

队伍编号	MC2406053
题号	C

基于随机森林和 PuLP 的物流预测和人员安排

摘要

本研究针对电子商务物流网络中分拣中心的货量预测及人员排班问题，提出一种基于**随机森林算法与 PuLP 线性规划**的混合方法。研究的初衷是通过精确预测每个分拣中心在未来一段时间内的货量，并基于这些预测结果有效地安排人员，从而优化物流操作效率和降低成本。

电子商务的高速发展对物流系统提出了**更高的效率和响应速度要求**。分拣中心作为物流网络的关键节点，承担着将包裹正确快速地分类并运往下一站的任务，其运作效率直接影响整个物流网络的性能。正确预测分拣中心的货量并基于这些预测进行高效的人员安排，是管理者面临的一大挑战。

在本研究中，我们首先分析了包括过去四个月每天货量和前三十天每小时货量的历史数据，采用随机森林算法来预测接下来三十天内 57 个分拣中心的每天及每小时的货量。随机森林被选中是因为其在多个预测模型比较中表现出**最佳的准确性和稳健性**。通过这种方式，我们能够处理大规模数据集并提取出影响货量变化的关键因素。

接下来，利用 PuLP 优化库，我们根据预测的货量需求，设计并实现了一人员排班模型。这一模型不仅要满足各个班次的货量处理需求，还要在此基础上尽可能地减少总人天数，同时确保每名员工的工作量平衡。此外，我们考虑到正式员工与临时工的不同成本和工作效率，优先使用正式员工，并在必要时添加临时工以满足额外的需求。

此外，研究还考虑了运输线路的调整对货量预测的影响。通过分析运输线路变更的数据，我们调整了预测模型以反映这些变化，进一步提高预测的准确性。

研究结果表明，所提出的方法能够有效预测分拣中心的货量，并据此制定出**人员成本效益高、响应迅速的排班计划**。这不仅优化了物流网络的运作效率，还为管理者提供了一个强大的决策支持工具，帮助他们在快速变化的市场环境中作出更好的资源配置决策。

最终，本文提出的综合方法为电子商务物流网络中的**货量预测与人员排班问题**提供了一种新的解决方案，展示了**机器学习与运筹学结合**的潜力，对提升物流效率和降低运营成本具有重要的理论和实际意义。

关键词：电商物流、分拣中心、货量预测、随机森林算法、人员排班、PuLP 优化、机器学习、运输线路分析。

目录

1 问题重述	1
1.1 问题背景	1
1.2 问题重述	1
1.2.1 问题一：建立货量预测模型	1
1.2.2 问题二：考虑运输线路调整的货量预测	1
1.2.3 问题三：基于货量预测的人员排班	1
1.2.4 问题四：特定分拣中心的排班优化	2
1.3 总体思路分析	2
1.3.1 模型假设	2
2 问题一的建模与求解	3
2.1 分析	3
2.1.1 LSTM	3
2.1.2 ARIMA	4
2.1.3 随机森林算法	4
2.2 按天的数据模拟与预测	4
2.3 按小时的数据模拟和预测	5
3 问题二的建模与求解	7
3.1 分析与建模求解	7
3.1.1 数据预处理	7
3.1.2 时间序列预测	8
3.1.3 退火算法	8
3.1.4 随机森林回归	9
4 问题三的建模与求解	10
4.1 分析	10
4.2 建模与求解	10
5 问题四的建模与求解	13
5.1 问题四的分析	13
5.2 建模求解	14
6 总结	15
6.1 使用随机森林的优点	15
6.2 问题一和问题二建模存在的缺点	15
6.3 PULP 效果评估	15
A 问题一代码	18

B 问题二代码	21
C 问题三代码	25
D 问题四代码	28

1 问题重述

1.1 问题背景

随着网购的流行，电商物流显得更加重要。在电商物流网络中，订单配送的过程包括多个环节，其中核心环节之一是分拣。分拣中心负责根据不同的目的地对包裹进行分类，并将它们发送到下一个目的地，最终交付给顾客。因此，提高分拣中心的管理效率对整个网络的订单交付效率和成本控制至关重要。

货量预测在电商物流网络中扮演着至关重要的角色。准确预测分拣中心的货物量是后续管理和决策的基础。通常，货量预测是根据历史货量、物流网络配置等信息，来预测每个分拣中心每天和每小时的货物量。

分拣中心的货量预测与网络的运输线路密切相关。通过分析各线路的运输货量，可以确定各分拣中心之间的网络连接关系。当线路关系发生变化时，可以根据调整信息来提高对各分拣中心货量的准确预测。

基于货量预测的人员排班是下一步需要解决的重要问题。分拣中心的人员包括正式员工和临时工两种类型。合理安排人员旨在完成工作的前提下尽可能降低人力成本。根据物流网络的情况，制定了人员安排的班次和小时人效指标。在确定人员安排时，优先考虑使用正式员工，必要时再增加临时工。

1.2 问题重述

1.2.1 问题一：建立货量预测模型

根据提供的历史数据建立一个模型，以预测 57 个分拣中心在未来 30 天内每天以及每小时的货量。数据来源包括过去四个月的每天货量和前三十天的每小时货量。预测的准确性是优化分拣中心运营效率的关键，它直接影响到资源的配置和人员的排班计划。

1.2.2 问题二：考虑运输线路调整的货量预测

继问题一之后，问题二进一步要求我们在考虑运输线路变化的情况下预测货量。附件中提供了过去 90 天的运输线路平均货量和未来 30 天预期的运输线路变化。运输线路的变化可能会显著影响各分拣中心的货量，因此，我们需要调整我们的预测模型以反映这些变化。这可能涉及到重新评估与特定线路相关的历史数据的权重，或者直接在模型中引入运输线路作为一个新的解释变量。通过这种方式，我们可以更准确地预测出受线路变更影响的货量变动，从而为人员调度和资源分配提供更为精确的数据支持。

1.2.3 问题三：基于货量预测的人员排班

问题三要求我们根据问题二中的货量预测结果，建立一个模型来安排未来 30 天内每个分拣中心每个班次的人员（包括正式工和临时工）。每个分拣中心每天有六个班次，人员配置必须足以处理预测的货量，同时尽量减少总人天数。这要求我们在确保每个班次有足够的手处理预定货量的同时，还要优化人员的总工作量，避免不必要的劳动力浪费。

1.2.4 问题四：特定分拣中心的排班优化

问题四针对特定的分拣中心（如 SC60），要求建立一个详细的排班模型，安排 200 名正式工在未来 30 天内的班次出勤计划，同时考虑雇佣临时工的需要。另外还要求每名正式工的出勤率不能超过 85%，且连续出勤天数不能超过 7 天。在保证分拣中心运营效率的同时，如何合理安排人员以保持工作负荷的平衡和公平是重要的考量指标。

1.3 总体思路分析

1.3.1 模型假设

1. 假设一：2023-12-01 至 2023-12-31 期间不会发生如购物节等可能导致货物量激增的特殊事件。
2. 假设二：每个分拣中心的货运量定义为在给定历史时间段中的平均值。对于每个分拣中心，平均发出货量和平均到达货量对于线路的变化应不敏感。
3. 假设三：对于同一到达分拣中心，不同始发分拣中心对其影响程度与两者之间的货量呈正相关。
4. 假设四：不考虑天气、交通情况等因素对分拣中心的影响。
5. 假设五：不考虑运输过程中损坏、丢失的情况对分拣中心货量的影响。
6. 假设六：每个数据点（每天的货量记录）被假设为独立同分布的。即假设前一天的货量与其他天的货量无直接关联。
7. 假设七：假设每个分拣中心的货量与日期（时间序列数据）具有一定的函数关系，使得随机森林等机器学习模型可以利用这些历史数据来预测未来的货量。

2 问题一的建模与求解

2.1 分析

问题一给出了分拣中心在过去的四个月内每天的货量以及过去 30 天内个小时的货量。在这些数据的基础上要对下一个月（30 天）的货量进行预测，我们先对给出的数据进行观察。我们选取，例如 SC6,SC41 来观察。利用 python/matplotlib 作图

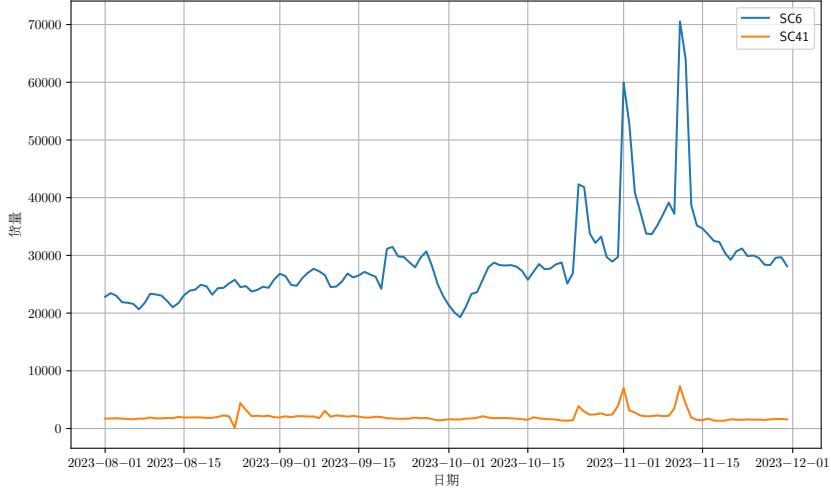


图 1: SC6,SC41 的日期-货量图线

根据观察，我们发现在特定的时间段会出现一些“异常值”，而这些异常值的出现很可能是由于节假日等因素引起的。因此为了预测 30 天后的货物量，我们选取几种深度学习的方法来进行预测。

1. LSTM
2. ARIMA
3. 随机森林算法

2.1.1 LSTM

长短期记忆（英语：Long Short-Term Memory, LSTM）是一种时间循环神经网络（RNN）。由于独特的设计结构，LSTM 适合于处理和预测时间序列中间隔和延迟非常长的重要事件。LSTM 通过引入“门”结构来调节信息的流动，这使得它在长序列数据上表现得更好。LSTM 的核心组件有遗忘门（Forget Gate）、输入门（Input Gate）、细胞状态（Cell State）、输出门（Output Gate）。LSTM 的每个时刻会通过上述四个门控制信息的流动。遗忘门决定丢弃哪些过去的信息，输入门和它的候选值共同决定将哪些新信息加入细胞状态。细胞状态随后更新，最后通过输出门确定应输出什么信息。^[3] ^[4]

通过这样的结构，LSTM 能够在处理序列数据时有效地保留长期依赖信息，解决了传统 RNN 在长序列上的梯度消失或爆炸问题。

2.1.2 ARIMA

ARIMA 模型（自回归积分滑动平均模型）是一种广泛应用于时间序列预测的统计模型。ARIMA 模型结合了自回归（AR）、差分（I）和移动平均（MA）三种主要组成部分，适用于分析和预测具有时间依赖性的数据。其核心组件包括自回归（AR）部分、差分（I）部分、移动平均（MA）部分。将这三部分结合起来，ARIMA 模型的完整形式可以表示为：

$$\phi(B)\nabla^d X_t = \theta(B)a_t$$

其中 $\phi(B)X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p}$, p 是自回归项的阶数, ϕ_i 是自回归系数, B 是退后算子。 $\theta(B)a_t$ 则类似, a_t 是时间序列的误差项。[5]

从式子中可以看出，通过对原始数据进行差分转换，ARIMA 能够将非平稳时间序列转换为平稳时间序列，因此模型能够适用于预测那些显示出趋势或季节性模式的数据。

2.1.3 随机森林算法

随机森林是一种集成学习方法，特别适合用于分类、回归和其他任务，通过构建多棵决策树在训练时并输出模式的类（分类）或平均预测（回归）。随机森林算法的核心思想是通过合并多个决策树的预测结果来提高整体模型的预测准确性和稳定性。[2]

随机森林算法的主要步骤有：自助采样（Bootstrap sampling），从原始数据集中随机（允许重复的）选择 N 个样本构成了一个训练集。 N 次采样产生的 N 个训练集被用于训练 N 棵决策树；构建决策树：对于每棵树的每个节点，随机选择 k 个特征（而不是所有特征），然后使用这些特征中的最佳分割方法来分割节点。这种“特征随机选择”的做法增加了树之间的差异性。决策树的生长：每棵树都尽可能地生长而不进行剪枝。每棵树都完全依赖于自助采样得到的训练数据集来建立。聚合预测：对于分类问题，使用多数投票法；对于回归问题，计算所有树的输出值的平均值。

随机森林算法的成功在于它的简单性和在多种数据集上表现出的高效性与准确性。它既能处理分类问题，也能处理回归问题，同时还能进行特征选择，这使它成为一种非常强大且灵活的机器学习算法。虽然随机森林通常不如专门的时间序列或图数据模型那样直接适用于处理分拣中心之间的货物流动问题，但它仍然可以在某些情况下提供有价值的见解和预测，特别是随机森林可以提供关于哪些特征（例如历史货物流量、时间因素、分拣中心之间的距离等）对预测货物流量最有影响的洞察。这对于理解货物流量模式和优化物流网络非常有用。

2.2 按天的数据模拟与预测

根据选择，我们对其进行数据模拟。我们通过使用 python 的 `sklearn` 中提供的模型来进行拟合。首先我们通过分类不同的仓库，获取所有的信息。在分类完成之后，我们利用几种方法来实现。首先是利用 LSTM 来生成预测的曲线。¹如图2。

相同的，我们选取随机森林进行预测可以得到图3。另外由于我们发现数据有较大的波动，而 ARIMA 模型适用于平稳时间序列，而从图中看，原始数据不是平稳的。因此直接使用 ARIMA 模型可能不会得到好的预测效果。我们尝试季节性 ARIMA (SARIMA) 模型。

¹具体代码在附录中。

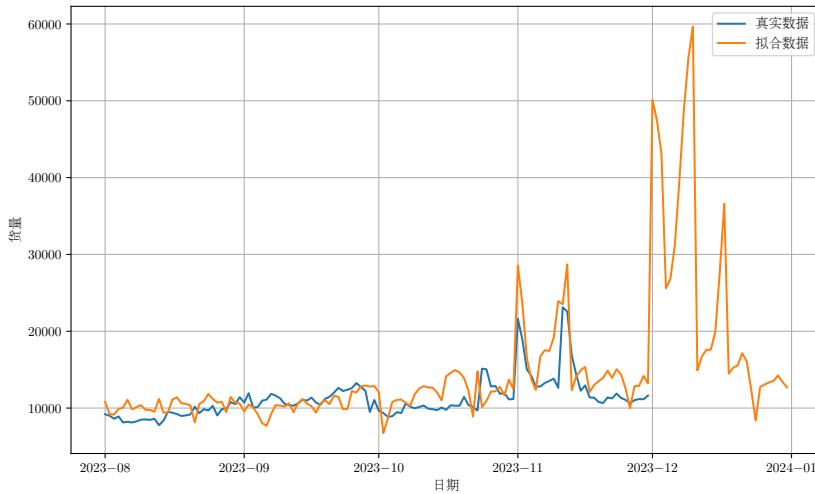


图 2: 利用 LSTM 预测图线示例

经过所输出的平均方差的对比，我们认为使用随机森林算法对于第一题是相对的最优解。因此我们对未来一个月之内每一个小时的货量预测也采取完全相同的方法预测。

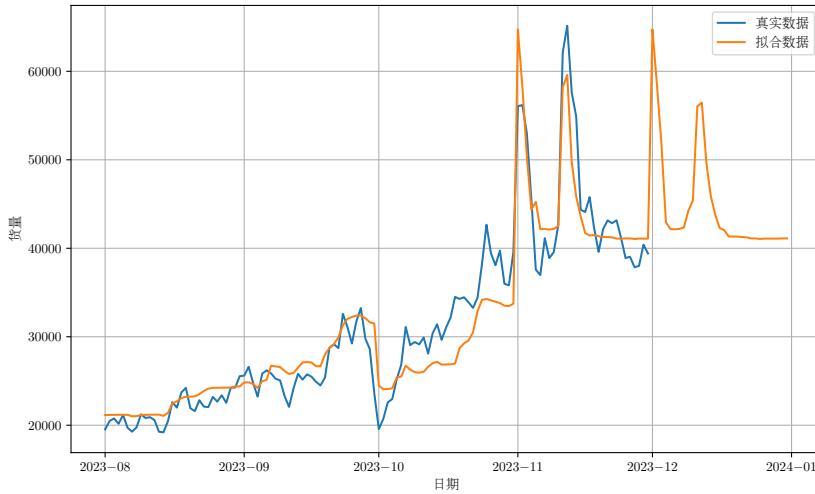


图 3: 利用随机森林预测图线示例 (SC10)

2.3 按小时的数据模拟和预测

按照之前的方法，相似的预测之后的数据，如图5。从图中可以看出，拟合效果符合数据波动的周期性。通过计算拟合模型与真实数据之间的平均方差，我们确定使用随机森林算法来拟合数据。

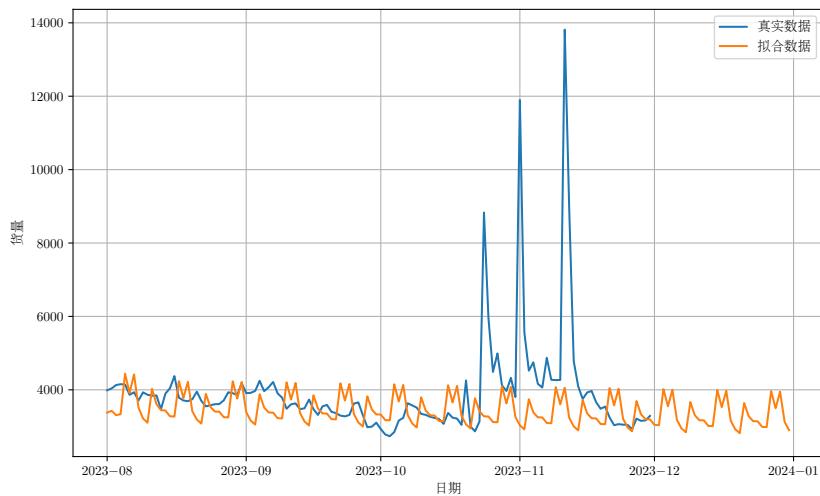


图 4: 利用 SARIMA 预测图线示例 (SC55)

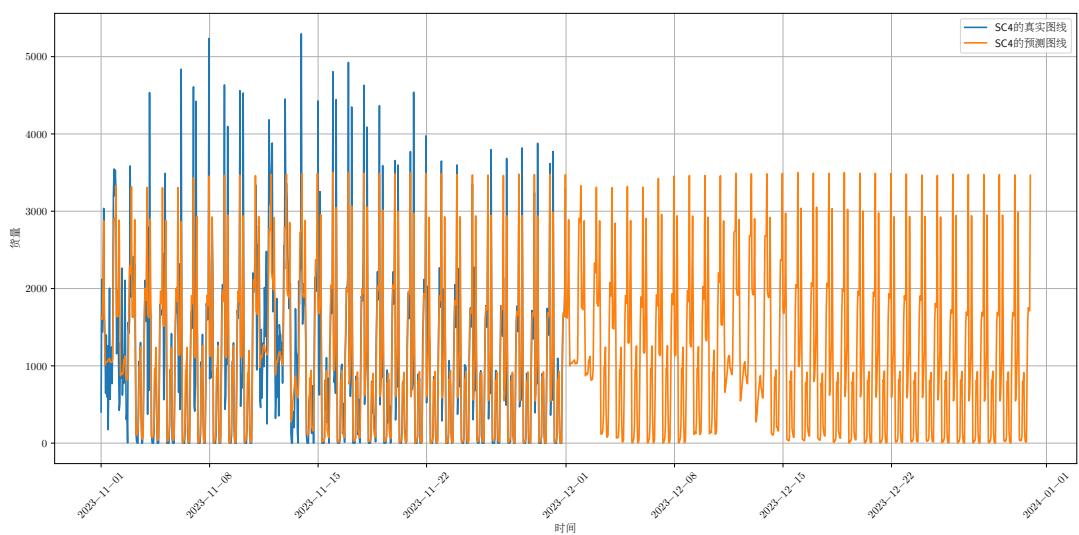


图 5: 利用随机森林模型预测小时图线示例 (SC4)

3 问题二的建模与求解

3.1 分析与建模求解

我们需要对已知的关系进行分析。首先先将仓库之间的关联进行可视化，我们选用弦图的形式，如图6。由于在现有的运输关系模型上发生了变动，于是我们需要使用模型在学习之后改变参数来进行预测。因此我们的流程大概为数据预处理、特征提取、模型训练和预测。

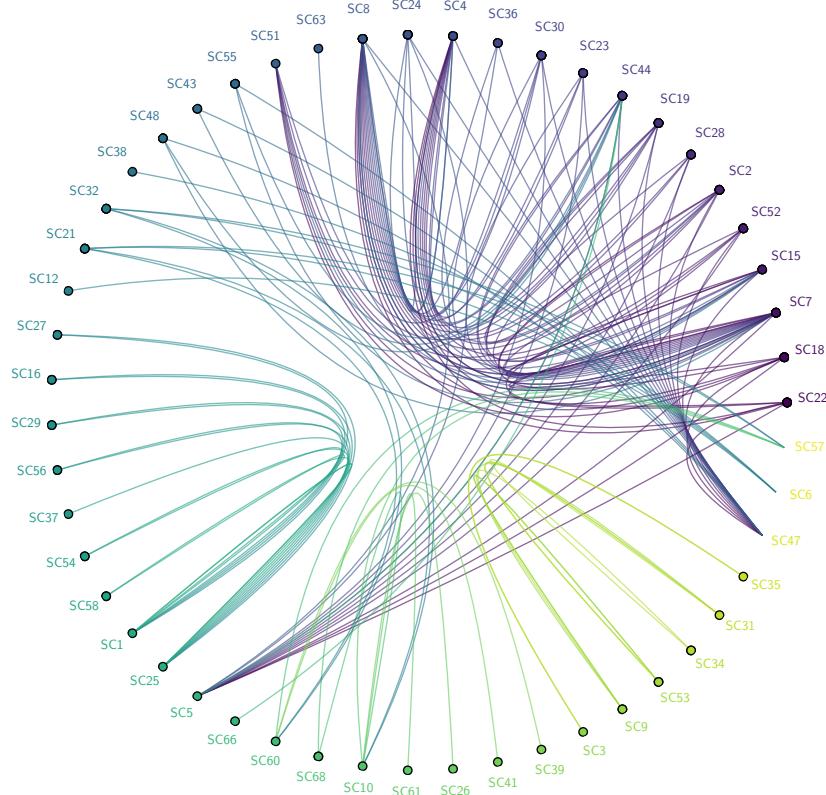


图 6: 分配中心之间的关系弦图

3.1.1 数据预处理

首先我们需要使用独热编码将分类变量转换为数值。独热编码（One-Hot Encoding）是一种处理分类数据的常用方法，由于许多算法更好地处理数值数据而不是文本数据，因此这种方法尤其适用于机器学习模型中。在独热编码中，每个类别值都被转换为一个二进制向量，该向量中只有一个位置是 1，其余位置都是 0。在这个问题中独热编码被用于“分拣中心”这个分类特征。这样做可以将这些文本标签转换为模型可解释的数值格式，同时保持不同分拣中心之间的独立性。

3.1.2 时间序列预测

时间序列预测主要涉及根据过去的数据点预测未来的值，也就是将时间转换为一个可以作为模型输入的特征。在这里我们将日期时间对象转换为相对于起始时间点的总秒数来表示的，这种转换使得模型能够理解时间的线性进展。

3.1.3 退火算法

在这里我们也可以选择使用退火算法求解，所谓退火算法 (Simulated Annealing, SA) 是一种用于寻找大规模搜索空间中的近似全局最优解的随机化算法，它受到物理中固体退火过程的启发。虽然该算法在很多优化问题中表现出了不错的效果，但它也存在一些缺点：

1. 收敛速度慢：退火算法的一个主要缺点是它的收敛速度相对较慢，尤其是在接近全局最优解时。算法需要执行大量的迭代和随机搜索步骤，这在大型问题上可能导致较高的计算成本。
2. 参数依赖性强：退火算法的效果在很大程度上依赖于其参数的设定，如初始温度、冷却速率和温度下降函数等。不恰当的参数设置可能导致算法效果不佳，如过早冷却可能使算法陷入局部最优解。
3. 随机性：作为一种基于概率的优化方法，退火算法的性能可能因随机性而波动。这意味着同一个问题的多次运行可能会得到不同的结果。
4. 解的质量保证问题：由于退火算法通常只能找到近似解，而不是确切的全局最优解。

尽管如此，我们依然实现了退火算法的代码。以下是大概的流程：

1. 生成初始解：随机为每个日期和班次分配 200 名员工中的员工，确保每名员工都至少被分配到一个班次。
2. 目标函数的定义：定义一个目标函数来计算解的总成本，包括：
 - 1) 每个员工的总天数成本（如果超过了允许的最大工作天数）。
 - 2) 每个班次中全职和临时员工数目的成本（如果班次的人员配置无法满足需求量）。
 - 3) 员工工作日数的方差惩罚，以平衡员工的工作负荷。
3. 生成邻域解：随机选择一个日期和两个班次，交换这两个班次中的一名员工，以生成新的邻域解。
4. 模拟退火算法：从初始解开始，通过温度控制和接受概率（根据成本差异和当前温度）逐步更新解，以期找到成本更低的解决方案。算法使用了温度衰减因子和迭代次数，直到温度低于最小阈值。

预期的结果是模拟退火算法自动化了员工排班过程，旨在平衡员工的工作时间并尽可能满足每个班次的需求量，同时保持成本效益。然而事实上效果并不如人意，因此我们放弃了这个结果，最后选择使用随机森林算法进行拟合。

3.1.4 随机森林回归

接下俩首先将时间（完整时间）转换为自起始时间以来的总秒数，这样可以将问题转换为一个回归问题，其中输入特征是连续的时间值。齐次模型训练中使用上述时间特征和历史货量数据训练随机森林模型。最后将未来的时间点同样转换为自起始时间以来的总秒数，使用训练好的随机森林模型预测这些时间点的货量。

这种方法利用了随机森林的强大能力来处理可能的非线性关系，并可以很好地应对由于特征与目标之间复杂关系造成的预测挑战。我们在随机森林中选取小时，日，周，月作为特征进行训练，最后以模型更改线路连接的参数获取预测的数据。具体代码详见附录。

4 问题三的建模与求解

4.1 分析

表 1: 符号说明

符号	说明
$R(c, d, s)$	表示第 c 个分拣中心在第 d 天的第 s 个班次中正式工的出勤人数。
$T(c, d, s)$	表示第 c 个分拣中心在第 d 天的第 s 个班次中临时工的出勤人数。
$V(c, d, s)$	表示第 c 个分拣中心在第 d 天的第 s 个班次中预测货物量

我们的目标是最小化总人天数，即所有分拣中心在所有班次中正式工和临时工的总人数。目标函数可以写为

$$\text{Minimize } Z = \sum_c \sum_d \sum_s (R(c, d, s) + T(c, d, s))$$

约束条件是

- 处理货物量约束：每个班次的员工人数必须足以处理该班次的预测货物量。如果假设一个正式员工每小时可以处理 25 个单位的货物，一个临时员工每小时可以处理 20 个单位，则每个班次的约束为：

$$25R(c, d, s) + 20T(c, d, s) \geq V(c, d, s)$$

- 正式员工数量限制：每个分拣中心每天的正式员工数量不能超过 60 人。对于每个日期 d 和分拣中心 c ，约束可以写为：

$$\sum_s R(c, d, s) \leq 60$$

4.2 建模与求解

针对这个问题，我们选择开源的 Python/PuLP 库求解。PuLP 作为一个 Python 库，可以定义问题、变量、目标函数和约束，然后使用 CBC 或其他求解器求解这些问题。这里我们选择在 PuLP 库中调用 CBC (Coin-or branch and cut) 解决这个问题。

我们将约束条件和目标函数录入 Python 脚本中，对每一个分拣中心单独进行运算。如下是运行时截取的一小段结果。可以看出效果较好，能够在可接受的时间内解决问题。（使用容忍度 Gap 为 0.01）

```
1 Result - Optimal solution found (within gap tolerance)
2
3 Objective value:          7569.00000000
4 Lower bound:              7514.735
5 Gap:                      0.01
```

```

6 Enumerated nodes:          167447
7 Total iterations:          849255
8 Time (CPU seconds):       101.51
9 Time (Wallclock seconds): 30.31
10
11 Option for printingOptions changed from normal to all
12 Total time (CPU seconds): 101.51   (Wallclock seconds): 30.31

```

我们选取一天的安排来查看。这是随机选取的一个分拣中心的预测货量。

表 2: SC14 的按小时预测货量

分配中心	日期	小时	预测货量
SC14	2023-12-01	0	71.81666666666666
SC14	2023-12-01	1	71.81666666666666
SC14	2023-12-01	2	71.81666666666666
SC14	2023-12-01	3	71.81666666666666
SC14	2023-12-01	4	71.81666666666666
SC14	2023-12-01	5	101.65666666666668
SC14	2023-12-01	6	117.33666666666666
SC14	2023-12-01	7	68.72666666666667
SC14	2023-12-01	8	76.79154761904765
SC14	2023-12-01	9	104.46576984126985
SC14	2023-12-01	10	106.61277777777778
SC14	2023-12-01	11	149.3447380952381
SC14	2023-12-01	12	2.7871190476190475
SC14	2023-12-01	13	102.0495714285714
SC14	2023-12-01	14	168.87423015873014
SC14	2023-12-01	15	148.3816865079365
SC14	2023-12-01	16	112.3909404761905
SC14	2023-12-01	17	99.52757570207568
SC14	2023-12-01	18	43.27497619047618
SC14	2023-12-01	19	132.01944444444447
SC14	2023-12-01	20	115.97620601620604
SC14	2023-12-01	21	77.28322222222221
SC14	2023-12-01	22	54.38195238095238
SC14	2023-12-01	23	17.495702380952384

这是安排好的排班表。

表 3: SC14 的排班表

分配中心	日期	排班时间表	正式员工	临时员工
SC14	2023-12-01	00:00-08:00	26.0	0.0
SC14	2023-12-01	05:00-13:00	9.0	11.0
SC14	2023-12-01	08:00-16:00	13.0	5.0
SC14	2023-12-01	12:00-20:00	2.0	17.0
SC14	2023-12-01	14:00-22:00	7.0	1.0
SC14	2023-12-01	16:00-24:00	3.0	0.0

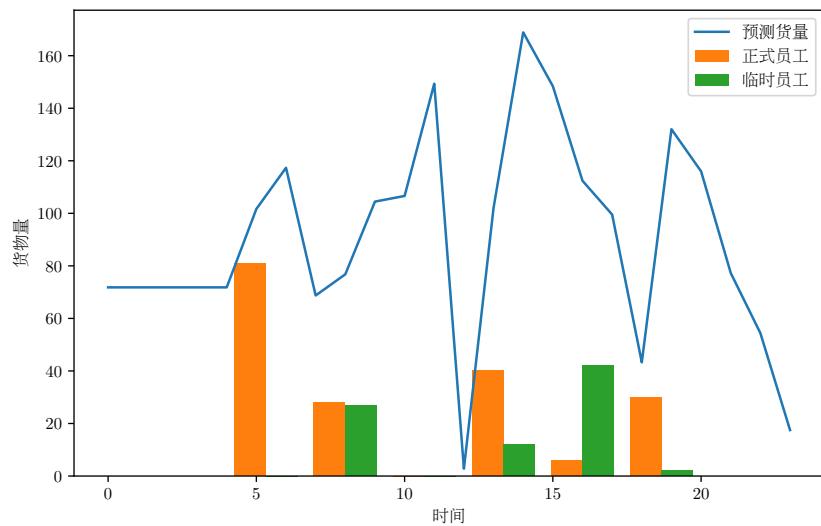


图 7: 员工的货物处理量与预测货物量示意图

可以看到满足所需的要求，并且能够按照预测的货物量准确的选取员工。如图所示，条形图部分是正式员工和临时员工处理货物量对排班时间的平均。需要注意的是，在排班间存在重合，在 15 小时附近可能受到左右四排班的叠加，是可以满足需求的。经过计算检验，可以初步认定效果良好。

5 问题四的建模与求解

5.1 问题四的分析

这个问题是关于如何调度员工的最优解。这是一个线性规划问题，目标是最小化总人力成本，并同时满足工作班次的需求。下面是关于一些变量的假设。

表 4: 符号说明

符号	说明
$F(i, d, s)$	二元变量，表示固定员工 i 在日期 d 和班次 s 是否工作
$T(d, s)$	整数变量，表示在日期 d 和班次 s 需要的临时工人数
$D(d, s)$	整数变量，表示在日期 d 和班次 s 所需的工作需求

我们的目标函数是

$$\text{Minimize } Z = \sum_{d \in \text{Dates}} \sum_{s \in \text{Shifts}} \left(\sum_{i=1}^{200} F(i, d, s) + T(d, s) \right)$$

即最小化全职和临时工的总人天数。这个最小化函数需要满足以下几个约束条件：

- 需求满足：

$$\forall d \in \text{Dates}, \forall s \in \text{Shifts} : \sum_{i=1}^{200} F(i, d, s) + 20 \times T(d, s) \geq D(d, s)$$

这个约束确保每个班次的工作需求被满足。这里，25 和 20 是每个工作的产出（或能力）。

- 正式工出勤率不超过 85%：

$$\forall i : \sum_{d \in \text{Dates}} \sum_{s \in \text{Shifts}} F(i, d, s) \leq 30 \times 0.85$$

此约束限制每个全职员工在一个月内的最大工作天数。

- 连续工作天数不超过 7 天：

$$\forall i, \forall d \in \text{Dates} \text{ if } d + 6 \leq \text{len}(\text{Dates}) : \sum_{k=0}^6 \sum_{s \in \text{Shifts}} F(i, d + k, s) \leq 7$$

这确保任何全职员工在连续 7 天内的工作天数不超过 7 天。

- 每人每天只能工作一个班次：每个员工每天只能安排一个班次

$$\sum_s F(i, d, s) \leq 1$$

5.2 建模求解

与问题三相同，我们使用 PuLP 求解。

由于数量较大，我们设置了一个容忍度 0.001，这是我们在这个容忍度下得到的结果。

```
1 Result - Optimal solution found (within gap tolerance)
2
3 Objective value:          204322.00000000
4 Lower bound:              204196.332
5 Gap:                      0.00
6 Enumerated nodes:         0
7 Total iterations:         0
8 Time (CPU seconds):      1.64
9 Time (Wallclock seconds): 1.85
10
11 Option for printingOptions changed from normal to all
12 Total time (CPU seconds): 1.72   (Wallclock seconds): 1.94
13
14 CSV file has been saved with the scheduling results.
```

可以看出运算结果符合预期。

6 总结

6.1 使用随机森林的优点

1. 随机森林算法 (RF) 不太容易过拟合，因为 RF 本质上是模型集成 (model ensemble)，从 Leo Breiman 的理论来看 RF 也不会因为树数量的增加，而导致过拟合，因为这些数都是集合在一起的单独模型，效果不好的树会被降低权重 (downvote)。[1]
2. 数据分布要求低。RF 不像线性模型，不要求数据分布符合正态分布，来得到统计结果上的近似。因此任意的数据分布都可以使用 RF。
3. 特征工程：对于一些简单的线性模型，为了增加特征，我们往往需要增加如 $x_1^2 + x_2^2$ 等特征来作为模型的输入，帮助模型构建更多的特征。但是在 RF 中，这些基础的特征工程是不必要的。但是在本题的求解过程中，我们使用了高阶特征工程，以此增加 RF 的精度（类似日期上的处理，提取出月份，周数等）。
4. 数据预处理：类似神经网络需要对数据作预处理来得到 0 ~ 1 之间的数据分布，在 RF 这里往往都不太需要，因此 RF 对于数据的要求也不高。
5. 求解速度快，求解准确性好 (mse 最小)

6.2 问题一和问题二建模存在的缺点

由于在统计过程中可以发现由于出现了一些异常值，这些异常值体现为节假日或者特殊情况。我们如果可以实现多模型集体学习，或许可以将各种模型的优点结合起来。同时，使用随机森林算法在本问题的求解过程中有以下不足：RF 的泛化能力较强，不太容易出现对极值的反应。当然我们也因此，在模型的求解过程中，我们添加了“2023-12-01 至 2023-12-31 期间不会发生如购物节等可能导致货物量激增的特殊事件”这一基本假设。

6.3 PULP 效果评估

使用 PULP (Python 用于优化) 模型来解决排班问题具有优点：

1. 灵活性:PULP 模型能够处理各种约束条件和目标函数，因此适用于各种不同的排班场景。在这个情景中，我们能够轻松地添加和修改约束条件和目标函数来满足特定的需求。
2. 求解器支持:PULP 支持多种优化求解器，包括 CBC、GLPK、GUROBI 等。这些求解器能够高效地解决中等规模的优化问题。

当然也存在以下缺点：

1. 性能受限:PULP 适用于中等规模的优化问题，对于大规模的问题可能存在性能上的限制。当问题规模较大时，求解可能会变得非常耗时。
2. 精确度问题: 由于某些算法的限制，PULP 可能不能总是找到全局最优解，尤其是对于复杂的非线性问题。在实践中，可能需要通过调整求解器参数或者使用其他方法来提高精确度。

3. 难以处理复杂约束: 虽然 PULP 提供了丰富的约束类型和灵活的约束表达方式, 但对于某些复杂的约束条件, 可能需要额外的技巧或者结合其他工具来处理。

结合情景分析: 在这个排班问题的情景下, 使用 PULP 是合适的选择。该问题的约束条件相对简单, 且规模适中, 不会导致求解器性能方面的问题。通过 PULP, 我们能够轻松地建立模型, 定义约束条件和目标函数, 并使用求解器求解得到最优的排班方案。然而, 如果问题规模变得非常大, 例如涉及数千名员工和数十个班次的情况, 那么可能需要考虑更高效的方法来解决。

参考文献

- [1] L. Breiman. Random forests, machine learning 45. Journal of Clinical Microbiology, 2:199–228, 2001.
- [2] Leo Breiman. Random forests. Machine learning, 45:5–32, 2001.
- [3] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks, 18(5):602–610, 2005. IJCNN 2005.
- [4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
- [5] Paul Newbold and Clive William John Granger. Experience with forecasting univariate time series and the combination of forecasts. 1974.

附录

A 问题一代码

随机森林调参代码

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.ensemble import RandomForestRegressor
4 from sklearn.model_selection import train_test_split, GridSearchCV
5 from sklearn.metrics import mean_squared_error
6 from sklearn.preprocessing import StandardScaler
7
8 # 假设存在的 SCid 列表
9 data_for_sc = pd.read_csv("../附件/附件1.csv", encoding="GB2312")
10 ALL_SC = list(set(data_for_sc["分拣中心"]))
11 existing_scs = list(map(lambda SC_: int(SC_[2:]), ALL_SC))
12 existing_scs.sort()
13
14
15 # 加载数据
16 def load_data(existing_scs):
17     all_data = []
18     for sc_id in existing_scs:
19         try:
20             data = pd.read_csv(f"SC{sc_id}.csv")
21             data["center_id"] = sc_id
22             all_data.append(data)
23         except FileNotFoundError:
24             print(f"File for center {sc_id} not found.")
25     return pd.concat(all_data, ignore_index=True) if all_data else pd.DataFrame()
26
27
28 # 数据清洗和预处理
29 def preprocess_data(data):
30     data["date"] = pd.to_datetime(data["date"], errors="coerce")
31     data.dropna(subset=["date", "value"], inplace=True)
32
33     data["year"] = data["date"].dt.year
34     data["month"] = data["date"].dt.month
35     data["day"] = data["date"].dt.day
36     data["weekday"] = data["date"].dt.weekday
37
38     scaler = StandardScaler()
39     data[["year", "month", "day", "weekday"]] = scaler.fit_transform(
40         data[["year", "month", "day", "weekday"]]
41     )
42
```

```

43     return data
44
45
46 # 模型训练和参数调整
47 def train_and_optimize_model(X_train, y_train):
48     param_grid = {
49         "n_estimators": [100 * i for i in range(1, 10)],
50         "max_depth": [10 * i for i in range(1, 10)],
51         "min_samples_split": [10 * i for i in range(1, 10)],
52     }
53     model = RandomForestRegressor(random_state=42)
54     grid_search = GridSearchCV(
55         model, param_grid, cv=5, scoring="neg_mean_squared_error", verbose=2
56     )
57     grid_search.fit(X_train, y_train)
58
59     print("Best parameters:", grid_search.best_params_)
60     return grid_search.best_estimator_
61
62
63 if __name__ == "__main__":
64     data = load_data(existing_scs)
65     if not data.empty:
66         data = preprocess_data(data)
67         features = data[["center_id", "year", "month", "day", "weekday"]]
68         target = data["value"]
69
70         X_train, X_test, y_train, y_test = train_test_split(
71             features, target, test_size=0.2, random_state=42
72         )
73
74         best_model = train_and_optimize_model(X_train, y_train)
75
76         y_pred = best_model.predict(X_test)
77         mse = mean_squared_error(y_test, y_pred)
78         print(f"Test MSE: {mse}")
79     else:
80         print("No data loaded, please check the data files.")

```

随机森林训练代码

```

1 import pandas as pd
2 import numpy as np
3 from sklearn.ensemble import RandomForestRegressor
4 from sklearn.model_selection import train_test_split
5 from sklearn.metrics import mean_squared_error
6
7 # 假设数据已经按照分拣中心编号和日期排列好
8 # 每个文件名格式为'SCi.csv'，其中 i 是分拣中心编号

```

```

9 # 假设存在的 SCid 列表
10 data_for_sc = pd.read_csv("../附件/附件1.csv", encoding="GB2312")
11 ALL_SC = list(set(data_for_sc["分拣中心"]))
12 existing_scs = list(map(lambda SC_: int(SC_[2:]), ALL_SC))
13 existing_scs.sort()
14
15 # 加载数据
16 def load_data(existing_scs):
17     all_data = []
18     for i in existing_scs:
19         data = pd.read_csv(f"SC{i}.csv")
20         data["SC_id"] = i
21         all_data.append(data)
22     return pd.concat(all_data, ignore_index=True)
23
24
25 # 数据预处理
26 def preprocess(data):
27     # 假设数据包含日期和货量两列，日期列名为'date'，货量列名为'value'
28     data["date"] = pd.to_datetime(data["date"])
29     data["year"] = (pd.to_datetime(data["date"])).dt.year
30     data["month"] = (pd.to_datetime(data["date"])).dt.month
31     data["day"] = (pd.to_datetime(data["date"])).dt.day
32     data["weekday"] = (pd.to_datetime(data["date"])).dt.weekday
33     return data
34
35
36 # 训练模型
37 def train_model(data):
38     features = data[["SC_id", "year", "month", "day", "weekday"]]
39     target = data["value"]
40     X_train, X_test, y_train, y_test = train_test_split(
41         features,
42         target,
43         test_size=0.2,
44         random_state=50,
45
46     )
47
48     model = RandomForestRegressor(n_estimators=300, random_state=42, min_samples_split=20)
49     model.fit(X_train, y_train)
50
51     # 预测和评估
52     y_pred = model.predict(X_test)
53     mse = mean_squared_error(y_test, y_pred)
54     print(f"Mean Squared Error: {mse}")
55     return model
56

```

```

57
58 # 预测未来的货量
59 def predict_future(model, start_date, num_days, existing_scs):
60     future_dates = pd.date_range(start_date, periods=num_days)
61     future_data = pd.DataFrame(
62         {
63             "date": np.repeat(future_dates, len(existing_scs)),
64             "SC_id": np.tile(existing_scs, num_days),
65         }
66     )
67     future_data = preprocess(future_data)
68     features = future_data[["SC_id", "year", "month", "day", "weekday"]]
69     predictions = model.predict(features)
70     future_data["predicted_volume"] = predictions
71     return future_data
72
73
74 # 保存结果到 CSV
75 def save_predictions_to_csv(predictions, file_name):
76     predictions["date"] = predictions["date"].dt.strftime("%Y/%m/%d")
77     predictions.to_csv(file_name, index=False)
78     print(f"Saved predictions to {file_name}")
79
80
81 # 主函数
82 def main():
83     data = load_data(existing_scs)
84     data = preprocess(data)
85     model = train_model(data)
86     future_predictions = predict_future(model, "2023-08-01", 153, existing_scs)
87     save_predictions_to_csv(future_predictions, "predicted_volumes.csv")
88
89
90 if __name__ == "__main__":
91     main()

```

ARIMA 和 LSTM 的代码未被采用，因此仅放在支撑材料中。

B 问题二代码

按小时预测

```

1 import pandas as pd
2 from sklearn.ensemble import RandomForestRegressor
3 from sklearn.preprocessing import OneHotEncoder
4
5
6 def load_data():
7     # 修改这里以加载附件 2

```

```

8     data_sc = pd.read_csv("附件2.csv", encoding="gb2312")
9     data_routes = pd.read_csv("附件3.csv", encoding="gb2312")
10    data_changes = pd.read_csv("附件4.csv", encoding="gb2312")
11    return data_sc, data_routes, data_changes
12
13
14
15 def preprocess_data(data_sc, data_changes):
16     # Combining date and hour into a single datetime column
17     data_sc["完整时间"] = pd.to_datetime(
18         data_sc["日期"].astype(str) + " " + data_sc["小时"].astype(str) + ":00"
19     )
20
21     # Adding new time-related features
22     data_sc["小时"] = data_sc["完整时间"].dt.hour # Hour of the day
23     data_sc["星期几"] = data_sc["完整时间"].dt.weekday # Day of the week (Monday=0,
Sunday=6)
24     data_sc["月份"] = data_sc["完整时间"].dt.month # Month of the year
25
26     # One-Hot Encoding for sorting centers
27     ohe = OneHotEncoder()
28     encoded_centers = ohe.fit_transform(data_sc[["分拣中心"]].astype(str)).toarray()
29     center_df = pd.DataFrame(encoded_centers, columns=ohe.get_feature_names_out())
30     data_sc = pd.concat([data_sc, center_df], axis=1)
31
32     return data_sc, ohe
33
34
35 def predict_future_volume(data_sc, ohe, data_changes):
36     # Generating data for the future date range within each hour
37     last_time = data_sc["完整时间"].max()
38     future_dates = pd.date_range(last_time + pd.Timedelta(hours=1), periods=744, freq="h")
39     future_data = pd.DataFrame({"完整时间": future_dates})
40     future_data["小时"] = future_data["完整时间"].dt.hour
41     future_data["星期几"] = future_data["完整时间"].dt.weekday
42     future_data["月份"] = future_data["完整时间"].dt.month
43
44     # Predicting future volume for each sorting center
45     future_volumes = []
46     centers = ohe.get_feature_names_out()
47     for center in centers:
48         center_data = data_sc[data_sc[center] == 1]
49         if not center_data.empty:
50             # Train a model to predict volume
51             X = center_data[["小时", "星期几", "月份"]].values
52             y = center_data["货量"].values
53             model = RandomForestRegressor(n_estimators=100, random_state=42)
54             model.fit(X, y)

```

```

55
56     # Use the model to predict future volume
57     future_X = future_data[["小时", "星期几", "月份"]].values
58     future_y = model.predict(future_X)
59     for date, volume in zip(future_dates, future_y):
60         cleaned_center_name = center.replace("x0_分拣中心_", "")
61         future_volumes.append(
62             [cleaned_center_name, date.strftime("%Y/%m/%d %H:%M"), volume]
63         )
64
65     future_volumes_df = pd.DataFrame(
66         future_volumes, columns=["分拣中心", "日期时间", "货量"]
67     )
68
69     # Considering changes in routing paths from Attachment 4
70     future_volumes_df.set_index(["分拣中心", "日期时间"], inplace=True)
71     for index, row in data_changes.iterrows():
72         from_center = "x0_分拣中心_" + row["始发分拣中心"]
73         to_center = "分拣中心_" + row["到达分拣中心"]
74         if from_center in centers:
75             future_volumes_df.loc[(to_center,), "货量"] += (
76                 future_volumes_df.loc[(from_center,), "货量"] * 0.1
77             )
78             future_volumes_df.loc[(from_center,), "货量"] *= 0.9
79
80     future_volumes_df.reset_index(inplace=True)
81     return future_volumes_df
82
83
84 def main():
85     data_sc, data_routes, data_changes = load_data()
86     data_sc, ohe = preprocess_data(data_sc, data_changes)
87     future_volumes_df = predict_future_volume(data_sc, ohe, data_changes)
88     future_volumes_df.to_csv("predicted_future_volumes_hours.csv", index=False)
89     print("预测完成, 结果已保存到 'predicted_future_volumes_hours.csv'")
90
91
92 if __name__ == "__main__":
93     main()

```

按天预测

```

1 import pandas as pd
2 from sklearn.ensemble import RandomForestRegressor
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import OneHotEncoder
5 from sklearn.metrics import mean_squared_error
6
7

```

```

8 def load_data():
9     # 加载数据
10    data_sc = pd.read_csv("附件1.csv", encoding="gb2312")
11    data_routes = pd.read_csv("附件3.csv", encoding="gb2312")
12    data_changes = pd.read_csv("附件4.csv", encoding="gb2312")
13    return data_sc, data_routes, data_changes
14
15
16 def preprocess_data(data_sc, data_changes):
17     # Convert '日期' to datetime and extract more features
18     data_sc["日期"] = pd.to_datetime(data_sc["日期"], format="%Y/%m/%d")
19     data_sc["月份"] = data_sc["日期"].dt.month # Month as a feature
20     data_sc["星期几"] = data_sc["日期"].dt.weekday # Day of the week as a feature
21     data_sc["年中日"] = data_sc["日期"].dt.dayofyear # Day of the year as a feature
22
23     # One-Hot Encoding for sorting centers
24     ohe = OneHotEncoder()
25     encoded_centers = ohe.fit_transform(data_sc[["分拣中心"]].astype(str)).toarray()
26     center_df = pd.DataFrame(encoded_centers, columns=ohe.get_feature_names_out())
27     data_sc = pd.concat([data_sc, center_df], axis=1)
28
29     return data_sc, ohe
30
31
32 def predict_future_volume(data_sc, ohe, data_changes):
33     # Generate future dates starting 120 days after the minimum date in the data
34     future_dates = pd.date_range(start=data_sc["日期"].max() + pd.Timedelta(days=1), periods
=31, freq="D")
35     future_data = pd.DataFrame({"日期": future_dates})
36     future_data["月份"] = future_data["日期"].dt.month
37     future_data["星期几"] = future_data["日期"].dt.weekday
38     future_data["年中日"] = future_data["日期"].dt.dayofyear
39
40     # Predict future volume for each sorting center
41     future_volumes = []
42     centers = ohe.get_feature_names_out()
43     for center in centers:
44         center_data = data_sc[data_sc[center] == 1]
45         if not center_data.empty:
46             # Train a model to predict volume
47             X = center_data[["年中日", "月份", "星期几"]]
48             y = center_data["货量"]
49             model = RandomForestRegressor(n_estimators=100, random_state=42)
50             model.fit(X, y)
51
52             # Predict future volume
53             future_X = future_data[["年中日", "月份", "星期几"]]
54             future_y = model.predict(future_X)

```

```

55     for date, volume in zip(future_dates, future_y):
56         cleaned_center_name = center.replace("x0_分拣中心_", "")
57         future_volumes.append(
58             [cleaned_center_name, date.strftime("%Y/%m/%d"), volume]
59         )
60
61     future_volumes_df = pd.DataFrame(
62         future_volumes, columns=["分拣中心", "日期", "货量"]
63     )
64
65     # Adjust predicted volumes based on changes from Attachment 4
66     future_volumes_df.set_index(["分拣中心", "日期"], inplace=True)
67     for index, row in data_changes.iterrows():
68         from_center = "x0_分拣中心_" + row["始发分拣中心"]
69         to_center = "分拣中心_" + row["到达分拣中心"]
70         if from_center in centers:
71             future_volumes_df.loc[(to_center,), "货量"] += (
72                 future_volumes_df.loc[(from_center,), "货量"] * 0.1
73             )
74             future_volumes_df.loc[(from_center,), "货量"] *= 0.9
75
76     future_volumes_df.reset_index(inplace=True)
77     return future_volumes_df
78
79
80 def main():
81     data_sc, data_routes, data_changes = load_data()
82     data_sc, ohe = preprocess_data(data_sc, data_changes)
83     future_volumes_df = predict_future_volume(data_sc, ohe, data_changes)
84     future_volumes_df.to_csv("predicted_future_volumes.csv", index=False)
85     print("预测完成, 结果已保存到 'predicted_future_volumes.csv'")
86
87
88 if __name__ == "__main__":
89     main()

```

C 问题三代码

```

1 import pandas as pd
2
3 # Load the provided CSV file to examine its structure
4 file_path = "../problem2/predicted_future_volumes_hours.csv"
5 predicted_volumes = pd.read_csv(file_path)
6 # Convert the date-time string to a datetime object
7 predicted_volumes["日期时间"] = pd.to_datetime(predicted_volumes["日期时间"])
8
9 # Define the shift times

```

```

10 shift_times = {
11     "00:00-08:00": ("00:00", "08:00"),
12     "05:00-13:00": ("05:00", "13:00"),
13     "08:00-16:00": ("08:00", "16:00"),
14     "12:00-20:00": ("12:00", "20:00"),
15     "14:00-22:00": ("14:00", "22:00"),
16     "16:00-24:00": ("16:00", "24:00"),
17 }
18
19
20 # Create a function to assign each hour to a shift
21 def assign_shift(hour):
22     for shift, (start, end) in shift_times.items():
23         if start <= hour.strptime("%H:%M") < end:
24             return shift
25     return None
26
27
28 # Apply the function to assign shifts
29 predicted_volumes["班次"] = predicted_volumes["日期时间"].apply(
30     lambda x: assign_shift(x)
31 )
32
33 # Group by center, date, and shift to sum up volumes
34 grouped_volumes = (
35     predicted_volumes.groupby(
36         ["分拣中心", predicted_volumes["日期时间"].dt.date, "班次"]
37     )["货量"]
38     .sum()
39     .reset_index()
40 )
41 grouped_volumes.rename(columns={"日期时间": "日期"}, inplace=True)
42
43 for special_SC in grouped_volumes["分拣中心"].unique():
44     pre_grouped_volumes = grouped_volumes[grouped_volumes["分拣中心"] == special_SC]
45
46     import pulp
47     from pulp import PULP_CBC_CMD
48
49     # Set up the linear programming problem to minimize the total number of person-days
50     model = pulp.LpProblem("Staff_Scheduling", pulp.LpMinimize)
51
52     # Decision variables
53     # Number of regular and temporary workers per shift, per day, per sorting center
54     regular_workers = pulp.LpVariable.dicts(
55         "Regular",
56         [
57             (center, date, shift)

```

```

58     for center, date, shift in zip(
59         pre_grouped_volumes["分拣中心"],
60         pre_grouped_volumes["日期"],
61         pre_grouped_volumes["班次"],
62         )
63     ],
64     lowBound=0,
65     cat="Integer",
66 )
67 temporary_workers = pulp.LpVariable.dicts(
68     "Temporary",
69     [
70         (center, date, shift)
71         for center, date, shift in zip(
72             pre_grouped_volumes["分拣中心"],
73             pre_grouped_volumes["日期"],
74             pre_grouped_volumes["班次"],
75             )
76     ],
77     lowBound=0,
78     cat="Integer",
79 )
80
81 # Objective: Minimize the total person-days
82 model += pulp.lpSum(
83     [
84         regular_workers[center, date, shift]
85         + temporary_workers[center, date, shift]
86         for center, date, shift in zip(
87             pre_grouped_volumes["分拣中心"],
88             pre_grouped_volumes["日期"],
89             pre_grouped_volumes["班次"],
90             )
91     ]
92 )
93
94 # Constraints
95 # Each shift's staffing needs must meet the volume requirements
96 for i, row in pre_grouped_volumes.iterrows():
97     center, date, shift, volume = (
98         row["分拣中心"],
99         row["日期"],
100        row["班次"],
101        row["货量"],
102        )
103     model += (
104         25 * regular_workers[center, date, shift]
105         + 20 * temporary_workers[center, date, shift]

```

```

106         >= volume
107     )
108
109     # Maximum of 60 regular workers per sorting center per day
110     for (center, date), group in pre_grouped_volumes.groupby(["分拣中心", "日期"]):
111         model += (
112             pulp.lpSum(regular_workers[center, date, shift] for shift in group["班次"])
113             <= 60
114         )
115
116     # Solve the model
117     model.solve(PULP_CBC_CMD(msg=1, threads=8, gapRel=0.05))
118     # model.solve()
119
120     # Output results
121     results = []
122     for center, date, shift in zip(
123         pre_grouped_volumes["分拣中心"], pre_grouped_volumes["日期"], pre_grouped_volumes["班
124         次"])
125     ):
126         result = {
127             "分拣中心": center,
128             "日期": date,
129             "班次": shift,
130             "正式工": regular_workers[center, date, shift].varValue,
131             "临时工": temporary_workers[center, date, shift].varValue,
132         }
133         results.append(result)
134
135     results_df = pd.DataFrame(results)
136     results_df.to_csv(f"sol3_/{special_SC}scheduling_results.csv", index=False)

```

D 问题四代码

```

1 import pandas as pd
2 from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpBinary, PULP_CBC_CMD
3
4 # 读取数据
5 df = pd.read_csv("for_read.csv")
6 df["date"] = pd.to_datetime(df["date"]).dt.date
7
8 # 班次及时间段
9 shifts = [(0, 8), (5, 13), (8, 16), (12, 20), (14, 22), (16, 24)]
10 shift_labels = ["Shift1", "Shift2", "Shift3", "Shift4", "Shift5", "Shift6"]
11 dates = sorted(df["date"].unique())
12
13 # 计算每个班次的需求

```

```

14 demand_per_shift = {date: {} for date in dates}
15 for date in dates:
16     daily_data = df[df["date"] == date]
17     for label, (start, end) in zip(shift_labels, shifts):
18         demand_per_shift[date][label] = daily_data[
19             (daily_data["hour"] >= start) & (daily_data["hour"] < end)
20         ]["value"].sum()
21
22 # 建立优化模型
23 model = LpProblem("Personnel Scheduling", LpMinimize)
24
25 # 定义变量
26 full_time = LpVariable.dicts(
27     "FullTime", (dates, shift_labels, range(200)), 0, 1, LpBinary
28 )
29 temp_workers = LpVariable.dicts(
30     "TempWorkers", (dates, shift_labels), 0, None, cat="Integer"
31 )
32
33 # 目标函数: 最小化总人天数
34 model += lpSum(
35     full_time[date][shift][i]
36     for date in dates
37     for shift in shift_labels
38     for i in range(200)
39 ) + lpSum(temp_workers[date][shift] for date in dates for shift in shift_labels)
40
41 # 每个班次的需求必须被满足的约束
42 for date in dates:
43     for shift in shift_labels:
44         model += (
45             25 * lpSum(full_time[date][shift][i] for i in range(200))
46             + 20 * temp_workers[date][shift]
47             >= demand_per_shift[date][shift]
48         )
49
50 # 正式工的出勤率不超过 85
51 for i in range(200):
52     model += (
53         lpSum(full_time[date][shift][i] for date in dates for shift in shift_labels)
54         <= 30 * 0.85
55     )
56
57 # 正式工连续出勤天数不超过 7 天
58 for i in range(200):
59     for d in range(len(dates) - 6):
60         model += (
61             lpSum(
62                 full_time[dates[d + k]][shift][i]

```

```

62         for k in range(7)
63             for shift in shift_labels
64                 )
65             <= 7
66         )
67
68 # 每个人每天只能上一个班次的约束
69 for date in dates:
70     for i in range(200):
71         model += (
72             lpSum(full_time[date][shift][i] for shift in shift_labels) <= 1
73         )
74
75 # 求解问题
76 model.solve(PULP_CBC_CMD(msg=1, threads=8, gapRel=0.001))
77
78 # 收集结果并输出为 CSV
79 results = []
80 for date in dates:
81     for shift in shift_labels:
82         for i in range(200):
83             if full_time[date][shift][i].varValue > 0:
84                 results.append(
85                     {
86                         "Sorting_Center": "SC60",
87                         "Date": date,
88                         "Shift": shift,
89                         "Employee": f"FullTime({i})",
90                     }
91                 )
92             temp_workers_count = int(temp_workers[date][shift].varValue)
93             for j in range(temp_workers_count):
94                 results.append(
95                     {
96                         "Sorting_Center": "SC60",
97                         "Date": date,
98                         "Shift": shift,
99                         "Employee": f"Temp({j})",
100                    }
101                )
102
103 # 创建 DataFrame 并保存到 CSV
104 results_df = pd.DataFrame(results)
105 results_df.to_csv("scheduling_results_2.csv", index=False)
106 print("CSV file has been saved with the scheduling results 2.")

```