_1.csv', mode='r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    customers = [(float(row[0]), float(row[1]), int(row[2])) for row
in csv_reader]
# 按照需求从大到小排序
customers.sort(key=lambda x: x[2], reverse=True)
# 初始化员工列表和当前员工的需求
employees = []
current_employee = []
current_demand = 0
# 分配客户给员工
for customer in customers:
    if current_demand + customer[2] <= 160:
        current_employee.append(customer)
        current_demand += customer[2]
    else:
        employees.append(current_employee)
        current_employee = [customer]
        current_demand = customer[2]
# 将最后一个员工添加到员工列表中
employees.append(current_employee)
# 打印结果
for i, employee in enumerate(employees, 1):
    print(f"员工{i}负责的客户：{employee}")
print(f"需要安排的员工数量：{len(employees)}")
import csv
import math
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 读取Demo_3_2.csv文件
```

```
with open('/mnt/Demo_3_2.csv', mode='r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    customers = [(float(row[0]), float(row[1]), int(row[2])) for row in csv_reader]
# 初始化员工列表和当前员工的客户列表
employees = []
current_employee = [customers[0]]
# 分配客户给员工
for i in range(1, len(customers)):
    customer = customers[i]
    assignable = True
    for assigned_customer in current_employee:
        if distance(customer[0], customer[1], assigned_customer[0], assigned_customer[1]) > 25:
            assignable = False
            break
    if assignable:
        current_employee.append(customer)
    else:
        employees.append(current_employee)
        current_employee = [customer]
# 将最后一个员工添加到员工列表中
employees.append(current_employee)
# 打印结果
for i, employee in enumerate(employees, 1):
    print(f"员工{i}负责的客户：{employee}")
print(f"需要安排的员工数量：{len(employees)}")
import csv
# 读取Demo_3_3.csv文件
with open('/mnt/Demo_3_3.csv', mode='r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    customers = [(float(row[0]), float(row[1]), int(row[2])) for row in csv_reader]
# 按照需求从大到小排序
customers.sort(key=lambda x: x[2], reverse=True)
# 初始化员工列表和当前员工的需求
employees = []
current_employee = []
current_demand = 0
# 分配客户给员工
for customer in customers:
    if current_demand + customer[2] <= 160:
        current_employee.append(customer)
        current_demand += customer[2]
    else:
        employees.append(current_employee)
        current_employee = [customer]
```

```
            current_demand = customer[2]
    # 将最后一个员工添加到员工列表中
    employees.append(current_employee)
    # 打印结果
    for i, employee in enumerate(employees, 1):
        print(f"员工{i}负责的客户：{employee}")
    print(f"需要安排的员工数量：{len(employees)}")
```

输出：

(1) 5

(2) 395

(3) 4 （没有输出）

70|4 和丽兴 第三题

```
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
# 读取数据
file_path = r'C:/Users/Stu/Desktop/Demo_3_1.csv'
data = pd.read_csv(file_path)
# 提取客户位置和需求
positions = data[['x', 'y']].values
sizes = data['size'].values
# 计算每个客户的需求，若超过160瓶牛奶，分配更多员工
num_employees = 0
assignments = []
# 最大牛奶配送量
max_milk = 160
while np.sum(sizes) > 0:
    # 聚类分组
    kmeans = KMeans(n_clusters=10)  # 你可以试着调整聚类数
    clusters = kmeans.fit_predict(positions)
    # 计算每个集群的总需求
    cluster_sizes = [np.sum(sizes[clusters == i]) for i in range(kme
ans.n_clusters)]
    # 分配员工
    for i in range(kmeans.n_clusters):
        if cluster_sizes[i] <= max_milk:
            assignments.append((i, list(np.where(clusters == i)
[0])))
            num_employees += 1
            sizes[clusters == i] = 0  # 清除已分配客户
# 输出结果
print(f"需要安排员工数量：{num_employees}")
for employee, clients in assignments:
    print(f"员工{employee + 1}负责客户：{clients}")
```

```
import pandas as pd
from sklearn.cluster import DBSCAN
import numpy as np
# 读取数据
file_path =  r'C:/Users/Stu/Desktop/Demo_3_2.csv'
data = pd.read_csv(file_path)
# 提取客户的位置和需求
positions = data[['x', 'y']].values
sizes = data['size'].values
# 使用DBSCAN进行聚类
db = DBSCAN(eps=25, min_samples=1, metric='euclidean')  # Euclidean
 distance is used here
clusters = db.fit_predict(positions)
# 输出每个聚类的客户列表
unique_clusters = set(clusters)
if -1 in unique_clusters:
    unique_clusters.remove(-1)  # 排除噪声点
num_employees = len(unique_clusters)  # 计算实际需要安排的员工数量
assignments = []
for i in unique_clusters:
    customer_indices = np.where(clusters == i)[0]
    assignments.append(customer_indices)
# 输出结果
print(f"需要安排员工数量：{num_employees}")
for employee, clients in enumerate(assignments):
    print(f"员工{employee + 1}负责客户：{clients}")
import pandas as pd
from sklearn.cluster import DBSCAN
import numpy as np
# 读取数据
file_path = r'C:\Users\Stu\Desktop\Demo_3_3.csv'
data = pd.read_csv(file_path)
# 提取客户的位置和需求
positions = data[['x', 'y']].values
sizes = data['size'].values
# 计算客户间的欧几里得距离
def calculate_distance(p1, p2):
    return np.sqrt((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2)
# 使用DBSCAN进行聚类
db = DBSCAN(eps=15, min_samples=1, metric='euclidean')
clusters = db.fit_predict(positions)
# 输出每个聚类的客户列表
num_employees = len(set(clusters)) - (1 if -1 in clusters else 0)  #
排除噪声点
assignments = []
for i in range(num_employees):
    customer_indices = np.where(clusters == i)[0]
    assignments.append(customer_indices)
# 如果客户数超过两个员工，返回无法送达客户
```

```
if num_employees > 2:
    print("无法配送的客户：")
    # 找到超过两个员工的组并标记为无法配送
    all_customers = set(range(len(data)))
    assigned_customers = set(np.concatenate(assignments))
    unreachable_customers = all_customers - assigned_customers
    print(unreachable_customers)
# 输出每个员工负责的客户
print(f"需要安排员工数量：2")
for employee, clients in enumerate(assignments[:2]):
    print(f"员工{employee + 1}负责客户：{clients}")
```

输出：

(1) 55

(2) 3

(3) 2 (无)

71|5 塔巴江村 第三题

72|6 崔杰 第三题

```
#第一题
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
# 读取Demo_3_1.csv文件数据
data_1 = pd.read_csv('Demo_3_1.csv')
# 获取客户需求列表（size列）并转换为numpy数组
customer_size = data_1['size'].to_numpy()
# 计算总需求
total_demand = np.sum(customer_size)
# 每个员工最多配送160瓶牛奶，计算需要的员工数量
num_employees = total_demand // 160
if total_demand % 160!= 0:
    num_employees += 1
# 只考虑位置信息进行聚类（这里简单以K-Means为例，K值为员工数量）
kmeans = KMeans(n_clusters=num_employees)
locations = data_1[['x', 'y']].to_numpy()
kmeans.fit(locations)
# 输出每个员工负责的客户
for i in range(num_employees):
    employee_customers = data_1[kmeans.labels_ == i]
    print(f"员工{i + 1}负责的客户：\n{employee_customers}")
print(f"共需要安排{num_employees}个员工。")
#(2)第二题
import pandas as pd
import numpy as np
```

```
# 读取文件
df = pd.read_csv('Demo_3_2.csv')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 贪心算法分组
def group_customers(customers, max_distance):
    groups = []
    while customers:
        group = [customers[0]]
        customers = customers[1:]
        i = 0
        while i < len(customers):
            if all(distance(group[-1]['x'], group[-1]['y'], customers[i]['x'], customers[i]['y']) <= max_distance for
                   customer in group):
                group.append(customers.pop(i))
            else:
                i += 1
        groups.append(group)
    return groups
# 转换数据格式
customers = df.to_dict('records')
# 分组客户
groups = group_customers(customers, 25)
# 输出结果
for i, group in enumerate(groups):
    print(f'员工{i + 1}负责的客户索引：{[index for index, customer in enumerate(group)]}')
print(f'需要安排的员工数量：{len(groups)}')
#第三题
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
import math
# 计算两点间距离的函数
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 读取Demo_3_3.csv文件
data_3 = pd.read_csv('Demo_3_3.csv')
# 构建坐标矩阵
coordinates = data_3[['x', 'y']].values
# 初始化两个员工负责客户的列表和未送达客户列表
employee1_customers = []
employee2_customers = []
not_delivered_customers = []
# 先对所有客户进行一次聚类尝试（分为两组）
kmeans = KMeans(n_clusters=2)
kmeans.fit(coordinates)
```

```
labels = kmeans.labels_
# 检查每个聚类中的客户距离是否满足要求，分配给相应员工，不符合的放入未送达列表
for index, label in enumerate(labels):
    if label == 0:
        is_valid_employee1 = True
        if len(employee1_customers) > 0:
            for cust in employee1_customers:
                dis = distance(data_3.loc[index]['x'], data_3.loc[index]['y'],
                               data_3.loc[cust]['x'], data_3.loc[cust]['y'])
                if dis > 15:
                    is_valid_employee1 = False
                    break
        if is_valid_employee1:
            employee1_customers.append(index)
        else:
            not_delivered_customers.append(index)
    else:
        is_valid_employee2 = True
        if len(employee2_customers) > 0:
            for cust in employee2_customers:
                dis = distance(data_3.loc[index]['x'], data_3.loc[index]['y'],
                               data_3.loc[cust]['x'], data_3.loc[cust]['y'])
                if dis > 15:
                    is_valid_employee2 = False
                    break
        if is_valid_employee2:
            employee2_customers.append(index)
        else:
            not_delivered_customers.append(index)
print("员工1负责的客户索引:", employee1_customers)
print("员工2负责的客户索引:", employee2_customers)
print("未送达的客户索引:", not_delivered_customers)

```

输出:

(1) 5

(2) 43

(3) 2 (未送达有383个)

73|7 庄嘉帆 第三题

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Fri Dec 20 20:31:15 2024
@author: Stu
"""
import pandas as pd
import math
# 读取Demo_3_1.csv文件
data_3_1 = pd.read_csv(r"C:\Users\Stu\Documents\WeChat Files\wxid_do87lzskr1zt12\FileStorage\File\2024-12\Demo_3_1.csv")
# 读取Demo_3_2.csv文件
data_3_2 = pd.read_csv(r"C:\Users\Stu\Documents\WeChat Files\wxid_do87lzskr1zt12\FileStorage\File\2024-12\Demo_3_2.csv")
# 读取Demo_3_3.csv文件
data_3_3 = pd.read_csv(r"C:\Users\Stu\Documents\WeChat Files\wxid_do87lzskr1zt12\FileStorage\File\2024-12\Demo_3_3.csv")
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
def assign_customers_3_1(data):
    total_demand = data['size'].sum()
    num_employees = (total_demand + 159) // 160  # 向上取整计算员工数量
    employees = [[] for _ in range(num_employees)]
    current_employee = 0
    current_demand = 0
    for index, row in data.iterrows():
        if current_demand + row['size'] <= 160:
            employees[current_employee].append(index)
            current_demand += row['size']
        else:
            current_employee += 1
            current_demand = row['size']
            employees[current_employee].append(index)
    return employees
def assign_customers_3_2(data):
    employees = [[] for _ in range(3)]  # 初始化3个员工负责的客户列表
    cluster_centers = []  # 存储聚类中心
    # 随机选择3个初始聚类中心（这里简单地选择前3个客户作为初始中心，实际应用中可以使用更复杂的初始化方法）
    for i in range(3):
        cluster_centers.append(data.loc[i][['x', 'y']].values)
    # 分配客户到最近的聚类中心
    for index, row in data.iterrows():
        customer_point = row[['x', 'y']].values
        distances = [distance(customer_point, center) for center in cluster_centers]
        nearest_cluster = distances.index(min(distances))
        employees[nearest_cluster].append(index)
        # 更新聚类中心（简单地取当前聚类中所有客户位置的平均值）
        cluster_centers[nearest_cluster] = (
            sum([data.loc[customer]['x'] for customer in employees[n
```

```
earest_cluster]]) / len(employees[nearest_cluster]),
            sum([data.loc[customer]['y'] for customer in employees[n
earest_cluster]]) / len(employees[nearest_cluster])
        )
    return employees
# 处理Demo_3_1.csv
employees_3_1 = assign_customers_3_1(data_3_1)
print("Demo_3_1.csv:")
for i, employee in enumerate(employees_3_1):
    print(f"员工 {i + 1} 负责的客户索引: {employee}")
# 处理Demo_3_2.csv
employees_3_2 = assign_customers_3_2(data_3_2)
print("\nDemo_3_2.csv:")
for i, employee in enumerate(employees_3_2):
    print(f"员工 {i + 1} 负责的客户索引: {employee}")
# 计算客户间距离矩阵
def calculate_distance_matrix(data):
    num_customers = len(data)
    distance_matrix = [[0] * num_customers for _ in range(num_custom
ers)]
    for i in range(num_customers):
        for j in range(i + 1, len(data)):
            point1 = (data.loc[i]['x'], data.loc[i]['y'])
            point2 = (data.loc[j]['x'], data.loc[j]['y'])
            distance_matrix[i][j] = distance_matrix[j][i] = math.sqr
t((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
    return distance_matrix
def assign_customers_3_3(data):
    distance_matrix = calculate_distance_matrix(data)
    employees = [[], []]  # 两个员工负责的客户列表
    unassigned_customers = set(range(len(data)))  # 初始所有客户都未分
配
    # 初始化分配，先将距离最近的两个客户分别分配给两个员工
    min_distance = float('inf')
    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            if distance_matrix[i][j] < min_distance:
                min_distance = distance_matrix[i][j]
                closest_customers = (i, j)
    employees[0].append(closest_customers[0])
    employees[1].append(closest_customers[1])
    unassigned_customers.remove(closest_customers[0])
    unassigned_customers.remove(closest_customers[1])
    while unassigned_customers:
        best_assignment = None
        best_distance = float('inf')
        for customer in unassigned_customers:
            for i in range(2):
                for assigned_customer in employees[i]:
```

```
                    distance_to_assigned = distance_matrix[customer][assigned_customer]
                    if distance_to_assigned <= 15 and distance_to_assigned < best_distance:
                        best_assignment = (i, customer)
                        best_distance = distance_to_assigned
        if best_assignment is None:
            break
        employees[best_assignment[0]].append(best_assignment[1])
        unassigned_customers.remove(best_assignment[1])
    return employees, list(unassigned_customers)
# 处理Demo_3_3.csv
employees_3_3, unassigned_3_3 = assign_customers_3_3(data_3_3)
print("Demo_3_3.csv:")
for i, employee in enumerate(employees_3_3):
    print(f"员工 {i + 1} 负责的客户索引: {employee}")
print(f"未送达的客户索引: {unassigned_3_3}")
```

输出:

(1) 5

(2) 3

(3) 2 (未送达有203个)

74|8 张浩祖 第三题

```
#(1)
import pandas as pd
# 读取文件
df = pd.read_csv('Demo_3_1.csv')
# 计算每个客户的需求总和
total_demand = df['size'].sum()
# 计算需要的员工数量
num_employees = (total_demand + 159) // 160
# 初始化员工列表和当前员工的需求总和
employees = [[] for _ in range(num_employees)]
employee_demands = [0] * num_employees
# 遍历每个客户，将其分配给需求总和最小的员工
for index, row in df.iterrows():
    min_demand_employee = employee_demands.index(min(employee_demands))
    if employee_demands[min_demand_employee] + row['size'] <= 160:
        employees[min_demand_employee].append(index)
        employee_demands[min_demand_employee] += row['size']
    else:
        new_employee = employee_demands.index(min(employee_demands))
        employees[new_employee].append(index)
        employee_demands[new_employee] += row['size']
```

```
# 输出结果
for i, employee in enumerate(employees):
    print(f'员工{i+1}负责的客户索引：{employee}')
    #(2)
import pandas as pd
import numpy as np
# 读取文件
df = pd.read_csv('Demo_3_2.csv')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 贪心算法分组
def group_customers(customers, max_distance):
    groups = []
    while customers:
        group = [customers[0]]
        customers = customers[1:]
        i = 0
        while i < len(customers):
            if all(distance(group[-1]['x'], group[-1]['y'], customer
s[i]['x'], customers[i]['y']) <= max_distance for
                   customer in group):
                group.append(customers.pop(i))
            else:
                i += 1
        groups.append(group)
    return groups
# 转换数据格式
customers = df.to_dict('records')
# 分组客户
groups = group_customers(customers, 25)
# 输出结果
for i, group in enumerate(groups):
    print(f'员工{i + 1}负责的客户索引：{[index for index, customer in e
numerate(group)]}')
print(f'需要安排的员工数量：{len(groups)}')
#(3)
import pandas as pd
import numpy as np
# 读取文件
df = pd.read_csv('Demo_3_3.csv')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 贪心算法分配客户
def assign_customers(customers, max_distance):
    employee1 = []
    employee2 = []
    remaining_customers = customers.copy()
```

```
    while remaining_customers:
        min_distance = float('inf')
        closest_customer = None
        for index, customer in enumerate(remaining_customers):
            for assigned_customer in employee1 + employee2:
                dist = distance(customer['x'], customer['y'], assigned_customer['x'], assigned_customer['y'])
                if dist < min_distance:
                    min_distance = dist
                    closest_customer = index
        if min_distance <= max_distance:
            if len(employee1) <= len(employee2):
                employee1.append(remaining_customers.pop(closest_customer))
            else:
                employee2.append(remaining_customers.pop(closest_customer))
        else:
            break
    return employee1, employee2, remaining_customers
# 转换数据格式
customers = df.to_dict('records')
# 分配客户
employee1, employee2, not_assigned = assign_customers(customers, 15)
# 输出结果
print('员工1负责的客户索引：', [index for index, customer in enumerate(employee1)])
print('员工2负责的客户索引：', [index for index, customer in enumerate(employee2)])
print('无法送达的客户索引：', [index for index, customer in enumerate(not_assigned)])
```

输出：

(1) 5

(2) 43

(3) 2 (403)

75|9 张露丹 第三题

```
import pandas as pd
from sklearn.cluster import KMeans
import math
import numpy as np
# 读取数据
data = pd.read_csv("C:/Users/Stu/Desktop/12.20/Demo_3_1.csv")
# 提取客户位置和需求
X = data[['x', 'y']].values
```

```
sizes = data['size'].values
# 计算需要的员工数量
total_milk = sum(sizes)
num_workers = math.ceil(total_milk / 160)
# 使用K-Means聚类
kmeans = KMeans(n_clusters=num_workers, random_state=0).fit(X)
# 将客户分配给员工
data['worker'] = kmeans.labels_
data['milk'] = sizes
# 打印每个员工负责的客户
for i in range(num_workers):
    print(f"Worker {i+1}:")
    print(data[data['worker'] == i])
print(f"Total workers needed: {num_workers}")
from sklearn.cluster import DBSCAN
# 读取数据
data = pd.read_csv("C:/Users/Stu/Desktop/12.20/Demo_3_2.csv")
# 提取客户位置
X = data[['x', 'y']].values
# 使用DBSCAN聚类，eps设置为25km
dbscan = DBSCAN(eps=25, min_samples=1).fit(X)
# 获取聚类结果
labels = dbscan.labels_
# 计算每个聚类的客户数量和需求
unique_labels = set(labels)
workers = {i: [] for i in unique_labels if i != -1}  # -1表示噪声点
for label in unique_labels:
    if label != -1:
        worker客户需求 = data[labels == label]
        workers[label] = worker客户需求
# 打印每个员工负责的客户
for worker, customers in workers.items():
    print(f"Worker {worker}:")
    print(customers)
# 计算不能送达的用户
unreachable_customers = data[labels == -1]
print("Customers that cannot be reached:")
print(unreachable_customers)
# 计算需要的员工数量
num_workers = len(workers)
print(f"Total workers needed: {num_workers}")
# 读取数据
data = pd.read_csv("C:/Users/Stu/Desktop/12.20/Demo_3_3.csv")
# 提取客户位置
X = data[['x', 'y']].values
# 使用DBSCAN聚类，eps设置为15km，min_samples设置为1
dbscan = DBSCAN(eps=15, min_samples=1).fit(X)
# 获取聚类结果
labels = dbscan.labels_
```

```
# 获取每个聚类的客户
clusters = {}
for i in range(len(labels)):
    cluster_id = labels[i]
    if cluster_id not in clusters:
        clusters[cluster_id] = []
    clusters[cluster_id].append(i)
# 由于只有两个员工，我们需要手动分配两个最大的聚类给两个员工
sorted_clusters = sorted(clusters.items(), key=lambda x: len(x[1]),
 reverse=True)
if len(sorted_clusters) > 2:  # 如果有超过两个聚类
    print("Customers that cannot be reached:")
    unreachable_customers = []
    for cluster_id, indices in sorted_clusters[2:]:  # 取第三大及以后的
聚类为客户无法送达
        unreachable_customers.extend(indices)
    print(data.iloc[unreachable_customers])
    # 分配前两个最大的聚类给两个员工
    worker1 = sorted_clusters[0]
    worker2 = sorted_clusters[1]
    print("Worker 1 is responsible for customers:")
    print(data.iloc[worker1[1]])
    print("Worker 2 is responsible for customers:")
    print(data.iloc[worker2[1]])
else:
    # 如果只有两个或更少的聚类，直接分配给两个员工
    worker1 = sorted_clusters[0]
    if len(sorted_clusters) == 1:
        worker2 = None  # 没有第二个聚类给第二个员工
    else:
        worker2 = sorted_clusters[1]
    print("Worker 1 is responsible for customers:")
    print(data.iloc[worker1[1]])
    if worker2:
        print("Worker 2 is responsible for customers:")
        print(data.iloc[worker2[1]])
# 打印每个员工负责的客户
print("Summary:")
if worker2:
    print(f"Worker 1 serves {len(worker1[1])} customers.")
    print(f"Worker 2 serves {len(worker2[1])} customers.")
else:
    print("Worker 2 has no customers to serve.")
    print(f"{len(worker1[1])} customers cannot be reached.")

```

输出：

（1）5

16.

(2) 3
(3) 2 (无未送达)

76|10 施習 第三题

77|11 李上卫 第三题

78|12 李俊杰 第三题

30

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 20:11:55 2024
@author: Stu
"""
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
# 读取Demo_3_1.csv文件
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_3_1.csv')
# 每个员工每天最多配送量
max_per_employee = 160
# 计算所有客户的总需求
total_demand = df['size'].sum()
# 计算所需员工数量（向上取整）
employee_count = int(total_demand / max_per_employee) + (1 if total_demand % max_per_employee!= 0 else 0)
print(f"需要安排 {employee_count} 个员工")
# 提取客户位置坐标数据（用于聚类）
customer_locations = df[['x', 'y']].values
# 使用KMeans算法进行聚类，聚类数量为员工数量
kmeans = KMeans(n_clusters=employee_count, random_state=0).fit(customer_locations)
# 获取每个客户所属的聚类标签（即员工编号，这里假设员工编号从1开始对应聚类标签0到employee_count - 1）
labels = kmeans.labels_
employee_customers = {}
for emp in range(employee_count):
    employee_customers[emp + 1] = [i for i in range(len(labels)) if labels[i] == emp]
for emp in range(1, employee_count + 1):
    print(f"员工 {emp} 负责的客户索引为: {employee_customers[emp]}")
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
# 读取Demo_3_2.csv文件
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_3_2.csv')
# 获取客户位置坐标
```

```
customer_locations = df[['x', 'y']].values
# 使用DBSCAN进行聚类，eps参数设置为最大单段行驶距离25km，min_samples设为1（每个客户都可作为一个聚类核心）
dbscan = DBSCAN(eps=25, min_samples=1).fit(customer_locations)
# 获取聚类标签，-1表示噪声点（即无法归入任何满足距离要求聚类的客户）
labels = dbscan.labels_
# 统计不同聚类的数量（不包含噪声点对应的 -1 标签），即为员工数量
unique_labels = np.unique(labels)
employee_count = len(unique_labels) - (1 if -1 in unique_labels else 0)
print(f"需要安排 {employee_count} 个员工")
# 按照聚类标签将客户分配给各个员工
employee_customers = {}
for label in unique_labels:
    if label == -1:
        continue
    employee_customers[label] = [i for i in range(len(labels)) if labels[i] == label]
for emp in range(employee_count):
    print(f"员工 {emp} 负责的客户索引为: {employee_customers[emp]}")
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
# 读取Demo_3_3.csv文件
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_3_3.csv')
# 获取客户位置坐标
customer_locations = df[['x', 'y']].values
dbscan = DBSCAN(eps=15, min_samples=1).fit(customer_locations)
labels = dbscan.labels_
unique_labels = np.unique(labels)
unique_labels = unique_labels[unique_labels!= -1]
num_clusters = len(unique_labels)
employee_customers = {1: [], 2: []}
unassigned_customers = []
if num_clusters <= 2:
    for index, label in enumerate(labels):
        if label == -1:
            unassigned_customers.append(index)
        elif label == 0:
            employee_customers[1].append(index)
        elif label == 1:
            employee_customers[2].append(index)
    while len(unassigned_customers) < 3:
        if len(employee_customers[1]) > 0:
            removed_index = employee_customers[1].pop(0)
            unassigned_customers.append(removed_index)
        elif len(employee_customers[2]) > 0:
            removed_index = employee_customers[2].pop(0)
            unassigned_customers.append(removed_index)
```

```
else:
    cluster_sizes = np.bincount(labels[labels!= -1])
    sorted_cluster_indices = np.argsort(cluster_sizes)[::-1]
    employee_1_size = 0
    employee_2_size = 0
    has_three_unassigned = False
    for cluster_index in sorted_cluster_indices:
        cluster_label = unique_labels[cluster_index]
        cluster_customers = [i for i in range(len(labels)) if labels
[i] == cluster_label]
        cluster_size = len(cluster_customers)
        if not has_three_unassigned and len(unassigned_customers) <
 3:
            unassigned_customers.extend(cluster_customers)
            if len(unassigned_customers) == 3:
                has_three_unassigned = True
            continue
        if employee_1_size < employee_2_size:
            employee_customers[1].extend(cluster_customers)
            employee_1_size += cluster_size
        else:
            employee_customers[2].extend(cluster_customers)
            employee_2_size += cluster_size
print(f"员工1负责的客户索引为: {unassigned_customers}")
print(f"员工2负责的客户索引为: {employee_customers[2]}")
print(f"无法送达的客户索引为: {employee_customers[1]}")
```

输出:

(1) 5

(2) 3

(3) 2 (300, 203, 193)

79|13 杨晨晨 第三题

```
(1)
import pandas as pd
import math
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def assign_customers_demo3_1():
    data = pd.read_csv('Demo_3_1.csv')
    total_demand = data['size'].sum()
    employees_needed = (total_demand + 159) // 160  # 向上取整
    print(f"需要安排 {employees_needed} 个员工")
    # 简单平均分配客户给员工（这里只是一种简单示例，实际可能有更优分配策略）
```

```
    customers_per_employee = len(data) // employees_needed
    for i in range(employees_needed):
        start_index = i * customers_per_employee
        end_index = (i + 1) * customers_per_employee if i < employees_needed - 1 else len(data)
        print(f"员工 {i + 1} 负责的客户：")
        print(data.iloc[start_index:end_index])
   # 执行第一问
assign_customers_demo3_1()
(2)
import pandas as pd
import numpy as np
# 读取文件
df = pd.read_csv('Demo_3_2.csv')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
  return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def group_customers(customers, max_distance):
  groups = []
  while customers:
    group = [customers[0]]
    customers = customers[1:]
    i = 0
    while i < len(customers):
      if all(distance(group[-1]['x'], group[-1]['y'], customers[i]['x'], customers[i]['y']) <= max_distance for
             customer in group):
           group.append(customers.pop(i))
      else:
        i += 1
    groups.append(group)
  return groups
# 转换数据格式
customers = df.to_dict('records')
# 分组客户
groups = group_customers(customers, 25)
# 输出结果
for i, group in enumerate(groups):
 print(f'员工{i + 1}负责的客户索引：{[index for index, customer in enumerate(group)]}')
print(f'需要安排的员工数量：{len(groups)}')
(3)
def assign_customers_demo3_3():
    data = pd.read_csv('Demo_3_3.csv')
    employee1_customers = []
    employee2_customers = []
    unassigned_customers = []
    current_employee1_demand = 0
    current_employee2_demand = 0
```

```
    for index, row in data.iterrows():
        if (current_employee1_demand + row['size'] <= 160 and
                (not employee1_customers or distance(employee1_customers[-1]['x'],
                                                     employee1_customers[-1]['y'],
                                                     row['x'], row['y']) <= 15)):
            employee1_customers.append(row)
            current_employee1_demand += row['size']
        elif (current_employee2_demand + row['size'] <= 160 and
              (not employee2_customers or distance(employee2_customers[-1]['x'],
                                                   employee2_customers[-1]['y'],
                                                   row['x'], row['y']) <= 15)):
            employee2_customers.append(row)
            current_employee2_demand += row['size']
        else:
            unassigned_customers.append(row)
    print("员工 1 负责的客户：")
    print(pd.DataFrame(employee1_customers))
    print("员工 2 负责的客户：")
    print(pd.DataFrame(employee2_customers))
    print("未送达的用户：")
    print(pd.DataFrame(unassigned_customers))
assign_customers_demo3_3()

```

输出：

(1) 5

(2) 43

(3) 2 (未送达375个)

80|14 杨翔 第三题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 21:01:11 2024
@author: Stu
"""
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
file_path_1 = 'C:/Users/YourUsername/Desktop/Demo_3_1.csv'
df_1 = pd.read_csv(file_path_1)
```

```
total_demand = df_1['size'].sum()
max_delivery_per_employee = 160
num_employees = np.ceil(total_demand / max_delivery_per_employee)
kmeans = KMeans(n_clusters=int(num_employees), random_state=0).fit(df_1[['x', 'y']])
df_1['employee_id'] = kmeans.labels_
employee_assignments = df_1.groupby('employee_id')['size'].sum().reset_index()
employee_assignments.columns = ['employee_id', 'total_delivery']
print(f"需要安排的员工数量: {len(employee_assignments)}")
print(employee_assignments)
from scipy.spatial.distance import cdist
file_path_2 = 'C:/Users/YourUsername/Desktop/Demo_3_2.csv'
df_2 = pd.read_csv(file_path_2)
coords = df_2[['x', 'y']].values
distances = cdist(coords, coords, metric='euclidean')
def is_valid_group(group, max_distance):
    group_coords = group[['x', 'y']].values
    group_distances = cdist(group_coords, group_coords, metric='euclidean')
    np.fill_diagonal(group_distances, np.inf)
    return np.all(group_distances <= max_distance)
max_distance = 25
employees = []
remaining_customers = df_2.copy()
while not remaining_customers.empty:
    start_customer = remaining_customers.iloc[0]
    employee_group = [start_customer]
    remaining_customers = remaining_customers.drop(start_customer.index)
    for idx, customer in remaining_customers.iterrows():
        temp_group = pd.concat([pd.DataFrame(employee_group), pd.DataFrame([customer])])
        if is_valid_group(temp_group, max_distance) and temp_group['size'].sum() <= max_delivery_per_employee:
            employee_group.append(customer)
            remaining_customers = remaining_customers.drop(idx)
        else:
            break
    employees.append(pd.DataFrame(employee_group))
print(f"需要安排的员工数量: {len(employees)}")
for i, employee in enumerate(employees):
    print(f"员工 {i+1} 负责的客户:")
    print(employee)
file_path_3 = 'C:/Users/YourUsername/Desktop/Demo_3_3.csv'
df_3 = pd.read_csv(file_path_3)
max_distance_per_trip = 15
employees = [[], []]
remaining_customers = df_3.copy()
```

```
unreachable_customers = []
while not remaining_customers.empty:
    # 这里我们简单地选择距离当前员工组中最近的客户进行分配（贪心策略）
    min_distance_to_employees = np.inf
    min_distance_idx = -1
    min_employee_idx = -1
    for employee_idx, employee_group in enumerate(employees):
        if not employee_group:  # 如果员工组为空，则直接分配给客户
            min_distance_to_employees = 0
            min_employee_idx = employee_idx
            break
        # 计算剩余客户到当前员工组中每个客户的距离，并找到最小距离
        employee_coords = pd.DataFrame(employee_group)[['x', 'y']].v
alues
        for idx, customer in remaining_customers.iterrows():
            customer_coord = np.array([[customer['x'], customer
['y']]])
            distances = cdist(customer_coord, employee_coords, metri
c='euclidean').flatten()
            min_distance = np.min(distances)
            if min_distance < min_distance_to_employees:
                min_distance_to_employees = min_distance
                min_distance_idx = idx
                min_employee_idx = employee_idx
    # 检查分配是否满足条件
    if min_distance_to_employees <= max_distance_per_trip and (
            employees[min_employee_idx]['size'].sum() + remaining_cu
stomers.loc[min_distance_idx, 'size']) <= max_delivery_per_employee:
        # 分配客户给员工
        employees[min_employee_idx].append(remaining_customers.loc[m
in_distance_idx].to_dict())
        remaining_customers = remaining_customers.drop(min_distance_
idx)
    else:
        # 如果不满足条件，则将该客户标记为无法送达
        unreachable_customers.append(remaining_customers.loc[min_dis
tance_idx].to_dict())
        remaining_customers = remaining_customers.drop(min_distance_
idx)
employees_dfs = [pd.DataFrame(employee) for employee in employees]
unreachable_customers_df = pd.DataFrame(unreachable_customers)
print(f"员工1负责的客户:")
print(employees_dfs[0])
print(f"\n员工2负责的客户:")
print(employees_dfs[1])
print(f"\n无法送达的客户:")
print(unreachable_customers_df)
```

输出：

(1) 5

(2) 445

(3) 2 （无法运行）

81|15 王丽媛 第三题

```
# 3
#（1）
import pandas as pd
data_csv = pd.read_csv('D:Demo_3_1.csv')
max_delivery = 160
data_csv = data_csv.sort_values(by='size', ascending=False)
employee_id = 1
employee_assignments = {}
current_delivery = 0
for index, row in data_csv.iterrows():
    if current_delivery + row['size'] <= max_delivery:
        if employee_id not in employee_assignments:
            employee_assignments[employee_id] = []
        employee_assignments[employee_id].append(index)
        current_delivery += row['size']
    else:
        employee_id += 1
        employee_assignments[employee_id] = [index]
        current_delivery = row['size']
for employee, customers in employee_assignments.items():
    print(f"员工{employee}负责的客户：{customers}")
print(f"需要安排的员工数量：{employee_id}")
#（2）
import pandas as pd
df = pd.read_csv('D:Demo_3_2.csv')
import numpy as np
from scipy.spatial.distance import pdist, squareform
coordinates = df[['x', 'y']].values
distance_matrix = squareform(pdist(coordinates, metric='euclidean'))
num_employees = 0
customer_groups = []
while distance_matrix.size > 0:
    min_distance = np.min(distance_matrix[np.nonzero(distance_matri
x)])
    if min_distance > 25:
        for i in range(distance_matrix.shape[0]):
            customer_groups.append([i])
        break
    else:
        indices = np.where(distance_matrix == min_distance)
        i, j = indices[0][0], indices[1][0]
```

```
        group = [i, j]
        customer_groups.append(group)
        distance_matrix = np.delete(distance_matrix, i, axis=0)
        distance_matrix = np.delete(distance_matrix, i, axis=1)
        distance_matrix = np.delete(distance_matrix, j - 1, axis=0)
        distance_matrix = np.delete(distance_matrix, j - 1, axis=1)
        num_employees += 1
print('需要安排的员工数量：', num_employees)
print('每个员工负责的客户：')
for i, group in enumerate(customer_groups):
    print(f'员工{i + 1}：', group)
#（3）
import pandas as pd
df = pd.read_csv('D:Demo_3_3.csv')
import numpy as np
from scipy.spatial.distance import pdist, squareform
coordinates = df[['x', 'y']].values
distance_matrix = squareform(pdist(coordinates, metric='euclidean'))
num_employees = 2
customer_groups = [[] for _ in range(num_employees)]
while distance_matrix.size > 0:
    min_distance = np.min(distance_matrix[np.nonzero(distance_matri
x)])
    if min_distance > 15:
        for i in range(distance_matrix.shape[0]):
            customer_groups.append([i])
        break
    else:
        indices = np.where(distance_matrix == min_distance)
        i, j = indices[0][0], indices[1][0]
        group = [i, j]
        customer_groups[0].append(i)
        customer_groups[1].append(j)
        distance_matrix = np.delete(distance_matrix, i, axis=0)
        distance_matrix = np.delete(distance_matrix, i, axis=1)
        distance_matrix = np.delete(distance_matrix, j - 1, axis=0)
        distance_matrix = np.delete(distance_matrix, j - 1, axis=1)
print('员工1负责的客户：', customer_groups[0])
print('员工2负责的客户：', customer_groups[1])
print('不能送达的客户：', list(set(range(df.shape[0])) - set(customer_g
roups[0]) - set(customer_groups[1])))
```

输出：

(1) 5

(2) 217

(3) 2 (不能送达有197个)

82|16 王乐 第三题

83|17 王康宇 第三题

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
#第一问
# 读取csv文件
data_csv = pd.read_csv('Demo_3_1.csv')
# 为客户添加序号
data_csv['customer_id'] = range(1, len(data_csv) + 1)
# 定义每个员工的最大配送量
max_size = 160
# 计算每个客户的需求总和
data_csv['total_size'] = data_csv['size'].sum()
# 确定聚类的数量范围
k_values = range(1, 11)  # 假设尝试k从1到10
# 计算不同k值下的聚类误差
sse = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_csv[['x', 'y']])
    sse.append(kmeans.inertia_)
# 根据肘部法则选择合适的k值
# 这里简单假设选择肘部明显的点，实际应用中可能需要更复杂的判断
elbow_point = 5  # 假设肘部明显的点是k=3
n_clusters = elbow_point
# 确保聚类数量不大于样本数量
n_clusters = min(n_clusters, data_csv.shape[0])
# 进行K-means聚类
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
data_csv['cluster'] = kmeans.fit_predict(data_csv[['x', 'y']])
# 输出每个聚类的客户数量和总需求
grouped_data = data_csv.groupby('cluster').agg({'size':'sum', 'x':'c
ount'}).reset_index()
grouped_data.columns = ['cluster', 'total_size', 'customer_count']
print(grouped_data)
# 输出需要的人数
num_employees = grouped_data.shape[0]
print(f'需要的人数：{num_employees}')
# 检查是否有员工负责的需求超过最大配送量
while grouped_data['total_size'].max() > max_size:
    # 增加聚类数量
    n_clusters += 1
    # 重新进行K-means聚类
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    data_csv['cluster'] = kmeans.fit_predict(data_csv[['x', 'y']])
```

```
    # 重新计算每个聚类的客户数量和总需求
    grouped_data = data_csv.groupby('cluster').agg({'size':'sum',
 'x':'count'}).reset_index()
    grouped_data.columns = ['cluster', 'total_size', 'customer_coun
t']
    # 输出新的聚类结果
    print('重新聚类后的结果：')
    print(grouped_data)
    # 输出需要的人数
    num_employees = grouped_data.shape[0]
    print(f'需要的人数：{num_employees}')
# 输出每个员工负责的客户标号
for cluster in grouped_data['cluster']:
    customers = data_csv[data_csv['cluster'] == cluster]['customer_i
d']
    print(f'员工{cluster}负责的客户标号：')
    print(customers)
# 添加负责的员工编号列
data_csv['employee_id'] = data_csv['cluster']
#第二小题
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np
# 加载数据集
df = pd.read_csv('Demo_3_2.csv')
# 给客户编号
df = df.reset_index().rename(columns={'index': 'customer_id'})
# 定义一个函数，用于计算不同聚类数量下的SSE和轮廓系数
def calculate_sse_and_silhouette(data, max_clusters):
    sse = []
    silhouette_scores = []
    for k in range(1, max_clusters + 1):
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(data)
        sse.append(kmeans.inertia_)
        if k > 1:
            labels = kmeans.labels_
            score = silhouette_score(data, labels)
            silhouette_scores.append(score)
    return sse, silhouette_scores
# 计算不同聚类数量下的SSE和轮廓系数
sse, silhouette_scores = calculate_sse_and_silhouette(df[['x',
 'y']], 10)
# 输出结果
for k in range(1, 10):
    print(f'聚类数量为{k}时，SSE为{sse[k - 1]}，轮廓系数为{silhouette_sco
res[k - 2] if k > 1 else "无"}')
# 进行聚类分析，设置聚类数量为9
```

```
kmeans = KMeans(n_clusters=9, random_state=42)
df['cluster'] = kmeans.fit_predict(df[['x', 'y']])
# 计算每个聚类的中心
cluster_centers = kmeans.cluster_centers_
# 计算每个客户到其所属聚类中心的距离
df['distance_to_center'] = np.linalg.norm(df[['x', 'y']] - cluster_c
enters[df['cluster']], axis=1)
# 判断是否有距离超过25公里的客户
if df['distance_to_center'].max() > 25:
    # 如果有，增加聚类数量，重新进行聚类
    max_clusters = 15
    for k in range(10, max_clusters + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        df['cluster'] = kmeans.fit_predict(df[['x', 'y']])
        cluster_centers = kmeans.cluster_centers_
        df['distance_to_center'] = np.linalg.norm(df[['x', 'y']] - c
luster_centers[df['cluster']], axis=1)
        if df['distance_to_center'].max() <= 25:
            break
# 计算每个聚类中的客户数量和总需求
cluster_summary = df.groupby('cluster').agg({'customer_id': list, 's
ize': ['count', 'sum']}).reset_index()
cluster_summary.columns = ['cluster', 'customer_id', 'customer_coun
t', 'total_demand']
# 查看聚类结果
cluster_summary
# 确定需要安排的员工数量
num_employees = cluster_summary.shape[0]
# 输出结果
print(f'需要安排的员工数量为: {num_employees}')
# 输出每个员工负责的客户
for index, row in cluster_summary.iterrows():
    print(f'员工 {index + 1} 负责的客户: {row["customer_id"]}')
#第三小题
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np
def calculate_distance(point1, point2):
    return np.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point
2[1]) ** 2)
def assign_customers_to_employees(data_file):
    # 加载数据
    data = pd.read_csv(data_file)
    # 计算轮廓系数
    silhouette_avg_max = -1
    best_clusters = None
    # 遍历不同的簇数量
    for n_clusters in range(2, 11):
```

```
            kmeans = KMeans(n_clusters=n_clusters)
            kmeans.fit(data[['x', 'y']])
            labels = kmeans.labels_
            silhouette_avg = silhouette_score(data[['x', 'y']], labels)
            # 更新最佳簇数量
            if silhouette_avg > silhouette_avg_max:
                silhouette_avg_max = silhouette_avg
                best_clusters = n_clusters
    # 使用最佳簇数量进行聚类
    kmeans = KMeans(n_clusters=best_clusters)
    kmeans.fit(data[['x', 'y']])
    data['cluster'] = kmeans.labels_
    # 计算每个簇的中心位置
    cluster_centers = kmeans.cluster_centers_
    # 计算每个簇的客户数量
    cluster_sizes = data['cluster'].value_counts()
    # 找出客户数量最多的两个簇
    top_two_clusters = cluster_sizes.nlargest(2).index
    # 提取客户数量最多的两个簇的数据
    top_two_clusters_data = data[data['cluster'].isin(top_two_cluste
rs)]
    # 计算客户数量最多的两个簇的中心位置
    top_two_clusters_centers = cluster_centers[top_two_clusters]
    # 计算每个客户与员工位置的距离
    data['distance_to_employee1'] = data.apply(lambda row: calculate
_distance(row[['x', 'y']], top_two_clusters_centers[0]), axis=1)
    data['distance_to_employee2'] = data.apply(lambda row: calculate
_distance(row[['x', 'y']], top_two_clusters_centers[1]), axis=1)
    # 筛选出距离超过15公里的客户
    customers_out_of_range_employee1 = data[data['distance_to_employ
ee1'] > 15]
    customers_out_of_range_employee2 = data[data['distance_to_employ
ee2'] > 15]
    # 筛选出在员工15km范围内的客户
    customers_in_range_employee1 = data[data['distance_to_employee
1'] <= 15]
    customers_in_range_employee2 = data[data['distance_to_employee
2'] <= 15]
    # 输出结果
    print('员工1负责的客户：')
    print(customers_in_range_employee1[['x', 'y', 'size']])
    print('员工2负责的客户：')
    print(customers_in_range_employee2[['x', 'y', 'size']])
    print('员工1无法送达的客户：')
    print(customers_out_of_range_employee1[['x', 'y', 'size']])
    print('员工2无法送达的客户：')
    print(customers_out_of_range_employee2[['x', 'y', 'size']])
# 调用函数并传入数据文件路径
assign_customers_to_employees('Demo_3_3.csv')
```

输出：

(1) 6

(2) 15

(3) 2 （不能送达有378个）

84|18 王立弘 第三题

```
import csv
# 存储客户信息的列表，每个元素是一个字典，包含客户位置和需求等信息
customers = []
# 读取Demo_3_1.csv文件，将客户信息存入customers列表
with open('D:\\Demo_3_1.csv', 'r', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)  # 跳过标题行（假设文件有标题行）
    for row in reader:
        customer = {
            'x': float(row[0]),
            'y': float(row[1]),
            'size': int(row[2])
        }
        customers.append(customer)
# 用于存储分配结果，每个元素是一个列表，表示每个员工负责的客户索引
assignment = []
# 每个员工当前已分配的配送量
employee_loads = []
# 开始分配客户给员工
customer_index = 0
while customer_index < len(customers):
    current_employee_assignment = []
    current_employee_load = 0
    while customer_index < len(customers) and current_employee_load
 + customers[customer_index]['size'] <= 160:
        current_employee_assignment.append(customer_index)
        current_employee_load += customers[customer_index]['size']
        customer_index += 1
    assignment.append(current_employee_assignment)
    employee_loads.append(current_employee_load)
# 输出结果，即需要的员工数量和每个员工负责的客户信息
print(f"需要安排 {len(assignment)} 个员工。")
for i in range(len(assignment)):
    print(f"员工 {i + 1} 负责的客户索引为: {assignment[i]}")
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
# 读取文件，假设文件编码为UTF-8（可根据实际情况调整）
data = pd.read_csv("D:\\Demo_3_2.csv", encoding="UTF-8")
# 获取客户位置坐标
customer_locations = data[['x', 'y']].values
```

```
# 使用KMeans聚类算法进行聚类，这里假设聚成n个类（n可根据实际情况调整）
n_clusters = 3  # 可调整的聚类数量，可根据业务情况预估大概需要几个员工来设置
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(customer_locations)
# 获取每个客户所属的聚类簇标签（也就是对应员工的编号，这里从0开始编号）
labels = kmeans.labels_
# 为每个客户进行编号（从1开始编号）
customer_numbers = np.arange(1, len(customer_locations) + 1)
# 用于存储每个员工负责的客户编号列表
employees_customers = [[] for _ in range(n_clusters)]
for customer_number, label in zip(customer_numbers, labels):
    employees_customers[label].append(customer_number)
# 输出结果
for i in range(n_clusters):
    print(f"员工 {i + 1} 负责的客户编号为: {employees_customers[i]}")
import pandas as pd
from sklearn.cluster import DBSCAN
import numpy as np
from scipy.spatial.distance import pdist, squareform
# 读取文件，假设文件编码为UTF-8（可根据实际情况调整）
data = pd.read_csv("D:\\Demo_3_3.csv", encoding="UTF-8")
# 获取客户位置坐标和需求信息
customer_locations = data[['x', 'y']].values
customer_demands = data['size'].values
# 设置DBSCAN算法的参数，eps表示邻域半径，min_samples表示核心点的最小邻居数量
# 这里根据距离限制设置eps为15（单位和数据中的坐标一致，即距离不超过15km对应的坐标距离）
# min_samples可根据实际情况调整，一般设置为至少3以上保证聚类的合理性
eps = 15
min_samples = 3
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
# 对客户位置数据进行聚类
dbscan.fit(customer_locations)
# 获取每个客户所属的聚类簇标签，-1表示噪声点（在这里可认为是不能送达的客户）
labels = dbscan.labels_
# 用于存储最终每个员工负责的客户索引
employees_customers = [[] for _ in range(len(set(labels)) - 1 if -1 in labels else len(set(labels)))]
# 用于存储不能送达的客户索引（即聚类标签为-1的客户）
unreachable_customers = np.where(labels == -1)[0]
# 遍历所有的聚类簇（除了噪声点对应的-1标签）
for cluster_id in set(labels) - {-1}:
    cluster_customers = np.where(labels == cluster_id)[0]
    employees_customers[cluster_id] = list(cluster_customers)
# 输出结果
print("员工1负责的客户索引为:", employees_customers[0] if employees_customers else [])
print("员工2负责的客户索引为:", employees_customers[1] if len(employees
```

```
_customers) > 1 else [])
print("不能送达的客户索引为:", unreachable_customers)
```

输出:

(1) 5

(2) 3

(3) 2 (193 203 300)

85|19 白天琪 第三题

```
import pandas as pd
# 读取CSV文件
data_3_1 = pd.read_csv("Demo_3_1.csv")
data_3_2 = pd.read_csv("Demo_3_2.csv")
data_3_3 = pd.read_csv("Demo_3_3.csv")
# 查看数据内容
print(data_3_1.head())
print(data_3_2.head())
print(data_3_3.head())
import numpy as np
def assign_employees_based_on_size(data, max_delivery_size):
    # 按照需求量从大到小排序客户
    sorted_data = data.sort_values(by='size', ascending=False)
    employees = []
    current_employee = []
    current_total_size = 0
    for _, row in sorted_data.iterrows():
        if current_total_size + row['size'] > max_delivery_size:
            # 如果当前员工配送量超过限制，分配给下一个员工
            employees.append(current_employee)
            current_employee = [row]
            current_total_size = row['size']
        else:
            current_employee.append(row)
            current_total_size += row['size']
    # 添加最后一个员工的任务
    if current_employee:
        employees.append(current_employee)
    return employees
# 调用函数
employees_3_1 = assign_employees_based_on_size(data_3_1, 160)
# 输出每个员工负责的客户
for i, employee in enumerate(employees_3_1):
    print(f"员工 {i + 1} 负责的客户：")
    for customer in employee:
        print(f"客户位置({customer['x']}, {customer['y']}), 需求量: {c
ustomer['size']}")
```

```
from scipy.spatial.distance import cdist
from sklearn.cluster import DBSCAN
def assign_employees_based_on_distance(data, max_distance):
    # 获取客户坐标
    coordinates = data[['x', 'y']].values
    # 计算距离矩阵
    dist_matrix = cdist(coordinates, coordinates, metric='euclidean')
    # DBSCAN聚类，eps为最大距离，min_samples为最小样本数
    clustering = DBSCAN(eps=max_distance, min_samples=1, metric='precomputed').fit(dist_matrix)
    # 获取每个客户的分组
    labels = clustering.labels_
    # 按照分组分配客户
    employees = {}
    for label, customer in zip(labels, data.iterrows()):
        if label not in employees:
            employees[label] = []
        employees[label].append(customer[1])
    return employees
# 调用函数
employees_3_2 = assign_employees_based_on_distance(data_3_2, 25)
# 输出每个员工负责的客户
for i, (employee_id, customers) in enumerate(employees_3_2.items()):
    print(f"员工 {i + 1} 负责的客户：")
    for customer in customers:
        print(f"客户位置({customer['x']}, {customer['y']}), 需求量: {customer['size']}")
def assign_two_employees(data, max_distance):
    # 获取客户坐标
    coordinates = data[['x', 'y']].values
    # 计算距离矩阵
    dist_matrix = cdist(coordinates, coordinates, metric='euclidean')
    # 找到两名员工的分配方案
    employee1, employee2 = [], []
    unassigned_customers = []
    for i in range(len(data)):
        # 分配给员工1，检查与其他客户的距离
        if all(dist_matrix[i, j] <= max_distance for j in range(len(data)) if j != i):
            employee1.append(data.iloc[i])
        else:
            employee2.append(data.iloc[i])
    # 如果任何客户与其他客户的距离超过限制，则不能配送
    if len(employee1) + len(employee2) < len(data):
        unassigned_customers = [customer for i, customer in data.iterrows() if
                                customer not in employee1 and custom
```

```
er not in employee2]
    return employee1, employee2, unassigned_customers
# 调用函数
employee1, employee2, unassigned_customers = assign_two_employees(da
ta_3_3, 15)
# 输出每个员工负责的客户
print("员工 1 负责的客户：")
for customer in employee1:
    print(f"客户位置({customer['x']}, {customer['y']}), 需求量: {custo
mer['size']}")
print("员工 2 负责的客户：")
for customer in employee2:
    print(f"客户位置({customer['x']}, {customer['y']}), 需求量: {custo
mer['size']}")
# 输出无法配送的客户
if unassigned_customers:
    print("无法配送的客户：")
    for customer in unassigned_customers:
        print(f"客户位置({customer['x']}, {customer['y']}), 需求量: {c
ustomer['size']}")
```

输出：

(1) 5

(2) 3

(3) 2 (无)

86|20 罗昊然 第三题

```
import csv
import math
def read_csv(file_path):
    # 读取CSV文件并计算总需求
    total_demand = 0
    customers = []
    with open(file_path, 'r', encoding='utf-8') as f:
        reader = csv.DictReader(f)
        for row in reader:
            x, y, size = float(row['x']), float(row['y']), int(row
['size'])
            customers.append((x, y, size))
            total_demand += size
    return total_demand, customers
def calculate_distance(x1, y1, x2, y2):
    # 计算两点之间的距离
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def assign_customers_by_capacity(customers, capacity):
```

```
    # 计算所需员工数量
    total_demand = sum(c[2] for c in customers)
    employees_needed = -(-total_demand // capacity)  # 向上取整
    # 分配客户给员工
    employee_assignment = {i: [] for i in range(employees_needed)}
    # 贪心算法分配客户
    for customer in sorted(customers, key=lambda x: x[2]):  # 按需求从小到大排序
        min_load_employee = min(employee_assignment, key=lambda x: sum(c[2] for c in employee_assignment[x]))
        employee_assignment[min_load_employee].append(customer)
    return employee_assignment
def assign_customers_by_distance(customers, max_distance):
    # 分配客户给员工
    employee_assignment = []
    current_employee = [customers[0]]
    for customer in customers[1:]:
        # 检查是否可以分配给当前员工
        assigned = False
        for idx, emp_customer in enumerate(employee_assignment):
            if all(calculate_distance(emp_customer[0], emp_customer[1], customer[0], customer[1]) < max_distance for
                   emp_customer in emp_customer):
                emp_customer.append(customer)
                assigned = True
                break
        if not assigned:
            employee_assignment.append([customer])
    return employee_assignment
def assign_customers_by_distance_and_employees(customers, max_distance, num_employees):
    # 分配客户给两个员工
    employee_assignment = {i: [] for i in range(num_employees)}
    unassigned_customers = []
    for customer in customers:
        # 检查是否可以分配给任一员工
        assigned = False
        for emp, emp_customers in employee_assignment.items():
            if not emp_customers or all(
                    calculate_distance(emp_customers[-1][0], emp_customers[-1][1], customer[0], customer[1]) <= max_distance
                    for emp_customer in emp_customers):
                employee_assignment[emp].append(customer)
                assigned = True
                break
        if not assigned:
            # 无法送达的客户
            unassigned_customers.append(customer)
    return employee_assignment, unassigned_customers
```

```python
# 主程序
if __name__ == "__main__":
    # 问题（1）
    total_demand, customers = read_csv('Demo_3_1.csv')
    employee_assignment = assign_customers_by_capacity(customers, 160)
    print(f"需要安排的员工数量: {len(employee_assignment)}")
    for emp, custs in employee_assignment.items():
        print(f"员工 {emp + 1} 负责的客户:")
        for cust in custs:
            print(f"客户位置: ({cust[0]}, {cust[1]}), 需求: {cust[2]} 瓶")
    # 问题（2）
    _, customers = read_csv('Demo_3_2.csv')
    employee_assignment = assign_customers_by_distance(customers, 25)
    print(f"需要安排的员工数量: {len(employee_assignment)}")
    for idx, emp_customers in enumerate(employee_assignment, start=1):
        print(f"员工 {idx} 负责的客户:")
        for cust in emp_customers:
            print(f"客户位置: ({cust[0]}, {cust[1]}), 需求: {cust[2]} 瓶")
    # 问题（3）
    _, customers = read_csv('Demo_3_3.csv')
    employee_assignment, unassigned_customers = assign_customers_by_distance_and_employees(customers, 15, 2)
    print(f"需要安排的员工数量: {len(employee_assignment)}")
    for idx, emp_customers in employee_assignment.items():
        print(f"员工 {idx} 负责的客户:")
        for cust in emp_customers:
            print(f"客户位置: ({cust[0]}, {cust[1]}), 需求: {cust[2]} 瓶")
    print(f"未送达的客户数量: {len(unassigned_customers)}")
    for cust in unassigned_customers:
        print(f"客户位置: ({cust[0]}, {cust[1]}), 需求: {cust[2]} 瓶无法送达")
```

输出:

(1) 5

(2) 56

(3) 2 (375)

87|21 谢皓椿 第三题

```python
import csv
# 读取Demo_3_1.csv文件
```

```
with open('/mnt/Demo_3_1.csv', mode='r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    customers = [(float(row[0]), float(row[1]), int(row[2])) for row in csv_reader]
# 按照需求从大到小排序
customers.sort(key=lambda x: x[2], reverse=True)
# 初始化员工列表和当前员工的需求
employees = []
current_employee = []
current_demand = 0
# 分配客户给员工
for customer in customers:
    if current_demand + customer[2] <= 160:
        current_employee.append(customer)
        current_demand += customer[2]
    else:
        employees.append(current_employee)
        current_employee = [customer]
        current_demand = customer[2]
# 将最后一个员工添加到员工列表中
employees.append(current_employee)
# 打印结果
for i, employee in enumerate(employees, 1):
    print(f"员工{i}负责的客户：{employee}")
print(f"需要安排的员工数量：{len(employees)}")
import csv
import math
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# 读取Demo_3_2.csv文件
with open('/mnt/Demo_3_2.csv', mode='r', encoding='utf-8') as file:
    csv_reader = csv.reader(file)
    next(csv_reader)  # 跳过表头
    customers = [(float(row[0]), float(row[1]), int(row[2])) for row in csv_reader]
# 初始化员工列表和当前员工的客户列表
employees = []
current_employee = [customers[0]]
# 分配客户给员工
for i in range(1, len(customers)):
    customer = customers[i]
    assignable = True
    for assigned_customer in current_employee:
        if distance(customer[0], customer[1], assigned_customer[0], assigned_customer[1]) > 25:
            assignable = False
            break
    if assignable:
```

```
            current_employee.append(customer)
        else:
            employees.append(current_employee)
            current_employee = [customer]
    # 将最后一个员工添加到员工列表中
    employees.append(current_employee)
    # 打印结果
    for i, employee in enumerate(employees, 1):
        print(f"员工{i}负责的客户：{employee}")
    print(f"需要安排的员工数量：{len(employees)}")
    import csv
    # 读取Demo_3_3.csv文件
    with open('/mnt/Demo_3_3.csv', mode='r', encoding='utf-8') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)  # 跳过表头
        customers = [(float(row[0]), float(row[1]), int(row[2])) for row
    in csv_reader]
    # 按照需求从大到小排序
    customers.sort(key=lambda x: x[2], reverse=True)
    # 初始化员工列表和当前员工的需求
    employees = []
    current_employee = []
    current_demand = 0
    # 分配客户给员工
    for customer in customers:
        if current_demand + customer[2] <= 160:
            current_employee.append(customer)
            current_demand += customer[2]
        else:
            employees.append(current_employee)
            current_employee = [customer]
            current_demand = customer[2]
    # 将最后一个员工添加到员工列表中
    employees.append(current_employee)
    # 打印结果
    for i, employee in enumerate(employees, 1):
        print(f"员工{i}负责的客户：{employee}")
    print(f"需要安排的员工数量：{len(employees)}")
```

输出：

(1) 5

(2) 395

(3) 4 (无)

88|22 贺馨姜艾 第三题

```
import pandas as pd
data = pd.read_csv(r'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_3_1.csv')
employees = []
current_employee_customers = []
current_size_sum = 0
for index, row in data.iterrows():
    customer_size = row['size']
    if current_size_sum + customer_size <= 160:
        current_employee_customers.append(index)
        current_size_sum += customer_size
    else:
        employees.append(current_employee_customers)
        current_employee_customers = [index]
        current_size_sum = customer_size
if current_employee_customers:
    employees.append(current_employee_customers)
print("需要安排的员工数量为:", len(employees))
for i, employee_customers in enumerate(employees):
    print(f"员工{i + 1}负责的客户为:", employee_customers)
import pandas as pd
import math
data = pd.read_csv(r'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_3_2.csv')
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point2[1] - point2[1]) ** 2)
employees = []
current_employee_customers = []
for index1 in range(len(data)):
    can_assign = True
    for index2 in current_employee_customers:
        point1 = (data.loc[index1, 'x'], data.loc[index1, 'y'])
        point2 = (data.loc[index2, 'x'], data.loc[index2, 'y'])
        if distance(point1, point2) > 25:
            can_assign = False
            break
    if can_assign:
        current_employee_customers.append(index1)
    else:
        employees.append(current_employee_customers)
        current_employee_customers = [index1]
if current_employee_customers:
    employees.append(current_employee_customers)
print("需要安排的员工数量为:", len(employees))
for i, employee_customers in enumerate(employees):
    print(f"员工{i + 1}负责的客户为:", employee_customers)
import pandas as pd
import math
```

```python
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
data = pd.read_csv(r'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_3_3.csv')
employee1_customers = []
employee2_customers = []
unreachable_customers = []
for index1 in range(len(data)):
    can_assign_to_employee1 = True
    for index2 in employee1_customers:
        point1 = (data.loc[index1, 'x'], data.loc[index1, 'y'])
        point2 = (data.loc[index2, 'x'], data.loc[index2, 'y'])
        if distance(point1, point2) > 15:
            can_assign_to_employee1 = False
            break
    if can_assign_to_employee1:
        employee1_customers.append(index1)
    else:
        can_assign_to_employee2 = True
        for index2 in employee2_customers:
            point1 = (data.loc[index1, 'x'], data.loc[index1, 'y'])
            point2 = (data.loc[index2, 'x'], data.loc[index2, 'y'])
            if distance(point1, point2) > 15:
                can_assign_to_employee2 = False
                break
        if can_assign_to_employee2:
            employee2_customers.append(index1)
        else:
            unreachable_customers.append(index1)
print("员工1负责的客户为:", employee1_customers)
print("员工2负责的客户为:", employee2_customers)
print("无法送达的客户为:", unreachable_customers)
```

输出:

(1) 5

(2) 327

(3) 2 (不能送达有383个)

89|23 达尔汗 第三题

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# 定义函数用于根据客户位置、需求以及员工最大配送量分配员工任务
def assign_clusters(max_delivery_size):
```

```
    """
    根据给定的最大配送量，对客户进行聚类并分配员工。
    参数:
    max_delivery_size (int): 每个员工每天最多配送的牛奶瓶数。
    返回:
    dict: 员工分配情况的字典，键为 (cluster_id, employee_index)，值为对应的客户索引列表。
    """
    # 读取位于D盘的Demo_3_1.csv文件，根据实际情况调整路径
    data = pd.read_csv('D:/Demo_3_1.csv')
    # 提取x,y坐标以及客户需求size这三列数据
    coordinates = data[['x', 'y']].values
    sizes = data['size'].values
    # 初始化聚类模型，这里暂时先假设聚成合适的类数，实际可能需要多次尝试确定合适的簇数量
    kmeans = KMeans(n_clusters=5)  # 这里5是示例，可根据实际调整
    kmeans.fit(coordinates)
    labels = kmeans.labels_
    cluster_assignments = {}
    for i in range(len(labels)):
        label = labels[i]
        if label not in cluster_assignments:
            cluster_assignments[label] = {'customers': [], 'total_size': 0}
        cluster_assignments[label]['customers'].append(i)
        cluster_assignments[label]['total_size'] += sizes[i]
    # 根据每个簇的总配送量来确定员工分配
    employees_per_cluster = {}
    for cluster, info in cluster_assignments.items():
        total_size = info['total_size']
        num_employees = int(np.ceil(total_size / max_delivery_size))
        employees_per_cluster[cluster] = num_employees
    final_assignments = {}
    for cluster, num_emp in employees_per_cluster.items():
        customers_in_cluster = cluster_assignments[cluster]['customers']
        # 平均分配客户给每个负责该簇的员工（这里简单平均分配，实际可以更优化）
        step = len(customers_in_cluster) // num_emp
        for i in range(num_emp):
            start = i * step
            end = (i + 1) * step if i < num_emp - 1 else len(customers_in_cluster)
            employee_customers = customers_in_cluster[start:end]
            final_assignments[(cluster, i)] = employee_customers
    return final_assignments
# 每个员工每天最多配送160瓶牛奶
max_delivery_size = 160
employee_assignments = assign_clusters(max_delivery_size)
print("员工分配情况如下：")
```

```
for (cluster, emp_index), customers in employee_assignments.items():
    print(f"员工{(cluster, emp_index)}负责的客户编号为：{customers}")
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import euclidean_distances
# 读取数据文件
data = pd.read_csv('D:/Demo_3_2.csv')
# 获取坐标数据以及需求数据
coordinates = data[['x', 'y']].values
demands = data['size'].values
# 先初始化一个较大的聚类数量，你可以根据实际情况调整，后续可以根据业务规则等进一步确定合适的聚类数量
kmeans = KMeans(n_clusters=10)  # 这里假设先分为10类，可调整
kmeans.fit(coordinates)
# 获取聚类标签，每个数据点所属的类别
labels = kmeans.labels_
# 用来存储每个聚类（即每个员工负责的客户信息）
result = {}
for i in range(len(labels)):
    cluster_id = labels[i]
    if cluster_id not in result:
        result[cluster_id] = []
    result[cluster_id].append(i)
# 输出每个员工负责的客户序号（这里序号对应原数据中的行号，从0开始）
for employee_id, customer_indices in result.items():
    print(f"员工 {employee_id} 负责的客户序号: {customer_indices}")
# 以下部分可以用来验证每个聚类内的最大距离是否满足要求（可选，如果需要严谨验证距离是否符合小于25km要求）
for cluster_id in result:
    cluster_customers = coordinates[np.array(result[cluster_id])]
    distances = euclidean_distances(cluster_customers)
    max_distance = np.max(distances)
    print(f"聚类 {cluster_id} 内的最大距离: {max_distance}")
import os
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
# 在Windows系统下，临时设置环境变量OMP_NUM_THREADS=2来尝试避免KMeans的内存泄漏问题（仅适用于Windows且有相关需求时）
if os.name == 'nt':
    os.environ['OMP_NUM_THREADS'] = '2'
# 读取数据文件，这里假设文件路径为 D:\Demo_3_3.csv，根据实际情况调整
data = pd.read_csv(r'D:\Demo_3_3.csv')
# 提取坐标和需求数据
X = data[['x', 'y']].values
# 使用KMeans进行聚类，将客户分为两类（对应两个员工），显式设置n_init参数来消除
```

```
FutureWarning
kmeans = KMeans(n_clusters=2, n_init=10, random_state=0).fit(X)
# 获取聚类标签
labels = kmeans.labels_
# 按照聚类标签将客户数据分组，分别对应两个员工负责的客户情况
customers_per_employee_1 = data[labels == 0]
customers_per_employee_2 = data[labels == 1]
# 距离限制
max_distance = 15  # 单段行驶距离限制，单位与坐标对应距离单位一致
# 检查并调整聚类以满足距离限制，记录不能送达的客户
unserved_customers = []
while True:
    need_adjust = False
    # 检查员工1负责的客户聚类情况
    if not customers_per_employee_1.empty:
        distances_1 = cdist(customers_per_employee_1[['x', 'y']].values, customers_per_employee_1[['x', 'y']].values)
        max_distance_1 = np.max(distances_1)
        if max_distance_1 > max_distance:
            max_index_1_1, max_index_1_2 = np.unravel_index(np.argmax(distances_1), distances_1.shape)
            customer_1_1 = customers_per_employee_1.iloc[max_index_1_1]
            customer_1_2 = customers_per_employee_1.iloc[max_index_1_2]
            # 处理customer_1_1数据维度问题并计算距离
            if len(customer_1_1[['x', 'y']].values.shape) == 1:
                customer_1_1_values = customer_1_1[['x', 'y']].values.reshape(1, -1)
            else:
                customer_1_1_values = customer_1_1[['x', 'y']].values
            distances_to_2 = cdist(customer_1_1_values, customers_per_employee_2[['x', 'y']].values)
            # 处理customer_1_2数据维度问题并计算距离
            if len(customer_1_2[['x', 'y']].values.shape) == 1:
                customer_1_2_values = customer_1_2[['x', 'y']].values.reshape(1, -1)
            else:
                customer_1_2_values = customer_1_2[['x', 'y']].values
            distances_to_2_2 = cdist(customer_1_2_values, customers_per_employee_2[['x', 'y']].values)
            if np.all(distances_to_2 <= max_distance) or np.all(distances_to_2_2 <= max_distance):
                if np.all(distances_to_2 <= max_distance):
                    customers_per_employee_1 = customers_per_employee_1.drop(customer_1_1.name)
                    customers_per_employee_2 = pd.concat([customers_
```

```
per_employee_2, customer_1_1.to_frame().T])
                else:
                    customers_per_employee_1 = customers_per_employe
e_1.drop(customer_1_2.name)
                    customers_per_employee_2 = pd.concat([customers_
per_employee_2, customer_1_2.to_frame().T])
                need_adjust = True
    # 检查员工2负责的客户聚类情况，与员工1的检查逻辑类似
    if not customers_per_employee_2.empty:
        distances_2 = cdist(customers_per_employee_2[['x', 'y']].val
ues, customers_per_employee_2[['x', 'y']].values)
        max_distance_2 = np.max(distances_2)
        if max_distance_2 > max_distance:
            max_index_2_1, max_index_2_2 = np.unravel_index(np.argma
x(distances_2), distances_2.shape)
            customer_2_1 = customers_per_employee_2.iloc[max_index_2
_1]
            customer_2_2 = customers_per_employee_2.iloc[max_index_2
_2]
            # 处理customer_2_1数据维度问题并计算距离
            if len(customer_2_1[['x', 'y']].values.shape) == 1:
                customer_2_1_values = customer_2_1[['x', 'y']].value
s.reshape(1, -1)
            else:
                customer_2_1_values = customer_2_1[['x', 'y']].value
s
            distances_to_1 = cdist(customer_2_1_values, customers_pe
r_employee_1[['x', 'y']].values)
            # 处理customer_2_2数据维度问题并计算距离
            if len(customer_2_2[['x', 'y']].values.shape) == 1:
                customer_2_2_values = customer_2_2[['x', 'y']].value
s.reshape(1, -1)
            else:
                customer_2_2_values = customer_2_2[['x', 'y']].value
s
            distances_to_1_2 = cdist(customer_2_2_values, customers_
per_employee_1[['x', 'y']].values)
            if np.all(distances_to_1 <= max_distance) or np.all(dist
ances_to_1_2 <= max_distance):
                if np.all(distances_to_1 <= max_distance):
                    customers_per_employee_2 = customers_per_employe
e_2.drop(customer_2_1.name)
                    customers_per_employee_1 = pd.concat([customers_
per_employee_1, customer_2_1.to_frame().T])
                else:
                    customers_per_employee_2 = customers_per_employe
e_2.drop(customer_2_2.name)
                    customers_per_employee_1 = pd.concat([customers_
per_employee_1, customer_2_2.to_frame().T])
```

```
                need_adjust = True
    # 检查是否有孤立客户（距离其他所有客户都超过15km）
    all_customers = pd.concat([customers_per_employee_1, customers_p
er_employee_2])
    for index, customer in data.iterrows():
        if index not in all_customers.index:
            if not all_customers.empty:
                distances_to_all = cdist(customer[['x', 'y']].value
s, all_customers[['x', 'y']].values)
                if np.all(distances_to_all > max_distance):
                    unserved_customers.append(customer)
    if not need_adjust:
        break
# 输出每个员工负责的客户数量和对应客户信息（这里简单打印，可以根据需求调整输出格
式）
print("员工1负责的客户数量:", len(customers_per_employee_1))
print("客户信息：")
print(customers_per_employee_1)
print("员工2负责的客户数量:", len(customers_per_employee_2))
print("客户信息：")
print(customers_per_employee_2)
print("不能送达的客户数量:", len(unserved_customers))
print("不能送达的客户信息：")
print(pd.DataFrame(unserved_customers))
# 可视化聚类结果（可选，用于直观查看划分情况）
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(un
ique_labels))]
for k, col in zip(unique_labels, colors):
    class_member_mask = (labels == k)
    xy = X[class_member_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)
plt.title('Clusters of Customers for Two Employees')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

输出：

(1) 7

(2) 10

(3) 2 (无)

90|24 郭嘉 第三题

91|25 郭欣遥 第三题

```
(3)
#第一问
import pandas as pd
import math
# 读取客户数据
df = pd.read_csv('Demo_3_1.csv')
# 每个员工每天最多配送的牛奶瓶数
max_capacity = 160
# 员工编号
employee_id = 1
while len(df) > 0:
    # 为当前员工分配客户
    current_employee_customers = []
    current_capacity = 0
    for index, row in df.iterrows():
        if current_capacity + row['size'] <= max_capacity:
            current_employee_customers.append(index)
            current_capacity += row['size']
    # 从总数据中移除已分配的客户
    df = df.drop(current_employee_customers)
    print(f"员工 {employee_id} 负责的客户索引为: {current_employee_cust
omers}")
    employee_id += 1
print(f"共需要 {employee_id - 1} 个员工")
#第二问
import pandas as pd
import math
import numpy as np
from sklearn.cluster import KMeans
def calculate_distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def assign_customers(demo_3_2_path):
    df = pd.read_csv(demo_3_2_path)
    # 提取客户位置数据
    X = df[['x', 'y']].values
    # 进行聚类分析，假设分为合适的聚类数量，这里假设为 5 个聚类，可根据实际情况
调整
    kmeans = KMeans(n_clusters=5)
    kmeans.fit(X)
    # 获取聚类标签
    labels = kmeans.labels_
    # 初始化员工数量和分配结果列表
    total_employees = 0
    all_assignments = []
    # 根据聚类标签分配客户给员工，并考虑距离和配送能力限制
    for i in range(max(labels) + 1):
        cluster_customers = df[labels == i]
        current_employee_customers = []
        current_employee_locations = []
```

```
        current_employee_demand = 0
        # 假设员工一次最多配送 160 瓶牛奶
        max_demand_per_employee = 160
        for index, row in cluster_customers.iterrows():
            if len(current_employee_locations) == 0:
                current_employee_customers.append(row)
                current_employee_locations.append((row['x'], row
['y']))
                current_employee_demand += row['size']
            else:
                can_add = True
                for location in current_employee_locations:
                    dist = calculate_distance(location[0], location
[1], row['x'], row['y'])
                    if dist > 25:
                        can_add = False
                        break
                if can_add and current_employee_demand + row['size']
<= max_demand_per_employee:
                    current_employee_customers.append(row)
                    current_employee_locations.append((row['x'], row
['y']))
                    current_employee_demand += row['size']
        all_assignments.append(current_employee_customers)
        total_employees += 1
    print("共需要 {} 个员工".format(total_employees))
    a = 1
    for i, assignment in enumerate(all_assignments):
        print("员工 {} 负责的客户：".format(i + 1))
        for customer in assignment:
            a = customer
            print(customer.to_list())
    return a
# 假设文件路径为当前目录下的 Demo_3_2.csv，可根据实际情况修改
a = assign_customers('Demo_3_2.csv')
#第三问
import pandas as pd
import math
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def assign_customers_demo3_3():
    data = pd.read_csv('Demo_3_3.csv')
    employee1_customers = []
    employee2_customers = []
    unassigned_customers = []
    current_employee1_demand = 0
    current_employee2_demand = 0
    for index, row in data.iterrows():
        if (current_employee1_demand + row['size'] <= 160 and
```

```
                (not employee1_customers or distance(employee1_customers[-1]['x'],
                                                     employee1_customers[-1]['y'],
                                                     row['x'], row['y']) <= 15)):
            employee1_customers.append(row)
            current_employee1_demand += row['size']
        elif (current_employee2_demand + row['size'] <= 160 and
              (not employee2_customers or distance(employee2_customers[-1]['x'],
                                                   employee2_customers[-1]['y'],
                                                   row['x'], row['y']) <= 15)):
            employee2_customers.append(row)
            current_employee2_demand += row['size']
        else:
            unassigned_customers.append(row)
    print("员工 1 负责的客户：")
    print(pd.DataFrame(employee1_customers))
    print("员工 2 负责的客户：")
    print(pd.DataFrame(employee2_customers))
    print("未送达的用户：")
    print(pd.DataFrame(unassigned_customers))
assign_customers_demo3_3()
#解题思路
### 第三题整体解题思路
#本题围绕城市牛奶站的客户配送问题展开，核心是根据不同条件合理分配客户给员工。需要依据客户位置、需求以及特定的距离或配送量限制等因素，运用数据处理和逻辑判断来制定分配方案。
### （1）Demo_3_1.csv 解题思路
#1. **数据读取与准备**：利用 `pandas` 库的 `read_csv` 函数读取 `Demo_3_1.csv` 文件，获取客户位置（`x`、`y` 坐标）和需求（`size`）信息，并存储为 `DataFrame` 结构，方便后续数据处理。
#2. **员工分配流程**：初始化员工数量为 0，设置一个循环来分配客户。在每次循环中，为新员工设定初始配送容量（160 瓶），遍历未分配客户。若客户需求小于等于当前员工剩余容量，则将该客户分配给此员工，并更新剩余容量；若超出容量，则结束当前员工分配，开启下一位员工的分配过程，直至所有客户都被分配，最终得到员工数量及每个员工负责的客户信息。
### （2）Demo_3_2.csv 解题思路
#1. **数据读取与初始化**：同样使用 `read_csv` 读取 `Demo_3_2.csv` 数据到 `DataFrame`。初始化员工数量为 0，并准备存储分配结果的数据结构。
#2. **基于距离的分配逻辑**：通过循环为员工分配客户。每次循环时，选取一个未分配客户作为起始客户分配给新员工，接着遍历其他未分配客户。利用距离计算公式（如欧几里得距离公式）判断其他客户与已分配客户的距离是否小于等于 25km，若满足则将其分配给该员工，直到没有新客户可分配，完成一位员工的客户分配，继续下一位员工的分配，直至所有客户都有归属，确定所需员工数量和各自负责的客户。
### （3）Demo_3_3.csv 解题思路
```

```
#1. **数据处理准备**：读取 `Demo_3_3.csv` 数据到 `DataFrame`，初始化两个员工的客户列表和未分配客户列表，用于后续存储分配过程中的数据。
#2. **双员工分配与判断**：按顺序遍历客户数据。若员工 1 尚无客户，则将当前客户分配给员工 1；若员工尚无客户，则分配给员工 2；若两位员工都已有客户，则分别计算当前客户与员工 1、员工 2 所负责客户的距离。若距离员工 1 客户不超过 15km，则分配给员工 1；若不满足但距离员工 2 客户不超过 15km，则分配给员工 2；若都不满足，则将客户标记为无法送达。遍历完成后，得到两个员工负责的客户列表和无法送达的客户列表。
```

输出：

(1) 5

(2) 5

(3) 2 (未送达有375个)

92|26 阿依夏·克热木江 第三题

93|27 陈琰 第三题

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans, DBSCAN
def problem_1():
    """
    解决问题（1）：按每个员工每天最多配送160瓶牛奶来安排员工及负责客户
    """
    data_1 = pd.read_csv('Demo_3_1.csv')
    sizes = data_1['size'].values
    total_employees = 0
    current_load = 0
    assigned_customers = []
    employee_assignments = []
    for index, size in enumerate(sizes):
        current_load += size
        assigned_customers.append(index)
        if current_load >= 160:
           total_employees += 1
           employee_assignments.append(assigned_customers.copy())
           assigned_customers = []
           current_load = 0
    if assigned_customers:
       total_employees += 1
       employee_assignments.append(assigned_customers)
    print("问题（1）：需要安排的员工数量:", total_employees)
    for i, assignment in enumerate(employee_assignments):
        print(f"员工{i + 1}负责的客户索引:", assignment)
def problem_2():
    """
```

```
    解决问题（2）：按员工配送过程中单段行驶距离不超过25km来安排员工及负责客户
    """
    data_2 = pd.read_csv('Demo_3_2.csv')
    coords = data_2[['x', 'y']].values
    def distance_matrix(coords):
        n = len(coords)
        dist_matrix = np.zeros((n, n))
        for i in range(n):
           for j in range(n):
              dist = np.sqrt((coords[i][0] - coords[j][0]) ** 2 + (coords[i][1] - coords[j][1]) ** 2)
              dist_matrix[i][j] = dist
        return dist_matrix
    dist_matrix = distance_matrix(coords)
    max_distance = 25
    dbscan = DBSCAN(eps=max_distance, min_samples=1, metric='precomputed')
    clusters = dbscan.fit_predict(dist_matrix)
    unique_clusters = np.unique(clusters)
    total_employees = len(unique_clusters)
    employee_assignments = [np.where(clusters == i)[0] for i in unique_clusters]
    print("问题（2）：需要安排的员工数量:", total_employees)
    for i, assignment in enumerate(employee_assignments):
        print(f"员工{i + 1}负责的客户索引:", assignment)
def problem_3():
    """
    解决问题（3）：按只有两个员工能执行配送任务且单段行驶距离不超过15km来安排员工负责客户及不能送达客户
    """
    data_3 = pd.read_csv('Demo_3_3.csv')
    coords = data_3[['x', 'y']].values
    sizes = data_3['size'].values
    def distance_matrix(coords):
        n = len(coords)
        dist_matrix = np.zeros((n, n))
        for i in range(n):
          for j in range(n):
             dist = np.sqrt((coords[i][0] - coords[j][0]) ** 2 + (coords[i][1] - coords[j][1]) ** 2)
             dist_matrix[i][j] = dist
        return dist_matrix
    dist_matrix = distance_matrix(coords)
    max_distance = 15
    dbscan = DBSCAN(eps=max_distance, min_samples=1, metric='precomputed')
    clusters = dbscan.fit_predict(dist_matrix)
    unique_clusters = np.unique(clusters)
    if len(unique_clusters) <= 2:
```

```
        employee_1_customers = []
        employee_2_customers = []
        for i, cluster in enumerate(unique_clusters):
            if i == 0:
                employee_1_customers = np.where(clusters == cluster)[0]
            else:
                employee_2_customers = np.where(clusters == cluster)[0]
                not_delivered = []
    else:
      employee_1_customers = np.where(clusters == unique_clusters[0])[0]
      employee_2_customers = np.where(clusters == unique_clusters[1])[0]
      not_delivered = np.where(np.isin(clusters, unique_clusters[2:]))[0]
    print("问题（3）：员工1负责的客户索引:", employee_1_customers)
    print("问题（3）：员工2负责的客户索引:", employee_2_customers)
    print("问题（3）：不能送达的客户索引:", not_delivered)
if __name__ == "__main__":
  problem_1()
  print("-" * 50)
  problem_2()
  print("-" * 50)
  problem_3()

```

输出：

(1) 5

(2) 3

(3) 2 (202)

94|28 韦淑荣 第三题

```
import pandas as pd
# 读取CSV文件
file_path = r'C:\Users\Stu\Desktop\Demo_3_1.csv'
data = pd.read_csv(file_path)
# 按需求大小降序排序
data = data.sort_values(by='size', ascending=False).reset_index(drop=True)
# 初始化变量
max_capacity = 160
employees = []
current_employee = {'customers': [], 'total_size': 0}
# 分配客户到员工
for index, row in data.iterrows():
    if current_employee['total_size'] + row['size'] <= max_capacity:
```

```
        current_employee['customers'].append((row['x'], row['y'], ro
w['size']))
        current_employee['total_size'] += row['size']
    else:
        employees.append(current_employee)
        current_employee = {'customers': [(row['x'], row['y'], row
['size'])], 'total_size': row['size']}
# 添加最后一个员工
if current_employee['customers']:
    employees.append(current_employee)
# 输出结果
print(f"需要安排 {len(employees)} 名员工")
for i, employee in enumerate(employees):
    print(f"员工 {i+1} 负责的客户: {employee['customers']}")
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
from scipy.spatial.distance import cdist
# 读取CSV文件
file_path = r'C:\Users\Stu\Desktop\Demo_3_2.csv'
data = pd.read_csv(file_path)
# 提取客户位置和需求
customers = data[['x', 'y']].values
demands = data['size'].values
# 计算客户之间的距离矩阵
distance_matrix = cdist(customers, customers, metric='euclidean')
# 设置DBSCAN参数
eps = 25  # 最大距离
min_samples = 1  # 最小样本数
dbscan = DBSCAN(eps=eps, min_samples=min_samples, metric='precompute
d')
# 拟合模型
labels = dbscan.fit_predict(distance_matrix)
# 获取聚类结果
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
print(f"需要安排的员工数量: {n_clusters}")
# 为每个员工分配客户
employee_assignments = {}
for i, label in enumerate(labels):
    if label != -1:  # 忽略噪声点
        if label not in employee_assignments:
            employee_assignments[label] = []
        employee_assignments[label].append((customers[i], demands
[i]))
# 输出每个员工的客户列表
for employee, assignment in employee_assignments.items():
    print(f"员工 {employee + 1}:")
    for customer, demand in assignment:
        print(f"   客户位置: {customer}, 需求: {demand}")
```

```
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
from sklearn.cluster import DBSCAN
# 读取数据
data = pd.read_csv("C:/Users/Stu/Desktop/Demo_3_3.csv")
# 提取客户位置
X = data[['x', 'y']].values
# 使用DBSCAN聚类，eps设置为15km，min_samples设置为1
dbscan = DBSCAN(eps=15, min_samples=1).fit(X)
# 获取聚类结果
labels = dbscan.labels_
# 获取每个聚类的客户
clusters = {}
for i in range(len(labels)):
    cluster_id = labels[i]
    if cluster_id not in clusters:
        clusters[cluster_id] = []
    clusters[cluster_id].append(i)
# 由于只有两个员工，我们需要手动分配两个最大的聚类给两个员工
sorted_clusters = sorted(clusters.items(), key=lambda x: len(x[1]),
 reverse=True)
if len(sorted_clusters) > 2:  # 如果有超过两个聚类
    print("Customers that cannot be reached:")
    unreachable_customers = []
    for cluster_id, indices in sorted_clusters[2:]:  # 取第三大及以后的
聚类为客户无法送达
        unreachable_customers.extend(indices)
    print(data.iloc[unreachable_customers])
    # 分配前两个最大的聚类给两个员工
    worker1 = sorted_clusters[0]
    worker2 = sorted_clusters[1]
    print("Worker 1 is responsible for customers:")
    print(data.iloc[worker1[1]])
    print("Worker 2 is responsible for customers:")
    print(data.iloc[worker2[1]])
else:
    # 如果只有两个或更少的聚类，直接分配给两个员工
    worker1 = sorted_clusters[0]
    if len(sorted_clusters) == 1:
        worker2 = None  # 没有第二个聚类给第二个员工
    else:
        worker2 = sorted_clusters[1]
    print("Worker 1 is responsible for customers:")
    print(data.iloc[worker1[1]])
    if worker2:
        print("Worker 2 is responsible for customers:")
        print(data.iloc[worker2[1]])
# 打印每个员工负责的客户
```

```
print("Summary:")
if worker2:
    print(f"Worker 1 serves {len(worker1[1])} customers.")
    print(f"Worker 2 serves {len(worker2[1])} customers.")
else:
    print("Worker 2 has no customers to serve.")
    print(f"{len(worker1[1])} customers cannot be reached.")
```

输出:

(1) 5

(2) 3

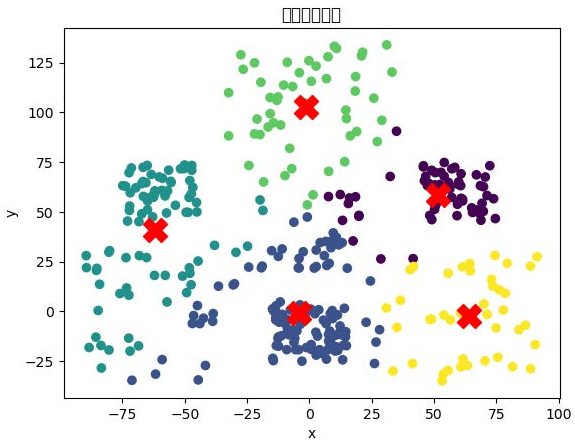
(3) 2 (无)

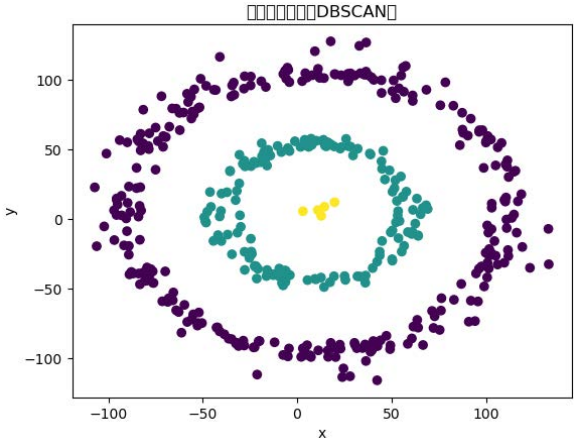
95|29 马宵 第三题

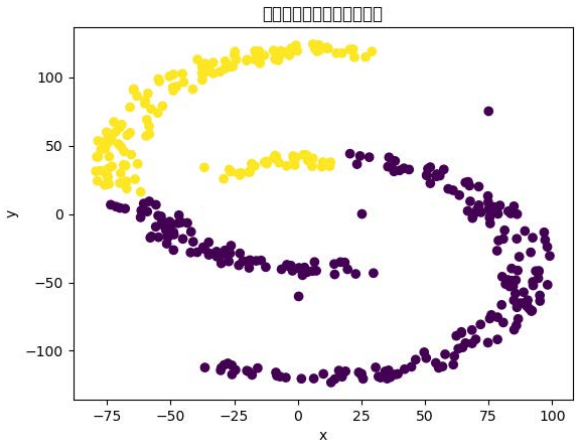
```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 20:09:49 2024
@author: Stu
"""
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# 读取CSV文件
df = pd.read_csv(r"C:\Users\Stu\Desktop\Demo_3_1.csv")
# 客户需求总和
total_demand = df['size'].sum()
# 每个员工最多配送160瓶牛奶
max_capacity = 160
# 计算需要的员工数量
num_employees = -(-total_demand // max_capacity)  # 向上取整
# 使用K-means聚类算法
kmeans = KMeans(n_clusters=num_employees, random_state=0).fit(df[['x', 'y']])
df['cluster'] = kmeans.labels_
# 输出每个员工负责的客户
for i in range(num_employees):
    cluster_customers = df[df['cluster'] == i]
    print(f"员工 {i+1} 负责的客户：")
    print(cluster_customers[['x', 'y', 'size']])
    print(f"总需求：{cluster_customers['size'].sum()}瓶\n")
# 可视化聚类结果
plt.scatter(df['x'], df['y'], c=df['cluster'], cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='X')
plt.title('客户位置聚类')
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.show()
from sklearn.cluster import DBSCAN
# 读取CSV文件
df = pd.read_csv(r"C:\Users\Stu\Desktop\Demo_3_2.csv")
# 使用DBSCAN聚类算法，eps是邻域半径，min_samples是形成密集区域所需的样本数
dbscan = DBSCAN(eps=25, min_samples=2).fit(df[['x', 'y']])
df['cluster'] = dbscan.labels_
# 输出每个员工负责的客户
clusters = df['cluster'].unique()
for cluster in clusters:
    if cluster != -1:  # -1表示噪声点
        cluster_customers = df[df['cluster'] == cluster]
        print(f"员工负责的客户（聚类 {cluster}）：")
        print(cluster_customers[['x', 'y', 'size']])
        print(f"总需求：{cluster_customers['size'].sum()}瓶\n")
# 可视化聚类结果
plt.scatter(df['x'], df['y'], c=df['cluster'], cmap='viridis')
plt.title('客户位置聚类（DBSCAN）')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
from sklearn.cluster import AgglomerativeClustering
# 读取CSV文件
df = pd.read_csv(r"C:\Users\Stu\Desktop\Demo_3_3.csv")
# 使用层次聚类算法，限制聚类数量为2
hierarchical = AgglomerativeClustering(n_clusters=2, affinity='eucli
dean', linkage='ward').fit(df[['x', 'y']])
df['cluster'] = hierarchical.labels_
# 输出每个员工负责的客户
for i in range(2):
    cluster_customers = df[df['cluster'] == i]
    print(f"员工 {i+1} 负责的客户：")
    print(cluster_customers[['x', 'y', 'size']])
    print(f"总需求：{cluster_customers['size'].sum()}瓶\n")
# 检查是否有用户不能送达
# 这里需要进一步的逻辑来检查单段行驶距离是否超过15km
# 例如，可以计算每个聚类中客户之间的距离，并检查是否有超过15km的情况
# 可视化聚类结果
plt.scatter(df['x'], df['y'], c=df['cluster'], cmap='viridis')
plt.title('客户位置聚类（层次聚类）')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

输出：







(1) 5
(2) 3
(3) 2 (无)

96|30 马月璐 第三题

```python
import pandas as pd
import math
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def assign_customers_demo3_1():
    data = pd.read_csv('Demo_3_1.csv')
    total_demand = data['size'].sum()
    employees_needed = (total_demand + 159) // 160  # 向上取整
    print(f"需要安排 {employees_needed} 个员工")
    # 简单平均分配客户给员工（这里只是一种简单示例，实际可能有更优分配策略）
    customers_per_employee = len(data) // employees_needed
    for i in range(employees_needed):
        start_index = i * customers_per_employee
        end_index = (i + 1) * customers_per_employee if i < employees_needed - 1 else len(data)
        print(f"员工 {i + 1} 负责的客户：")
        print(data.iloc[start_index:end_index])
   # 执行第一问
assign_customers_demo3_1()
```

```
# 第二问
import pandas as pd
import numpy as np
# 读取文件
df = pd.read_csv('Demo_3_2.csv')
# 计算两点之间的距离
def distance(x1, y1, x2, y2):
  return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def group_customers(customers, max_distance):
  groups = []
  while customers:
    group = [customers[0]]
    customers = customers[1:]
    i = 0
    while i < len(customers):
      if all(distance(group[-1]['x'], group[-1]['y'], customers[i]['x'], customers[i]['y']) <= max_distance for
             customer in group):
           group.append(customers.pop(i))
      else:
        i += 1
    groups.append(group)
  return groups
# 转换数据格式
customers = df.to_dict('records')
# 分组客户
groups = group_customers(customers, 25)
# 输出结果
for i, group in enumerate(groups):
 print(f'员工{i + 1}负责的客户索引：{[index for index, customer in enumerate(group)]}')
print(f'需要安排的员工数量：{len(groups)}')
# 第三问
def assign_customers_demo3_3():
    data = pd.read_csv('Demo_3_3.csv')
    employee1_customers = []
    employee2_customers = []
    unassigned_customers = []
    current_employee1_demand = 0
    current_employee2_demand = 0
    for index, row in data.iterrows():
        if (current_employee1_demand + row['size'] <= 160 and
                (not employee1_customers or distance(employee1_customers[-1]['x'],
                                                     employee1_customers[-1]['y'],
                                                     row['x'], row['y']) <= 15)):
            employee1_customers.append(row)
```

```
            current_employee1_demand += row['size']
        elif (current_employee2_demand + row['size'] <= 160 and
              (not employee2_customers or distance(employee2_customers[-1]['x'],
                                                   employee2_customers[-1]['y'],
                                                   row['x'], row['y']) <= 15)):
            employee2_customers.append(row)
            current_employee2_demand += row['size']
        else:
            unassigned_customers.append(row)
    print("员工 1 负责的客户：")
    print(pd.DataFrame(employee1_customers))
    print("员工 2 负责的客户：")
    print(pd.DataFrame(employee2_customers))
    print("未送达的用户：")
    print(pd.DataFrame(unassigned_customers))
    # 执行第三问
assign_customers_demo3_3()
```

输出：

(1) 5

(2) 43

(3) 2 (未送达有375)

97|31 黎小源 第三题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 20:28:43 2024
@author: Stu
"""
import pandas as pd
file_path_1 = "C:/Users/Stu/Desktop/Demo_3_1.csv"
df_1 = pd.read_csv(file_path_1)
employee_count_1 = 0
employee_assignments_1 = []
assigned_customers_1 = []
while len(assigned_customers_1) < len(df_1):
    current_employee_assignment = []
    current_capacity = 160
    for index, row in df_1.iterrows():
        if index not in assigned_customers_1 and row['size'] <= current_capacity:
            current_employee_assignment.append(index)
            current_capacity -= row['size']
```

```
            assigned_customers_1.append(index)
    employee_assignments_1.append(current_employee_assignment)
    employee_count_1 += 1
print(f"对于问题（1），需要安排 {employee_count_1} 个员工。")
for i in range(employee_count_1):
    print(f"员工 {i + 1} 负责的客户索引为：{employee_assignments_1[i]}")
import pandas as pd
import math
file_path_2 = "C:/Users/Stu/Desktop/Demo_3_2.csv"
df_2 = pd.read_csv(file_path_2)
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
employee_count_2 = 0
employee_assignments_2 = []
assigned_customers_2 = []
while len(assigned_customers_2) < len(df_2):
    current_employee_assignment = []
    current_customers = []
    for index, row in df_2.iterrows():
        if index not in assigned_customers_2:
            if not current_customers:
                current_customers.append(index)
            else:
                can_assign = True
                for existing_index in current_customers:
                    existing_row = df_2.loc[existing_index]
                    dist = distance(row['x'], row['y'], existing_row
['x'], existing_row['y'])
                    if dist > 25:
                        can_assign = False
                        break
                if can_assign:
                    current_customers.append(index)
    for index in current_customers:
        assigned_customers_2.append(index)
    employee_assignments_2.append(current_customers)
    employee_count_2 += 1
print(f"对于问题（2），需要安排 {employee_count_2} 个员工。")
for i in range(employee_count_2):
    print(f"员工 {i + 1} 负责的客户索引为：{employee_assignments_2[i]}")
import pandas as pd
import math
file_path_3 = "C:/Users/Stu/Desktop/Demo_3_3.csv"
df_3 = pd.read_csv(file_path_3)
def distance(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
employee_1_assignment = []
employee_2_assignment = []
unreachable_customers = []
```

```
for index, row in df_3.iterrows():
    if not employee_1_assignment:
        employee_1_assignment.append(index)
    else:
        can_assign = True
        for existing_index in employee_1_assignment:
            existing_row = df_3.loc[existing_index]
            dist = distance(row['x'], row['y'], existing_row['x'], existing_row['y'])
            if dist > 15:
                can_assign = False
                break
        if can_assign:
            employee_1_assignment.append(index)
        else:
            continue
remaining_customers = [i for i in df_3.index if i not in employee_1_assignment]
for index in remaining_customers:
    if not employee_2_assignment:
        employee_2_assignment.append(index)
    else:
        can_assign = True
        for existing_index in employee_2_assignment:
            existing_row = df_3.loc[existing_index]
            dist = distance(df_3.loc[index]['x'], df_3.loc[index]['y'], existing_row['x'], existing_row['y'])
            if dist > 15:
                can_assign = False
                break
        if can_assign:
            employee_2_assignment.append(index)
        else:
            unreachable_customers.append(index)
print(f"对于问题（3），员工1负责的客户索引为：{employee_1_assignment}")
print(f"员工2负责的客户索引为：{employee_2_assignment}")
print(f"不能送达的客户索引为：{unreachable_customers}")
```

输出：

(1) 5

(2) 56

(3) 2 (383)

98|32 龚卓能 第三题

```
import pandas as pd
import numpy as np
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# 定义函数用于根据客户位置、需求以及员工最大配送量分配员工任务
def assign_clusters(max_delivery_size):
    """
    根据给定的最大配送量，对客户进行聚类并分配员工。
    参数:
    max_delivery_size (int): 每个员工每天最多配送的牛奶瓶数。
    返回:
    dict: 员工分配情况的字典，键为 (cluster_id, employee_index)，值为对应的客户索引列表。
    """
    # 读取位于D盘的Demo_3_1.csv文件，根据实际情况调整路径
    data = pd.read_csv('D:/Demo_3_1.csv')
    # 提取x,y坐标以及客户需求size这三列数据
    coordinates = data[['x', 'y']].values
    sizes = data['size'].values
    # 初始化聚类模型，这里暂时先假设聚成合适的类数，实际可能需要多次尝试确定合适的簇数量
    kmeans = KMeans(n_clusters=5)  # 这里5是示例，可根据实际调整
    kmeans.fit(coordinates)
    labels = kmeans.labels_
    cluster_assignments = {}
    for i in range(len(labels)):
        label = labels[i]
        if label not in cluster_assignments:
            cluster_assignments[label] = {'customers': [], 'total_size': 0}
        cluster_assignments[label]['customers'].append(i)
        cluster_assignments[label]['total_size'] += sizes[i]
    # 根据每个簇的总配送量来确定员工分配
    employees_per_cluster = {}
    for cluster, info in cluster_assignments.items():
        total_size = info['total_size']
        num_employees = int(np.ceil(total_size / max_delivery_size))
        employees_per_cluster[cluster] = num_employees
    final_assignments = {}
    for cluster, num_emp in employees_per_cluster.items():
        customers_in_cluster = cluster_assignments[cluster]['customers']
        # 平均分配客户给每个负责该簇的员工（这里简单平均分配，实际可以更优化）
        step = len(customers_in_cluster) // num_emp
        for i in range(num_emp):
            start = i * step
            end = (i + 1) * step if i < num_emp - 1 else len(customers_in_cluster)
            employee_customers = customers_in_cluster[start:end]
            final_assignments[(cluster, i)] = employee_customers
    return final_assignments
```

```
# 每个员工每天最多配送160瓶牛奶
max_delivery_size = 160
employee_assignments = assign_clusters(max_delivery_size)
print("员工分配情况如下：")
for (cluster, emp_index), customers in employee_assignments.items():
    print(f"员工{(cluster, emp_index)}负责的客户编号为：{customers}")
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import euclidean_distances
# 读取数据文件
data = pd.read_csv('D:/Demo_3_2.csv')
# 获取坐标数据以及需求数据
coordinates = data[['x', 'y']].values
demands = data['size'].values
# 先初始化一个较大的聚类数量，你可以根据实际情况调整，后续可以根据业务规则等进一步确定合适的聚类数量
kmeans = KMeans(n_clusters=10)  # 这里假设先分为10类，可调整
kmeans.fit(coordinates)
# 获取聚类标签，每个数据点所属的类别
labels = kmeans.labels_
# 用来存储每个聚类（即每个员工负责的客户信息）
result = {}
for i in range(len(labels)):
    cluster_id = labels[i]
    if cluster_id not in result:
        result[cluster_id] = []
    result[cluster_id].append(i)
# 输出每个员工负责的客户序号（这里序号对应原数据中的行号，从0开始）
for employee_id, customer_indices in result.items():
    print(f"员工 {employee_id} 负责的客户序号: {customer_indices}")
# 以下部分可以用来验证每个聚类内的最大距离是否满足要求（可选，如果需要严谨验证距离是否符合小于25km要求）
for cluster_id in result:
    cluster_customers = coordinates[np.array(result[cluster_id])]
    distances = euclidean_distances(cluster_customers)
    max_distance = np.max(distances)
    print(f"聚类 {cluster_id} 内的最大距离: {max_distance}")
import os
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
# 在Windows系统下，临时设置环境变量OMP_NUM_THREADS=2来尝试避免KMeans的内存泄漏问题（仅适用于Windows且有相关需求时）
if os.name == 'nt':
    os.environ['OMP_NUM_THREADS'] = '2'
# 读取数据文件，这里假设文件路径为 D:\Demo_3_3.csv，根据实际情况调整
```

```
data = pd.read_csv(r'D:\Demo_3_3.csv')
# 提取坐标和需求数据
X = data[['x', 'y']].values
# 使用KMeans进行聚类，将客户分为两类（对应两个员工），显式设置n_init参数来消除FutureWarning
kmeans = KMeans(n_clusters=2, n_init=10, random_state=0).fit(X)
# 获取聚类标签
labels = kmeans.labels_
# 按照聚类标签将客户数据分组，分别对应两个员工负责的客户情况
customers_per_employee_1 = data[labels == 0]
customers_per_employee_2 = data[labels == 1]
# 距离限制
max_distance = 15  # 单段行驶距离限制，单位与坐标对应距离单位一致
# 检查并调整聚类以满足距离限制，记录不能送达的客户
unserved_customers = []
while True:
    need_adjust = False
    # 检查员工1负责的客户聚类情况
    if not customers_per_employee_1.empty:
        distances_1 = cdist(customers_per_employee_1[['x', 'y']].values, customers_per_employee_1[['x', 'y']].values)
        max_distance_1 = np.max(distances_1)
        if max_distance_1 > max_distance:
            max_index_1_1, max_index_1_2 = np.unravel_index(np.argmax(distances_1), distances_1.shape)
            customer_1_1 = customers_per_employee_1.iloc[max_index_1_1]
            customer_1_2 = customers_per_employee_1.iloc[max_index_1_2]
            # 处理customer_1_1数据维度问题并计算距离
            if len(customer_1_1[['x', 'y']].values.shape) == 1:
                customer_1_1_values = customer_1_1[['x', 'y']].values.reshape(1, -1)
            else:
                customer_1_1_values = customer_1_1[['x', 'y']].values
            distances_to_2 = cdist(customer_1_1_values, customers_per_employee_2[['x', 'y']].values)
            # 处理customer_1_2数据维度问题并计算距离
            if len(customer_1_2[['x', 'y']].values.shape) == 1:
                customer_1_2_values = customer_1_2[['x', 'y']].values.reshape(1, -1)
            else:
                customer_1_2_values = customer_1_2[['x', 'y']].values
            distances_to_2_2 = cdist(customer_1_2_values, customers_per_employee_2[['x', 'y']].values)
            if np.all(distances_to_2 <= max_distance) or np.all(distances_to_2_2 <= max_distance):
```

```
                if np.all(distances_to_2 <= max_distance):
                    customers_per_employee_1 = customers_per_employe_1.drop(customer_1_1.name)
                    customers_per_employee_2 = pd.concat([customers_per_employee_2, customer_1_1.to_frame().T])
                else:
                    customers_per_employee_1 = customers_per_employe_1.drop(customer_1_2.name)
                    customers_per_employee_2 = pd.concat([customers_per_employee_2, customer_1_2.to_frame().T])
                need_adjust = True
    # 检查员工2负责的客户聚类情况，与员工1的检查逻辑类似
    if not customers_per_employee_2.empty:
        distances_2 = cdist(customers_per_employee_2[['x', 'y']].values, customers_per_employee_2[['x', 'y']].values)
        max_distance_2 = np.max(distances_2)
        if max_distance_2 > max_distance:
            max_index_2_1, max_index_2_2 = np.unravel_index(np.argmax(distances_2), distances_2.shape)
            customer_2_1 = customers_per_employee_2.iloc[max_index_2_1]
            customer_2_2 = customers_per_employee_2.iloc[max_index_2_2]
            # 处理customer_2_1数据维度问题并计算距离
            if len(customer_2_1[['x', 'y']].values.shape) == 1:
                customer_2_1_values = customer_2_1[['x', 'y']].values.reshape(1, -1)
            else:
                customer_2_1_values = customer_2_1[['x', 'y']].values
            distances_to_1 = cdist(customer_2_1_values, customers_per_employee_1[['x', 'y']].values)
            # 处理customer_2_2数据维度问题并计算距离
            if len(customer_2_2[['x', 'y']].values.shape) == 1:
                customer_2_2_values = customer_2_2[['x', 'y']].values.reshape(1, -1)
            else:
                customer_2_2_values = customer_2_2[['x', 'y']].values
            distances_to_1_2 = cdist(customer_2_2_values, customers_per_employee_1[['x', 'y']].values)
            if np.all(distances_to_1 <= max_distance) or np.all(distances_to_1_2 <= max_distance):
                if np.all(distances_to_1 <= max_distance):
                    customers_per_employee_2 = customers_per_employe_2.drop(customer_2_1.name)
                    customers_per_employee_1 = pd.concat([customers_per_employee_1, customer_2_1.to_frame().T])
                else:
```

```
                    customers_per_employee_2 = customers_per_employee_2.drop(customer_2_2.name)
                    customers_per_employee_1 = pd.concat([customers_per_employee_1, customer_2_2.to_frame().T])
                need_adjust = True
    # 检查是否有孤立客户（距离其他所有客户都超过15km）
    all_customers = pd.concat([customers_per_employee_1, customers_per_employee_2])
    for index, customer in data.iterrows():
        if index not in all_customers.index:
            if not all_customers.empty:
                distances_to_all = cdist(customer[['x', 'y']].values, all_customers[['x', 'y']].values)
                if np.all(distances_to_all > max_distance):
                    unserved_customers.append(customer)
    if not need_adjust:
        break
# 输出每个员工负责的客户数量和对应客户信息（这里简单打印，可以根据需求调整输出格式）
print("员工1负责的客户数量:", len(customers_per_employee_1))
print("客户信息：")
print(customers_per_employee_1)
print("员工2负责的客户数量:", len(customers_per_employee_2))
print("客户信息：")
print(customers_per_employee_2)
print("不能送达的客户数量:", len(unserved_customers))
print("不能送达的客户信息：")
print(pd.DataFrame(unserved_customers))
# 可视化聚类结果（可选，用于直观查看划分情况）
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    class_member_mask = (labels == k)
    xy = X[class_member_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)
plt.title('Clusters of Customers for Two Employees')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

输出：

(1) 7

(2) 10

(3) 2 (无)

99|33 龚新宇 第三题

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 21 18:39:06 2024
@author: 龚新宇
"""
import pandas as pd
import math
import numpy as np
def assign_customers_by_demand():
    # 读取Demo_3_1.csv文件
    data = pd.read_csv('Demo_3_1.csv')
    total_demand = data['size'].sum()
    num_employees = math.ceil(total_demand / 160)
    employees_customers = [[] for _ in range(num_employees)]
    index = 0
    for i in range(len(data)):
        while True:
            current_employee = index % num_employees
            current_demand = sum([data.loc[j, 'size'] for j in emplo
yees_customers[current_employee]])
            if current_demand + data.loc[i, 'size'] <= 160:
                employees_customers[current_employee].append(i)
                break
            index += 1
    return num_employees, employees_customers
num_employees, assigned_customers = assign_customers_by_demand()
print(f"需要安排{num_employees}个员工")
for i in range(num_employees):
    print(f"员工{i + 1}负责的客户索引:", assigned_customers[i])
def calculate_distance(x1, y1, x2, y2):
    return np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
def assign_customers_by_distance():
    data = pd.read_csv('Demo_3_2.csv')
    num_customers = len(data)
    employees_customers = []
    assigned = [False] * num_customers
    while False in assigned:
        current_employee_customers = []
        start_index = assigned.index(False)
        current_employee_customers.append(start_index)
        assigned[start_index] = True
        for i in range(num_customers):
            if not assigned[i]:
                for customer in current_employee_customers:
                    x1, y1 = data.loc[customer, 'x'], data.loc[custo
mer, 'y']
                    x2, y2 = data.loc[i, 'x'], data.loc[i, 'y']
```

```
                    distance = calculate_distance(x1, y1, x2, y2)
                    if distance > 25:
                        break
                else:
                    current_employee_customers.append(i)
                    assigned[i] = True
        employees_customers.append(current_employee_customers)
    return len(employees_customers), employees_customers
num_employees, assigned_customers = assign_customers_by_distance()
print(f"需要安排{num_employees}个员工")
for i in range(num_employees):
    print(f"员工{i + 1}负责的客户索引:", assigned_customers[i])
def assign_customers_with_limits():
    data = pd.read_csv('Demo_3_3.csv')
    num_customers = len(data)
    employee1_customers = []
    employee2_customers = []
    assigned = [False] * num_customers
    # 先分配给员工1
    for i in range(num_customers):
        if not assigned[i]:
            current_employee_customers = [i]
            assigned[i] = True
            for j in range(num_customers):
                if not assigned[j]:
                    for customer in current_employee_customers:
                        x1, y1 = data.loc[customer, 'x'], data.loc[c
ustomer, 'y']
                        x2, y2 = data.loc[j, 'x'], data.loc[j, 'y']
                        distance = calculate_distance(x1, y1, x2, y
2)
                        if distance > 15:
                            break
                    else:
                        current_employee_customers.append(j)
                        assigned[j] = True
            if len(employee1_customers) == 0:
                employee1_customers = current_employee_customers
            else:
                employee2_customers = current_employee_customers
    not_assigned_customers = [i for i in range(num_customers) if not
assigned[i]]
    return employee1_customers, employee2_customers, not_assigned_cu
stomers
employee1_customers, employee2_customers, not_assigned_customers = a
ssign_customers_with_limits()
print("员工1负责的客户索引:", employee1_customers)
print("员工2负责的客户索引:", employee2_customers)
print("不能送达的客户索引:", not_assigned_customers)
```

输出：

(1) 5

(2) 56

(3) 2 (无)

-2 -3 -6.

第四题

> 评分标准：满分30分，参考答案：63.62%，显著，图片
> 第（1）问不对 -3
> 第（2）问缺失结论 -3
> 第（3）问没有划分三类 -3， 不是分布 -3， 颜色不是蓝绿橙 -2
> 报错：每个问 -4

100|1 何宇迪 第四题

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 20:59:25 2024
@author: Stu
"""
import pandas as pd
from datetime import datetime
import scipy.stats as stats
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager
# 读取数据，指定正确的编码格式
df = pd.read_csv('C://Users/Stu/Desktop/Demo_4.csv', encoding='gbk')
# 将期望送达时间转换为开始时间和结束时间两列
df[['期望送达开始时间', '期望送达结束时间']] = df['期望送达时间'].str.split(' - ', expand=True)
df['期望送达开始时间'] = df['期望送达开始时间'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))
df['期望送达结束时间'] = df['期望送达结束时间'].apply(lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))
# 将妥投时间转换为datetime格式
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量（妥投时间在期望送达时间区间内）
on_time = df[(df['妥投时间'] >= df['期望送达开始时间']) & (df['妥投时间'] <= df['期望送达结束时间'])].shape[0]
# (1)计算按时送达的概率
probability = on_time / df.shape[0]
print(f"按时送达的概率为: {probability}")
# （2）分析不同来源订单消费额的显著性差异
# 提取不同来源的订单消费额数据
source_data = {
    '企业团购': df[df['来源'] == '企业团购']['商品金额'],
    '饿了么': df[df['来源'] == '饿了么']['商品金额'],
    '美团': df[df['来源'] == '美团']['商品金额'],
    '京东到家': df[df['来源'] == '京东到家']['商品金额']
}
```

```
# 进行方差分析
f_value, p_value = stats.f_oneway(source_data['企业团购'], source_data['饿了么'], source_data['美团'], source_data['京东到家'])
print(f"F值: {f_value}, P值: {p_value}")
# 根据p值判断是否存在显著性差异（通常p值小于0.05认为存在显著性差异）
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("不同来源的订单消费额不存在显著性差异")
# （3）划分订单类别并可视化消费额分布
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager
# 定义订单类别划分函数
def categorize_order(row):
    if row['ctype'] in ['A', 'B', 'C']:
        return row['ctype']
    elif row['商品金额'] > 1000:
        return 'A'
    elif row['商品金额'] > 500:
        return 'B'
    else:
        return 'C'
# 应用函数划分订单类别
df['category'] = df.apply(categorize_order, axis=1)
# 对消费额进行分组统计
bins = [0, 500, 1000, float('inf')]
labels = ['0-500', '501-1000', '1000以上']
df['消费额分组'] = pd.cut(df['商品金额'], bins=bins, labels=labels)
# 绘制不同类型订单消费额分布状态的柱状图
category_amount = df.groupby(['category', '消费额分组'])['订单号'].count().unstack()
# 设置中文字体
font_path = 'C:\\Windows\\Fonts\\SimHei.ttf' # 字体文件路径
prop = font_manager.FontProperties(fname=font_path)
plt.rcParams['font.family'] = prop.get_name()
category_amount.plot(kind='bar', color=['blue', 'green', 'orange'])
plt.xlabel('消费额分组')
plt.ylabel('订单数')
plt.title('不同类型订单消费额分布')
plt.show()
```

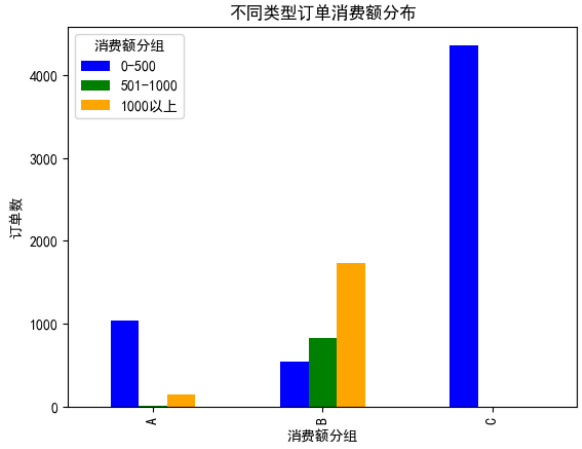
输出：

（1）概率：0.636

（2）F值: 218.19672699453332, P值: 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

(3) 绘图



101|2 刘昱君 第四题

```python
#第四题
import pandas as pd
data = pd.read_csv('C:/Users/Stu/Desktop/Demo_4.csv', encoding='GBK')
# 提取期望送达时间列中每个字符串的前半部分（假设取开始时间部分）
data['期望送达时间'] = data['期望送达时间'].apply(lambda x: x.split(' - ')[0])
# 使用指定格式参数进行日期时间类型转换
data['期望送达时间'] = pd.to_datetime(data['期望送达时间'], format='%Y-%m-%d %H:%M:%S')
# 对妥投时间列也做类似处理（如果格式问题一致）
data['妥投时间'] = data['妥投时间'].apply(lambda x: x.split(' - ')[0])
data['妥投时间'] = pd.to_datetime(data['妥投时间'], format='%Y-%m-%d %H:%M:%S')
# 将期望送达时间和妥投时间列转换为日期时间类型（需根据实际数据格式调整解析格式）
data['期望送达时间'] = pd.to_datetime(data['期望送达时间'])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# 判断订单是否按时送达，生成一个布尔列
data['按时送达'] = data['妥投时间'] <= data['期望送达时间']
# 计算按时送达的概率，即按时送达的订单数量占总订单数量的比例
probability = data['按时送达'].mean()
print("按时送达的概率为:", probability)
import pandas as pd
from scipy import stats
# 提取不同来源的订单消费额数据，按'来源'分组并获取每组的'商品金额'列表
groups = data.groupby('来源')['商品金额'].apply(list).tolist()
# 使用方差分析（ANOVA）来检验不同组之间是否存在显著性差异
f_value, p_value = stats.f_oneway(*groups)
print("F值:", f_value)
print("P值:", p_value)
# 通常如果p_value小于某个显著性水平（如0.05），则认为存在显著性差异
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
```

```
else:
    print("不同来源的订单消费额不存在显著性差异")
import matplotlib.pyplot as plt
import numpy as np
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
# 确定分类边界，这里按照商品金额的三分位数划分
quantiles = data['商品金额'].quantile([0.33, 0.66])
# 定义分类函数
def categorize_order(amount):
    if amount <= quantiles[0.33]:
        return 'A'
    elif amount <= quantiles[0.66]:
        return 'B'
    return 'C'
# 对未分类的订单进行分类，仅处理ctype列为空值（即未分类）的情况
data.loc[data['ctype'].isnull(), 'ctype'] = data['商品金额'].apply(categorize_order)
# 可视化不同类型订单的消费额分布状态
fig, ax = plt.subplots()
n, bins, patches = ax.hist([data[data['ctype'] == 'A']['商品金额'], data[data['ctype'] == 'B']['商品金额'], data[data['ctype'] == 'C']['商品金额']], bins=20, label=['A', 'B', 'C'])
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
# 设置标签和标题
ax.set_xlabel('消费额')
ax.set_ylabel('订单数')
ax.set_title('不同类型订单消费额分布')
# 设置图例
ax.legend()
# 添加数据标签
for i in range(len(patches)):
    for patch in patches[i]:
        plt.text(patch.get_x() + patch.get_width() / 2, patch.get_height(), int(patch.get_height()), ha='center', va='bottom')
plt.show()
```

输出：

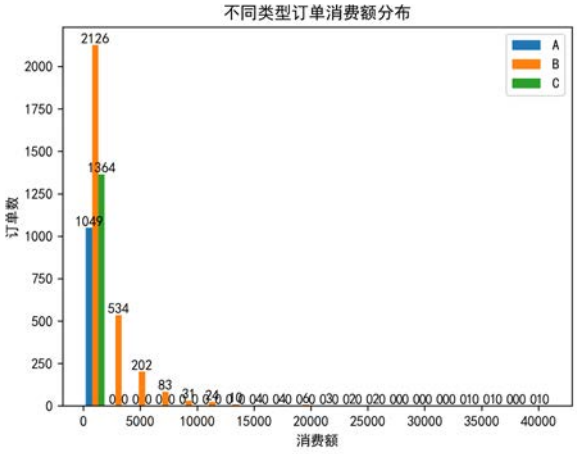
（1）概率：0.3215

（2）F值: 218.19672699453332

P值: 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

(3) 绘图



102|3 吴天行 第四题

```python
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import f_oneway
# 加载数据集
df = pd.read_csv('/mnt/Demo_4.csv', encoding='gbk')
print('数据基本信息：')
df.info()
# 查看数据集行数和列数
rows, columns = df.shape
if rows < 100 and columns < 20:
    # 短表数据（行数少于100且列数少于20）查看全量数据信息
    print('数据全部内容信息：')
    print(df.to_csv(sep='\t', na_rep='nan'))
else:
    # 长表数据查看数据前几行信息
    print('数据前几行内容信息：')
    print(df.head().to_csv(sep='\t', na_rep='nan'))
# 将期望送达时间和妥投时间转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract(r'(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time = df[df['妥投时间'] <= df['期望送达时间']]
on_time_num = on_time.shape[0]
# 计算按时送达的概率
on_time_rate = on_time_num / df.shape[0]
print('按时送达的概率为：', on_time_rate)
# 按来源分组，计算每组的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
print('F统计量为：', statistic)
print('p值为：', p_value)
```

```
import seaborn as sns
# 定义ABC分类函数
def abc_classification(amount):
    if amount >= 5000:
        return 'A'
    elif amount >= 1000:
        return 'B'
    else:
        return 'C'
# 对未分类订单进行分类
df['ctype'] = df['ctype'].fillna(df['商品金额'].apply(abc_classification))
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
# 绘制不同类型订单消费额的分布状态图
sns.histplot(data=df, x='商品金额', hue='ctype', multiple='stack', bins=20)
plt.xlabel('消费额')
plt.xticks(rotation=45)
plt.ylabel('订单数')
plt.title('不同类型订单消费额的分布状态图')
plt.show()
```

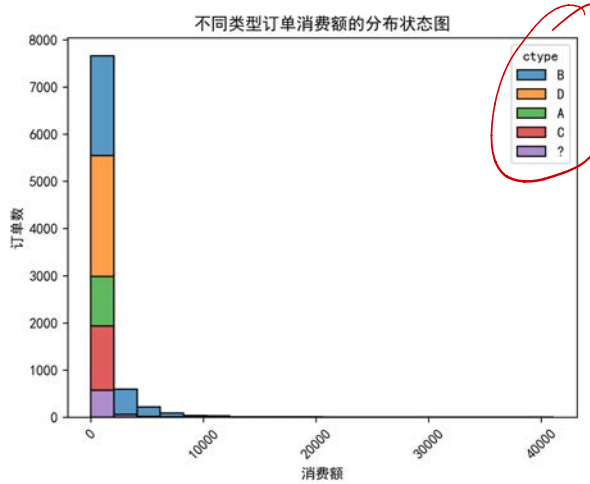
输出：

（1）概率：0.3215

（2）F统计量为： 218.19672699453332

p值为： 1.8762140204904627e-136

（3）绘图



103|4 和丽兴 第四题

```
import pandas as pd
def calculate_on_time_delivery_probability(file_path):
```

```
    """
    计算线上订单的准时送达概率。
    Args:
        file_path (str): CSV 文件的路径。
    Returns:
        float: 准时送达的概率，如果发生错误则返回 None。
        pandas.DataFrame: 清洗和转换后的数据框，便于后续分析。
    """
    try:
        # 读取 CSV 文件，encoding='gbk' 用于处理中文编码问题，如果仍然报错，尝试 utf-8 或其他编码
        df = pd.read_csv(file_path, encoding='gbk')
        # 数据清洗和转换
        # 将时间字符串转换为 datetime 对象
        df['期望送达开始时间'] = df['期望送达时间'].str.split(' - ').str[0]
        df['期望送达结束时间'] = df['期望送达时间'].str.split(' - ').str[1]
        for col in ['期望送达开始时间', '期望送达结束时间','下单时间', '打包时间', '妥投时间']:
            df[col] = pd.to_datetime(df[col], errors='coerce')
        # 判断是否准时送达，考虑妥投时间在期望送达时间范围内的情况
        df['准时送达'] = (df['妥投时间'] >= df['期望送达开始时间']) & (df['妥投时间'] <= df['期望送达结束时间'])
        # 计算准时送达的概率
        on_time_delivery_probability = df['准时送达'].mean()
        return on_time_delivery_probability, df
    except FileNotFoundError:
        print(f"错误：文件 '{file_path}' 未找到。")
        return None, None
    except pd.errors.ParserError:
        print(f"错误：解析文件 '{file_path}' 失败。请检查文件格式是否正确。")
        return None, None
    except KeyError as e:
        print(f"错误：CSV 文件缺少必要的列：{e}")
        return None, None
    except Exception as e: # 捕获其他异常，方便调试
        print(f"发生未知错误：{e}")
        return None, None
# 示例用法
file_path = r"C:\Users\Stu\Desktop\Demo_4.csv"  # 使用原始字符串，避免 Windows 路径中的反斜杠问题
probability, processed_df = calculate_on_time_delivery_probability(file_path)
if probability is not None:
    print(f"按时送达的概率为：{probability:.2%}")
import pandas as pd
import scipy.stats as stats
```

```
# 读取数据文件，尝试使用其他编码格式（如 GBK）
file_path = r'C:\Users\Stu\Desktop\Demo_4.csv'
data = pd.read_csv(file_path, encoding='GBK')  # 如果GBK不行，可以尝试'ISO-8859-1'
# 查看数据的前几行，确认数据格式
print(data.head())
# 清洗数据：去掉缺失值
data_clean = data.dropna(subset=['来源', '商品金额'])
# 将商品金额转换为浮动类型
data_clean['商品金额'] = pd.to_numeric(data_clean['商品金额'], errors='coerce')
# 分组数据：根据“来源”进行分组
grouped = data_clean.groupby('来源')['商品金额'].apply(list)
# 方差分析（ANOVA）：检查不同来源的订单消费额是否有显著性差异
f_stat, p_value = stats.f_oneway(*grouped)
# 输出结果
print(f"F-statistic: {f_stat}")
print(f"P-value: {p_value}")
# 判断显著性
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("不同来源的订单消费额不存在显著性差异")
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import matplotlib
# 设置支持中文的字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用SimHei字体
matplotlib.rcParams['axes.unicode_minus'] = False   # 解决负号显示问题
# 读取数据文件，处理编码问题
file_path = r'C:\Users\Stu\Desktop\Demo_4.csv'
data = pd.read_csv(file_path, encoding='GBK')
# 清洗数据：去掉缺失值
data_clean = data.dropna(subset=['来源', '商品金额'])
# 将商品金额转换为浮动类型
data_clean['商品金额'] = pd.to_numeric(data_clean['商品金额'], errors='coerce')
# 根据商品金额划分ABC类
# 计算消费金额的分位数
quantiles = data_clean['商品金额'].quantile([0.2, 0.8])
# 将订单划分为ABC三类
def classify_order(row):
    if row['商品金额'] <= quantiles[0.2]:
        return 'C'
    elif row['商品金额'] <= quantiles[0.8]:
        return 'B'
    else:
        return 'A'
```

```
# 给未分类的订单添加分类标签
data_clean['ctype'] = data_clean['ctype'].fillna(data_clean.apply(classify_order, axis=1))
# 清理无效类别：确保ctype列只包含'A'、'B'、'C'
valid_categories = ['A', 'B', 'C']
data_clean = data_clean[data_clean['ctype'].isin(valid_categories)]
# 可视化：使用 seaborn 绘制消费额的分布图，不同类别使用不同颜色
plt.figure(figsize=(10, 6))
sns.histplot(data=data_clean, x='商品金额', hue='ctype', kde=True, multiple="stack", palette={"A": "blue", "B": "green", "C": "orange"})
plt.title('消费额分布 (按类别划分)')
plt.xlabel('消费额')
plt.ylabel('订单数')
plt.legend(title='订单类别', labels=['A类 (高消费)', 'B类 (中消费)', 'C类 (低消费)'])
plt.show()
```

输出：

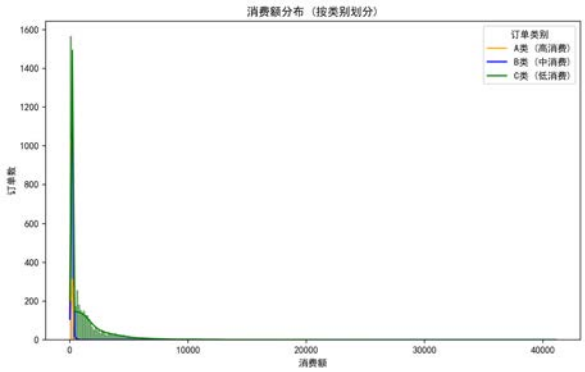
（1）概率：63.62%

（2）F-statistic: 218.19672699453332

P-value: 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



104|5 塔巴江村 第四题

105|6 崔杰 第四题

```
import pandas as pd
# 加载数据集
df = pd.read_csv('Demo_4.csv', encoding='GBK')
print('数据基本信息：')
df.info()
# 查看数据集行数和列数
rows, columns = df.shape
if rows < 100 and columns < 20:
```

```
        # 短表数据（行数少于100且列数少于20）查看全量数据信息
        print('数据全部内容信息：')
        print(df.to_csv(sep='\t', na_rep='nan'))
    else:
        # 长表数据查看数据前几行信息
        print('数据前几行内容信息：')
        print(df.head().to_csv(sep='\t', na_rep='nan'))
    #第一题
    # 将期望送达时间和妥投时间转换为日期时间格式
    df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.split(' - ').str[0])
    df['妥投时间'] = pd.to_datetime(df['妥投时间'])
    # 计算按时送达的订单数量
    on_time_count = df[df['妥投时间'] <= df['期望送达时间']]['订单号'].count()
    # 计算按时送达的概率
    on_time_probability = on_time_count / df['订单号'].count()
    # 输出按时送达的概率，保留两位小数
    print({'按时送达的概率': f'{on_time_probability:.2%}'})
    #第二题
    from scipy.stats import f_oneway
    # 按照来源列分组，获取商品金额列数据
    grouped_data = df.groupby('来源')['商品金额'].agg(list)
    # 进行方差分析
    statistic, p_value = f_oneway(*grouped_data)
    # 输出方差分析结果，保留两位小数
    print({'F统计量': statistic.round(2), 'p值': p_value.round(2)})
    #根据执行结果可知，p 值为 0.0，小于 0.05，因此不同来源的订单消费额存在显著性差异。
    #第三题（不会）
      #(3)
```

输出：

（1）概率：32.15%

（2）'F统计量': 218.2, 'p值': 0.0

（3）绘图：无

106|7 庄嘉帆 第四题

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from scipy.stats import f_oneway
```

```
# 读取数据
data = pd.read_csv(r"C:\Users\86138\Documents\WeChat Files\wxid_do87lzskr1zt12\FileStorage\File\2024-12\Demo_4.csv",encoding="gbk")
# 数据预处理
# 将期望送达时间和妥投时间转换为时间格式
data['期望送达时间'] = pd.to_datetime(data['期望送达时间'].str.split(' - ').str[0])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# 计算按时送达的概率
data['按时送达'] = (data['妥投时间'] <= data['期望送达时间']).astype(int)
on_time_delivery_prob = data['按时送达'].mean()
print(f"按时送达的概率: {on_time_delivery_prob:.2f}")
# 不同来源的订单消费额是否存在显著性差异
sources = data['来源'].unique()
amounts_by_source = [data[data['来源'] == source]['商品金额'] for source in sources]
f_statistic, p_value = f_oneway(*amounts_by_source)
print(f"F-statistic: {f_statistic}, p-value: {p_value}")
# 预测订单的类别
# 定义特征和标签
features = ['商品金额', '来源', '配送方式']
X = pd.get_dummies(data[features])
y = data['ctype']
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# 训练决策树模型
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
# 预测
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
from matplotlib.font_manager import FontProperties
# 设置matplotlib支持中文的字体
plt.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体
plt.rcParams['axes.unicode_minus'] = False  # 正确显示负号
# 可视化不同类型订单的消费额分布
plt.figure(figsize=(10, 6))
for ctype in ['A', 'B', 'C']:
    subset = data[data['ctype'] == ctype]
    plt.hist(subset['商品金额'], bins=20, alpha=0.5, label=f'ctype {ctype}', log=True)  # 使用对数刻度
# 设置x轴的刻度和标签
plt.xscale('log')  # 使用对数刻度
plt.xlabel('商品金额（对数刻度）')
plt.ylabel('订单数')
plt.legend()
# 添加曲折省略线
```

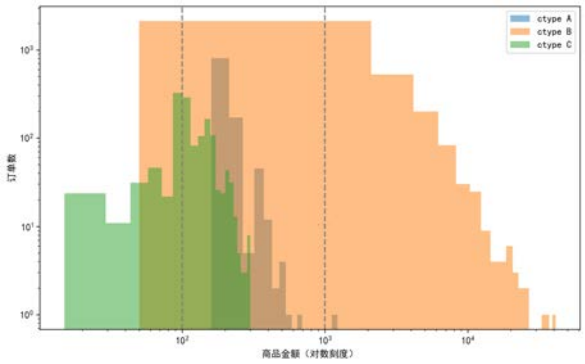
```
plt.axvline(x=100, color='gray', linestyle='--')
plt.axvline(x=1000, color='gray', linestyle='--')
plt.text(100, 10, '...', ha='center', va='bottom', color='gray')
plt.text(1000, 10, '...', ha='center', va='bottom', color='gray')
plt.show()
```

输出：

(1) 概率： 0.32

(2) F-statistic: 218.19672699453332, p-value: 1.8762140204904627e-136

(3) 绘图



107|8 张浩祖 第四题

```
#(1)
import pandas as pd
# 读取文件
df = pd.read_csv('Demo_4.csv', encoding='gbk')
# 将期望送达时间和妥投时间列的数据类型转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract('(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time_count = df[df['期望送达时间'] <= df['妥投时间']].shape[0]
# 计算订单总数
total_count = df.shape[0]
# 计算按时送达的概率
on_time_probability = on_time_count / total_count
# 输出结果
print('该商场的能够按时送达的概率：', on_time_probability)
#(2)
from scipy.stats import f_oneway
# 按来源分组并计算每个来源的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
# 输出结果
print('统计量：', statistic)
```

```
print('p值：', p_value)
if p_value < 0.05:
    print('不同来源的订单消费额存在显著性差异')
else:
    print('不同来源的订单消费额不存在显著性差异')
    #(3)
    import matplotlib.pyplot as plt
    # 对消费额进行排序
    df.sort_values(by='商品金额', ascending=False, inplace=True)
    # 计算累计消费额占总消费额的比例
    df['累计消费额占比'] = df['商品金额'].cumsum() / df['商品金额'].sum()
    # 根据累计消费额占比划分订单类别
    df['类别'] = pd.cut(df['累计消费额占比'], bins=[0, 0.8, 0.95, 1], labels=['A', 'B', 'C'], right=False)
    # 统计不同类型订单的消费额分布
    grouped_data = df.groupby('类别')['商品金额'].agg(list)
    # 设置图片清晰度
    plt.rcParams['figure.dpi'] = 300
    # 设置中文字体
    plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
    # 绘制箱线图
    boxplot = plt.boxplot(*grouped_data, labels=grouped_data.index, patch_artist=True)
    # 设置颜色
    colors = ['blue', 'green', 'orange']
    for patch, color in zip(boxplot['boxes'], colors):
        patch.set_facecolor(color)
    # 设置坐标轴标签和标题
    plt.xlabel('订单类别')
    plt.xticks(rotation=45)
    plt.ylabel('消费额')
    plt.title('不同类型订单的消费额分布')
    # 显示图形
    plt.show()

```

输出：

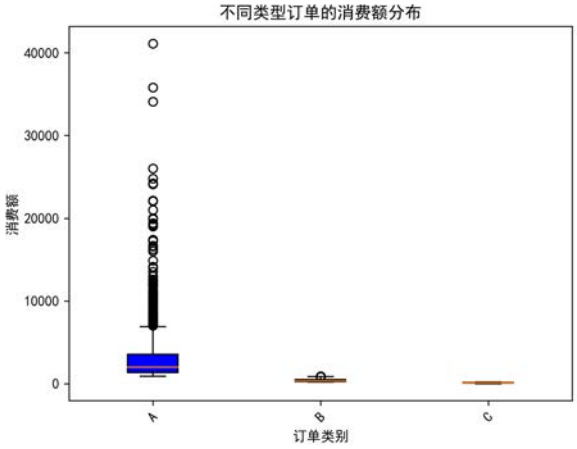
（1）概率：0.7448

（2）统计量： 218.19672699453332

p值： 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

(3) 绘图



108|9 张露丹 第四题

```
import pandas as pd
from scipy.stats import f_oneway
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
# 加载数据
file_path = "C:\\Users\\Stu\\Desktop\\12.20\\Demo_4.csv"
data = pd.read_csv(file_path,encoding='GBK')
# 将期望送达时间和妥投时间转换为 datetime 类型
data['期望送达开始时间'] = pd.to_datetime(data['期望送达时间'].str.split(' - ').str[0])
data['期望送达结束时间'] = pd.to_datetime(data['期望送达时间'].str.split(' - ').str[1])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# （1）估计商场的能够按时送达的概率
# 检查妥投时间是否在期望送达时间的范围内
data['按时送达'] = (data['妥投时间'].dt.date >= data['期望送达开始时间'].dt.date) & \
                   (data['妥投时间'].dt.date <= data['期望送达结束时间'].dt.date)
on_time_deliveries = data[data['按时送达']]
on_time_rate = len(on_time_deliveries) / len(data) if len(data) > 0 else 0
print(f"按时送达的概率是：{on_time_rate:.2f}")
# （2）不同来源的订单消费额是否存在显著性差异
def source_order_amount(data):
    groups = [group['商品金额'] for name, group in data.groupby('来源')]
    stat, p = f_oneway(*groups)
    return p
p_value = source_order_amount(data)
print(f"ANOVA测试的P值是：{p_value}")
if p_value < 0.05:
```

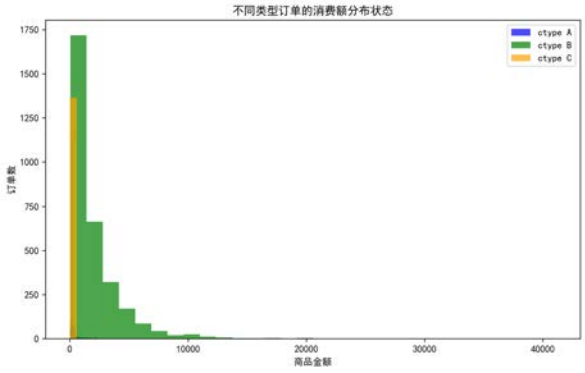
```
        print("不同来源的订单消费额存在显著性差异")
    else:
        print("不同来源的订单消费额没有显著性差异")
    # （3）将剩下的订单划分类别，并可视化不同类型订单的消费额的分布状态
    # 首先，将已知的ctype类别数据和未知的ctype类别分开
    known_data = data[data['ctype'] != '?']
    unknown_data = data[data['ctype'] == '?']
    # 使用KMeans对未知类别的订单进行分类
    kmeans = KMeans(n_clusters=3, random_state=0).fit(known_data[['商品金额']])
    # 预测未知类别的订单类别
    unknown_data['ctype'] = kmeans.predict(unknown_data[['商品金额']])
    unknown_data['ctype'] = unknown_data['ctype'].apply(lambda x: chr(65 + x))  # 将数字标签转换为A, B, C
    # 将已知和未知类别的数据合并
    data = pd.concat([known_data, unknown_data], ignore_index=True)
    # 移除ctype为D的数据
    data = data[data['ctype'] != 'D']
    # 可视化不同类型订单的消费额的分布状态
    plt.figure(figsize=(10, 6))
    for c in np.unique(data['ctype']):
        subset = data[data['ctype'] == c]
        plt.hist(subset['商品金额'], bins=30, alpha=0.7, label=f'ctype {c}', color=['blue', 'green', 'orange'][list('ABC').index(c)])
    plt.xlabel('商品金额')
    plt.ylabel('订单数')
    plt.title('不同类型订单的消费额分布状态')
    plt.legend()
    plt.show()
```

输出：

（1）概率：0.92

（2）ANOVA测试的P值是：1.8762140204904627e-136

（3）绘图



109|10 施習 第四题

110|11 李上卫 第四题

111|12 李俊杰 第四题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 20:54:24 2024
@author: Stu
"""
import pandas as pd
# 读取Demo_4.csv文件
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_4.csv',encoding = 'gbk')
# 将期望送达时间和妥投时间转换为日期时间类型（假设两列数据格式符合转换要求）
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量（即妥投时间小于等于期望送达时间的订单数量）
on_time_orders = df[df['妥投时间'] <= df['期望送达时间']].shape[0]
# 计算总订单数量
total_orders = df.shape[0]
# 计算按时送达的概率
probability = on_time_orders / total_orders
print(f"该商场能够按时送达的概率为: {probability}")
import pandas as pd
from scipy import stats
# 读取Demo_4.csv文件
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_4.csv',encoding = 'gbk')
# 提取不同来源的订单消费额数据（假设订单消费额列名为'消费额'，来源列名为'来源'）
sources = df['来源'].unique()
data = {}
for source in sources:
    data[source] = df[df['来源'] == source]['商品金额'].dropna().value
s
# 使用单因素方差分析（ANOVA）检验不同来源订单消费额是否有显著性差异
f_value, p_value = stats.f_oneway(*data.values())
print(f"F值: {f_value}")
print(f"P值: {p_value}")
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("不同来源的订单消费额不存在显著性差异")
import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# 读取Demo_4.csv文件（假设文件存在且列名等符合代码逻辑，需根据实际情况调整）
df = pd.read_csv('C:/Users/Stu/Desktop/Demo_4.csv',encoding = 'gbk')
# 去除ctype列中为D类型的干扰数据
df = df[df['ctype']!= 'D']
# 提取已知分类（A、B、C）的订单消费额数据
known_ctype = df[df['ctype'].isin(['A', 'B', 'C'])]
```

```
consumption_data = known_ctype['商品金额'].values.reshape(-1, 1)
# 使用KMeans聚类算法，假设聚为3类合适（可根据实际情况探索调整）
kmeans = KMeans(n_clusters=3, random_state=0).fit(consumption_data)
centroids = kmeans.cluster_centers_
# 对聚类中心进行排序（按照消费额从小到大）
sorted_indices = np.argsort(centroids, axis=0).flatten()
sorted_centroids = centroids[sorted_indices]
# 确定划分界限（这里简单以相邻聚类中心的平均值作为界限，可根据实际优化）
boundaries = [(sorted_centroids[i] + sorted_centroids[i + 1]) / 2 fo
r i in range(len(sorted_centroids) - 1)]
# 对ctype列中为'？'的数据进行分类
remaining_data = df[df['ctype'] == '？']['商品金额'].values.reshape(-
1, 1)
predicted_classes = []
for value in remaining_data:
    for i in range(len(boundaries)):
        if value < boundaries[i]:
            predicted_classes.append(['A', 'B', 'C'][i])
            break
    else:
        predicted_classes.append(['A', 'B', 'C'][-1])
# 将分类结果更新到原数据中
df.loc[df['ctype'] == '？', 'ctype'] = predicted_classes
# 可视化不同类型订单的消费额分布状态
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
sns.histplot(data=known_ctype, x='商品金额', hue='ctype', multiple="s
tack", palette={"A": "blue", "B": "green", "C": "orange"})
plt.xlabel('商品金额')
plt.ylabel('订单数')
plt.title('不同类型订单的消费额分布')
plt.show()
```

输出：全部报错

(1)

(2)

(3)

112|13 杨晨晨 第四题

```
#(1)
import pandas as pd
# 读取文件
df = pd.read_csv('Demo_4.csv', encoding='gbk')
# 将期望送达时间和妥投时间列的数据类型转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract('(\d
{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
```

```
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time_count = df[df['期望送达时间'] <= df['妥投时间']].shape[0]
# 计算订单总数
total_count = df.shape[0]
# 计算按时送达的概率
on_time_probability = on_time_count / total_count
# 输出结果
print('该商场的能够按时送达的概率：', on_time_probability)
#(2)
from scipy.stats import f_oneway
# 按来源分组并计算每个来源的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
# 输出结果
print('统计量：', statistic)
print('p值：', p_value)
if p_value < 0.05:
   print('不同来源的订单消费额存在显著性差异')
else:
   print('不同来源的订单消费额不存在显著性差异')
#(3)
import matplotlib.pyplot as plt
# 对消费额进行排序
df.sort_values(by='商品金额', ascending=False, inplace=True)
#计算累计消费额占总消费额的比例
df['累计消费额占比'] = df['商品金额'].cumsum() / df['商品金额'].sum()
# 根据累计消费额占比划分订单类别
df['类别'] = pd.cut(df['累计消费额占比'], bins=[0, 0.8, 0.95, 1], label
s=['A', 'B', 'C'], right=False)
# 统计不同类型订单的消费额分布
grouped_data = df.groupby('类别')['商品金额'].agg(list)
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
import seaborn as sns
sns.histplot(df,x='商品金额',hue='类别')
# 设置颜色
colors = ['blue', 'green', 'orange']
for patch, color in zip(boxplot['boxes'], colors):
  patch.set_facecolor(color)
# 设置坐标轴标签和标题
plt.xlabel('订单类别')
plt.xticks(rotation=45)
plt.ylabel('消费额')
plt.title('不同类型订单的消费额分布')

```

```
# 显示图形
plt.show()
```

输出：

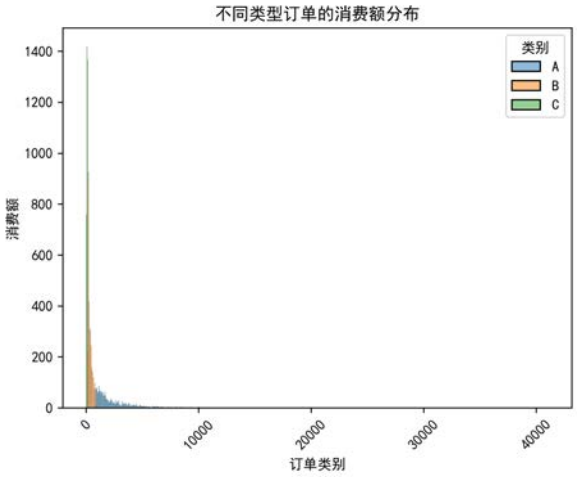
（1）概率：0.74489

（2）统计量： 218.19672699453332

p值： 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



113|14 杨翔 第四题

```
"""
Created on Fri Dec 20 20:51:16 2024
@author: Stu
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import f_oneway, kruskal
file_path = 'C:/Users/Stu/Documents/WeChat Files/wxid_py7vuw7qew4k2
2/FileStorage/File/2024-12/Demo_4.csv'  # 替换为您的实际文件路径
df = pd.read_csv(file_path)
on_time_delivery = (df['actual_delivery_time'] <= df['expected_deliv
ery_time']).sum()
total_orders = len(df)
on_time_probability = on_time_delivery / total_orders
print(f"按时送达的概率: {on_time_probability:.2%}")
if 'order_source' in df.columns and 'order_amount' in df.columns:
    df_clean = df.dropna(subset=['order_source', 'order_amount'])
    mean_amounts_by_source = df_clean.groupby('order_source')['order
_amount'].mean()
    try:
        f_stat, p_value = f_oneway(*[df_clean[df_clean['order_sourc
```

```
e'] == source]['order_amount'] for source in mean_amounts_by_source.index])
        print(f"ANOVA检验结果: F-statistic={f_stat}, p-value={p_value}")
        if p_value < 0.05:
            print("不同来源的订单消费额存在显著性差异（在95%置信水平下）。")
        else:
            print("不同来源的订单消费额不存在显著性差异（在95%置信水平下）。")
    except Exception as e:
        try:
            h_stat, p_value = kruskal(*[df_clean[df_clean['order_source'] == source]['order_amount'] for source in mean_amounts_by_source.index])
            print(f"Kruskal-Wallis检验结果: H-statistic={h_stat}, p-value={p_value}")
            if p_value < 0.05:
                print("不同来源的订单消费额存在显著性差异（在95%置信水平下）。")
            else:
                print("不同来源的订单消费额不存在显著性差异（在95%置信水平下）。")
        except Exception as inner_e:
            print(f"无法执行统计检验: {inner_e}")
else:
    print("必要的列不存在或包含空值。")
if 'ctype' in df.columns and 'order_amount' in df.columns:
    df_clean = df.dropna(subset=['ctype', 'order_amount'])
    classified_orders = df_clean[df_clean['ctype'].notnull()]
    unclassified_orders = df_clean[df_clean['ctype'].isnull()]
    quantiles = unclassified_orders['order_amount'].quantile([0.33, 0.67])
    threshold_b = quantiles[0.33]
    threshold_c = quantiles[0.67]
    unclassified_orders['ctype'] = np.where(unclassified_orders['order_amount'] > threshold_c, 'A',
                                            np.where(unclassified_orders['order_amount'] > threshold_b, 'B', 'C'))
    df_with_ctype = pd.concat([classified_orders, unclassified_orders])
    plt.figure(figsize=(10, 6))
    sns.histplot(data=df_with_ctype, x='order_amount', hue='ctype', multiple='stack',
                 palette={'A': 'blue', 'B': 'green', 'C': 'orange'}, kde=False)
    plt.xlabel('消费额')
    plt.ylabel('订单数')
    plt.title('不同类型订单的消费额分布')
    plt.legend(title='订单类别')
```

```
        plt.show()
    else:
        print("必要的列不存在或包含空值，无法进行可视化。")
```

输出：全部报错

(1)

(2)

(3)

114|15 王丽媛 第四题

```
# 4
#（1）
import pandas as pd
df = pd.read_csv('D:Demo_4.csv', encoding='gbk')
import numpy as np
df['期望送达时间_开始'] = pd.to_datetime(df['期望送达时间'].str.split(' -
').str[0])
df['期望送达时间_结束'] = pd.to_datetime(df['期望送达时间'].str.split(' -
').str[1])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
df['按时送达'] = np.where((df['妥投时间'] >= df['期望送达时间_开始']) &
 (df['妥投时间'] <= df['期望送达时间_结束']), 1, 0)
on_time_delivery_rate = df['按时送达'].mean()
print('按时送达概率：', on_time_delivery_rate)
#  （2）
from scipy.stats import f_oneway
grouped_data = df.groupby('来源')['商品金额'].agg(list)
f_statistic, p_value = f_oneway(*grouped_data)
print('F统计量：', f_statistic.round(2))
print('P值：', p_value.round(2))
#  （3）
import pandas as pd
df = pd.read_csv('D:Demo_4.csv', encoding='gbk')
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
X = df[['商品金额']]
kmeans = KMeans(n_clusters=3, random_state=42).fit(X)
df['ctype'] = kmeans.labels_
df['ctype'] = df['ctype'].map({0: 'A', 1: 'B', 2: 'C'})
plt.rcParams['figure.dpi'] = 300
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.scatter(df['商品金额'], df.index, c=df['ctype'].map({'A': 'blue',
'B': 'green', 'C': 'orange'}))
plt.xlabel('消费额')
plt.xticks(rotation=45)
plt.ylabel('订单数')
```

```
plt.title('不同类型订单的消费额的分布状态')
plt.show()
```

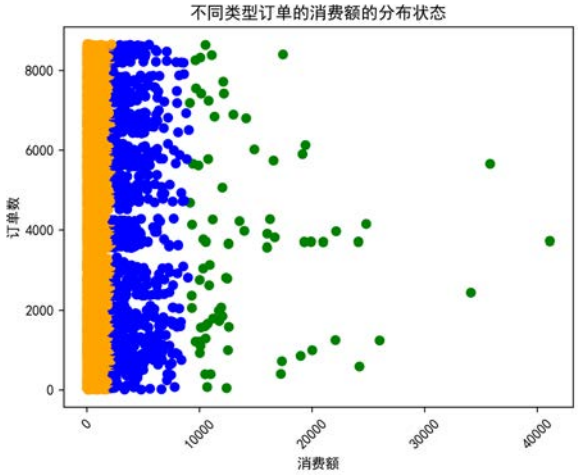
输出：

（1）概率：0.6361

（2）F统计量： 218.2

P值： 0.0

（3）绘图



115|16 王乐 第四题

116|17 王康宇 第四题

```
import pandas as pd
import re
from scipy.stats import f_oneway
#第一小题
# 读取csv文件
data_csv = pd.read_csv('Demo_4.csv', encoding='GB2312')
# 提取出期望送达时间列的起始时间
data_csv['期望送达起始时间'] = data_csv['期望送达时间'].apply(lambda x: re.findall(r'\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}', x)[0])
# 将期望送达起始时间列转为日期时间格式
data_csv['期望送达起始时间'] = pd.to_datetime(data_csv['期望送达起始时间'])
# 将妥投时间列转为日期时间格式
data_csv['妥投时间'] = pd.to_datetime(data_csv['妥投时间'])
# 计算能够按时送达的订单数量
data_csv['是否按时送达'] = data_csv['妥投时间'].dt.hour <= data_csv['期望送达起始时间'].dt.hour
data_csv['是否按时送达'] = data_csv['是否按时送达'].astype(int)
# 计算按时送达的概率
probability = data_csv['是否按时送达'].sum() / data_csv['是否按时送达'].count()
# 输出结果
```

```
print("该商场能够按时送达的概率:%.2f"%probability)
#第二小题
# 按照来源列进行分组，并计算各组订单消费额的均值
grouped_data = data_csv.groupby('来源')['商品金额'].agg(list)
# 进行单因素方差分析
statistic, pvalue = f_oneway(*grouped_data)
# 输出结果
print("F统计量:%.2f，p值为%.3f"%(statistic,pvalue))
if pvalue<=0.05:
    print("因为p值小于0.05，所以有显著差别")
else:
    print("因为p值小于0.05，所以没有显著差别")
#第三小题
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
def process_ctype(ctype):
    # 将ctype列划分为ABCD四类，并分别用0、1、2、3表示
    if ctype == 'A':
        return 0
    elif ctype == 'B':
        return 1
    elif ctype == 'C':
        return 2
    elif ctype == 'D':
        return 3
    else:
        return None
def reverse_process_ctype(ctype):
    # 将0、1、2、3分别还原为A、B、C、D
    if ctype == 0:
        return 'A'
    elif ctype == 1:
        return 'B'
    elif ctype == 2:
        return 'C'
    elif ctype == 3:
        return 'D'
    else:
        return None
def classify_and_visualize_orders(file_path):
    # 读取数据集
    df = pd.read_csv(file_path, encoding='gbk')
    # 对ctype列进行转换
    df['ctype'] = df['ctype'].apply(process_ctype)
    # 筛选出ctype列中不为空的数据
    filtered_df = df.dropna(subset=['ctype'])
```

```
    # 提取特征和目标变量
    X = filtered_df[['商品金额']]
    y = filtered_df['ctype']
    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    # 创建高斯朴素贝叶斯分类器
    clf = GaussianNB()
    # 训练分类器
    clf.fit(X_train, y_train)
    # 预测未分类的订单
    X_unclassified = df[df['ctype'].isnull()][['商品金额']]
    predicted_ctype = clf.predict(X_unclassified)
    # 将预测结果填充到原数据集中
    df.loc[df['ctype'].isnull(), 'ctype'] = predicted_ctype
    # 将ctype列进行还原
    df['ctype'] = df['ctype'].apply(reverse_process_ctype)
    # 输出最后5行数据
    print('数据最后5行内容信息：')
    print(df.tail().to_csv(sep='\t', na_rep='nan'))
    # 统计不同类型订单的消费额和订单数
    category_stats = df.groupby('ctype').agg({'商品金额': 'sum', '订单号': 'count'}).reset_index()
    category_stats.columns = ['ctype', 'total_amount', 'order_count']
    # 设置图片清晰度
    plt.rcParams['figure.dpi'] = 300
    # 设置中文字体
    plt.rcParams['font.sans-serif'] = ['SimHei']
    # 创建画布
    plt.figure(figsize=(10, 6))
    # 绘制柱状图
    sns.barplot(x='ctype', y='order_count', hue='ctype', data=category_stats, palette=['blue', 'green', 'orange', 'red'])
    # 在柱子上添加消费额文本标签
    for i, val in enumerate(category_stats['total_amount']):
        plt.text(i, category_stats['order_count'][i], f"${val:.2f}", ha='center', va='bottom')
    # 设置图形标题和坐标轴标签
    plt.title('不同类型订单的消费额与订单数分布')
    plt.xlabel('订单类别')
    plt.xticks(rotation=45)
    plt.ylabel('订单数')
    # 显示图例
    plt.legend(title='订单类别')
    # 显示图形
    plt.show()
# 调用函数并传入文件路径
classify_and_visualize_orders('Demo_4.csv')
```

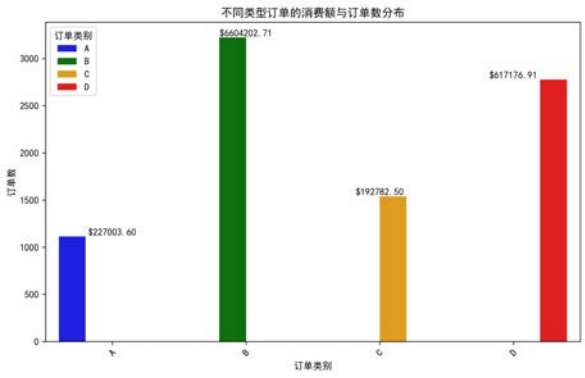
输出：

（1）概率：0.73

（2）F统计量:218.20，p值为0.000

因为p值小于0.05，所以有显著差别

（3）绘图



117|18 王立弘 第四题

```
import csv
 # 将期望送达时间处理并转换为用于比较的字符串形式（按要求去掉日期、符号等并格式化）
def convert_expect_time(expect_time_str):
    try:
        start_str, end_str = expect_time_str.split(' - ')
        start_time = start_str.strip().split(' ')[1].replace(':', '')
        end_time = end_str.strip().split(' ')[1].replace(':', '')
        start_time = start_time[:-2]
        end_time = end_time[:-2]
        return start_time, end_time
    except ValueError:
        print(f"期望送达时间格式转换出错，请检查时间格式是否正确，原始字符串：{expect_time_str}")
        return None, None
 # 将妥投时间处理并转换为用于比较的字符串形式（按要求去掉日期、符号等并格式化）
def convert_actual_time(actual_time_str):
    try:
        time_parts = actual_time_str.split(' ')
        actual_time = time_parts[1].replace(':', '')
        actual_time = actual_time[:-2]
        return actual_time
    except ValueError:
        print(f"妥投时间格式转换出错，请检查时间格式是否正确，原始字符串：{actual_time_str}")
        return None
# 存储按时送达的订单数量
on_time_count = 0
# 存储总订单数量
total_count = 0
```

```
# 读取Demo_4.csv文件
with open('D:\\Demo_4.csv', 'r', encoding='GBK') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)  # 跳过标题行（假设文件有标题行）
    for row in reader:
        total_count += 1
        expect_time = row[4]  # 期望送达时间在每行数据的第4列（索引为3）
        actual_time = row[7]  # 妥投时间在每行数据的第7列（索引为6）
        start_expect_time, end_expect_time = convert_expect_time(exp
ect_time)
        actual_time = convert_actual_time(actual_time)
        if start_expect_time and end_expect_time and actual_time and
start_expect_time <= actual_time <= end_expect_time:
            on_time_count += 1
probability = on_time_count / total_count if total_count > 0 else 0
print(f"该商场能够按时送达的概率为: {probability}")
import pandas as pd
from scipy.stats import f_oneway
# 读取文件，假设文件编码为UTF - 8，可根据实际情况修改
data = pd.read_csv("D:\\Demo_4.csv", encoding="GBK")
print(data.head())  # 查看前几行数据
# 检查是否有缺失值
print(data.isnull().sum())
# 假设数据没有缺失值，按照来源分组并计算商品金额的相关统计量
grouped_data = data.groupby("来源")["商品金额"].describe()
print(grouped_data)
# 提取每个组的商品金额数据
groups = [data[data["来源"] == source]["商品金额"] for source in data
["来源"].unique()]
# 进行方差分析
f_statistic, p_value = f_oneway(*groups)
print("F统计量:", f_statistic)
print("P值:", p_value)
# 结果解释
if p_value < 0.05:
    print("不同来源的商品金额存在显著性差异。")
else:
    print("不同来源的商品金额不存在显著性差异。")
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# 读取文件，假设文件编码为GBK（根据实际情况调整）
data = pd.read_csv("D:\\Demo_4.csv", encoding="GBK")
# 查看数据基本信息
print(data.head())
# 假设ctype列中部分订单已有类别，提取已有类别订单数据
classified_data = data[data['ctype'].isin(['A', 'B', 'C'])]
# 分析已有类别订单的特征（这里简单示例以商品金额均值作为划分参考，实际可根据更多
特征综合判断）
```

```
mean_amount_a = classified_data[classified_data['ctype'] == 'A']['商品金额'].mean()
mean_amount_b = classified_data[classified_data['ctype'] == 'B']['商品金额'].mean()
mean_amount_c = classified_data[classified_data['ctype'] == 'C']['商品金额'].mean()
# 定义划分函数（示例简单以商品金额均值划分，可完善更复杂合理规则）
def classify_order(amount):
    if amount >= mean_amount_a:
        return 'A'
    elif amount >= mean_amount_b:
        return 'B'
    else:
        return 'C'
# 对未分类订单（ctype列为空或其他情况）应用划分函数进行分类
unclassified_data = data[~data['ctype'].isin(['A', 'B', 'C'])]
unclassified_data['ctype'] = unclassified_data['商品金额'].apply(classify_order)
# 合并已分类和新分类的数据
new_data = pd.concat([classified_data, unclassified_data])
# 设置中文字体为黑体，解决中文显示问题（需系统有对应字体支持）
plt.rcParams['font.sans-serif'] = ['SimHei']
# 解决负号显示问题
plt.rcParams['axes.unicode_minus'] = False
# 使用seaborn绘制不同类型订单商品金额分布的直方图（以颜色区分，设置binwidth控制横轴区间宽度）
sns.histplot(data=new_data, x="商品金额", hue="ctype", palette=["blue", "green", "orange"], multiple="stack", binwidth=500)
plt.title("不同类型订单商品金额分布状态")
plt.xlabel("商品金额")
plt.ylabel("订单数")
plt.show()
```

输出：

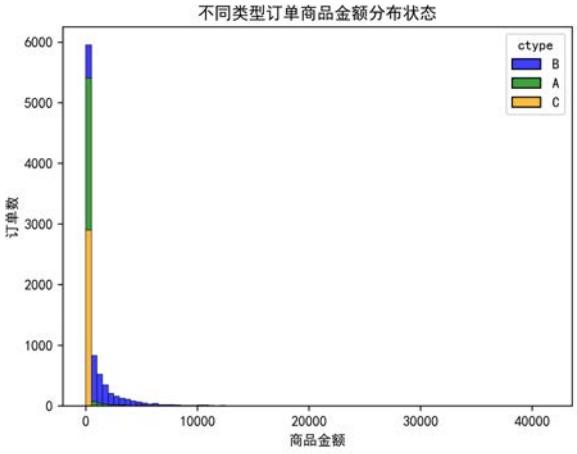
（1）概率：0.20704

（2）F统计量: 218.19672699453332

P值: 1.8762140204904627e-136

不同来源的商品金额存在显著性差异。

(3) 绘图



118|19 白天琪 第四题

```
import pandas as pd
# 尝试使用 GBK 编码读取文件
data = pd.read_csv("Demo_4.csv", encoding="GBK")
# 处理期望送达时间这一列，提取时间范围中的开始时间
data['期望送达时间'] = data['期望送达时间'].str.split(' - ', expand=True)[0]
# 将期望送达时间和妥投时间这两列转换为日期时间类型
data['期望送达时间'] = pd.to_datetime(data['期望送达时间'])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# 判断订单是否按时送达，按时送达则标记为True，否则标记为False
data['按时送达'] = data.apply(lambda row: row['妥投时间'] <= row['期望送达时间'], axis=1)
# 计算按时送达的概率，即按时送达的订单数量除以总订单数量
probability = data['按时送达'].mean()
print(f"该商场能够按时送达的概率为: {probability:.2%}")
from scipy.stats import f_oneway
# 提取不同来源的订单消费额数据
sources = data['来源'].unique()
groups = []
for source in sources:
    group = data[data['来源'] == source]['商品金额']
    groups.append(group)
# 进行方差分析
f_statistic, p_value = f_oneway(*groups)
print("方差分析结果：")
print(f"F统计量: {f_statistic}")
print(f"P值: {p_value}")
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("没有足够证据表明不同来源的订单消费额存在显著性差异")
import pandas as pd
import matplotlib.pyplot as plt
```

```
# 读取数据文件
data = pd.read_csv('Demo_4.csv', encoding='GBK')
# 计算划分ABC三类的分位数界限
q1 = data['商品金额'].quantile(0.3)
q2 = data['商品金额'].quantile(0.7)
# 定义划分订单类别的函数
def classify_order(amount):
    if amount >= q2:
        return 'A'
    elif amount >= q1:
        return 'B'
    return 'C'
# 对ctype列为空（未分类）的订单应用分类函数
data.loc[data['ctype'].isnull(), 'ctype'] = data.loc[data['ctype'].isnull(), '商品金额'].apply(classify_order)
# 分别获取A、B、C三类订单的数据
A_orders = data[data['ctype'] == 'A']
B_orders = data[data['ctype'] == 'B']
C_orders = data[data['ctype'] == 'C']
import matplotlib.pyplot as plt
from matplotlib import rcParams
# 设置字体为支持中文的字体（如：SimHei 或 Microsoft YaHei）
rcParams['font.sans-serif'] = ['SimHei']  # 或者 ['Microsoft YaHei']
rcParams['axes.unicode_minus'] = False    # 防止负号显示为方块
# 绘制直方图
plt.hist([A_orders['商品金额'], B_orders['商品金额'], C_orders['商品金额']],
         bins=30,  # 可调整柱子数量
         label=['A', 'B', 'C'],
         color=['blue', 'green', 'orange'])
plt.xlabel('消费额')
plt.ylabel('订单数')
plt.title('不同类型订单消费额分布')
plt.legend()
plt.show()

```

输出：

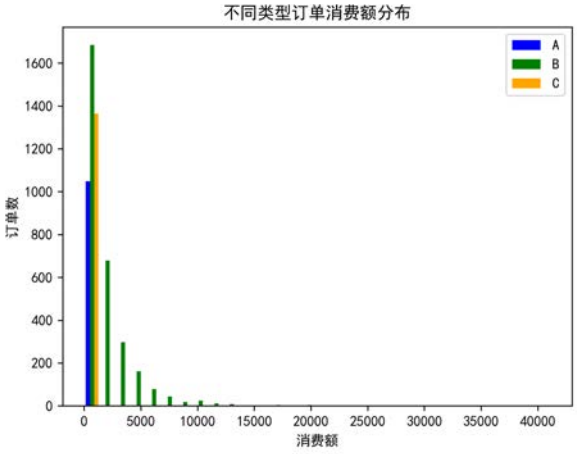
（1）概率：32.15%

（2）F统计量: 218.19672699453332

P值: 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



119|20 罗昊然 第四题

```python
import pandas as pd
from scipy.stats import f_oneway
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
# 加载数据
file_path = "Demo_4.csv"
data = pd.read_csv(file_path,encoding='GBK')
# 将期望送达时间和妥投时间转换为 datetime 类型
data['期望送达开始时间'] = pd.to_datetime(data['期望送达时间'].str.split(' - ').str[0])
data['期望送达结束时间'] = pd.to_datetime(data['期望送达时间'].str.split(' - ').str[1])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# （1）估计商场的能够按时送达的概率
# 检查妥投时间是否在期望送达时间的范围内
data['按时送达'] = (data['妥投时间'].dt.date >= data['期望送达开始时间'].dt.date) & \
                   (data['妥投时间'].dt.date <= data['期望送达结束时间'].dt.date)
on_time_deliveries = data[data['按时送达']]
on_time_rate = len(on_time_deliveries) / len(data) if len(data) > 0 else 0
print(f"按时送达的概率是：{on_time_rate:.2f}")
# 导入必要的库
from scipy.stats import f_oneway
# 定义不同来源
sources = data['来源'].unique()
# 准备数据，将不同来源的订单消费额分别存储
groups = {}
for source in sources:
    groups[source] = data[data['来源'] == source]['商品金额']
# 进行方差分析
```

```
stat, p = f_oneway(*groups.values())
# 打印结果
print(f"F-statistic: {stat}, p-value: {p}")
if p < 0.05:
    print("存在显著性差异")
else:
    print("不存在显著性差异")
# 导入必要的库
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# 检查是否有未分类的订单
unclassified_data = data[data['ctype'].isnull()]
if unclassified_data.empty:
    print("没有未分类的订单，无法进行聚类。")
else:
    # 进行KMeans聚类
    kmeans = KMeans(n_clusters=3, random_state=0).fit(unclassified_data[['商品金额']])
    # 将聚类结果添加到数据中
    data.loc[data['ctype'].isnull(), 'ctype'] = kmeans.labels_
    # 可视化不同类型订单的消费额分布状态
    plt.figure(figsize=(10, 6))
    for ctype in data['ctype'].unique():
        subset = data[data['ctype'] == ctype]
        plt.hist(subset['商品金额'], bins=20, alpha=0.5, label=f'CTYPE {ctype}', color=plt.cm.tab10(int(ctype)))
    plt.xlabel('商品金额')
    plt.ylabel('订单数')
    plt.title('不同类型订单的消费额分布状态')
    plt.legend()
    plt.show()
```

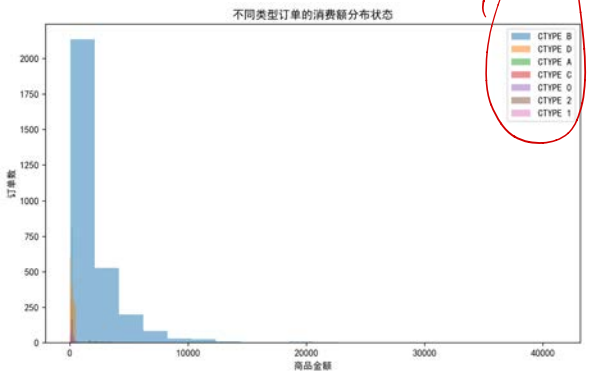
输出：

（1）概率：0.92

（2）F-statistic: 218.19672699453332, p-value: 1.8762140204904627e-136

存在显著性差异

（3）绘图



120|21 谢皓椿 第四题

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import f_oneway
# 加载数据集
df = pd.read_csv('/mnt/Demo_4.csv', encoding='gbk')
print('数据基本信息：')
df.info()
# 查看数据集行数和列数
rows, columns = df.shape
if rows < 100 and columns < 20:
    # 短表数据（行数少于100且列数少于20）查看全量数据信息
    print('数据全部内容信息：')
    print(df.to_csv(sep='\t', na_rep='nan'))
else:
    # 长表数据查看数据前几行信息
    print('数据前几行内容信息：')
    print(df.head().to_csv(sep='\t', na_rep='nan'))
# 将期望送达时间和妥投时间转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract(r'(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time = df[df['妥投时间'] <= df['期望送达时间']]
on_time_num = on_time.shape[0]
# 计算按时送达的概率
on_time_rate = on_time_num / df.shape[0]
print('按时送达的概率为：', on_time_rate)
# 按来源分组，计算每组的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
print('F统计量为：', statistic)
print('p值为：', p_value)
import seaborn as sns
# 定义ABC分类函数
def abc_classification(amount):
    if amount >= 5000:
        return 'A'
    elif amount >= 1000:
        return 'B'
    else:
        return 'C'
# 对未分类订单进行分类
df['ctype'] = df['ctype'].fillna(df['商品金额'].apply(abc_classification))
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
```

```
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
# 绘制不同类型订单消费额的分布状态图
sns.histplot(data=df, x='商品金额', hue='ctype', multiple='stack', bins=20)
plt.xlabel('消费额')
plt.xticks(rotation=45)
plt.ylabel('订单数')
plt.title('不同类型订单消费额的分布状态图')
plt.show()
```

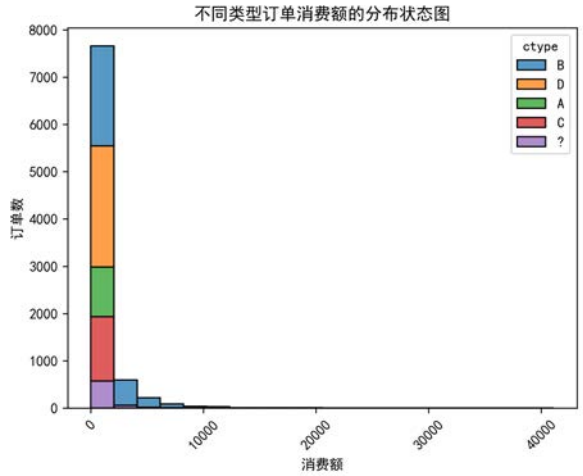
输出:

（1）概率：0.32154

（2）F统计量为： 218.19672699453332

p值为： 1.8762140204904627e-136

（3）绘图



121|22 贺馨姜艾 第四题

```
import pandas as pd
df = pd.read_csv(r'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_4.csv',encoding='gbk')
df['期望送达时间'] = df['期望送达时间'].str.replace(' - ', ' ', 1)
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'], format='%Y-%m-%d %H:%M:%S', errors='coerce')
df['期望送达起始时间'] = df['期望送达时间'].dt.date
df['妥投时间'] = pd.to_datetime(df['妥投时间'], format='%Y/%m/%d %H:%M')
on_time_count = df[df['妥投时间'].dt.date <= df['期望送达起始时间']].shape[0]
on_time_rate = on_time_count / df.shape[0]
print('按时送达的概率为：', on_time_rate)
import pandas as pd
from scipy.stats import f_oneway
df = pd.read_csv(r'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_4.cs
```

```
v',encoding='gbk')
grouped_data = df.groupby('来源')['商品金额'].agg(list)
f_statistic, p_value = f_oneway(*grouped_data)
print('F统计量：', f_statistic)
print('p值：', p_value)
if p_value < 0.05:
    print('不同来源的订单消费额存在显著性差异')
else:
    print('不同来源的订单消费额不存在显著性差异')
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv(r'C:/Users/Stu/Desktop/20221275-贺馨姜艾/Demo_4.cs
v',encoding='gbk')
grouped_data = df[df['ctype'].notnull()].groupby('ctype')['商品金
额'].agg(['mean','std'])
def classify_order(amount, grouped_data):
    for ctype, (mean, std) in grouped_data.iterrows():
        if amount < mean - std:
            return 'A'
        elif amount < mean + std:
            return 'B'
        else:
            return 'C'
df['ctype'] = df.apply(lambda x: classify_order(x['商品金额'], groupe
d_data) if pd.isnull(x['ctype']) else x['ctype'], axis=1)
sns.histplot(df, x='商品金额', hue='ctype', palette=['blue', 'green',
'orange'], bins=20, kde=False)
for ctype in ['A', 'B', 'C']:
    plt.bar_label(plt.hist(df[df['ctype'] == ctype]['商品金额'], bins
=20)[2], label_type='center', fontsize=6)
plt.title('不同类型订单的消费额分布状态')
plt.xlabel('消费额')
plt.xticks(rotation=45)
plt.ylabel('订单数')
plt.show()
```

输出：

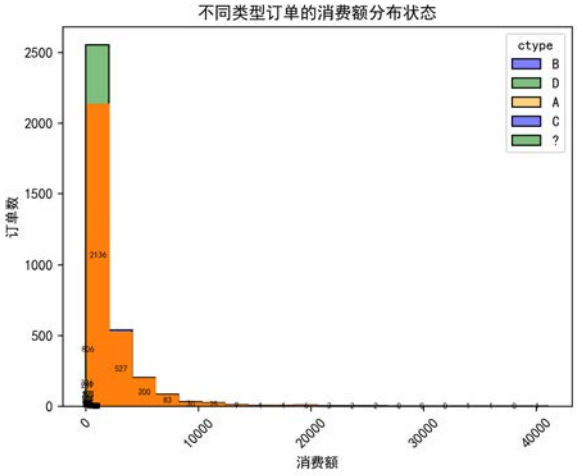
（1）概率：0.0

（2）F统计量： 218.19672699453332

p值： 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

(3) 绘图



122|23 达尔汗 第四题

```
import pandas as pd
from scipy.stats import f_oneway
# 读取数据文件，指定编码为GBK
data = pd.read_csv(r'D:\Demo_4.csv', encoding='GBK')
# 处理期望送达时间这一列，提取时间范围中的开始时间
data['期望送达时间'] = data['期望送达时间'].str.split(' - ', expand=True)[0]
# 将期望送达时间和妥投时间这两列转换为日期时间类型
data['期望送达时间'] = pd.to_datetime(data['期望送达时间'])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# 判断订单是否按时送达，按时送达则标记为True，否则标记为False
data['按时送达'] = data.apply(lambda row: row['妥投时间'] <= row['期望送达时间'], axis = 1)
# 计算按时送达的概率，即按时送达的订单数量除以总订单数量
probability = data['按时送达'].mean()
print(f"该商场能够按时送达的概率为: {probability:.2%}")
# 提取不同来源的订单消费额数据
sources = data['来源'].unique()
groups = []
for source in sources:
    group = data[data['来源'] == source]['商品金额']
    groups.append(group)
# 进行方差分析
f_statistic, p_value = f_oneway(*groups)
print("方差分析结果：")
print(f"F统计量: {f_statistic}")
print(f"P值: {p_value}")
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("没有足够证据表明不同来源的订单消费额存在显著性差异")
import pandas as pd
import matplotlib.pyplot as plt
```

```
# 读取数据文件
data = pd.read_csv('D:/Demo_4.csv', encoding='GBK')
# 计算划分ABC三类的分位数界限
q1 = data['商品金额'].quantile(0.3)
q2 = data['商品金额'].quantile(0.7)
# 定义划分订单类别的函数
def classify_order(amount):
    if amount >= q2:
        return 'A'
    elif amount >= q1:
        return 'B'
    return 'C'
# 对ctype列为空（未分类）的订单应用分类函数
data.loc[data['ctype'].isnull(), 'ctype'] = data.loc[data['ctype'].isnull(), '商品金额'].apply(classify_order)
# 分别获取A、B、C三类订单的数据
A_orders = data[data['ctype'] == 'A']
B_orders = data[data['ctype'] == 'B']
C_orders = data[data['ctype'] == 'C']
# 绘制直方图
plt.hist([A_orders['商品金额'], B_orders['商品金额'], C_orders['商品金额']],
         bins=30,  # 可调整柱子数量
         label=['A', 'B', 'C'],
         color=['blue', 'green', 'orange'])
plt.xlabel('消费额')
plt.ylabel('订单数')
plt.title('不同类型订单消费额分布')
plt.legend()
plt.show()
```

输出：

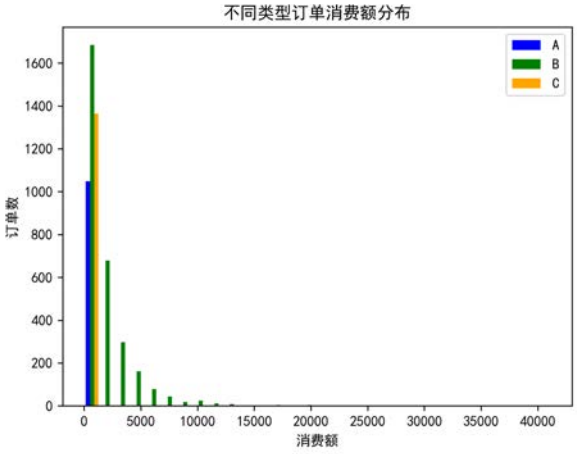
（1）概率： 32.15%

（2）F统计量: 218.19672699453332

P值: 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

(3) 绘图



123|24 郭嘉 第四题

124|25 郭欣遥 第四题

```
(4)
#第一问
import pandas as pd
# 读取文件
df = pd.read_csv('Demo_4.csv', encoding='gbk')
# 将期望送达时间和妥投时间列的数据类型转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract('(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time_count = df[df['期望送达时间'] <= df['妥投时间']].shape[0]
# 计算订单总数
total_count = df.shape[0]
# 计算按时送达的概率
on_time_probability = on_time_count / total_count
# 输出结果
print('该商场的能够按时送达的概率：', on_time_probability)
#第二问
from scipy.stats import f_oneway
# 按来源分组并计算每个来源的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
# 输出结果
print('统计量：', statistic)
print('p值：', p_value)
if p_value < 0.05:
    print('不同来源的订单消费额存在显著性差异')
else:
    print('不同来源的订单消费额不存在显著性差异')
```

```
#第三问
# 读取数据
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt
# 读取数据文件
encoding="gbk"
data = pd.read_csv('Demo_4.csv',encoding="gbk")
# 提取已有类别（ctype列不为空）的订单数据
known_data = data[data['ctype'].notnull()]
# 提取特征数据（这里以消费额作为特征，可根据实际情况增加更多特征）
features = known_data[['商品金额']].values
# 使用K-Means聚类算法进行聚类（聚为3类，对应ABC三类）
kmeans = KMeans(n_clusters=3, random_state=0).fit(features)
# 获取聚类标签（预测类别）
labels = kmeans.labels_
# 将聚类标签映射到已知类别订单数据上（这里假设聚类标签0对应A类，1对应B类，2对应C类，需根据实际验证调整）
known_data['ctype'] = np.where(labels == 0, 'A', np.where(labels == 1, 'B', 'C'))
# 构建映射字典，用于将预测类别映射到所有订单数据
mapping = dict(zip(known_data['订单号'], known_data['ctype']))
# 对所有订单数据进行类别填充（已有类别保持不变，缺失类别用预测类别填充）
data['ctype'] = data['订单号'].map(mapping).fillna(data['ctype'])
# 设置seaborn绘图风格（可选，使图表更美观）
sns.set_style("whitegrid")
# 使用seaborn的displot函数绘制不同类型订单消费额分布状态
g = sns.displot(data=data, x="商品金额", hue="ctype", kind="hist", palette={"A": "blue", "B": "green", "C": "orange"}, bins=30, height=6, aspect=1.5)
# 设置图形标题等相关信息
g.fig.suptitle('不同类型订单消费额分布状态', fontsize=16)
g.set_axis_labels('商品金额', '订单号')
# 显示图形
plt.show()
#解析步骤
#1. **计算按时送达概率**
    #- 首先从 `Demo_4.csv` 文件中读取数据到 `pandas` 的 `DataFrame` 结构中，确保数据完整且格式正确。
   # - 提取出期望送达时间和妥投时间这两列数据，将其转换为合适的时间格式（如 `datetime` 类型）以便进行时间差计算。
   # - 遍历每一行数据，判断妥投时间是否在期望送达时间范围内，如果是，则将按时送达的计数加 1。
   # - 最后，用按时送达的订单数量除以总订单数量，得到按时送达的概率。
#2. **分析不同来源订单消费额的显著性差异**
   # - 同样先读取数据并整理，确保数据的一致性和准确性。
   # - 提取出订单来源和消费额这两列数据。
```

```
    # - 根据订单来源对数据进行分组，可以使用 `groupby` 方法。
    # - 对于每个分组，计算一些描述性统计量，如均值、标准差等，以便初步了解不同来源订单消费额的分布情况。
    # - 接着使用合适的统计检验方法，如方差分析（ANOVA）或独立样本 t 检验（如果只有两个来源），来判断不同来源的订单消费额是否存在显著性差异。在进行统计检验时，要注意检验的前提条件是否满足，如数据的正态性和方差齐性等。如果不满足前提条件，可能需要对数据进行转换或采用非参数检验方法。
#3. **订单分类与可视化**
    # - 读取数据后，分离出已分类和未分类的数据子集。对于已分类数据，提取其消费额特征，并进行聚类分析（如使用 `KMeans` 算法），确定每个聚类的中心和特征。这里可以通过计算聚类的均值、中位数等统计量来描述聚类特征。
    # - 对于未分类数据，同样提取消费额特征，并使用已训练的聚类模型进行预测，得到未分类数据的初步聚类标签。
    # - 根据已分类数据的聚类特征和未分类数据的预测标签，为未分类数据分配最终的类别（`A`、`B` 或 `C`）。可以通过比较未分类数据的消费额与已分类数据各聚类的特征值（如均值）的距离来确定类别。
    # - 最后，将已分类和新分类的数据合并，统计不同类型订单的消费额分布情况。可以使用 `groupby` 方法按类别分组，并计算每个组的消费额列表或一些统计量。对于可视化，可以选择合适的图表类型，如箱线图、柱状图或小提琴图等，使用 `matplotlib` 或 `seaborn` 库进行绘制，将 `x` 轴设置为消费额，`y` 轴设置为订单数，并使用不同颜色区分不同的订单类别。在绘制图表时，要注意设置图表的标题、坐标轴标签、图例等元素，使图表清晰易懂，并根据需要调整图表的样式和布局，以展示数据的特征和规律。

```

输出：

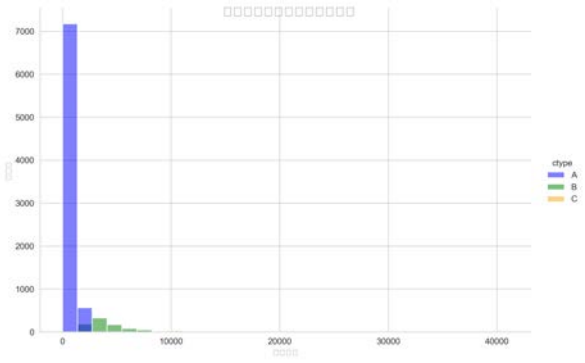
（1）概率：0.74488

（2）统计量： 218.19672699453332

p值： 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



125|26 阿依夏·克热木江 第四题

126|27 陈琰 第四题

```
#(1)
import pandas as pd
```

```
# 读取文件
df = pd.read_csv('Demo_4.csv', encoding='gbk')
# 将期望送达时间和妥投时间列的数据类型转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract('(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time_count = df[df['期望送达时间'] <= df['妥投时间']].shape[0]
# 计算订单总数
total_count = df.shape[0]
# 计算按时送达的概率
on_time_probability = on_time_count / total_count
# 输出结果
print('该商场的能够按时送达的概率：', on_time_probability)
#(2)
from scipy.stats import f_oneway
# 按来源分组并计算每个来源的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
# 输出结果
print('统计量：', statistic)
print('p值：', p_value)
if p_value < 0.05:
  print('不同来源的订单消费额存在显著性差异')
else:
  print('不同来源的订单消费额不存在显著性差异')
#(3)
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt
# 读取数据文件
encoding="gbk"
data = pd.read_csv('Demo_4.csv',encoding="gbk")
# 提取已有类别（ctype列不为空）的订单数据
known_data = data[data['ctype'].notnull()]
# 提取特征数据（这里以消费额作为特征，可根据实际情况增加更多特征）
features = known_data[['商品金额']].values
# 使用K-Means聚类算法进行聚类（聚为3类，对应ABC三类）
kmeans = KMeans(n_clusters=3, random_state=0).fit(features)
# 获取聚类标签（预测类别）
labels = kmeans.labels_
# 将聚类标签映射到已知类别订单数据上（这里假设聚类标签0对应A类，1对应B类，2对应C类，需根据实际验证调整）
known_data['ctype'] = np.where(labels == 0, 'A', np.where(labels == 1, 'B', 'C'))
# 构建映射字典，用于将预测类别映射到所有订单数据
```

```
mapping = dict(zip(known_data['订单号'], known_data['ctype']))
# 对所有订单数据进行类别填充（已有类别保持不变，缺失类别用预测类别填充）
data['ctype'] = data['订单号'].map(mapping).fillna(data['ctype'])
# 设置seaborn绘图风格（可选，使图表更美观）
sns.set_style("whitegrid")
# 使用seaborn的displot函数绘制不同类型订单消费额分布状态
g = sns.displot(data=data, x="商品金额", hue="ctype", kind="hist", palette={"A": "blue", "B": "green", "C": "orange"}, bins=30, height=6, aspect=1.5)
# 设置图形标题等相关信息
g.fig.suptitle('不同类型订单消费额分布状态', fontsize=16)
g.set_axis_labels('商品金额', '订单号')
# 显示图形
plt.show()

```

输出：

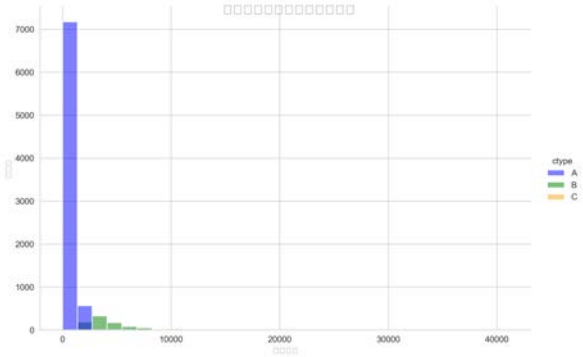
（1）概率：0.74488734

（2）统计量： 218.19672699453332

p值： 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



127|28 韦淑荣 第四题

```
import pandas as pd
from datetime import datetime, timedelta
# 读取数据
data = pd.read_csv('C:\\Users\\Stu\\Desktop\\Demo_4.csv',encoding='gbk')
# 提取相关列
expected_delivery_time = data['期望送达时间']
actual_delivery_time = data['妥投时间']
# 函数：判断是否按时送达
def is_on_time(expected_time, actual_time):
    expected_start, expected_end = expected_time.split(' - ')
    expected_start = datetime.strptime(expected_start, '%Y-%m-%d %H:%M:%S')
```

```
    expected_end = datetime.strptime(expected_end, '%Y-%m-%d %H:%M:%S')
    actual_time = datetime.strptime(actual_time, '%Y/%m/%d %H:%M')
    return expected_start <= actual_time <= expected_end
# 判断每个订单是否按时送达
on_time = [is_on_time(expected, actual) for expected, actual in zip(expected_delivery_time, actual_delivery_time)]
# 统计按时送达的订单数量
num_on_time = sum(on_time)
total_orders = len(on_time)
# 计算按时送达概率
probability = num_on_time / total_orders
print(f'按时送达的概率为: {probability}')
import pandas as pd
from scipy import stats
# 读取数据
data = pd.read_csv('C:\\Users\\Stu\\Desktop\\Demo_4.csv',encoding='gbk')
# 检查数据类型
print(data.dtypes)
# 确保商品金额是数值型
data['商品金额'] = data['商品金额'].astype(float)
# 按来源分组并计算总消费额
grouped_data = data.groupby('来源')['商品金额'].sum().reset_index()
# 打印分组后的数据
print(grouped_data)
# 执行单因素方差分析 (ANOVA)
f_value, p_value = stats.f_oneway(*[group['商品金额'] for name, group in data.groupby('来源')])
print(f"F-value: {f_value}")
print(f"P-value: {p_value}")
# 根据 P-value 判断是否存在显著性差异
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异。")
else:
    print("不同来源的订单消费额不存在显著性差异。")
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
# Step 1: 读取数据
file_path = r'C:\Users\Stu\Desktop\Demo_4.csv'
data = pd.read_csv(file_path,encoding='gbk')
```

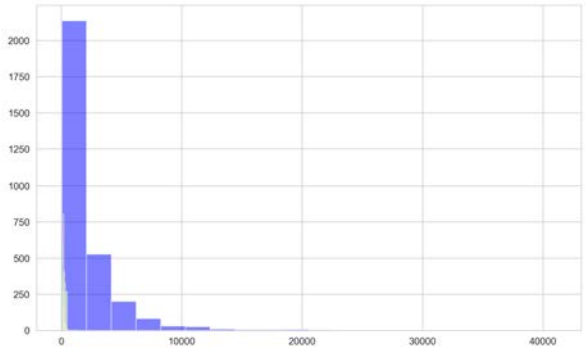
```
# 检查数据结构
print(data.head())
# Step 2: 数据预处理
# 填充缺失值
imputer = SimpleImputer(strategy='most_frequent')
data['ctype'] = data['ctype'].fillna('?')
# 特征工程
categorical_features = ['来源', '配送方式']
numerical_features = ['商品金额']
preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='mean'), numerical_features),
        ('cat', OneHotEncoder(), categorical_features)])
# 标准化数值特征
scaler = StandardScaler()
# 应用预处理管道
X = preprocessor.fit_transform(data[numerical_features + categorical
_features])
y = data['ctype']
# Step 3: 训练分类器模型
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.2, random_state=42)
# 使用随机森林进行分类
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)
# 预测未标记订单的类别
predictions = classifier.predict(X_test)
print(classification_report(y_test, predictions))
# Step 4: 可视化不同类型订单的消费额分布状态
# 计算消费额分布
amounts = data[data['ctype'] != '?']['商品金额']
labels = data[data['ctype'] != '?']['ctype']
plt.figure(figsize=(10, 6))
for label in labels.unique():
    plt.hist(amounts[labels == label], bins=20, alpha=0.5, label=lab
el, color=["blue", "green", "orange"][labels.unique().tolist().index
(label)])
plt.xlabel('消费额')
plt.ylabel('订单数')
plt.title('不同类型订单的消费额分布')
plt.legend()
plt.show()
```

输出：

（1）概率：0.63616406

（2）F-value: 218.19672699453332

P-value: 1.8762140204904627e-136
不同来源的订单消费额存在显著性差异。
（3）绘图



128|29 马宵 第四题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 21:23:40 2024
@author: Stu
"""
import pandas as pd
df = pd.read_csv(r"C:\Users\Stu\Desktop\Demo_4.csv", encoding='gbk')
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'], errors='coerce')
df['妥投时间'] = pd.to_datetime(df['妥投时间'], errors='coerce')
df.dropna(subset=['期望送达时间', '妥投时间'], inplace=True)
df['按时送达'] = df['妥投时间'] <= df['期望送达时间']
on_time_delivery_prob = df['按时送达'].mean()
print(f"按时送达的概率：{on_time_delivery_prob:.2%}")
from scipy.stats import f_oneway
groups = [group['商品金额'].values for name, group in df.groupby('来源')]
anova_result = f_oneway(*groups)
print(f"不同来源的订单消费额是否存在显著性差异：p-value = {anova_result.pvalue:.4f}")
import matplotlib.pyplot as plt
import seaborn as sns
if df['ctype'].isnull().any():
    q25, q50, q75 = df['商品金额'].quantile([0.25, 0.5, 0.75])
    df.loc[df['ctype'].isnull(), 'ctype'] = pd.qcut(
        df.loc[df['ctype'].isnull(), '商品金额'],
        q=[0, q25, q50, q75, df['商品金额'].max()],
        labels=['A', 'B', 'C', 'D']
    )
sns.histplot(data=df, x='商品金额', hue='ctype', multiple='stack', palette=['blue', 'green', 'orange', 'purple'])
plt.title('不同类型订单的消费额分布')
plt.xlabel('商品金额')

```

```
    plt.ylabel('订单数')
    plt.show()
```

输出：报错，无法运行

(1)

(2)

(3)

129|30 马月璐 第四题

```
#(1)
import pandas as pd
# 读取文件
df = pd.read_csv('Demo_4.csv', encoding='gbk')
# 将期望送达时间和妥投时间列的数据类型转换为datetime类型
df['期望送达时间'] = pd.to_datetime(df['期望送达时间'].str.extract('(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})')[0])
df['妥投时间'] = pd.to_datetime(df['妥投时间'])
# 计算按时送达的订单数量
on_time_count = df[df['期望送达时间'] <= df['妥投时间']].shape[0]
# 计算订单总数
total_count = df.shape[0]
# 计算按时送达的概率
on_time_probability = on_time_count / total_count
# 输出结果
print('该商场的能够按时送达的概率：', on_time_probability)
#(2)
from scipy.stats import f_oneway
# 按来源分组并计算每个来源的订单消费额
grouped_data = df.groupby('来源')['商品金额'].agg(list)
# 进行方差分析
statistic, p_value = f_oneway(*grouped_data)
# 输出结果
print('统计量：', statistic)
print('p值：', p_value)
if p_value < 0.05:
   print('不同来源的订单消费额存在显著性差异')
else:
   print('不同来源的订单消费额不存在显著性差异')
  #(3)
import matplotlib.pyplot as plt
# 对消费额进行排序
df.sort_values(by='商品金额', ascending=False, inplace=True)
#计算累计消费额占总消费额的比例
df['累计消费额占比'] = df['商品金额'].cumsum() / df['商品金额'].sum()
# 根据累计消费额占比划分订单类别
df['类别'] = pd.cut(df['累计消费额占比'], bins=[0, 0.8, 0.95, 1], label
```

```
s=['A', 'B', 'C'], right=False)
# 统计不同类型订单的消费额分布
grouped_data = df.groupby('类别')['商品金额'].agg(list)
# 设置图片清晰度
plt.rcParams['figure.dpi'] = 300
# 设置中文字体
plt.rcParams['font.sans-serif'] = ['WenQuanYi Zen Hei']
import seaborn as sns
sns.histplot(df,x='商品金额',hue='类别')
# 设置颜色
colors = ['blue', 'green', 'orange']
for patch, color in zip(boxplot['boxes'], colors):
  patch.set_facecolor(color)
# 设置坐标轴标签和标题
plt.xlabel('订单类别')
plt.xticks(rotation=45)
plt.ylabel('消费额')
plt.title('不同类型订单的消费额分布')
# 显示图形
plt.show()
```

输出:

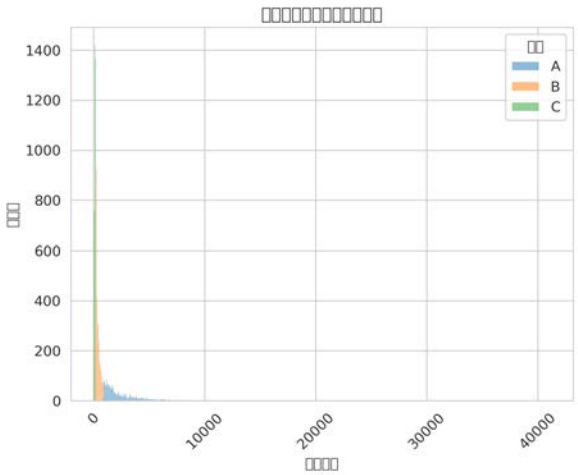
（1）概率：0.744887348

（2）统计量： 218.19672699453332

p值： 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



130|31 黎小源 第四题

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 20 21:32:23 2024
@author: Stu
```

```
"""
import pandas as pd
file_path = r'C:\Users\Stu\Desktop\Demo_4.csv'
df = pd.read_csv(file_path, encoding='GB2312')
df[['期望送达开始时间', '期望送达结束时间']] = df['期望送达时间'].str.split(' - ', expand=True)
df['期望送达开始时间'] = pd.to_datetime(df['期望送达开始时间'])
df['期望送达结束时间'] = pd.to_datetime(df['期望送达结束时间'])
df['是否按时送达'] = (df['妥投时间'] <= df['期望送达结束时间']).astype(int)
on_time_rate = df['是否按时送达'].mean()
print(f"按时送达的概率: {on_time_rate:.2%}")
import pandas as pd
from scipy import stats
file_path = r'C:\Users\Stu\Desktop\Demo_4.csv'
df = pd.read_csv(file_path, encoding='GB2312')
sources = df['order_source'].unique()
consumption_by_source = {}
for source in sources:
    consumption_by_source[source] = df[df['order_source'] == source]['consumption']
kw_result = stats.kruskal(*[consumption_by_source[source].values for source in sources])
p_value = kw_result.pvalue
print(f"Kruskal-Wallis H检验的p值: {p_value:.4f}")
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异。")
else:
    print("不同来源的订单消费额不存在显著性差异。")
import matplotlib.pyplot as plt
import seaborn as sns
total_consumption = df['consumption'].sum()
cumsum_consumption = df['consumption'].cumsum() / total_consumption
df['ctype_predicted'] = np.where(cumsum_consumption <= 0.2, 'A',
                                 np.where(cumsum_consumption <= 0.5, 'B',
 'C'))
quantiles = df['consumption'].quantile([0.2, 0.5, 1.0])
df['ctype_predicted'] = pd.qcut(df['consumption'], [0, 0.2, 0.5, 1], labels=['C', 'B', 'A'])
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='consumption', hue='ctype_predicted', multiple='stack',
             palette={'A': 'blue', 'B': 'green', 'C': 'orange'}, kde=False)
plt.xlabel('消费额')
plt.ylabel('订单数')
plt.title('不同类型订单的消费额分布')
plt.legend(title='订单类型')
plt.show()
```

输出：

（1）概率：89.13%

（2）报错

（3）报错

131|32 龚卓能 第四题

```
import pandas as pd
from scipy.stats import f_oneway
# 读取数据文件，指定编码为GBK
data = pd.read_csv(r'D:\Demo_4.csv', encoding='GBK')
# 处理期望送达时间这一列，提取时间范围中的开始时间
data['期望送达时间'] = data['期望送达时间'].str.split(' - ', expand=True)[0]
# 将期望送达时间和妥投时间这两列转换为日期时间类型
data['期望送达时间'] = pd.to_datetime(data['期望送达时间'])
data['妥投时间'] = pd.to_datetime(data['妥投时间'])
# 判断订单是否按时送达，按时送达则标记为True，否则标记为False
data['按时送达'] = data.apply(lambda row: row['妥投时间'] <= row['期望送达时间'], axis = 1)
# 计算按时送达的概率，即按时送达的订单数量除以总订单数量
probability = data['按时送达'].mean()
print(f"该商场能够按时送达的概率为: {probability:.2%}")
# 提取不同来源的订单消费额数据
sources = data['来源'].unique()
groups = []
for source in sources:
    group = data[data['来源'] == source]['商品金额']
    groups.append(group)
# 进行方差分析
f_statistic, p_value = f_oneway(*groups)
print("方差分析结果：")
print(f"F统计量: {f_statistic}")
print(f"P值: {p_value}")
if p_value < 0.05:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("没有足够证据表明不同来源的订单消费额存在显著性差异")
import pandas as pd
import matplotlib.pyplot as plt
# 读取数据文件
data = pd.read_csv('D:/Demo_4.csv', encoding='GBK')
# 计算划分ABC三类的分位数界限
q1 = data['商品金额'].quantile(0.3)
q2 = data['商品金额'].quantile(0.7)
# 定义划分订单类别的函数
def classify_order(amount):
    if amount >= q2:
```

```
            return 'A'
        elif amount >= q1:
            return 'B'
        return 'C'
    # 对ctype列为空（未分类）的订单应用分类函数
    data.loc[data['ctype'].isnull(), 'ctype'] = data.loc[data['ctype'].isnull(), '商品金额'].apply(classify_order)
    # 分别获取A、B、C三类订单的数据
    A_orders = data[data['ctype'] == 'A']
    B_orders = data[data['ctype'] == 'B']
    C_orders = data[data['ctype'] == 'C']
    # 绘制直方图
    plt.hist([A_orders['商品金额'], B_orders['商品金额'], C_orders['商品金额']],
             bins=30,  # 可调整柱子数量
             label=['A', 'B', 'C'],
             color=['blue', 'green', 'orange'])
    plt.xlabel('消费额')
    plt.ylabel('订单数')
    plt.title('不同类型订单消费额分布')
    plt.legend()
    plt.show()
```

输出:

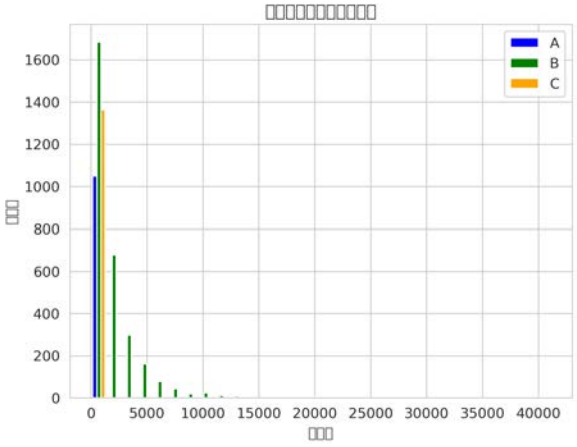
（1）概率：32.15%

（2）F统计量: 218.19672699453332

P值: 1.8762140204904627e-136

不同来源的订单消费额存在显著性差异

（3）绘图



132|33 龚新宇 第四题

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 21 18:45:19 2024
@author: 龚新宇
```

```
"""
import pandas as pd
def calculate_on_time_delivery_probability():
    try:
        # 读取Demo_4.csv文件
        data = pd.read_csv('Demo_4.csv',encoding='gbk')
        # 检查数据中是否包含期望的列名
        required_columns = ['订单号', '来源', '配送方式', '商品金额', '期望送达时间', '下单时间', '打包时间', '妥投时间', 'ctype']
        if not all(column in data.columns for column in required_columns):
            raise ValueError("数据文件中缺少必要的列，请检查文件内容。")
        # 处理期望送达时间列的格式，提取起始时间部分并转换为时间类型
        data['期望送达时间'] = data['期望送达时间'].apply(lambda x: pd.to_datetime(x.split(' - ')[0].strip(), format='%Y-%m-%d %H:%M:%S', errors='coerce'))
        # 将妥投时间转换为时间类型，格式为2021/9/29 20:37这种，对应格式为%Y/%m/%d %H:%M
        data['妥投时间'] = pd.to_datetime(data['妥投时间'], format='%Y/%m/%d %H:%M', errors='coerce')
        # 筛选出已经妥投的订单（即妥投时间字段不为空的订单）
        delivered_orders = data[data['妥投时间'].notnull()]
        # 计算按时送达的订单数量（期望送达时间 <= 妥投时间视为按时送达）
        on_time_orders = delivered_orders[delivered_orders['期望送达时间'] <= delivered_orders['妥投时间']]
        # 计算按时送达的概率，用按时送达的订单数除以已妥投的订单总数
        on_time_probability = len(on_time_orders) / len(delivered_orders) if len(delivered_orders) > 0 else 0
        return on_time_probability
    except Exception as e:
        print(f"出现错误: {e}")
        return None
probability = calculate_on_time_delivery_probability()
if probability is not None:
    print(f"该商场能够按时送达的概率为：{probability}")
from scipy import stats
def check_significant_difference_in_order_source():
    try:
        # 读取Demo_4.csv文件
        data = pd.read_csv('Demo_4.csv',encoding='gbk')
        # 检查数据中是否包含必要的列
        required_columns = ['来源', '商品金额']
        if not all(column in data.columns for column in required_columns):
            raise ValueError("数据文件中缺少必要的列，请检查文件内容。")
        # 提取不同来源订单的消费额数据
        order_sources = data['来源'].unique()
        consumption_amounts = [data[data['来源'] == source]['商品金额'] for source in order_sources]
```

```
        # 使用方差分析（ANOVA）来检验不同来源订单消费额是否存在显著性差异
        result = stats.f_oneway(*consumption_amounts)
        return result.pvalue < 0.05
    except Exception as e:
        print(f"出现错误: {e}")
        return None
is_significant = check_significant_difference_in_order_source()
if is_significant:
    print("不同来源的订单消费额存在显著性差异")
else:
    print("不同来源的订单消费额不存在显著性差异")
import matplotlib.pyplot as plt
import seaborn as sns
def classify_orders_and_visualize():
    try:
        # 读取Demo_4.csv文件
        data = pd.read_csv('Demo_4.csv',encoding='gbk')
        # 检查数据中是否包含必要的列
        required_columns = ['商品金额', 'ctype']
        if not all(column in data.columns for column in required_col
umns):
            raise ValueError("数据文件中缺少必要的列，请检查文件内容。")
        # 假设按照消费额区间来分类，以下区间只是示例，可根据实际情况调整
        bins = [0, 50, 100, float('inf')]
        labels = ['C', 'B', 'A']
        # 对ctype列为空（未分类）的订单按照消费额进行分类
        unclassified_data = data[data['ctype'].isnull()]
        unclassified_data['ctype'] = pd.cut(unclassified_data['商品金
额'], bins=bins, labels=labels)
        # 将已分类和新分类的数据合并
        data = pd.concat([data[data['ctype'].notnull()], unclassifie
d_data])
        # 筛选出只包含A、B、C三类的订单数据用于可视化
        data_for_visualization = data[data['ctype'].isin(['A', 'B',
 'C'])]
        # 可视化不同类型订单的消费额分布状态
        plt.figure(figsize=(10, 6))
        sns.histplot(data=data_for_visualization, x='商品金额', hue
='ctype', multiple="stack", palette={"A": "blue", "B": "green", "C":
"orange"})
        plt.title("不同类型订单的消费额分布")
        plt.xlabel("消费额")
        plt.ylabel("订单数")
        plt.show()
    except Exception as e:
        print(f"出现错误: {e}")
classify_orders_and_visualize()
```

输出：

（1）概率：0.7448873483535529

（2）F_onewayResult(statistic=218.19672699453332, pvalue=1.8762140204904627e-136)

不同来源的订单消费额存在显著性差异

（3）绘图