基于二元正态统计分布-LSTM 预测与非线性规划的蔬菜商品自动定价与补货决策模型

摘要

本文通过分析蔬菜各品类及单品销售量的分布规律和相互关系，一方面**实现了对蔬菜商品销售流水数据的可视化**，另一方面构建了基于**销售量与定价间的二元正态统计分布规律**和 **LSTM 神经网络预测**的**非线性规划模型**，该模型可以实现销售量与定价 [1] 直接的相互约束和预测，同时可以实现**对利润的非线性规划与补货和定价决策**。这对于中小型蔬菜商品超市的经营者和管理者来说具有重要的现实意义。

针对问题一，利用可视分析探寻了**特征的分布规律与相关性**。首先将附件中的数据进行匹配连接，并对一些**销售方式发生变化的蔬菜商品**将进行**近似处理**，通过 **Pearson 相关系数**来衡量两蔬菜品类或单品间的相关关系，并将商品销售量的分布规律和相关性分析结果进行**可视分析**，以此为出发点，可以更深层次地剖析蔬菜类商品的市场销售情况，对商超的营销策略做出了一定的指导。

针对问题二，利用二元正态统计模型**约束了销售总量与定价**，利用 LSTM 实现了对**成本和销售总量的预测**，利用非线性规划的最优化模型实现了**补货与定价决策**。首先，我们利用**滑动窗口检验**确定了销售总量、定价与时间的序列相关性。其次，我们检验了销售总量与定价数据的正态性，确立了二者联合服从二元正态分布。利用条件分布下的**“3σ 原则”**，我们获得了销售总量与定价的**约束关系**。然后，我们构建 LSTM 模型在时间序列上预测成本价与销售总量，并将这些经验预测结果用于决策。最后，我们考量了蔬菜类商品的运输损耗率，通过**非线性规划**得到了补货量与定价决策 [2]，以实现利润最大化。经检验，我们的模型获得了 **30% 左右的利润率**，和**现实中的蔬菜市场利润率相符**。

针对问题三，我们将问题二中的决策模型细化到每个蔬菜单品，并在**有限销售空间**下完成了**利润提升策略**。结果显示，当对决策进货量小于 2.5kg 的蔬菜单品进行剔除后，我们预测所获利润提高了 180.5 元，实现了一定程度的仓库优化。

针对问题四，我们确定了**天气状况、节假日状况、环境温度和通货膨胀率**为可采集的有益于决策的数据。我们从**自然环境和人文因素**两个方面着手，结合**供给侧与需求侧因素**，调研出了四个重要的数据特征。

关键字： 二元正态统计分布，LSTM，非线性规划，补货与定价决策

一、 问题重述

1.1 问题背景

在生鲜超市中，蔬菜的保质期通常很短，而且如果不能在当天卖出，第二天的品相就会受到影响，难以再销售。因此，超市每天都会根据过往的销售数据和需求来决定补货量。进货的时间一般是凌晨 3:00 到 4:00，而且由于蔬菜种类多样，产地也各不相同，商家需要在不完全了解具体商品和成本的情况下做出补货决策。蔬菜的售价通常是基于“成本加成”的方法来设定的。对于那些因运输损坏或品相不佳的商品，超市会进行打折销售。因此，精准的市场需求分析对于决定补货和定价策略非常关键。从需求方面看，蔬菜的销售量与时间有一定的关系；从供应方面来看，4 月到 10 月是蔬菜种类相对丰富的时期，而超市的销售空间有限，因此合理的商品组合显得尤为重要。

1.2 问题提出

问题的设置是层层递进并且服务于同一个主题的——如何分析蔬菜各品类或单品之间定价和销量的关系，并以此制定补货和销售计划，以期达到利润最大化的目的。

问题一：蔬菜类商品不同品类或者不同单品之间可能并不是各自独立的，而是存在有一定的关联关系，请分析蔬菜各品类及单品销售量的分布规律及相互关系。

问题二：由于商超在制定补货计划时会以品类为单位，请分析各蔬菜品类的销售总量与成本加成定价的关系，并给出各蔬菜品类未来一周（2023 年 7 月 1 日——7 日）的日补货总量和定价策略，使得商超收益最大。

问题三：由于蔬菜类商品的销售空间有限，因此商超希望进一步制定单品的补货计划，要求可售单品的总数控制在 27——33 个，并且满足各单品订购量满足最小陈列量 2.5 千克的要求。根据给出的 2023 年 6 月 24 日——30 日的可售品种，给出 7 月 1 日的单品补货量和对应的定价策略，在尽量满足市场上对各品类蔬菜商品需求的前提下，使得商超收益最大。

问题四：为了能够更好地制定蔬菜商品的补货和定价决策，商超还需要采集哪些相关的数据，这些数据对解决上述问题有何帮助？请给出相应的意见和理由。

二、 问题分析

2.1 总体分析

本题目是一个关于蔬菜类商品的**数据挖掘与决策**问题。图 1 展示了我们问题的分析流程。

从决策目的来看，本题旨在利用过去的销售经验来实现对蔬菜类商品的自动定价与补货，可将模型分为时间序列预测与决策生成两大部分。因此，本题需要完成三方面任务——其一，对蔬菜各品类以及单品的销售分布进行统计分析，并挖掘潜在的关联规则；其二，利用历史数据进行时间序列建模，以实现对未来某一时间段内蔬菜品类或单品的库存需求进行准确预测；其三，对成本、销量与定价之间的关联性进行深度分析，并据此构建最优定价模型以实现收益最大化。

从数据特性来看，本题给出的商品数据表现为高度结构化，索引清晰，标签明确的特征。数据中不存在缺失值、但存在少量的异常值。其中，销售额数据数据量较大，因此需要进行时间序列转换。同时，为了便于分析，需要将某些特征做拼接处理，因此需要对数据做出一定的预处理。

从模型的选择来看，销售明细的流水量样本接近百万级，且与实际情景息息相关。考虑到机器学习在现实应用场景中的普遍性和可行性，本题选用长短期记忆循环神经网络模型实现补货决策。而定价问题一般需要较高的解释性，不宜使用过于复杂的模型，因此采取基于规则的模型来进行定价。

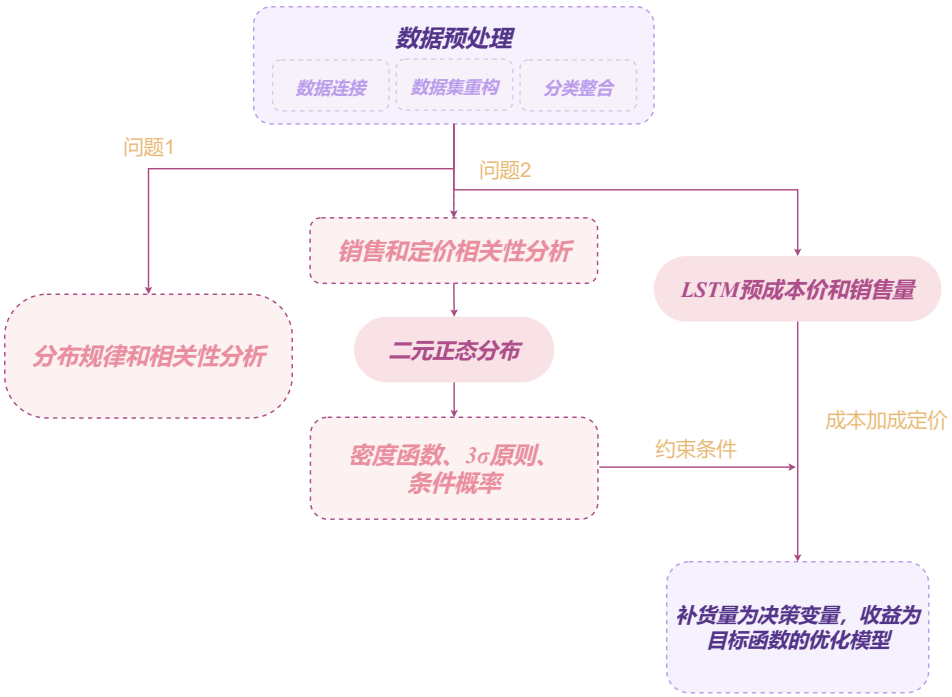

图 **1**　总体分析流程

2.2 问题一的分析

问题一的核心问题是将商品销量的分布进行可视化分析，并对各蔬菜品类及单品间的销售量进行相关性分析。我们首先需要进行预处理，将数据集进行对应连接和分类整合，再对其进行可视化操作。对于蔬菜品类或单品间的销量相关性，由于数据量较为庞

大，我们选择使用 Pearson 相关 [3] 来衡量两蔬菜品类或单品间可能存在的相关关系。

2.3 问题二的分析

问题二的核心问题是构建以最终受益为目标函数，以日补货量为决策变量的优化模型，对日补货总量以及相应的定价策略进行求解。在进行优化模型的构建之前，我们首先需要对个蔬菜品类的销售总量与成本加成定价进行相关性分析，并基于这种相关关系得到这两个变量间的约束关系作为优化模型的一个约束条件。之后，我们需要利用 LSTM 神经网络模型对销售总量和批发价格进行预测，最终写出以商超收益为目标变量，日补货量、日实际销量、商品定价和商品批发价为自变量的目标函数，并对其进行求解，得到商超在要求日期内的日补货量与定价决策。

2.4 问题三的分析

问题三的核心在于补货单品种类的选择、补货量决策和定价策略。由于问题三只需要对未来一天的补货情况做出规划，因此我们可以在问题二的基础上对数据集进行一些简化假设，如进行预测时仅考虑近期一段时间的销售情况，这样有利于提高单日预测的精准度。在做出预测后我们按照分类，构建出 6 个品类下各蔬菜单品的各项指标的矩阵，我们使用这些矩阵中的数据来重新构建问题二中的优化模型，由于问题二涉及到的时间窗口极短，因此在问题三的优化模型中取消了正态分布的约束，并依据不同的要求更新了约束体系。在得到各蔬菜单品补货与定价的优化结果后，我们依照题目中“最小陈列量 2.5 千克”和“可售单品总数控制在 27-33 个”的需求，对蔬菜单品补货种类进行筛选，最终得到 7 月 1 日各单品补货量和定价策略。

2.5 问题四的分析

问题四的核心问题在于探讨进行哪些更多元的数据采集来更精确地制定商超蔬菜商品的补货与定价决策。除了传统的销售数据，商家还需要考虑多维度的其他数据，如气象条件、节假日因素、环境温度以及宏观经济稳定性指标（通货膨胀率）。我们深入探讨了上述四个外部因素对于生鲜蔬菜的供需关系产生的直接或间接的影响。

三、 模型假设

1. 假设商品的只存在售出和损耗两种状态，不考虑其他可能性；
2. 假设当天未售出的商品也视为损耗，不计入第二天的数据计算；
3. 假设不同供应来源的商品具有同一属性，即同名商品可以进行数据合并。
4. 假设蔬菜的销售情况在短期内不会发生剧烈改变

四、 符号说明

符号	说明	单位
$\boldsymbol{M}$	蔬菜品类日实际销售量矩阵	千克
M_i	蔬菜品类某日实际销售量	千克
$\bar{M}$	蔬菜品类日预测销售量矩阵	千克
$\boldsymbol{\bar{M}_i}$	蔬菜品类某日预测销售量	千克
$\boldsymbol{X}$	蔬菜品类成本加成定价矩阵	元/千克
X_i	蔬菜品类成本加成定价	元/千克
$\boldsymbol{C}$	蔬菜品类批发价格矩阵	元/千克
C_i	蔬菜品类批发价格	元/千克
$\boldsymbol{Y}$	蔬菜品类补货量矩阵	千克
Y_i	蔬菜品类补货量	千克
$\boldsymbol{A}$	蔬菜品类损耗率矩阵	
A_i	蔬菜品类损耗率	
i	销售日期	
z	销售记录总天数	天
r	相关系数	
w	滑动窗口大小	
$\sum$	协方差矩阵	
μ	均值	
σ	标准差	
W	收益	元
α	花叶类蔬菜	
β	花采类蔬菜	
γ	辣椒类蔬菜	
δ	茄类蔬菜	
ζ	食用菌蔬菜	
ϵ	水声根茎类蔬菜	

五、 数据预处理

数据连接：将附件 1 中蔬菜品类的商品信息与附件 2 中的流水数据进行数据连接，按照“单品编码”连接，得到一个含有一个详细商品信息的销售流水记录。

数据集重构：将连接好的数据集进行进一步处理，我们将销售类型为“退货”所对

应的销售记录视为无效数据，即删除掉“退货”和与退货对应的“销售”两条数据同时删除，之后按照商品的不同类别，按日划分，最终得到一个包含商品详细信息的日销售总量数据集。若出现无法匹配到“销售”数据的“退货”数据，也将其视为无效数据，将其删除。

分类整合：将上述处理得到的日销售量数据集按照 6 个不同的蔬菜种类进行分类，得到每个蔬菜种类的日销售量数据集。

六、 问题一模型的建立与求解

6.1 各蔬菜品类及单品销售量的可视化

为了分析蔬菜各品类及单品销售量的分布规律和相互关系，我们首先对预处理后的数据进行可视分析，企图寻找销量的季节性、周期性等规律。

蔬菜各品类的时间序列分析：我们使用预处理过后的数据绘制了花菜类、花叶类、辣椒类、茄类、食用菌类和水生根茎类 6 个蔬菜品类在时间上分布的散点图。结果如图 2 所示。观察图像我们得到，花菜类蔬菜的销量并不存在明显的分布规律，茄类和水生

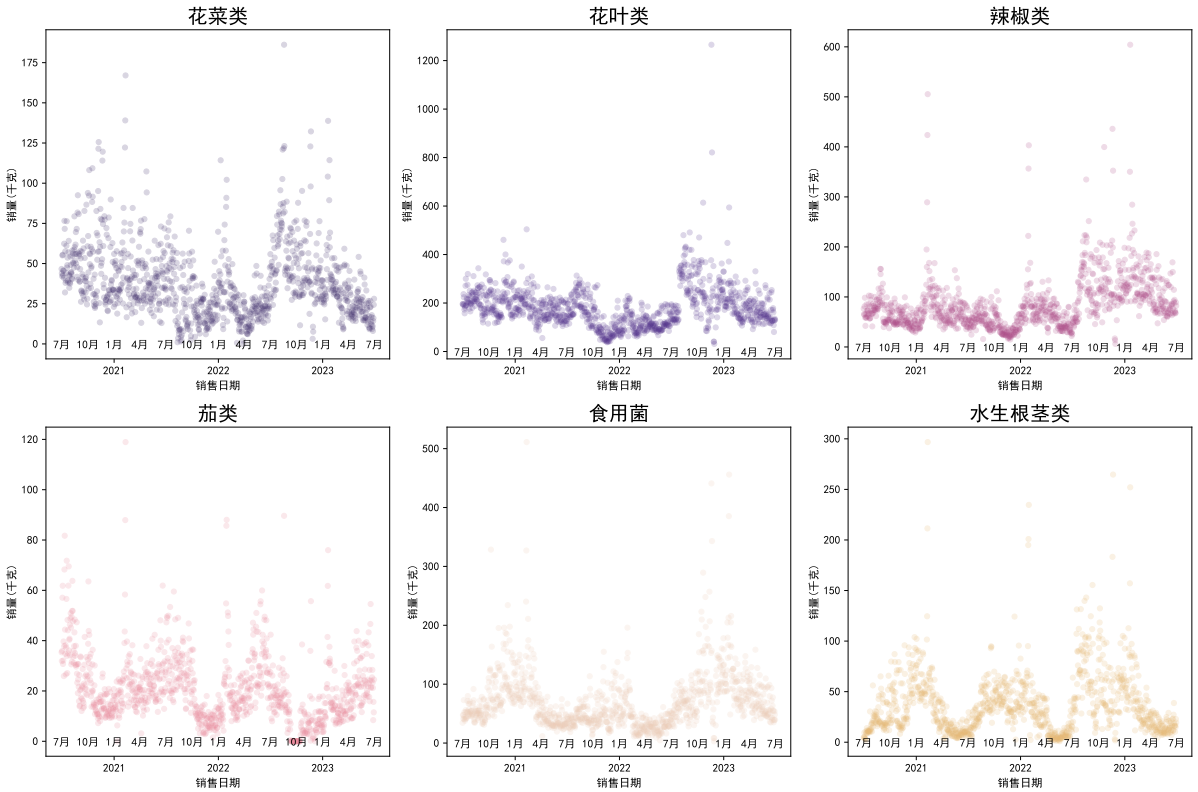

图 **2**　蔬菜各品类的日销量分布

根茎类蔬菜的销量表现出明显的季节性，花叶类、辣椒类和食用菌类蔬菜的销量分布在 2022 年 7 月之前表现出较明显的季节性规律，但 2022 年 7 月之后的数据发生明显断层，

故我们重新审视数据，发现以上三类蔬菜在 2022 年 7 月之后出现了按“份”或“袋”等为单位售卖的情况，因此与仅按照千克售卖的数据产生了偏差。

为解决上述问题，我们将“份”、“小份”、“包”等封装销售的商品数据全部进行替换，每一份封装商品均替换为 0.25 千克。

讨论我们近似估计的合理性，有如下两个方面：

- 我们使用往年同时期的数据，将售卖价格与往年对应商品单价进行结合，得到一个较为合理的估计值。
- 我们通过市场大数据分析修正我们的近似估计值。

将数据进行修改后，我们再次进行可视化操作，得到结果如图 3 所示。观察图 3，我

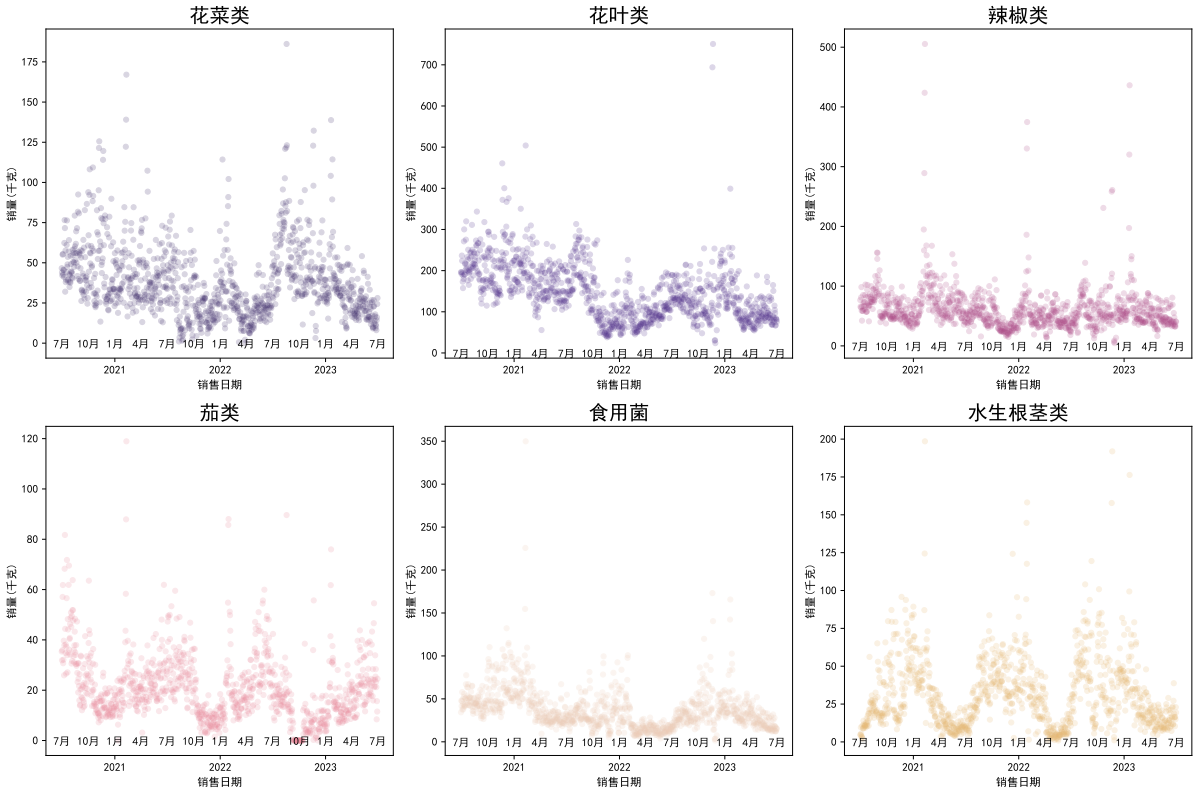

图 **3**　修正后的蔬菜各品类的日销量分布

们发现花叶类、辣椒类和食用菌分布中的断层现象已经消失，且后续的变化趋势符合图像的整体趋势，由此我们可以进一步我们确认的单位替换是合理且成功的。

在得到各蔬菜品类的分布规律后，我们在每一类中选取一个蔬菜单品的销售情况进行可视化，得到的结果如图 4 所示。对比观察图 2 图 3，我们发现图 3 中蔬菜单品的销售量分布规律与图 2 中各蔬菜品类的销售量分布规律吻合的很好，进一步印证了进行数据修正的必要性和正确性。

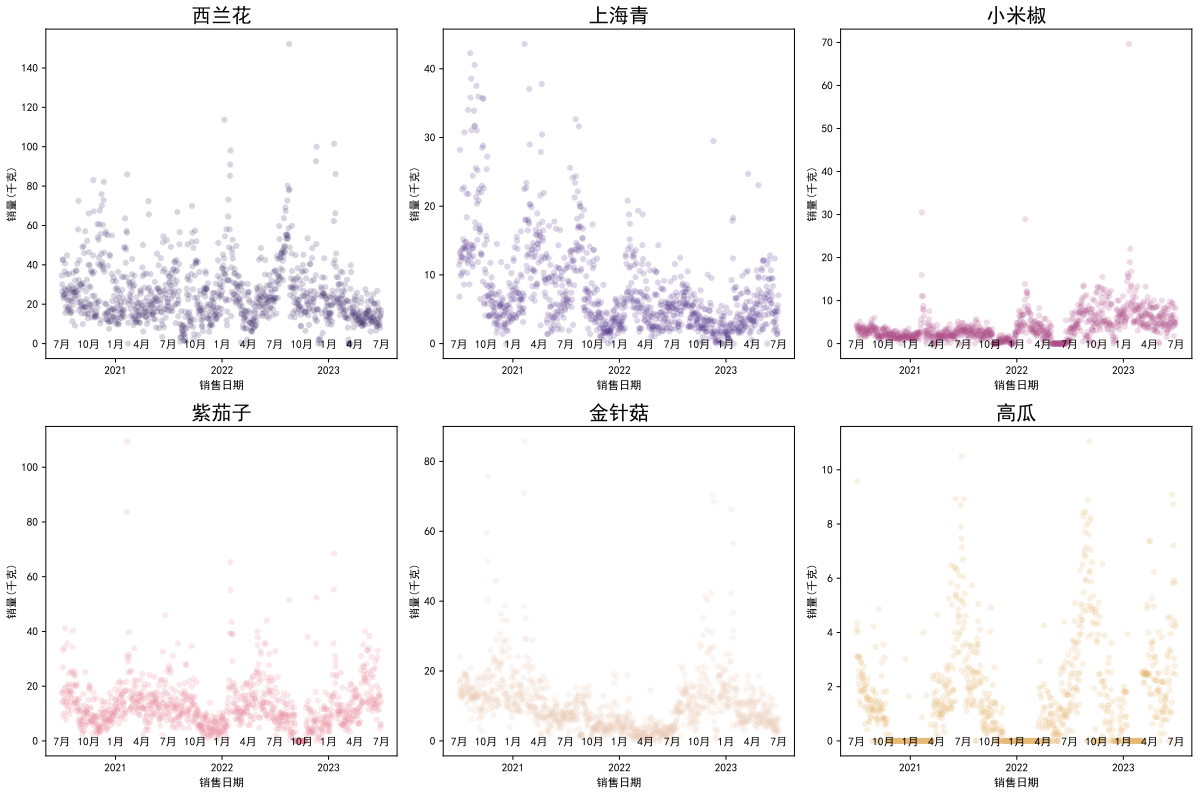

图 **4**　各品类中某蔬菜单品的日销量分布

6.2 各蔬菜品类及单品销售量间的相互关系

此题我们采用 Pearson 相关系数分析来分析各蔬菜品类及单品销售量间的相互关系。现有 6 个蔬菜品类，可以构成数据矩阵 $M = (m_{ij})$，现在要研究矩阵中第 a 列 x_a 和第 b 列 x_b 的相关性，设 a,b 列的相关系数为 $r(a,b)$，其中 m_a 和 m_b 代表某蔬菜品类的日销售量，$\bar{m_a}$ 和 $\bar{m_b}$ 分别是 a 列和 b 列的均值。

$$r(a,b) = \frac{\sum_{i=1}^{z}(m_{a,i} - \bar{m_a})(x_{b,i} - \bar{x_b})}{\sum_{i=1}^{z}(m_{a,i} - \bar{m_a})^2 \sum_{j=1}^{z}(m_{b,i} - \bar{m_b})^2} \tag{1}$$

相关系数的值的范围是从 -1 到 $+1$. 值 -1 表示完全负相关，值 $+1$ 表示完全正相关，值 0 表示列之间没有相关性。

相关系数的求解过程如下。

- 计算均值：

$$\bar{m_a} = \frac{\sum_{i=1}^{z} m_{a,i}}{z} \tag{2}$$

$$\bar{m_b} = \frac{\sum_{i=1}^{z} m_{b,i}}{z} \tag{3}$$

- 代入上文 $r(a,b)$ 的表达式计算相关系数。

据此画出的各蔬菜品类销量间的热力图如图 5 所示。

各蔬菜单品销量与时间的相关系数如表 1 所示。

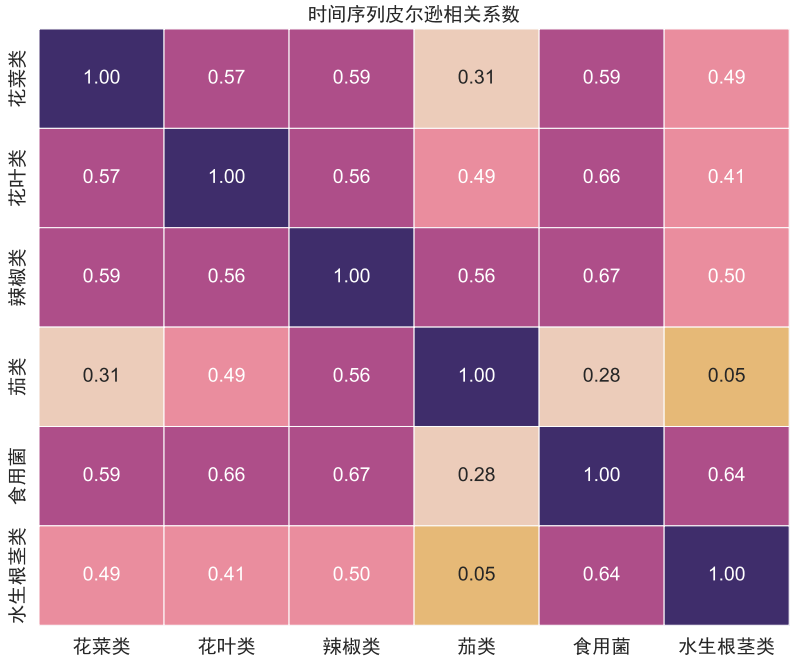

图 **5**　各蔬菜品类销量间的热力图

观察图 5 和表 1 我们可以得到以下结论：

- 大部分蔬菜品类销量间的相关系数都在 0.5 以上，这代表这些蔬菜品类销售的旺季有着较大的重叠。
- 茄类和水声根茎类的相关性仅有 0.05，经查询资料发现两者的成熟期十分接近，因此造成两者基本不相关的原因可能是营养价值带来的差异，即由于两类蔬菜所含营养成分的差别较大，消费者购买时不存在相互替代的情况。
- 大部分蔬菜单品与时间序列成弱相关性，这代表这些蔬菜的销售受季节影响较小，是常年蔬菜，储存和运输相对容易。
- 少数和时间序列成强相关的蔬菜单品（如小青菜、大白菜等），这些蔬菜都属于时令蔬菜，生产、储存和运输受季节的影响较大。

七、 问题二模型的建立与求解

7.1 各蔬菜品类的销售总量与成本加成定价的相关性分析

我们首先对各蔬菜品类的销售量和成本加成定价进行滑动窗口同步性分析 [4]，这是一种用于分析两个或多个时间序列数据之间动态相关性的方法，这种方法能够展示时间序列在不同时间窗口下的局部相关性。

现在有两个时间序列 $X = \{x_1, x_2, ..., x_N\}$ 和 $Y = \{y_1, y_2, ..., y_N\}$，其中 N 是时间序

表 2　各蔬菜单品销量与时间的 **Pearson** 相关系数

七彩椒	上海青	东门口小白菜	丝瓜尖	云南油麦菜	云南生菜	余干椒	保康高山大白菜	冰草	净藕
0.24	-0.46	-0.28	-0.03	-0.46	-0.39	-0.04	0.38	0.01	-0.11
南瓜尖	双孢菇	双沟白菜	和丰阳光海鲜菇 (包)	四川红香椿	圆茄子	外地茼蒿	大白菜	大白菜秧	大芥兰
-0.05	0.36	-0.01	0.07	0.02	-0.34	-0.02	-0.55	0.07	-0.02
大龙茄子	奶白菜	奶白菜苗	姜蒜小米椒组合装	姬菇 (包)	姬菇	娃娃菜	小白菜	小皱皮	小米椒
-0.46	0.33	-0.03	0.64	-0.55	-0.28	0.16	-0.44	0.62	0.4
小青菜	平菇	快菜	春菜	木耳菜	本地上海青	本地小毛白菜	本地黄心油菜	杏鲍菇 (250 克)	杏鲍菇
0.67	-0.45	-0.15	0.04	0.11	-0.15	0.15	0.29	-0.03	-0.36
枝江红菜苔	枝江青梗散花	槐花	水果辣椒	油菜苔	泡泡椒	洪山菜苔	珍品手提袋	莲藕拼装礼盒	洪湖莲藕 (粉藕)
0.02	0.58	0.07	0.14	-0.1	-0.63	0.04	0.04	0.06	0.16
洪湖莲藕 (脆藕)	洪湖藕带	活体银耳	海鲜菇 (包)	海鲜菇	灯笼椒	牛排菇	牛首油菜	牛首生菜	猪肚菇
0.06	0.26	-0.03	0.34	-0.24	-0.18	-0.0	-0.38	-0.08	0.0
猴头菇	甘蓝叶	甜白菜	田七	白玉菇	白菜苔	白蒿	秀珍菇	竹叶菜	紫圆茄
-0.03	-0.03	-0.41	-0.18	-0.38	-0.12	0.15	-0.12	-0.02	-0.01
紫尖椒	紫白菜	紫苏	紫茄子	紫螺丝椒	紫贝菜	红尖椒	红杭椒	红椒	红橡叶
-0.0	-0.09	0.01	-0.03	0.08	-0.12	0.15	-0.37	-0.05	0.01
红灯笼椒	红珊瑚	红线椒	红莲藕带	红薯尖	组合椒系列	绣球菌	绿牛油	艾蒿	芜湖青椒
-0.07	0.01	-0.0	0.29	-0.26	-0.26	-0.09	0.01	0.09	0.22
芝麻苋菜	芥兰	芥菜	花茄子	花菇 (一人份)	苋菜	茶树菇	茼蒿	荠菜	荸荠
0.01	-0.02	-0.04	-0.1	0.07	0.02	0.21	-0.13	-0.1	-0.11
莲蓬 (个)	菊花油菜	菌菇火锅套餐	菌蔬四宝	菜心	菠菜	菱角	萝卜叶	蒲公英	蔡甸藜蒿
0.13	-0.03	0.07	0.06	-0.53	-0.04	0.17	-0.18	0.07	-0.09
薄荷叶	藕尖	虫草花	螺丝椒	蟹味菇	蟹味菇白玉菇	襄甜红菜苔	西兰花	西峡花菇	西峡香菇
0.08	-0.0	0.03	0.02	-0.16	0.43	0.01	-0.12	0.05	0.19
西峡香菇	豌豆尖	赤松茸	辣妹子	野生粉藕	野藕	金针菇	银耳	长线茄	随州泡泡青
-0.28	-0.04	0.07	-0.04	-0.12	0.44	-0.18	0.25	0.07	0.01
青尖椒	青杭椒	青梗散花	青红尖椒组合装	青红杭椒组合装	青线椒	青茄子	青菜苔	面条菜	马兰头
-0.51	-0.01	-0.69	0.17	0.46	-0.36	-0.33	-0.05	0.09	0.02
马齿苋	高瓜	鱼腥草	鲜木耳	鲜粽叶	鲜粽子叶	鲜藕带	鸡枞菌	鹿茸菇	黄心菜
0.04	0.15	0.28	-0.08	0.14	-0.05	-0.05	-0.07	0.05	-0.1
黄白菜	黄花菜	黑油菜	黑牛肝菌	黑皮鸡枞菌	龙牙菜				
-0.37	0.09	0.09	0.04	-0.12	0.2				

列的长度。我们首先选择窗口大小 w，它是一个重要的参数，决定了局部相关性的粒度。接下来根据窗口大小，对于每一个时间 t，选择 X 和 Y 在 $[t-w/2, t+w/2]$ 范围内的子序列。对于每个窗口，计算 X 和 Y 的子序列的 Pearson 相关系数 r_t，相关系数的计算方法上文中已给出。

最后我们得到一个新的时间序列 $R = \{r_1, r_2, ..., r_{N-w+1}\}$，其中每个 r_t 都是一个窗口内的局部 Pearson 相关系数。由图像可知两者呈现强相关，考虑到两组数据都是时间

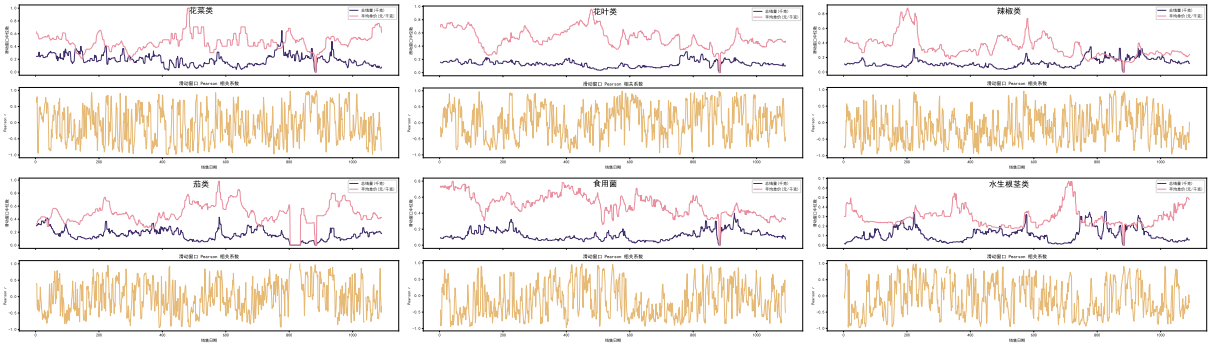

图 **6**　各蔬菜品类的销售量和成本加成定价的局部 **Pearson** 相关系数

序列，我们首先使用 Pearson 相关分别考察销售总量和成本加成定价与时间的关系，部分结果如图 7 所示。观察图像可知，销售总量与成本加成定价都和时间有可能存在正态

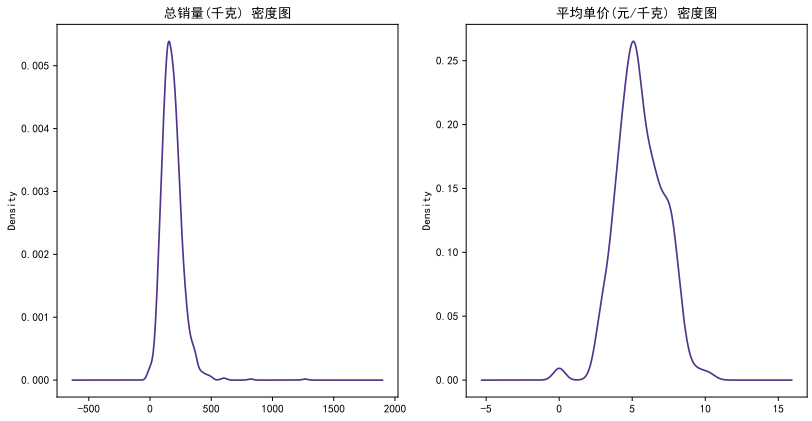

图 **7**　各蔬菜品类的销售量和成本加成定价关于时间的相关

相关性。为进一步确定猜想，我们绘制出了上述两个变量对时间的 Q-Q 图，如图 8 所示。据此，我们判断销售总量和成本加成定价间存在二元正态分布关系 [5]。二元正态分布的联合概率密度函数可由以下公式表示：

$$f(x,y)=\frac{1}{2\pi\sqrt{|\sum|}}\exp(-\frac{1}{2}(X-\mu)^T\sum^{-1}(X-\mu)) \tag{4}$$

其中 X 是一个列向量，包含变量 $\bar{m}$（预测销售量）和 x（成本加成定价），μ 是均值向量，包含 $\mu_{\bar{m}}$（预测总销量的均值）和 μ_x（平均单价的均值），$\sum$ 为协方差矩阵，具体形式为：

$$\sum=\begin{pmatrix}\sigma_{\bar{m}}^2 & r\sigma_{\bar{m}}\sigma_x\\ r\sigma_{\bar{m}}\sigma_x & \sigma_x^2\end{pmatrix} \tag{5}$$

$\sigma_{\bar{m}}$ 和 σ_x 分别是 $\bar{m}$（预测总销量）和 x（成本加成定价）的样本标准差，r 是 $\bar{m}$ 和 x 的样本相关系数，$|\sum|$ 是 $\sum$ 的行列式，$\sum^{-1}$ 是 $\sum$ 的逆矩阵。

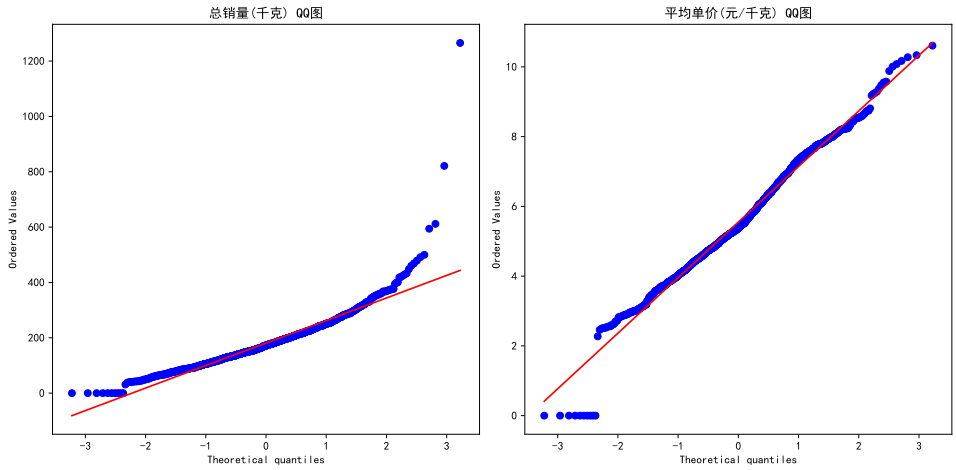

图 **8**　各蔬菜品类的销售量和成本加成定价对时间的 **QQ** 图

通过计算销售总量和成本加成定价的均值和标准差，我们可以得到 $\mu_{\bar{m}}$，μ_x，$\sigma_{\bar{m}}$ 和 σ_x，由此我们可以算出二元正态分布密度函数的表达式，由此得到两个变量间的关系，如图 9 所示。

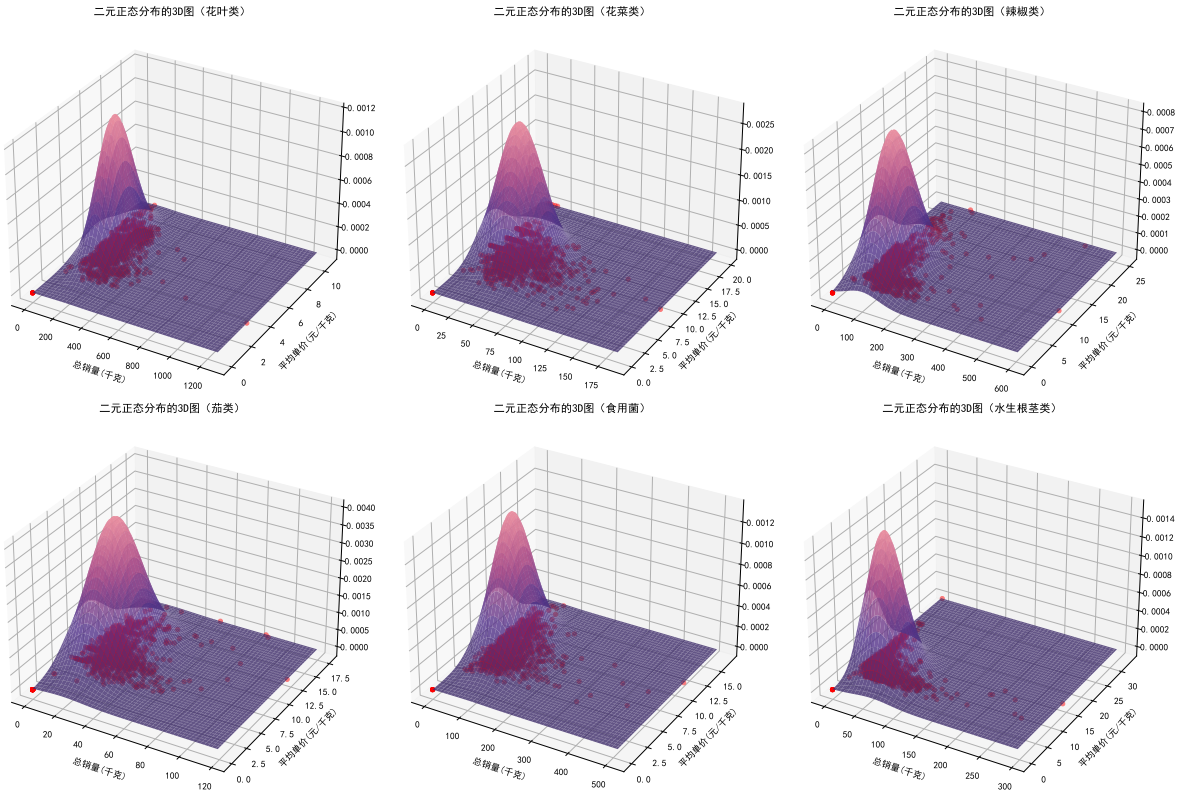

图 **9**　二元正态分布图像

为了找寻到两个变量间的约束关系，我们利用条件概率分布，当我们已知销售总量时，可以计算成本加成定价的条件期望和条件方差。在二元正态分布中，条件期望和条

件方差可以用以下公式表示：

$$E(X|\bar{M}=\bar{m})=\mu_X+\rho\frac{\sigma_X}{\sigma_{\bar{m}}}(\bar{m}-\mu_{\bar{M}}) \tag{6}$$

$$Var(X|\bar{M}=\bar{m})=(1-\rho^2)\sigma_Y^2 \tag{7}$$

其中 E 是期望，Var 是方差，μ 是均值，σ 是标准差，ρ 是两个变量之间的相关系数。在该条件分布下，X 的值应当遵循“3σ 原则”，即

$$E(X|\bar{M}=\bar{m})-3\sqrt{Var(X|\bar{M}=\bar{m})}\le X\le E(X|\bar{M}=\bar{m})+3\sqrt{Var(X|\bar{M}=\bar{m})} \tag{8}$$

因此，我们可以用上式来描述两个变量间的约束关系。与此同时，我们注意到，我们可以通过时序预测来得到预期销售量，但在后续实际的运算过程中，计算销售额时应使用实际销售量来代入计算，于是我们考虑，利用二元正态分布的实际含义，将预期销售量乘上不同定价所对应的概率，以能够完成预测销售量的概率来描述预测销售量和实际销售量之间的关系。对此，我们只需要将二元正态分布的概率密度对定价进行积分，积分结果乘以一侧销售量，即可得到实际销售量，表达式如下：

$$M=\begin{cases}2\bar{m}\int_{-\infty}^{x}f(X|\bar{M}=\bar{m}),\ x<EX\\ 2\bar{m}(1-\int_{x}^{+\infty}f(X|\bar{M}=\bar{m}),\ x>EX)\end{cases} \tag{9}$$

7.2 LSTM 神经网络 [6] 模型预测销售总量和批发价格

由于各蔬菜品类每天的销售总量是一个时间序列，因此我们可以采用 LSTM 神经网络对未来一段时间内的销售总量情况进行预测。LSTM 神经网络的数学原理如下。

我们首先构建 LSTM 神经网络模型的结构，包括输入层、隐藏层（LSTM 层）和输出层。其中 LSTM 层是 LSTM 模型的核心组成部分，它通过一种特殊的单元结构来解决传统循环神经网络（RNN）中的长期依赖问题。LSTM 层包含一系列 LSTM 单元，每个单元包含三个重要的门：输入门、遗忘门和输出门，以及一个内部的单元状态。

假设输入为 x_t，上一个单元的输出（隐藏状态）为 h_{t_1}，上一个单元状态为 c_{t-1}，下面是 LSTM 单元内部的主要数学运算：

- **遗忘门**：决定哪些信息从单元状态中丢弃。

$$f_t=\sigma(W_f\cdot[h_{t-1},x_t]+b_f) \tag{10}$$

- **输入门**：决定更新哪些新信息到单元状态。

$$i_t=\sigma(W_i\cdot[h_{t-1},x_t]+b_i) \tag{11}$$

- **候选单元状态**：创建一个候选状态来加入到实际单元状态中。

$$\tilde{c}_t=\tanh(W_C\cdot[h_{t-1},x_t]+b_C) \tag{12}$$

- **更新单元状态**：结合遗忘门和输入门的信息，更新单元状态。

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{13}$$

- **输出门**：决定输出哪些单元状态信息。

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{14}$$

- **隐藏状态（单元输出）**：

$$h_t = o_t * \tanh(c_t) \tag{15}$$

其中，σ 是 sigmoid 函数，$[h_{t-1}, x_t]$ 是是前一时刻的隐藏状态和当前输入的拼接，$*$ 表示逐元素乘法，W 和 b 是需要学习的权重和偏置。

接下来我们对 LSTM 神经网络模型进行求解，这包括向前传播和反向传播。

在向前传播过程中，我们首先初始化单元状态 c_0 和隐藏状态 h_0，通常这两者都初始化为零向量。然后，对于输入序列的每一个时间步 t，我们进行以下操作：

- 使用前一个时间步 $t-1$ 的隐藏状态 h_{t-1} 和当前输入 x_t，计算遗忘门 f_t、输入门 i_t、输出门 o_t 和候选单元状态 $\tilde{c}_t$。
- 使用这些门的输出和前一个单元状态 c_{t-1}，更新单元状态 c_t。
- 使用新的单元状态 c_t 和输出门 o_t，计算新的隐藏状态 h_t。

反向传播用于计算损失函数 J 对网络参数的梯度。具体来说，目标是找出如何改变参数以减少输出 h_t 和实际目标 y_t 之间的差距。

- **梯度初始化**：首先，对每一个时间步，我们计算损失函数 J 和对隐藏状态 h_t 的梯度 $\frac{\partial J}{\partial h_t}$。损失函数 J 的计算公式如下：

$$J = -\frac{1}{T}\sum_{t=1}^{T}[y_t log(\hat{y}_t) + (1-y_t) log(1-\hat{y}_t)] \tag{16}$$

- **计算门和状态的梯度**：对于每一个时间步 t，我们根据 $\frac{\partial J}{\partial h_t}$ 和当前单元状态 c_t 来计算损失函数 J 对输入门、遗忘门、输出门以及单元状态的梯度。例如，输出门 o_t 的梯度 $\frac{\partial J}{\partial o_t}$ 可通过链式法则来计算：

$$\frac{\partial J}{\partial o_t} = \frac{\partial J}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t} \tag{17}$$

- **参数梯度**：接着，我们使用这些局部梯度来计算损失函数 J 对网络参数 W_f、W_i、W_o、b_f、b_i、b_o 的梯度。
- **权重更新**：最后，我们使用随机梯度下降法来更新所有网络参数。

这个过程会迭代进行，通常会遍历多个周期（epochs）以最小化损失函数。其中随机梯度下降法的原理如下：

表 **3**　**7** 月 **1** 日——**7** 日各蔬菜品类的预测销售总量

	7 月 1 日	7 月 2 日	7 月 3 日	7 月 4 日	7 月 5 日	7 月 6 日	7 月 7 日
花菜类	15.488	14.133	13.530	13.325	13.267	13.251	13.246
花叶类	184.585	218.549	258.691	245.258	255.268	252.600	254.876
辣椒类	96.619	120.776	116.431	124.908	116.180	125.340	116.745
茄类	18.428	16.722	15.923	15.283	14.368	13.014	11.418
食用菌	46.523	44.875	43.162	41.352	39.452	37.555	35.844
水生根茎类	18.350	16.975	15.325	13.682	12.371	11.507	11.003

首先，随机初始化模型的权重和偏置，将训练数据集打乱进行数据准备，之后进行迭代更新，对于训练集中的每一个样本 (x_i, y_i)，计算模型对当前样本的预测 $\hat{y}_i$，计算损失 J_i，求出损失函数 J_i 对每个参数 θ 的梯度 $\frac{\partial J_i}{\partial \theta}$，最终更新参数：$\theta = \theta - \alpha \frac{\partial J_i}{\partial \theta}$。其中，$\alpha$ 是学习率，用于控制每一步更新的大小。

最终得到各蔬菜品类 7 月 1 日——7 日的预测销售量如表 2 和图 10 所示。

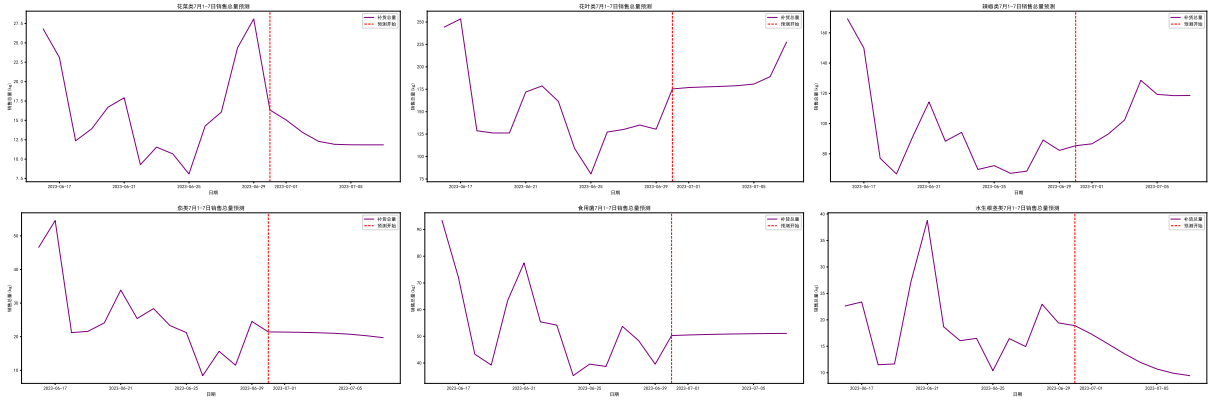

图 **10**　各蔬菜品类的预测销售总量

各蔬菜品类 7 月 1 日——7 日的预测批发价格如表 3 和图 11 所示。

7.3 补货量与定价决策

对于一家超市来说，制定合适的补货量与定价决策，其目标为销售利益最大化，同时决策策略应具有较好的稳定性，能够应对多变的市场变化情况。

结合题目已知条件和假设，该规划模型需要满足以下约束和假设：

- 假设我们上述使用的 LSTM 预测模型的误差不超过 10%。
- 预期补货量减去损耗量大于等于预测销售量。
- 预期补货量和预测批发价均大于零，预期定价大于等于预测批发价。

表 **4**　**7** 月 **1** 日——**7** 日各蔬菜品类的预测批发价格

	7 月 1 日	7 月 2 日	7 月 3 日	7 月 4 日	7 月 5 日	7 月 6 日	7 月 7 日
花菜类	8.182	8.104	8.030	7.960	7.893	7.829	7.767
花叶类	3.936	4.052	4.163	4.270	4.372	4.469	4.651
辣椒类	6.386	6.500	6.608	6.711	6.807	6.897	6.979
茄类	4.674	4.710	4.743	4.772	4.800	4.824	4.847
食用菌	4.804	4.840	4.872	4.902	4.930	4.956	4.980
水生根茎类	11.862	11.964	12.082	12.221	12.386	12.583	12.820

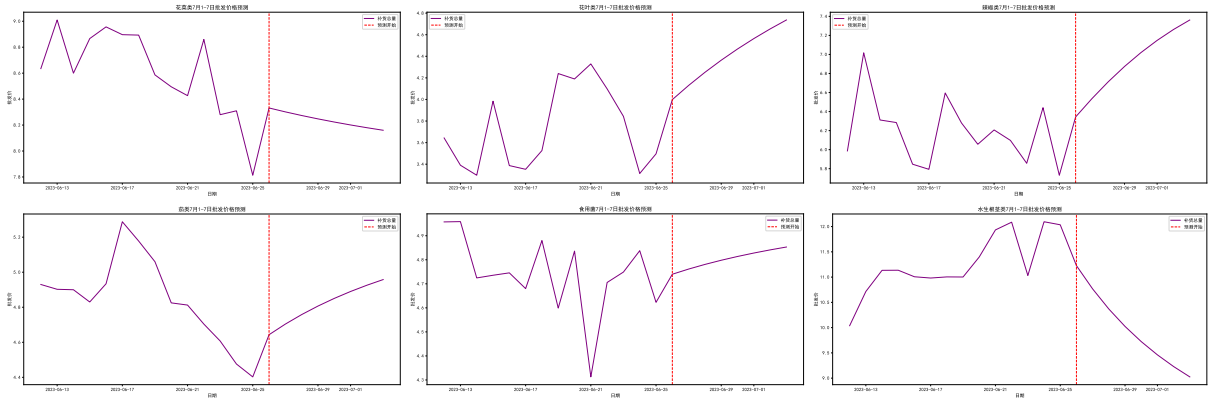

图 **11**　各蔬菜品类的预测批发价格

- 在确定预测销售量条件分布下，预期定价满足“3σ 原则”。
- 实际销售量和预测销售量满足条件概率分布。

则最终的目标函数为：

$$max(\boldsymbol{W}) = max(\boldsymbol{X^T} \cdot \boldsymbol{M} - C^T \cdot \boldsymbol{Y}) \tag{18}$$

需要满足的约束条件为：

$$\boldsymbol{Y} * (1 - \boldsymbol{A}) \geq \boldsymbol{M} \tag{19}$$

$$1.1\boldsymbol{M} \geq Y \geq 0.9\boldsymbol{M} \tag{20}$$

$$C_i, Y_i \geq 0 \tag{21}$$

$$X_i \geq C_i \tag{22}$$

$$E(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}) - 3\sqrt{Var(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}} \leq X \leq E(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}) + 3\sqrt{Var(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}})} \tag{23}$$

$$M = \begin{cases} 2\bar{m} \int_{-\infty}^{x} f(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}),\ x < E(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}) \\ 2\bar{m}(1 - \int_{x}^{+\infty} f(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}),\ x > E(\boldsymbol{X}|\bar{\boldsymbol{M}} = \bar{\boldsymbol{m}}) \end{cases} \tag{24}$$

表 **5**　日补货总量、定价决策和预期收益

日期	加成成本定价 X	补货量 Y	预期最大收益 W
7 月 1 日	[5.998, 9.880, 8.778, 8.438, 8.736, 11.862]	[183.86, 13.94, 86.96, 16.59, 41.87, 16.52]	633.68
7 月 2 日	[6.019, 9.913, 8.721, 8.425, 8.753, 11.964]	[196.69, 12.72, 108.70, 15.05, 40.39, 15.28]	633.37
7 月 3 日	[5.893, 9.927, 8.671, 8.419, 8.771, 12.082]	[232.82, 12.18, 104.79, 14.33, 38.85, 13.79]	608.39
7 月 4 日	[5.928, 9.932, 8.648, 8.414, 8.790, 12.221]	[220.73, 11.99, 112.42, 13.75, 37.22, 12.31]	564.72
7 月 5 日	[5.389, 9.934, 9.218, 8.789, 9.083, 12.386]	[280.79, 11.94, 104.56, 12.93, 35.51, 11.13]	456.30
7 月 6 日	[5.890, 9.935, 8.654, 8.397, 8.829, 12.583]	[227.34, 11.93, 112.81, 11.71, 33.80, 10.36]	479.05
7 月 7 日	[5.885, 9.934, 8.665, 8.386, 8.847, 12.820]	[229.39, 11.92, 105.07, 10.28, 32.26, 9.90]	412.66

其中 C_i, Y_i, X_i 分别为蔬菜的批发价格、补货量和成本加成定价。

最终解得的日补货总量、定价决策和预期收益如表 4 所示。向量中蔬菜品类的排列顺序均为花叶类、花菜类、辣椒类、茄类、食用菌和水声根茎类。

八、 问题三模型的建立与求解

8.1 补货量与定价决策优化模型 [7] 的构建

由问题三题意可知，我们要根据过去一个星期蔬菜商品的销售情况来做出单品补货与定价的决策，我们假设蔬菜的销售情况在短期内不会发生剧烈改变，因此区别于问题二中的 LSTM 时间序列预测，我们使用 6 月 24 日到 6 月 30 日数据的均值作为 7 月 1 日对应指标的预测值，由此在不影响精确度的情况下大大简化了运算过程。

首先我们需要对数据进行筛选，得到 2023 年 6 月 24-30 日的可售蔬菜单品总数为 49 个，并按类构建出 6 个品类下各蔬菜单品各项指标的矩阵，并作出如下定义：α 代表花叶类蔬菜，β 代表花菜类蔬菜，γ 代表辣椒类蔬菜，δ 代表茄类蔬菜，ζ 代表食用菌蔬菜，ϵ 代表水生根茎类蔬菜。据此我们将本问题中的成本加成定价 $\boldsymbol{X}$，损耗率 $\boldsymbol{A}$，成本价 $\boldsymbol{C}$ 和预测销量 $\boldsymbol{M}$ 做如下的拼接:

$$\boldsymbol{X} = [\boldsymbol{X_\alpha} : \boldsymbol{X_\beta} : \boldsymbol{X_\gamma} : \boldsymbol{X_\delta} : \boldsymbol{X_\zeta} : \boldsymbol{X_\epsilon}] \tag{25}$$

$$\boldsymbol{A} = [\boldsymbol{A_\alpha} : \boldsymbol{A_\beta} : \boldsymbol{A_\gamma} : \boldsymbol{A_\delta} : \boldsymbol{A_\zeta} : \boldsymbol{A_\epsilon}] \tag{26}$$

$$\boldsymbol{C} = [\boldsymbol{C_\alpha} : \boldsymbol{C_\beta} : \boldsymbol{C_\gamma} : \boldsymbol{C_\delta} : \boldsymbol{C_\zeta} : \boldsymbol{C_\epsilon}] \tag{27}$$

$$\boldsymbol{M} = [\boldsymbol{M_\alpha} : \boldsymbol{M_\beta} : \boldsymbol{M_\gamma} : \boldsymbol{M_\delta} : \boldsymbol{M_\zeta} : \boldsymbol{M_\epsilon}] \tag{28}$$

接着，我们根据问题二中已经构建的最优化模型，在有限约束下计算利润的最大值。在问题三中，利润具有如下约束：

表 **6**　各个单品在 **7** 月 **1** 日的补货量

菠菜	菠菜 (份)	菜心	红薯尖	木耳菜	云南生菜	云南生菜 (份)
2.79	10.82	2.572	4.921	6.422	3.446	35.647
长线茄	青茄子 (1)	双孢菇 (盒)	海鲜菇 (包)	白玉菇 (袋)	红莲藕带	菱角
4.527	3.286	10.02	8.857	1.07	0.788	2.254
枝江青梗散花	西兰花	七彩椒 (2)	姜蒜小米椒组合装 (小份)	小皱皮 (份)	圆茄子 (2)	紫茄子 (1)
6.957	13.839	1.344	7.729	13.335	1.33	0.565
木耳菜 (份)	奶白菜	上海青	娃娃菜	外地茼蒿	苋菜	小青菜 (1)
32.934	8.887	4.512	10.694	2.951	10.951	5.465
虫草花 (份)	蟹味菇与白玉菇双拼 (盒)	西峡花菇 (1)	金针菇 (盒)	鲜木耳 (份)	净藕 (1)	洪湖藕带
2.524	2.017	5.184	16.216	4.416	6.38	5.31
小米椒 (份)	红椒 (2)	芜湖青椒 (1)	螺丝椒	螺丝椒 (份)	青红杭椒组合装 (份)	青线椒 (份)
23.66	2.248	15.094	7.618	12.461	2.366	1.104
紫茄子 (2)	野生粉藕	云南油麦菜	竹叶菜	高瓜 (1)	云南油麦菜 (份)	高瓜 (2)
11.607	0.931	1.949	15.393	4.247	23.502	1.324

- 预期补货量减去损耗量大于等于预测销售量。
- 每个类别的补货总量不应低于问题二中决策出的补货量。

目标函数：

$$max(\boldsymbol{W}) = max(\boldsymbol{X^T} \cdot \boldsymbol{M} - C^T \cdot \boldsymbol{Y}) \tag{29}$$

约束条件：

$$Y \times (1 - A) \geq M \tag{30}$$

$$\sum Y_\alpha \geq 183.86 \tag{31}$$

$$\sum Y_\beta \geq 13.94 \tag{32}$$

$$\sum Y_\gamma \geq 86.96 \tag{33}$$

$$\sum Y_\delta \geq 16.59 \tag{34}$$

$$\sum Y_\zeta \geq 41.87 \tag{35}$$

$$\sum Y_\epsilon \geq 16.51 \tag{36}$$

利用 Python 的优化工具箱求解上述线性规划问题，得到的 7 月 1 日各个单品的补货量结果如表 6 所示。在此条件下，我们计算可能获得的总利润 W=683.632, 与问题二中 7 月 1 日的总获利偏差不大，说明了我们决策的合理性。

表 7　选择后的 **33** 个单品补货量

螺丝椒 (份)	菠菜 (份)	红薯尖	木耳菜	云南生菜	紫茄子 (2)
12.461	10.82	4.921	6.422	3.446	11.607
长线茄	青茄子 (1)	双孢菇 (盒)	海鲜菇 (包)	苋菜	竹叶菜
4.527	3.286	10.02	8.857	10.951	15.393
枝江青梗散花	西兰花	姜蒜小米椒组合装 (小份)	小皱皮 (份)	小青菜 (1)	高瓜 (1)
6.957	13.839	7.729	13.335	5.465	4.247
木耳菜 (份)	奶白菜	上海青	娃娃菜	外地茼蒿	云南油麦菜 (份)
32.934	8.887	4.512	10.694	2.951	23.502
西峡花菇 (1)	金针菇 (盒)	鲜木耳 (份)	净藕 (1)	洪湖藕带	云南生菜 (份)
5.184	16.216	4.416	6.38	5.31	35.647
小米椒 (份)	芜湖青椒 (1)	螺丝椒			
23.66	15.094	7.618			

8.2 单品种类选择

由于仓库原因和场地限制，蔬菜类商品的销售空间有限 [8]，需要满足各单品订购量最小陈列量为 2.5 千克的要求。因此我们需要对 8.1 中的结果进行选择。

我们首先按照补货量对所有单品进行排序，然后从高到低依次选择出补货量满足 2.5kg 的单品，直到单品数目超过 33 个为止。最终，我们选择出的单品如表 7 所示。

值得一提的是，我们发现，在去除补货量小于 2.5kg 的单品后，预计获利有所上升。此时，总获利 W=864.2。这是因为进货量较少的单品，其市场需求量较低，且损耗率较高，导致了这些单品处于负盈利状态，减少此类单品的补货有助于提高利润。

九、 问题四模型的建立与求解

为了更精准地进行蔬菜商品的补货与定价决策，存在几个关键的外部变量需被详细考量。**这部分旨在深入探讨四个外部因素——天气状况、节假日、环境温度以及宏观经济指标（通货膨胀率）——对于生鲜蔬菜进货量的影响。**通过增加这些影响因子，可以从更多维度指导商家在不同情境下做出更为合理的进货决策。

9.1 气象条件

在补货决策中，气象条件 [9] 是一个复杂而具有多重影响的因素。首先，气象因素直接影响蔬菜的生长和供应量。例如，持续的降雨可能导致蔬菜生长受阻，甚至出现烂

根和死亡，从而造成供应短缺和批发价格上涨。其次，气象条件还间接影响消费者行为。降雨通常会减少人们的外出意愿，进而导致蔬菜销量下降。然而，这一现象并非一致，因为在雨季，适用于炖食和烹饪的蔬菜（如土豆、萝卜等）的需求可能反而增加。

因此对于商家而言，与气象部门或相关数据供应商建立合作关系是至关重要的。这不仅能帮助商家实时获取并分析气象数据，还能更精准地进行补货决策，以应对气象条件带来的多重影响。

9.2 节假日因素 [10]

节假日因素在销售量短期内的波动模式上表现出显著的影响，这种影响常常是与特定节假日背后的文化因素和饮食传统紧密相关的。为了应对这种波动，商家可以通过对节假日与销售数据进行时序分析来预测需求变化，从而更精准地制定补货策略。具体而言，商家应针对与某一特定节假日（如春节、端午节等）相关联的蔬菜品种进行特别考量。例如，在春节期间，大葱、大蒜和白菜等蔬菜常用于制作传统节日食品，因此在节假日前应适当增加这些品种的进货量。这不仅可以满足消费者节日期间的特殊需求，还能避免节假日期间因供应不足而导致的销售机会损失。

9.3 环境温度

环境温度不仅直接影响蔬菜的生理活动和保质期，还在一定程度上塑造了消费者的购买行为和需求模式。具体来说，在高温季节，含水量较高的蔬菜（如黄瓜等）通常会受到更高的消费者青睐，因为这些蔬菜具有显著的降温和解暑效果。反之，在低温季节，适用于炖食和烹饪的蔬菜（如土豆、白菜、萝卜等）的需求有可能增加，这主要是因为这类蔬菜通常会用于制作更为丰富和高热量的食品。

由于温度因素的这种多维性影响，商家需要采取一种更为灵活和响应式的补货策略。这可能包括但不限于实时监控气象预报数据，以预测即将到来的温度变化，并据此调整进货计划。同时，商家还需要考虑到温度对蔬菜保质期的影响，特别是对于那些易腐败或者对温度敏感的品种。在必要情况下，商家应当重新配置冷藏和冷冻设备的温度设置，以最大限度地延长蔬菜的保质期。

进一步地，商家可以考虑与供应链合作伙伴共同研发温度敏感的物流解决方案，以确保在不同温度条件下蔬菜的质量和新鲜度都能得到妥善保证。这种温度敏感的补货和物流策略可以有效优化库存管理。

9.4 宏观经济稳定性指标（通货膨胀率)

通货膨胀率作为一项宏观经济指标，其波动会直接影响商品的价格水平和消费者的购买力 [11]。高通胀环境下，消费者的购买力可能受到影响，进而导致销售量的不稳定。

因此，商超在进行补货决策时，应充分考虑到这一经济环境因素。具体而言，可以通过数据分析来预测在不同通胀水平下，各类蔬菜的需求量可能出现的变化，从而更准确地进行补货。

商家可能需要更为灵活的定价策略，以在不同通胀水平下平衡成本和销售量。因此，在进行补货决策时，应考虑到经济稳定性因素，实时关注当下的物价和消费趋势，以实现成本和效益的最优化。

以上的四个外部因素——气象条件、节假日、环境温度以及宏观经济稳定性指标（如通货膨胀率）——为商家在补货和定价策略中带来了更多的多维度考量。采集这些相关数据并将其整合进现有的决策模型不仅能够提高补货精度，还能更好地优化商品定价。最终，这种多维度的数据分析和决策模型不仅有助于商家降低库存成本和提高销售额，还能更好地满足消费者需求，从而建立和维护一个更为健康和可持续的商业生态系统。因此，对于商家而言，多元数据采集与分析是一项至关重要的长期战略任务。

十、 模型的评价

10.1 模型优点

- **大数据与统计规律双驱动.** 本模型采用了一种混合优化策略，集成了数据挖掘和机器学习技术，以针对性地解决蔬菜商品供应链中的各种问题。该模型综合运用了多元统计分析和关联规则挖掘，实现了库存需求的多维度预测。

  在模型构建过程中，我们依照最优化理论和计算复杂性理论，将问题层次化并采用模块化设计。这不仅保证了模型的系统性和一致性，也为后续的功能扩展和算法优化提供了便利。
- **可解释性与实用性.** 在决策层面，模型采用了一种基于规则的推理框架，该框架具有很强的解释性。弥补了机器学习的不足。在供应链管理和其他商业应用中，解释性常常与准确性同等重要。

  模型进一步通过整合现有的业务流程和约束条件（如最小陈列量、可售单品总数等）来增加其实用性。这样的综合考虑不仅强化了模型的实际应用潜力，也增强了其在学术界的影响力。

  此外，模型还嵌入了成本效益分析和非线性约束优化，以模拟与真实世界复杂的供需关系更为接近。
- **准确性与效率.** 为了提高预测精度，本模型采用了长短期记忆（LSTM）循环神经网络，并针对大数据环境进行了优化。通过大量的实验验证，该算法在大规模流水量样本下表现出显著的优越性。

  在数据预处理方面，模型运用了一系列先进的技术，包括异常值检测、特征工程、和数据融合等，这些都极大地增加了模型的鲁棒性和准确性。

10.2 模型缺点

- **需大量的历史数据驱动.** 神经网络模型往往需要大量的时序数据才能够进行良好的拟合和预测，统计规律也需要从大量的数据中才能得出，而现实中获取到巨额的历史数据不易，因此在模型的驱动能力上有待提高。
- **精细化程度不足.** 在现实中，商家往往会根据当前的销售情况动态调整定价策略，进行适时的打折促销活动。我们的模型没有将定价策略细化到具体的时间，因此精细化程度仍有欠缺。

参考文献

[1] 李亚丽. (2023). 考虑保鲜技术投资的联合补货与定价协同决策 [D]. 重庆交通大学. DOI:10.27671/d.cnki.gcjtc.2022.000163.

[2] 李常青. B2C 电子商务竞争环境下 3C 产品动态定价策略研究 [D]. 西南交通大学, 2012.

[3] 刘子军. 基于 Pearson 相关系数的低渗透砂岩油藏重复压裂井优选方法 [J]. 油气地质与采收率, 2022, 29(02):140-144. DOI:10.13673/j.cnki.cn37-1359/te.2022.02.017.

[4] 王立辉, 许扬, 陆于平等. 数字化变电站过程层采样值时间同步性分析及应用 [J]. 电力自动化设备, 2010, 30(08):37-40+44.

[5] 陈智君, 郝奇, 伍永健等. 基于二元正态分布匹配和非线性优化的激光 SLAM 研究 [J]. 组合机床与自动化加工技术, 2021(09):19-23. DOI:10.13462/j.cnki.mmtamt.2021.09.005.

[6] 李明伟. LSTM 模型预测不同水保措施条件下土壤侵蚀量 [J/OL]. 水土保持通报:1-8[2023-09-10]. DOI:10.13961/j.cnki.stbctb.20230508.010.

[7] 潘晓飞, 解志恒, 王淑云. 考虑损失规避的生鲜品商超保鲜努力和定价的优化决策 [J]. 公路交通科技, 2022, 39(06):177-185+190.

[8] 蒋渊, 马士华. (2019). 考虑销售时机的季节性商品批发价定价和质量披露管理. 运筹与管理, 28(02), 148-153.

[9] 朱慧娟. 气象因子对仓储保鲜及商品销售的影响研究 [D]. 安徽农业大学, 2014.

[10] 秦志勇. 天气和假日效应对生鲜农产品网络销售的影响研究 [D]. 华中科技大学, 2012.

[11] 涂涛涛, 李崇光. 中国蔬菜价格波动与通货膨胀——基于波动来源的分解 [J]. 华中农业大学学报 (社会科学版), 2014(01):37-43. DOI:10.13300/j.cnki.hnwkxb.2014.01.008.

附录 A　按照类划分数据源程序

```
import pandas as pd
import os
import zipfile
import shutil

# Step 1: Read the CSV file
# Replace 'RQ1step2-chan.csv' with the path of your actual CSV file
df = pd.read_csv('merged3.csv', encoding='utf-8')

# Step 2: Group the data by '分类名称' and '销售日期', and sum up the '销量(千克)'
grouped_df = df.groupby(['分类名称', '日期'])['批发价格(元/千克)'].mean().reset_index()

# Step 3: Find all unique '销售日期'
all_unique_dates = grouped_df['日期'].unique()

# Step 4: For each '分类名称', generate a DataFrame with all dates and fill missing
    values with 0
filled_dfs = {}
for category in grouped_df['分类名称'].unique():
category_df = grouped_df[grouped_df['分类名称'] == category].copy()

# Create a DataFrame with all dates
full_dates_df = pd.DataFrame({'日期': all_unique_dates})
full_dates_df['分类名称'] = category
full_dates_df = full_dates_df.merge(category_df, on=['分类名称', '日期'], how='left')

# Fill missing '销量(千克)' values with 0
full_dates_df['批发价格(元/千克)'].fillna(0, inplace=True)

filled_dfs[category] = full_dates_df

# Step 5: Create a directory to store the CSV files, remove it first if it exists
output_directory = 'Grouped_Sales_By_Category_Filled_cost'
if os.path.exists(output_directory):
shutil.rmtree(output_directory)
os.makedirs(output_directory, exist_ok=True)

# Save the filled DataFrames
for category, filled_df in filled_dfs.items():
output_file_path = os.path.join(output_directory, f"{category}.csv") #
    文件名现在是分类名称
filled_df.to_csv(output_file_path, index=False, encoding='gbk')
```

附录 B　按单品划分数据源程序

```
import pandas as pd
import os
import zipfile
import shutil

# Step 1: Read the CSV file
# Replace 'your_file_path.csv' with the path of your actual CSV file
df = pd.read_csv('RQ1step2-chan.csv', encoding='gbk')

# Step 2: Group the data by '单品名称' and '销售日期', and sum up the '销量(千克)'
grouped_df = df.groupby(['单品名称', '销售日期'])['销量(千克)'].sum().reset_index()

# Step 3: Find all unique '销售日期'
all_unique_dates = grouped_df['销售日期'].unique()

# Step 4: For each '单品名称', generate a DataFrame with all dates and fill missing
    values with 0
filled_dfs = {}
for item in grouped_df['单品名称'].unique():
item_df = grouped_df[grouped_df['单品名称'] == item].copy()

# Create a DataFrame with all dates
full_dates_df = pd.DataFrame({'销售日期': all_unique_dates})
full_dates_df['单品名称'] = item
full_dates_df = full_dates_df.merge(item_df, on=['单品名称', '销售日期'], how='left')

# Fill missing '销量(千克)' values with 0
full_dates_df['销量(千克)'].fillna(0, inplace=True)

filled_dfs[item] = full_dates_df

# Step 5: Create a directory to store the CSV files, remove it first if it exists
output_directory = 'Grouped_Sales_By_Item_Filled'
if os.path.exists(output_directory):
shutil.rmtree(output_directory)
os.makedirs(output_directory, exist_ok=True)

# Save the filled DataFrames
for item, filled_df in filled_dfs.items():
output_file_path = os.path.join(output_directory, f"{item}.csv") # File name is now the
    item name
filled_df.to_csv(output_file_path, index=False, encoding='gbk')

# Step 6: Zip all the CSV files
```

```
zip_file_path = 'Grouped_Sales_By_Item_Filled.zip'
with zipfile.ZipFile(zip_file_path, 'w') as zipf:
for root, _, files in os.walk(output_directory):
for file in files:
zipf.write(os.path.join(root, file), file)
```

附录 C　问题一绘制相关系数矩阵

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# 设置字体以支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei'] # 黑体
plt.rcParams['axes.unicode_minus'] = False # 解决无法显示符号的问题



# 读取数据文件并存储在字典中
file_paths = [
"花菜类.csv",
"花叶类.csv",
"辣椒类.csv",
"茄类.csv",
"食用菌.csv",
"水生根茎类.csv"
]
data_dict = {}
for file_path in file_paths:
file_name = os.path.basename(file_path).split(".")[0]
data_dict[file_name] = pd.read_csv(file_path, encoding='GBK')

# 计算皮尔逊相关系数矩阵
correlation_data = {key: df['销量(千克)'] for key, df in data_dict.items()}
correlation_df = pd.DataFrame(correlation_data)
correlation_matrix = correlation_df.corr()

# 定义颜色
colors = [
"#E6B977",
"#ECCCBA",
"#EA8D9E",
"#AE4E89",
"#533380",
```

```
    "#3F2D6B"
    ]

    # 绘制皮尔逊相关系数矩阵图
    plt.figure(figsize=(10, 8))
    sns.set(font_scale=1.2)
    ax = sns.heatmap(correlation_matrix, annot=True, cmap=colors, fmt=".2f", linewidths=.5,
        cbar=False)

    # 使用matplotlib的原生方法设置轴标签
    ax.set_xticklabels(ax.get_xticklabels(), fontproperties='SimHei')
    ax.set_yticklabels(ax.get_yticklabels(), fontproperties='SimHei')

    # 使用指定的字体显示标题
    plt.title('时间序列皮尔逊相关系数', fontproperties='SimHei')

    # 保存图表为PDF文件
    plt.savefig('correlation_matrix.pdf', format='pdf')
```

附录 D　问题二滑动窗口相关性

```
import pandas as pd
import matplotlib.pyplot as plt
# 设置字体以支持中文显示
plt.rcParams['font.sans-serif'] = ['SimHei'] # 黑体
plt.rcParams['axes.unicode_minus'] = False # 解决无法显示符号的问题
# Step 1: Read the CSV file with GBK encoding
df = pd.read_csv('花菜类.csv', encoding='gbk')

# Step 2: Convert "销售日期" to datetime and sort the DataFrame
df['销售日期'] = pd.to_datetime(df['销售日期'])
df = df.sort_values(by='销售日期')

# Step 3: Normalize the data using Min-Max normalization
df['总销量(千克)_归一化'] = (df['总销量(千克)'] - df['总销量(千克)'].min()) /
    (df['总销量(千克)'].max() - df['总销量(千克)'].min())
df['平均单价_归一化'] = (df['平均单价(元/千克)'] - df['平均单价(元/千克)'].min()) /
    (df['平均单价(元/千克)'].max() - df['平均单价(元/千克)'].min())

# Step 4: Calculate the rolling median with a window size of 7 days
window_size = 7
df['滑动窗口中位数_总销量'] = df['总销量(千克)_归一化'].rolling(window=window_size,
    center=True).median()
df['滑动窗口中位数_平均单价'] = df['平均单价_归一化'].rolling(window=window_size,
    center=True).median()
```

```
# Step 5: Calculate the rolling Pearson correlation
rolling_r = df['总销量(千克)_归一化'].rolling(window=window_size,
    center=True).corr(df['平均单价_归一化'])

# Step 6: Plotting
f, ax = plt.subplots(2, 1, figsize=(14, 6), sharex=True)

df[['滑动窗口中位数_总销量', '滑动窗口中位数_平均单价']].plot(ax=ax[0])
ax[0].set(xlabel='销售日期', ylabel='滑动窗口中位数')
ax[0].legend(["总销量(千克)", "平均单价(元/千克)"])

rolling_r.plot(ax=ax[1])
ax[1].set(xlabel='销售日期', ylabel='Pearson r')
ax[1].set_title('滑动窗口 Pearson 相关系数')

plt.tight_layout()
plt.savefig("Sliding_Window_Analysis.pdf")
plt.show()
```

附录 E　问题二正态统计模型

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

file_list = ['花叶类', '花菜类', '辣椒类', '茄类', '食用菌', '水生根茎类']

# 创建一个 2x3 的子图布局
fig, axes = plt.subplots(2, 3, figsize=(18, 12), subplot_kw={'projection': '3d'})

for ax, file_name in zip(axes.flat, file_list):
df = pd.read_csv(f'{file_name}.csv', encoding='gbk')
x = df['总销量(千克)']
y = df['平均单价(元/千克)']
mean = np.mean(df[['总销量(千克)', '平均单价(元/千克)']], axis=0)
cov_matrix = np.cov(x, y)
rv = multivariate_normal(mean, cov_matrix)

x_range = np.linspace(min(x), max(x), 100)
```

```
y_range = np.linspace(min(y), max(y), 100)
x_mesh, y_mesh = np.meshgrid(x_range, y_range)
pos = np.empty(x_mesh.shape + (2,))
pos[:, :, 0] = x_mesh
pos[:, :, 1] = y_mesh
pdf_values = rv.pdf(pos)

ax.plot_surface(x_mesh, y_mesh, pdf_values, cmap='coolwarm', alpha=0.8)
ax.scatter(x, y, np.zeros_like(x), c='r', marker='o', alpha=0.5)
ax.set_xlabel('总销量(千克)')
ax.set_ylabel('平均单价(元/千克)')
# ax.set_zlabel('概率密度')
ax.set_title(f'二元正态分布的3D图（{file_name}）')

plt.tight_layout()
plt.show()

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import probplot

plt.rcParams['font.sans-serif'] = ['SimHei'] # 黑体
plt.rcParams['axes.unicode_minus'] = False # 解决无法显示符号的问题

# 读取数据（请确保CSV文件与此脚本在同一目录下）
df = pd.read_csv('花叶类.csv', encoding='gbk')

# 提取两个变量
x = df['总销量(千克)']
y = df['平均单价(元/千克)']

# 绘制总销量(千克)的QQ图
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
probplot(x, dist="norm", plot=plt)
plt.title("总销量(千克) QQ图")

# 绘制平均单价(元/千克)的QQ图
plt.subplot(1, 2, 2)
probplot(y, dist="norm", plot=plt)
plt.title("平均单价(元/千克) QQ图")

plt.tight_layout()
plt.show()

import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import probplot

plt.rcParams['font.sans-serif'] = ['SimHei'] # 黑体
plt.rcParams['axes.unicode_minus'] = False # 解决无法显示符号的问题

# 读取数据（请确保CSV文件与此脚本在同一目录下）
df = pd.read_csv('花叶类.csv', encoding='gbk')

# 提取两个变量
x = df['总销量(千克)']
y = df['平均单价(元/千克)']
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
x.plot(kind='kde')
plt.subplot(1, 2, 2)
y.plot(kind='kde')
plt.show()
```

附录 F　问题二获得条件分布

```
import numpy as np
import pandas as pd

# 给定的7天数据列表
M = [[184.585, 15.488, 96.616, 18.482, 46.523, 18.350],
[218.549, 14.133, 120.776, 16.722, 44.875, 16.975],
[258.691, 13.53, 116.431, 15.923, 43.162, 15.325],
[245.258, 13.325, 124.908, 15.283, 41.352, 13.682],
[255.268, 13.267, 116.18, 14.368, 39.452, 12.371],
[252.6, 13.251, 125.34, 13.014, 37.555, 11.507],
[254.876, 13.246, 116.745, 11.418, 35.844, 11.003]]

file_list = ['花叶类', '花菜类', '辣椒类', '茄类', '食用菌', '水生根茎类']

# 循环遍历每一天的数据
for day, given_x_values in enumerate(M, 1):
day_reasonable_ranges = [] # 初始化空列表，用于存储当天所有类别的合理范围
print(f"Day {day}:\n")
for file_name, given_x in zip(file_list, given_x_values):
df = pd.read_csv(f'{file_name}.csv', encoding='gbk')

x = df['总销量(千克)']
y = df['平均单价(元/千克)']
```

```
mean = np.mean(df[['总销量(千克)', '平均单价(元/千克)']], axis=0)
cov_matrix = np.cov(x, y)
corr_coeff = np.corrcoef(x, y)[0, 1]

mu_x, mu_y = mean
sigma_x, sigma_y = np.sqrt(np.diag(cov_matrix))

# 计算条件期望和条件方差
conditional_mean = mu_y + corr_coeff * (sigma_y / sigma_x) * (given_x - mu_x)
conditional_var = sigma_y ** 2 * (1 - corr_coeff ** 2)
conditional_std = np.sqrt(conditional_var)

print('条件期望',conditional_mean,'条件方差',conditional_var)
# 计算平均单价的合理范围
reasonable_range = [conditional_mean - 3 * conditional_std, conditional_mean + 3 *
    conditional_std]
day_reasonable_ranges.extend(reasonable_range)
#
# print(f"在总销量为 {given_x} 时，{file_name}的平均单价的99.7%合理范围是
    {reasonable_range}\n")

# print(f"Day {day} 所有类别的合并合理范围: \n{day_reasonable_ranges}")
print("=" * 50)
```

附录 G　问题二最优化模型

```
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from scipy.stats import norm
from scipy.integrate import quad

# 定义损耗率矩阵 A
A = np.array([1 - 0.103, 1 - 0.141, 1 - 0.085, 1 - 0.071, 1 - 0.081, 1 - 0.12])

# 读取预测批发价和预测销量数据
df_price = pd.read_excel("预测批发价.xlsx", index_col=0)
df_sales = pd.read_excel("预测销量.xlsx", index_col=0)

# 读取约束条件和正态分布参数
bounds_dates = {
   '7月1日': [0.7921259741098146, 10.288913004034374, 2.1349826623269346, 17.624765257833015,
       0,
   19.682661265198842, 0, 17.445907381152058, 1.2542865217329044, 16.21755479421607,
   0.08602856521430624, 20.613587301203097],
```

```
    '7月2日': [0.6999354471801418, 10.1967224771047, 2.1678900231125935, 17.657672618618673,0,
    19.450174176451654, 0, 17.43276782713199, 1.271389845432772, 16.234658117915938,
    0.13830557658164544, 20.665864312570434],
    '7月3日': [0.5909756034629448, 10.087762633387502, 2.1825344058090907, 17.67231700131517,
        0,
    19.491985285748285, 0, 17.42680276823311, 1.2891677541086644, 16.25243602659183,
    0.20103799022245283, 20.728596726211244],
    '7月4日': [0.6274376028813373, 10.124224632805895, 2.1875130102083977, 17.677295605714477,
        0,
    19.41041272567751, 0, 17.42202474858945, 1.3079523511333964, 16.271220623616564,
    0.2635042663508447, 20.791063002339634],
    '7月5日': [0.6002668581349582, 10.097053888059516, 2.1889215909652746, 17.678704186471354,
        0,
    19.49440061104942, 0, 17.415193673630156, 1.3276709888941642, 16.290939261377332,
    0.3133480204618131, 20.840906756450604],
    '7月6日': [0.6075087709205063, 10.104295800845065, 2.1893101649671705, 17.67909276047325,
        0,
    19.406255671772758, 0, 17.405085175821537, 1.3473584919637318, 16.310626764446898,
    0.34619699342281685, 20.873755729411606],
    '7月7日': [0.6013308872998513, 10.09811791722441, 2.1894315943427634, 17.679214189848842,
        0,
    19.488963723419367, 0, 17.39316998933516, 1.3651156441788252, 16.32838391666199,
    0.36535889431673496, 20.892917630305526],
}
normal_params = {
    '7月1日': {
        'a': [5.540519489072094, 9.879873960079975, 8.441698409992837, 8.43827848149936,
            8.735920657974487, 10.349807933208702],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
            6.219427705397261, 11.705018546096381]
    },
    '7月2日': {
        'a': [5.448328962142421, 9.912781320865633, 8.20921132124565, 8.425138927479294,
            8.753023981674355, 10.40208494457604],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
            6.219427705397261, 11.705018546096381]
    },
    '7月3日': {
        'a': [5.339369118425224, 9.92742570356213, 8.251022430542278, 8.419173868580412,
            8.770801890350247, 10.464817358216848],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
            6.219427705397261, 11.705018546096381]
    },
    '7月4日': {
        'a': [5.3758311178436164, 9.932404307961438, 8.169449870471503, 8.414395848936753,
            8.78958648737498, 10.52728363434524],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
```

```
            6.219427705397261, 11.705018546096381]
    },
    '7月5日': {
        'a': [5.348660373097237, 9.933812888718315, 8.253437755843418, 8.407564773977457,
            8.809305125135747, 10.577127388456208],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
            6.219427705397261, 11.705018546096381]
    },
    '7月6日': {
        'a': [5.355902285882785, 9.93420146272021, 8.165292816566753, 8.397456276168839,
            8.828992628205315, 10.609976361417212],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
            6.219427705397261, 11.705018546096381]
    },
    '7月7日': {
        'a': [5.34972440226213, 9.934322892095803, 8.248000868213362, 8.385541089682462,
            8.846749780420408, 10.62913826231113],
        'b': [2.5052489969928695, 6.664815690445642, 14.039916212457907, 9.015264265984271,
            6.219427705397261, 11.705018546096381]
    }
}

# 计算 K 值
from scipy.stats import norm

# Define objective function and constraints
def calc_K(a, b, X, M):
cdf_value = norm.cdf(X, a, np.sqrt(b))

def segment1(x):
return M * 2 * (1 - cdf_value)

def segment2(x):
return M * 2 * cdf_value

return np.piecewise(X, [X >= a, X < a], [segment1, segment2])

def objective(XY):
X = XY[:6]
Y = XY[6:]
K = np.array([calc_K(a[i], b[i], Xi, M[i]) for i, Xi in enumerate(X)])
# print("K:", K) # Debugging line
return -np.sum(X * K - C * Y)


# Update constraints
def constraint1(XY):
```

```
X = XY[:6]
Y = XY[6:]
K = np.array([calc_K(a[i], b[i], X[i], M[i]) for i in range(6)])
return np.sum(Y * A) - np.sum(K)

def constraint2(XY):
X = XY[:6]
Y = XY[6:]
K = np.array([calc_K(a[i], b[i], X[i], M[i]) for i in range(6)])
return Y - 0.9 * M

def constraint3(XY):
X = XY[:6]
Y = XY[6:]
K = np.array([calc_K(a[i], b[i], X[i], M[i]) for i in range(6)])
return 1.1 * M - Y
def constraint4(XY):
X = XY[:6]
Y = XY[6:]
return X - C

# Update constraints list


# Define constraints
con1 = {'type': 'eq', 'fun': constraint1}
con2 = {'type': 'ineq', 'fun': constraint2}
con3 = {'type': 'ineq', 'fun': constraint3}
con4 = {'type': 'ineq', 'fun': constraint4}
constraints = [con1, con2, con3, con4]

# 循环执行优化
for date in df_price.columns:
C = df_price[date].values
M = df_sales[date].values
a = normal_params[date]['a']
b = normal_params[date]['b']

bounds_X = list(zip(bounds_dates[date][::2], bounds_dates[date][1::2]))
bounds_Y = [(0, None) for _ in range(6)]
bounds = bounds_X + bounds_Y
initial_guess = np.concatenate([M, M])

print(f"Optimizing for {date}:")
print("M:", M)
print("C:", C)
```

```
result = minimize(objective, initial_guess, bounds=bounds, constraints=constraints,
    method='SLSQP')

optimized_XY = result.x
optimized_X = optimized_XY[:6]
optimized_Y = optimized_XY[6:]

# Calculate and print K values
optimized_K = np.array([calc_K(a[i], b[i], optimized_X[i], M[i]) for i in range(6)])
print("Optimized K:", optimized_K)

print("Optimized X:", optimized_X)
print("Optimized Y:", optimized_Y)
print('销售额',np.sum(optimized_K*optimized_X))
print("Optimized objective function value:", -result.fun)
print("------")
```

附录 H　问题三优化模型

```
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from scipy.stats import norm
from scipy.integrate import quad

# 花叶类17种A 花菜类2个B 辣椒类10个C 茄类5个D 食用菌8个E 水生根茎类7个F
# 下面定义成本向量c
Ac = [9.67, 4.07, 4.62, 2.6, 3.12, 1.53, 2.56, 4.09, 4.4, 9.19, 2.21, 2.7, 5.68, 3.49, 2.66,
    2.84, 2.15]
Bc = [8.39, 7.59]
Cc = [12.13, 2.33, 2.1, 2.11, 12.72, 3.63, 8.97, 4.29, 3.32, 2.54]
Dc = [3.2, 4.34, 3.43, 7, 3.98]
Ec = [3.41, 1.95, 3.29, 2.6, 3.12, 15.6, 1.45, 1.3]
Fc = [10.38, 18, 5.29, 9.16, 16.06, 11.67, 13.69]
# 下面定义损耗率矩阵s
As = [18.51, 9.43, 13.7, 8.42, 7.61, 9.43, 15.68, 14.43, 2.48, 26.16, 18.52, 10.33, 15.25,
    9.43, 12.81, 9.43, 13.62]
Bs = [9.43, 9.26]
Cs = [9.43, 9.43, 9.43, 9.43, 9.43, 5.7, 10.18, 9.43, 9.43, 9.43]
Ds = [6.71, 9.43, 6.07, 6.9, 5.01]
Es = [0.2, 0, 6.57, 9.43, 0.84, 10.8, 0.45, 9.43]
Fs = [5.54, 24.05, 16.63, 9.61, 12.69, 29.25, 9.43]
# 下面定义销量矩阵M
AM = [2.273333333, 9.8, 2.21925, 4.506571429, 5.933428571, 1, 7.493166667, 3.861285714,
    10.42857143, 2.179333333, 8.923,
```

```
4.900857143, 2.92075, 32.28571429, 1.699333333, 21.28571429, 13.29671429]
BM = [6.30075, 12.55771429]
CM = [1.217, 7, 11.28571429, 21.42857143, 2.035714286, 14.23342857, 6.842428571,
    11.28571429, 2.142857143, 1]
DM = [1.2405, 0.512, 10.90285714, 4.214428571, 3.121166667]
EM = [10, 8.857142857, 1, 2.285714286, 2, 4.624142857, 16.14285714, 4]
FM = [6.026285714, 4.033142857, 0.656571429, 2.037333333, 0.813, 3.004428571, 1.1992]

# 下面定义销售单价X
AX = [14, 6.8, 6, 6, 6, 3.9, 5.2, 8, 6.8, 13.2, 4, 5.2, 9.2, 5.8, 7.2, 4.5, 4]
BX = [14, 14]
CX = [20.8, 4.8, 2.8, 5.8, 20, 5.2, 12, 5.9, 5.8, 4.8]
DX = [8, 9, 6, 12, 6]
EX = [5.5, 3, 7.8, 3.8, 5.8, 24, 2, 2.6]
FX = [16, 26, 9.2, 14, 26, 16, 18]

from scipy.optimize import minimize

# 成本向量
c = np.array(Ac + Bc + Cc + Dc + Ec + Fc)

# 损耗率矩阵
s = np.array(As + Bs + Cs + Ds + Es + Fs)

# 销量矩阵
M = np.array(AM + BM + CM + DM + EM + FM)

# 销售单价
X = np.array(AX + BX + CX + DX + EX + FX)


# 定义目标函数
def objective(Y):
return - np.sum(X * M) + np.sum(c * Y)


# 定义新的约束条件
def class_constraint(Y, class_indices, target_sum):
return np.sum(Y[class_indices]) - target_sum


# 定义新的约束条件：Y都大于等于2.5或等于0
def all_Y_geq_25(Y):
return np.min(np.where(Y == 0, 2.5, Y) - 2.5)


# 定义新的约束条件：Y不等于0的个数在27-33之间
```

```
def count_Y_non_zero(Y):
count = np.count_nonzero(Y)
return 33 - count # 上界


def count_Y_non_zero_lower(Y):
count = np.count_nonzero(Y)
return count - 27 # 下界

def all_Y_geq_0(Y):
return Y
Y_initial = M



# 约束条件列表
constraints = [
{'type': 'ineq', 'fun': lambda Y: Y * (1 - s/100) - M},
{'type': 'ineq', 'fun': lambda Y: -183.85762252 + np.sum(Y[list(range(0, 17))])},
{'type': 'ineq', 'fun': lambda Y: -13.9392 + np.sum(Y[list(range(17, 19))])},
{'type': 'ineq', 'fun': lambda Y: -86.9571 + np.sum(Y[list(range(19, 29))])},
{'type': 'ineq', 'fun': lambda Y: -16.5852 + np.sum(Y[list(range(29, 34))])},
{'type': 'ineq', 'fun': lambda Y: -41.8707 + np.sum(Y[list(range(34, 42))])},
{'type': 'ineq', 'fun': lambda Y: -16.515 + np.sum(Y[list(range(42, 49))])},
{'type': 'ineq', 'fun': lambda Y: Y - 0}, # 硬性约束：所有的Y都应该大于或等于0
]

# 使用Scipy的minimize函数进行优化
result = minimize(objective, Y_initial, constraints=constraints, method='SLSQP',
    options={'maxiter': 10000})

print(np.sum(result.x>2.5))
# 输出结果
if result.success:
print("优化成功!")
print(f"最优Y值：{result.x}")
print(f"最优目标函数值：{-result.fun}")
else:
print("优化失败")
print("原因:", result.message)
# 按照值的大小排序Y，返回排序后的索引
sorted_indices = np.argsort(result.x)

# 获取最大的33个Y值的索引
top_33_indices = sorted_indices[-33:]

# 创建一个全为0的数组
```

```
Y_top_33 = np.zeros_like(result.x)

# 只保留最大的33个Y值
Y_top_33[top_33_indices] = result.x[top_33_indices]

# 计算使用这33个Y值时目标函数的值
objective_value_top_33 = objective(Y_top_33)
print(f"替换后的Y: {Y_top_33}")
print(f"只保留最大的33个Y值时的目标函数值: {-objective_value_top_33}")
```