基于 SARIMA 和双层贪心算法的蔬菜定价补货决策

摘要

本文对蔬菜类商品的自动定价与补货决策进行研究。分别从品类整体和单品局部两个角度出发，分析蔬菜销售的分布规律和相互关系，继而考虑蔬菜损耗与市场需求，制订了使收益最大化的最优定价补货策略。

针对问题一，本文对各品类、单品销售量的分布规律和相互关系进行分析。首先统计各品类和部分单品不同季节的销售量，发现存在**季节性**和**突发性**。接着分析关联关系，通过各品类销量的时间序列进行 **DTW 多尺度相关性分析**，发现水生根茎类、花菜类、茄类销量随时间的变化具有相似性；通过对单品销量利用 Apriori 算法进行**关联规则挖掘**，发现部分单品常一同出现，如西兰花与静藕、紫茄子等。

针对问题二，本文求取以品类为单位的收益最大化的补货决策。首先，本文利用历史数据分析销售总量与成本加成定价的关系。假设理想状态，将历史当日总销量看作日补货量，结合**供需关系**，分析加成比例在需求量固定时随日补货量的变化。然后，建立**最大收益策略模型**，寻找最优自动定价机制与补货决策。利用历史数据确定损耗商品价格折扣，考虑销量与时间的关系，再利用 **SARIMA** 预测未来需量，对模型进行灵敏度分析后进行需量预测。最后通过分析日补货量在 **3 个不同大小阶段变化**对收益的影响求得未来一周的最优策略。求解得到其中 2023 年 7 月 1 日的最大收益为 **610.8277** 元。

针对问题三，本文基于问题二的结论建立**可售单品静态补货定价决策模型**。考虑到实际生活中大多数人的购买欲望会受价格和促销情况的影响，故在模型中加入**价格敏感系数**和**需求奖励度**，并加入可售总数约束、单品陈列量约束，并考虑第二问品类最佳日补货量。接着，经分析发现收益由日补货量和单位补货量收益共同决定，基于此，架构**双层贪心**，求解得到 7 月 1 日的最大收益为 **1109.8847** 元。

针对问题四，基于上述问题求解并联系实际，本文希望商超还可以采集平均每天不同时段商超的人流量和单品购买次数、平均每天不同时段单品的剩余比例、不同保鲜方式的成本信息和损耗率信息、每个单品近期的供应上限数据。本文基于这些数据建立了**动态单品补货定价策略模型**，可帮助商超更好制定蔬菜商品的补货定价策略。

本文的亮点有：1、针对大数据集设计高效省时的相关性分析算法；2、针对不确定和动态因素设计容错率高，适应性强的双层贪心优化策略.

关键词：最大收益策略　DTW　关联规则挖掘　SARIMA　双层次贪心算法

1 问题重述

1.1 问题背景

生鲜食品是人们生活中不可或缺的一部分，《“十四五”冷链物流发展规划》中提出要构建生鲜食品供应链生态，并注重生鲜食品的商贸营销方面。在人们需求和政策推动下，生鲜食品的零售业不断发展。然而由于该类食品本身的易变质性，定价策略和补货决策成为零售商家重点需要考虑的问题。本文主要考虑生鲜食品中的蔬菜类。蔬菜类商品常采用“成本加成定价”法确定销售定价，并对损耗的产品采取限时打折策略。对历史销售和需求情况进行分析，可有助于商超在单品和进货价格未知的情况下作出补货和定价策略[1] [6]。问题背景图如下。

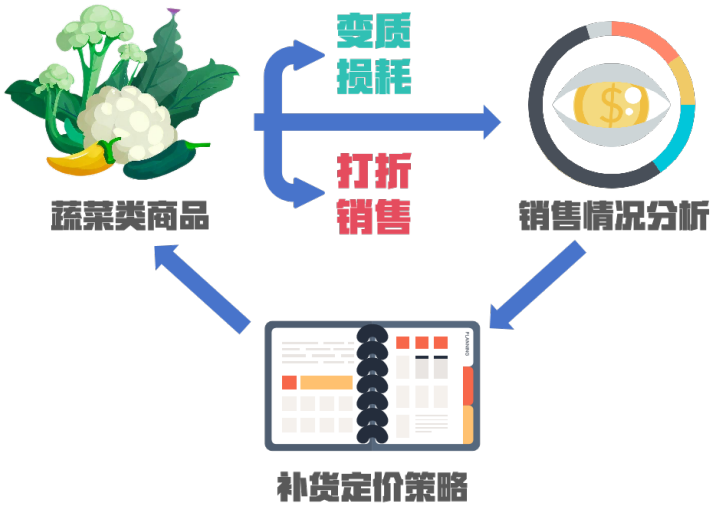

图 1　问题背景图

1.2 问题提出

四个附件是关于某商超的商品信息和历史销售情况的数据。附件 1 是 6 个蔬菜品类的商品信息；附件 2 是各类商品 2020 年 7 月 1 日至 2023 年 6 月 30 日的销售流水数据；附件 3 是蔬菜类商品 2020 年 7 月 1 日至 2023 年 6 月 30 日的批发价格信息；附件 4 是各蔬菜类商品的近期损耗率。根据附件数据完成以下问题：

问题一：不同蔬菜类商品之间可能有潜在的关联关系，要求对各品类及单品销售量的相互关系和分布规律进行分析。

问题二：若商超以品类为单位进行补货，要求对各蔬菜品类的项受总量和成本加成定价的关系进行分析，并求取使商超收益最大的 2023 年 7 月 1 日至 7 日的各类蔬菜品类的日补货总量与定价策略。

问题三：由于蔬菜类商品的销售空间有限，商超想制定进一步的补货计划，规定可售单品总数处于 27 至 33 个范围内，要求单品订购量陈列量最小为 2.5 千克。基于 2023 年 6 月 24 至 30 日的可售品种，要求 7 月 1 日的单品捕获量和定价策略，希望在尽可能满足各品类蔬菜商品的市场需求前提下，使商超能获得最大收益。

问题四：除附件数据外商超还需要采集哪些相关数据，来更好制定蔬菜的补货和定价决策，并帮助解决上述问题，要求给出意见与理由。

2 模型假设

1. 假设附件数据经过预处理后均真实可靠;
2. 假设当日补货的蔬菜商品仅在当日销售，当天运损和变质的蔬菜总量记为损耗量，隔日不会再进行售出。
3. 假设当日蔬菜商品销售完后不会进行补货，隔日再进行补货;
4. 假设历史的销售总量可以认为是当天的单品需求量。

3 符号说明

符号	说明
a_{ijk}	第 k 天第 i 个品类第 j 个单品的定价
b_{ijk}	第 k 天第 i 个品类第 j 个单品的定价
N_{ik}	第 k 天第 i 个品类的总销量
β	损耗率
α	加成比例
H	日补货量
S	收益
r	打折折扣
w	价格价格敏感系数
Z	需求奖励度

注：表中未列出及重复的符号均以首次出现处为准。

4 问题一模型的建立及求解

4.1 问题一分析

问题一要求分析蔬菜各品类及单品销售量的分布规律及相互关系。本文先对数据进行了初步分析，对缺失值和异常值进行了分析和处理。然后先从分布规律出发，对各品类、部分单品进行数据统计分析，分析它们的分布规律往往存在一定的季节性和突发性；之后从相互关系出发，对不同品类进行 DTW 多层次相关性分析销量对应的时间序列的相似性，对不同单品采用关联规则挖掘分析相互的影响程度。

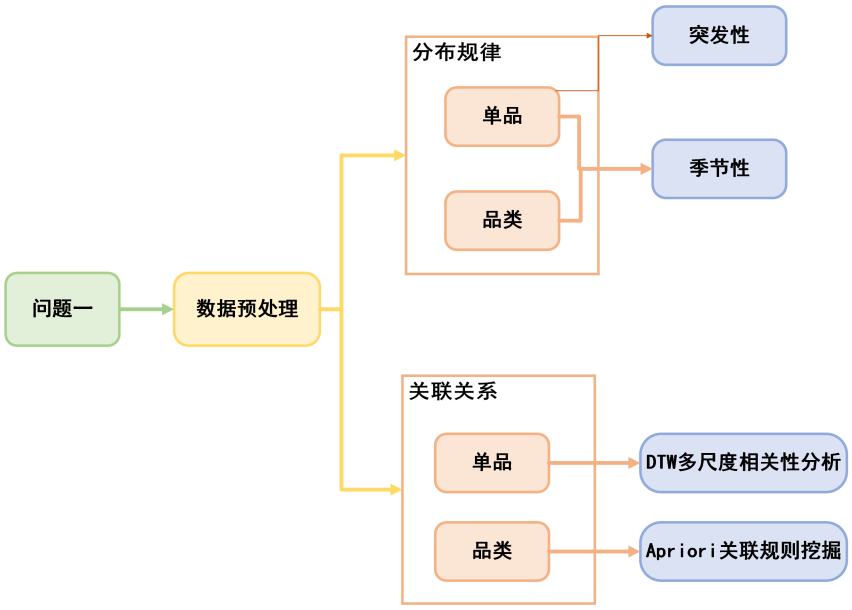

图 2　问题一思路图

4.2 数据预处理

附件 2 数据可能有缺失值与异常值存在，本文对其进行数据预处理操作。

4.2.1 缺失值检验与处理

单品：

经统计，发现部分单品缺失情况较严重，在根据实际情况分析不同单品可能存在季节性以及某些突发原因，故认为缺失是代表当天该单品无供应，销量为 0。

品类：

通过遍历各品类蔬菜的销售量数据，本文发现茄类的销售数据存在时间不连续的情况，具体缺失时间为 2021 年 1 月 13 日、2022 年 9 月 12 日至 2022 年 10 月 14 日、2022 年 12 月 5 日。由于附件 3 存在这些天的批发价格数据，本文认为当天有销售该类商品，于是认定该部分数据是由某种原因而丢失的销售流水数据，需要填补。

经过统计分析，发现不同年份的茄类销量大小与变化趋势相差不大，因此其它年份的数据具有参考意义，本文决定用其它年份在相同日期销量的平均值来对缺失值进行填补。

4.2.2 异常值检验与处理

针对异常值检验，本文主要分为两部分，对于数值型数据采取 IQR（Interquartile Range，四分位距）法，对于非数值型数据采用人为判断。

IQR 检验：

本文对销量数据进行采用 IQR 检验，主要通过绘制各品类每日销量的小提琴图来识别离群值，将超过小提琴上下方即处于下四分数和上四分数区间之外的数据数据记为异常值。绘出的小提琴图如下所示。

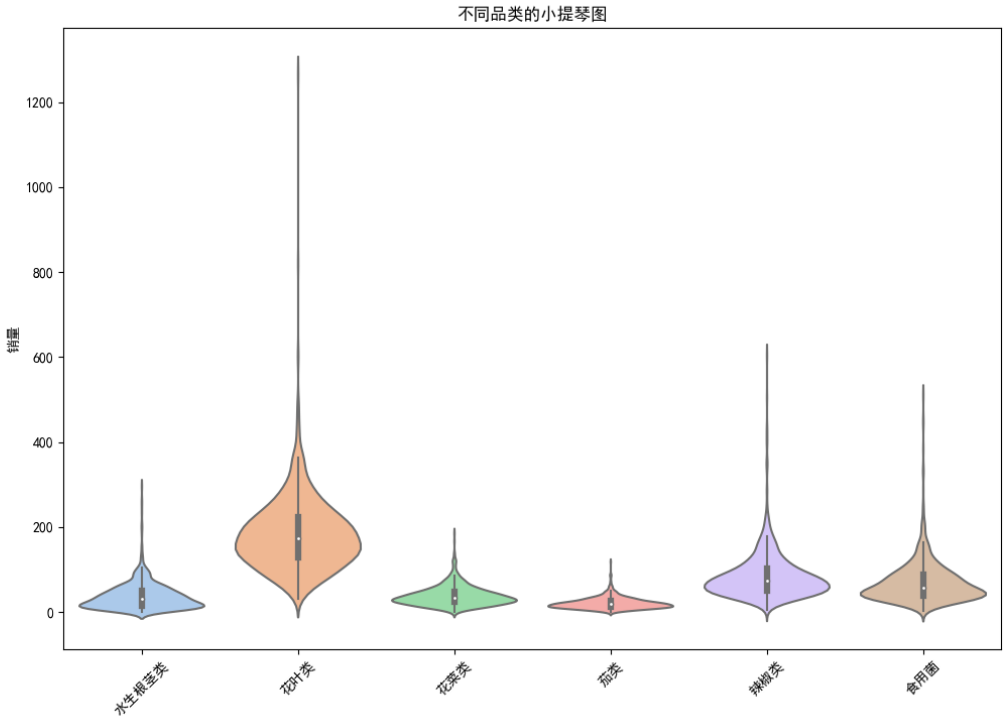

图 3　各品类每日销量的小提琴图

人为判断：

借助 EXCEL 并结合人为判断，本文发现部分非数值型数据存在异常，部分数据销售单价与是否打折存在不对应的情况。如 2023 年 6 月 30 日单品编号为 102900051000944 的洪湖藕带当日的销售单价为 26 元/千克，打折后的单价为 13 元/千克，但扫码销售时间的 19:17:40.764 的订单销售单价为 26 元/千克，却被记为打折销售，这明显存在异常。本文以实际销售单价为准，对是否进行打折销售信息进行修正。针对判定的异常值，本

文决定用其它年份在相同日期销量的平均值来对异常值进行替换。

4.3 初步统计分析

根据附件 2，我们可得知销售类型存在销售和退货两种销售类型。基于此，本文将蔬菜的最终销售量定义为销售类型为销售的正销量与销售类型为退货的负销量之和。

本文首先根据附件 2 的销售流水进行统计，得到各蔬菜品类的销售总量的占比情况如下。

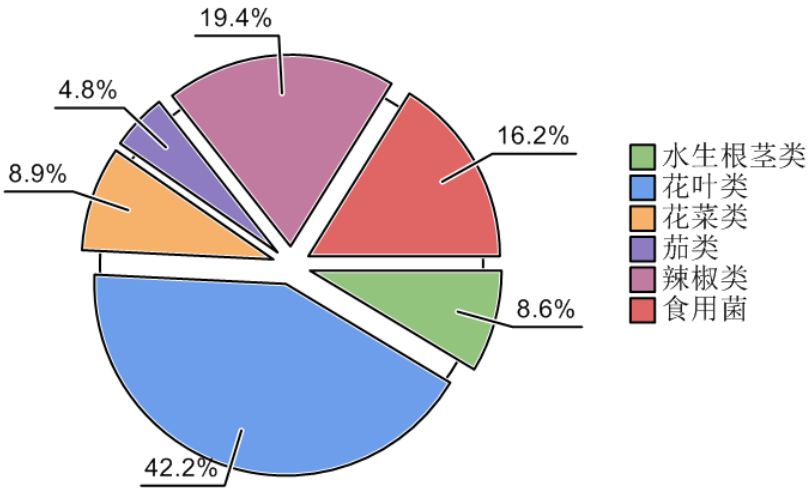

图 4　各蔬菜品类的销售总量统计占比图

从图中可以看出，花叶类的占比较大，接近一半，其次为辣椒类、食用菌、花菜类、水生根茎类，其中茄类的销量总量在各蔬菜品类中占比最小。

4.4 蔬菜商品销售量分布规律

4.4.1 各品类的销售量分布规律

季节性分析

考虑到蔬菜具有时令性，本文猜测各品类蔬菜商品销售在时间维度上存在一定分布规律。本文将针对各品类蔬菜的销售量的季节性对附件 2 进行数据挖掘。

附件 2 中销售流水数据截取的时间区间为 2020 年 7 月 1 日—2023 年 6 月 30 日，由于 2020 年、2023 年的数据月份不完整，本文在时间维度上主要分析 2021 年、2022 两年间的各蔬菜品类销售的季节性倾向情况。本文将 3—5 月记为春季、6—8 月记为夏季、9-11 月记为秋季、12 月—次年 2 月记为冬季。统计各蔬菜品类每一个季节下的月平均销售量，具体情况如下图所示。

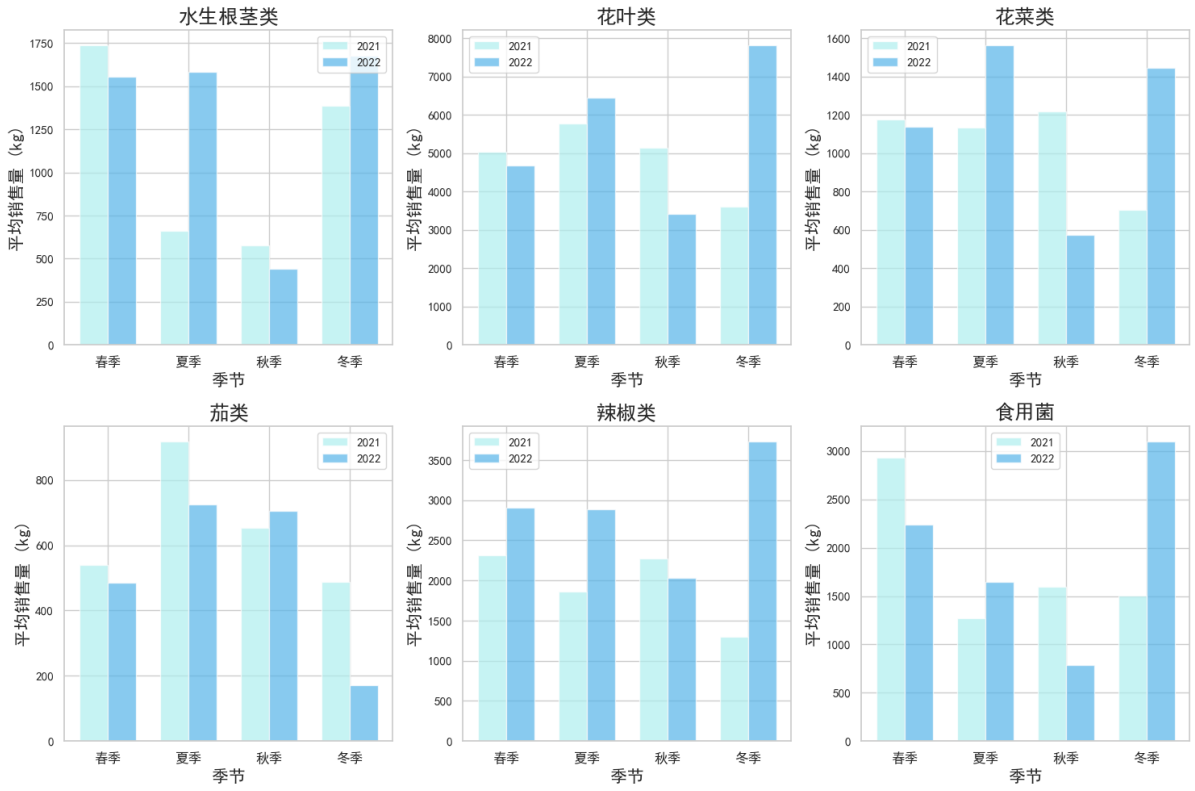

图 5　各蔬菜品类不同季节的平均销量条形图

从图中可以看出，水生根茎类秋季和冬季的销售量较高，夏季的销售量相对较低；花叶类夏季的销售量相对较高，夏季和秋季的销售量相对较低；花菜类夏季销售量最高，而秋季的销售量有所下降；茄类夏季和秋季的销售量相对较高，冬季的销售量最低；辣椒类春季和夏季的销售量较高，而秋季的销售量有所下降；食用菌春季的销售量最高，夏季和秋季的销售量较低。

总体上，各品类蔬菜的季节性趋势较明显。对于大多数品类蔬菜，春季的销售量都相对较高，这可能是由于春季是农作物生长的季节，新鲜蔬菜的供应量和品质都较好。夏季和冬季的销售量因品类而异，但对于某些品类，如茄类和花菜类蔬菜，夏季的销售量较高。而对于大多数品类，秋季的销售量都相对较低。

这些季节性趋势[2]可能是天气、节日、促销活动等多种因素的共同影响的结果，但很大程度上是由蔬菜的生长周期所决定的。

4.4.2 各单品的销售量分布规律

通过分析数据发现，不同单品的销售量差别较大。部分蔬菜商品可能由于受众人群较少、价格昂贵等原因，销量较少，可研究数据有限，如虫草花 (盒)(2)、四川红香椿。为了反映单品间销售量的普遍分布规律，本文选取了六种常见蔬菜进行研究，分别是红薯尖、菠菜、小白菜、西兰花、净藕 (1)、西峡香菇 (1)。考虑到蔬菜单品更可能存在季

节性，本文统计六者 2021 至 2022 的每月销售总量进行季节性分析，如下图所示。

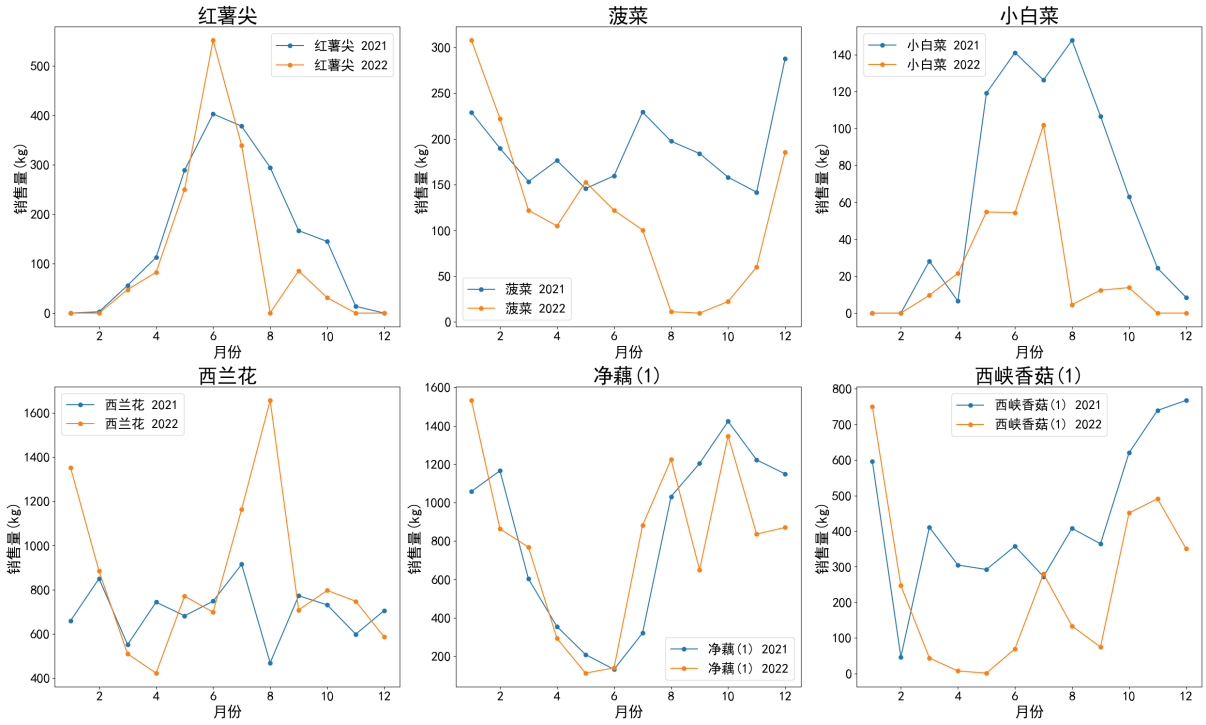

图 6　单品季节性分析

从局部上看，上述六种单品蔬菜中，红薯尖、菠菜、小白菜均属于花叶类品类，明显地，从图中可以看出三者的销售量变化趋势并不同，这一定程度上反映了同一品类的不同单品存在不同销售量的分布规律。西兰花属于花菜类，净藕 (1) 属于水生根茎类，西峡香菇 (1) 属于食用菌。对比来看不同品类的单品间的差异性更大。

从整体上看，上图中 2021 年和 2022 年的每个单品之间的销售量变化大体相同，这可以验证各蔬菜单品销售仍具有周期性。但不可忽视的是 2022 年的 8 月上述六种蔬菜单品的销售量均出现了突变情况。经查找资料，这也许是由于 2022 年 8 月部分地区出现高温情况，导致很多蔬菜歉收，蔬菜供应链出现断裂，商超蔬菜供不应求，从而出现部分单品蔬菜被迫销售量下降的情况。然而部分蔬菜可能留有库存，消费者只能选择该类蔬菜，如西兰花、净藕 (1) 等，从而出现部分单品销售量骤升的情况。

4.5 蔬菜商品间的销售关联关系

4.5.1 各品类间的销售关联关系

不同品类蔬菜销量之间可能存在某种潜在联系，本文对各品类进行 DTW 多尺度相关性分析。

DTW 多尺度相关性分析

DTW（动态时间弯曲）是一种用于序列数据的距离度量[5]，它可以捕捉两个序列之

间的相似性，即使它们在时间上有些位移或速度上有所变化。多尺度相关性分析可以帮助我们了解不同时间尺度下类别与类别之间的关系。本文选取一个月（即 30 天）作为窗口大小，使用 DTW 分别计算不同品类、单品之间时间序列的 DTW 距离。然后使用层次聚类，可以通过 DTW 距离将各类别分组，然后使用树状图表示它们之间的关系。

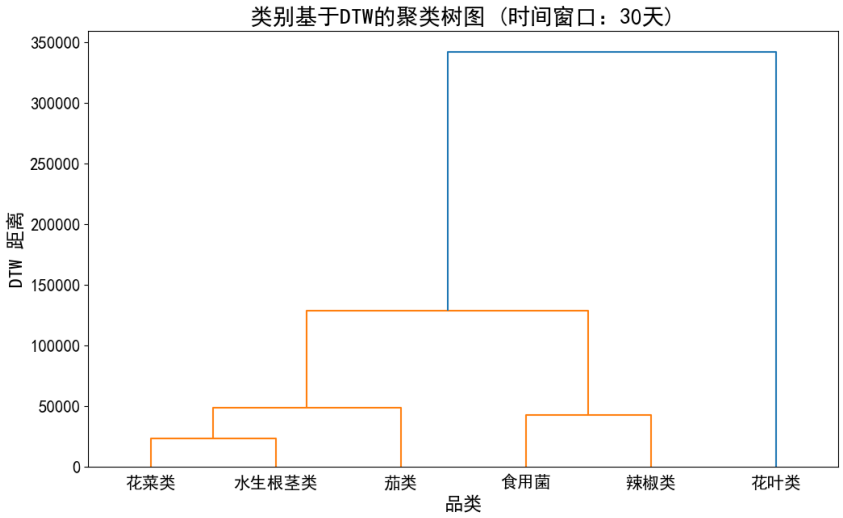

图 7　品类 DTW 分析结果图

由结果可以分析，水生根茎类、花菜类和茄类的销售量随时间序列的变化在某种程度上是相似的，可能存在某种共同的销售模式或趋势。

4.5.2 各单品间的销售关联关系

Apriori(关联规则挖掘) 算法常应用于零售与市场分析，用于寻找商品之间的关联性[3]，适用于本文分析各单品间的销售关联关系。并且，由于该算法采用的是前向逐层搜索的方法，这可以避免搜索整个组合空间，减小计算复杂度，在处理附件 2 这样的大规模数据集时具有较高的效率。因此，本文决定采用 Apriori 法来研究各单品间的销售量关联关系。

然而，依靠传统的 Apriori 仅能反映单品蔬菜间的购买关联性，无法反映销售量之间的相互关系，因此本文针对该题对 Apriori 稍作改进。具体实现步骤如下：

Step1 统计日销售量:

在统计不同单品的销售量时，若按扫码销售时间以时刻为单位，统计单品实时销售量，会出现计算资源消耗较高的问题，而且可能会生成大量无关或无用的的规则。

若以月或年为单位，统计单品月销售量或年销售量，会出现关联度高的两个单品实际销售时间相差较大，从而无法根据关联规则来反映销量间的相互关系。

因此本文决定以天为单位，分析每天的订单，统计各单品的日销售量。

Step2 归一化去量纲:

附件 2 中销量是通过售出的蔬菜重量进行衡量，但不同蔬菜的重量差异性较大，所以需要对各单品销量进行去量纲操作，本文利用如下公式对各单品的日销售量进行归一化处理。

$$y_{ij}^{'}=\frac{y_{ij}-\min\{y_i\}}{\max\{y_i\}-\min\{y_i\}}. \tag{1}$$

其中 x_{ij} 表示蔬菜品类 i 第 j 天的日销售总量，$i=1,2,3,4,5,6$ 分别代表花叶类、花菜类、水生根茎类、茄类、辣椒类、食用菌。

Step3 销量离散化：

销量是连续型数据，为将销量引入关联规则挖掘中，本文将销量离散化，转为二进制变量。将每个品类销量的中位数作为阈值，将大于该阈值的日销量记为 1，表示销售量较大；将小于该阈值的日销量记为 0，表示销售量较小。

Step4 关联规则挖掘：

通过在逐层搜索的迭代过程中不断链接和剪枝，找出数据集中的相继的关系以形成规则。从而，我们可以根据最终形成的规则来判断数据集中各项之间隐藏的属性之间的关系。

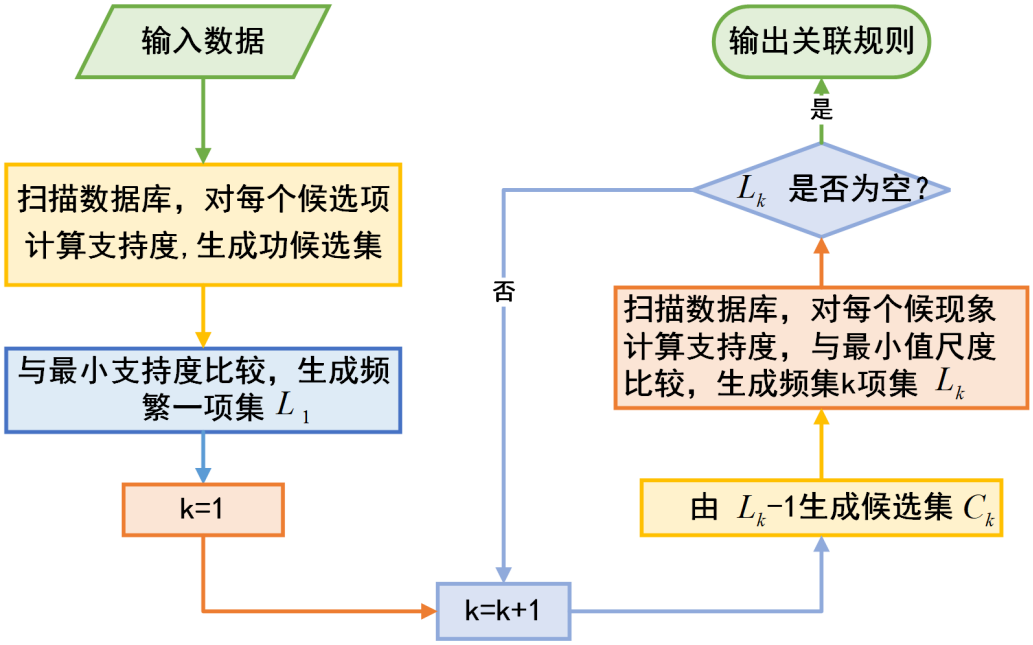

图 8　Apriori 算法具体步骤流程图

在上图中，生成频繁集的具体方式用伪代码表示如下。

Algorithm 1: Mining Regulation

Input: *D:dataSets;I:frequentSet;β:Confidence threhold;*

Output: *R:association rules;*

```
begin
    compute the non empty and proper subset of I: subSets;
    for S in subsets do
        if Confidence(S → I − S ≥)β then
            R+ = (S → I − S)
        end
    end
end
```

根据关联规则挖掘探究，本文筛选了关联规则结果中的频繁二项集作为分析对象，其中部分结果如下表所示，详细见附件 A，支持度越高，则两个单品同时出现的概率越大，则更能证明两个单品的存在一定的相互影响。根据下面的数据可以分析出西兰花和部分其他单品的关系，西兰花的出现，往往伴随着静藕 (1)、紫茄子 (2) 等单品蔬菜的同时出现。

表 1　频繁二项集信息表 (部分)

支持度	二项集
0.964976959	西兰花, 净藕 (1)
0.932718894	西兰花, 紫茄子 (2)
0.838709677	西兰花, 上海青
0.824884793	西兰花, 黄白菜 (2)
0.785253456	螺丝椒, 西兰花
0.780645161	芜湖青椒 (1), 西兰花
0.771428571	西兰花, 青茄子 (1)
0.749308756	西峡香菇 (1), 西兰花
0.717050691	云南油麦菜, 西兰花

5 问题二模型的建立与求解

5.1 问题二分析

问题二主要分为两小问。

针对第一小问，我们根据历史数据来分析销售总量和成本加成定价的关系。销售总量即是指当日补货量，由于过去补货量情况未知，本文假设过去处于每日所补均销的理想状态，将总销量近似为日补货量并分析定价策略，各品类蔬菜在新鲜期按成本加成定价法定价，损耗商品在定价基础上打折销售，因定价主要由加成比例决定，故对加成比例与销售总量的关系进行分析。

针对第二小问，要求以品类为单位，求取使未来一周收益最大的补货和定价策略。首先经过分析，欲使总收益最大，即保证每个品类的收益最大。然后，根据过去一个月的损耗量确定各品类的损耗率。接着，考虑到问题一各品类销售量具有周期性，本文采用 SARIMA 模型对未来一周的需求量进行预测。最后，分析不同日补货量与收益之间的函数关系，最后求得最优策略。

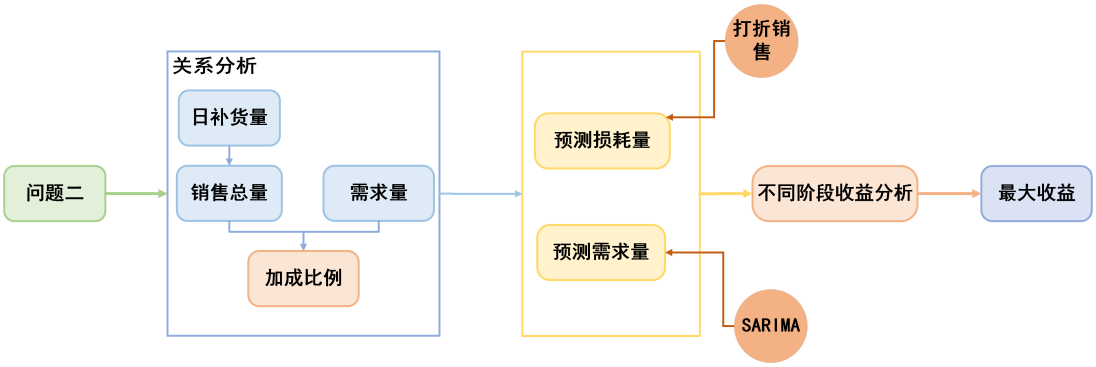

图 9　问题二思路图

5.2 销售总量与成本加成定价的关系分析

5.2.1 销售总量说明

销售总量指商超未来某一天打算销售的蔬菜量即当日的补货量，是商超需要规划的值。总销量指商超的当天的实际销售量，是不确定值，它由销售总量和需求量共同决定，并取其中的较小者，可表示如下

$$N_{ik} = \begin{cases} N_{ik}^{0}, N_{ik}^{0} \leq H_{ik}; \\ H_{ik}, H_{ik} \leq N_{ik}^{0}. \end{cases} \tag{2}$$

其中 N_{ik} 表示第 i 个品类在第 k 天的总销量；H_{ik} 表示第 i 个品类在第 k 天的日补货量；N_{ik}^{0} 表示第 i 个品类在第 k 天的需求量。

5.2.2 成本加成定价

根据题目，商超采用成本加成定价法进行定价[7] [8]，即在进货成本加上预期收益作为最终定价，可表示为

$$a = b \times (1 + \alpha)\,. \tag{3}$$

其中 a 指蔬菜商品定价，b 指蔬菜商品成本，α 指加成比例。

由于本题考虑商超按品类进行补货，基于上式，以品类作为单位, 定价与成本应满足如下关系：

$$A_{ik} = B_{ik} \times (1 + \alpha_{ik})\,. \tag{4}$$

其中 A_{ik} 指第 i 个品类的第 k 天的定价；B_{ik} 指第 i 个品类的第 k 天的成本；α_{ik} 表示第 i 个品类的第 k 天的加成比例。

5.2.3 损耗分析

由题目可知，蔬菜商品自身具有易变质性，属于易逝品，在运输过程和售卖过程中不可避免会出现损耗情况。附件 4 给出了商超通过近期盘点周期的数据计算得到的各单品蔬菜的损耗率。经分析，本文认为损耗率是指商品损耗量与日补货量的比值，但由于题目所给数据有限，我们无法详细得知以往各蔬菜商品的日补货量，因此本文假设过去商超销售处于理想状态，每天补货的蔬菜商品在当天可全部售出，即认为过去的日补货量等于总销量。从而，各蔬菜商品的以往损耗量可表示为

$$n_{ijk}^{'} = N_{ijk} \times \beta_{ij}. \tag{5}$$

其中 $n_{ijk}^{'}$ 表示第 i 个品类的第 j 个单品蔬菜在第 k 天损耗量；N_{ijk} 表示第 i 个品类的第 j 个单品蔬菜在第 k 天的总销售量；β_{ij} 表示第 i 个品类的第 j 个单品蔬菜的损耗率。

5.2.4 定价策略

为增加消费者对损耗商品的购买意愿以减少损失，商超会对损耗商品采取打折销售。因此，本文认为在销售过程中商超会将蔬菜商品分为两部分，分别采取不同的定价策略。

对于新鲜商品直接定价。

$$A_{ik}^{1} = \sum_{j=1}^{m_i} a_{ijk} \times \left(N_{ijk} - n_{ijk}^{'}\right) \tag{6}$$

对于已损耗商品将在定价基础上给予一定折扣。

$$A_{ik}^{2} = \sum_{j=1}^{m_i} a_{ijk} \times n_{ijk}^{'} \times \gamma_i \tag{7}$$

从而，每种品类蔬菜的定价可表示为

$$A_{ik} = A_{ik}^1 + A_{ik}^2 = \sum_{j=1}^{m_i} a_{ijk} \times N_{ijk} \times (1 - \beta_{ijk} + \beta_{ij} \times \gamma_i). \tag{8}$$

上式中 γ_i 表示第 i 个品类蔬菜的折扣；a_{ijk} 表示第 i 个品类的第 j 个单品的定价;b_{ijk} 表示第 i 个品类的第 j 个单品的成本;m_i 表示第 i 个品类包含的单品数量。

5.2.5 关系分析

在对蔬菜商品进行定价时，由于成本固定，定价是由加成比例决定的。下面对加成比例与销售总量之间的关系展开分析。

首先，根据上文可将加成比例表示为

$$\alpha_{ik} = \frac{A_{ik}}{B_{ik}} - 1, \tag{9}$$

即

$$\alpha_{ik} = \frac{\sum_{j=1}^{m_i} \left[a_{ijk} N_{ijk} \left(1 - \beta_{ij} + \beta_{ij} \times \gamma_i\right)\right]}{\sum_{j=1}^{m_i} b_{ijk} H_{ijk}} - 1. \tag{10}$$

其中 b_{ijk} 表示第 i 个品类的的第 j 个单品的在第 k 天的成本。

由于本问题要求以品类为单位，将上面公式转化为

$$\alpha_{ik} = \frac{\bar{a}_{ik} N_{ik} \left(1 - \bar{\beta}_{ik} + \bar{\beta}_{ik} \times \gamma_i\right)}{\overline{b_{ik}} H_{ik}} - 1. \tag{11}$$

其中 $\bar{a}_{ik}$ 表示第 i 个品类在第 k 天的定价；$\bar{b}_{ik}$ 表示第 i 个品类在第 k 天的成本。

当 $N_{ik}^0 \leq H_{ik}$ 即销售总量小于等于需求量时，补货的蔬菜可全部售出，最终总销量等于开始的销售总量。通过公式可以看出，此时增加销售总量，加成比例会减小。

当 $N_{ik}^0 > H_{ik}$ 即销售总量大于需求量时，补货的蔬菜不能全部售出，最后会有部分蔬菜被丢弃的情况，此时总销量等于需求量。经分析公式，改变销售总量加成比例不会发生变化，为恒定值 $\frac{\bar{a}_{ik}\left(1-\bar{\beta}_{ik}+\bar{\beta}_{ik}\times\gamma_i\right)}{\overline{b_{ik}}}$.

5.3 最大收益策略模型

5.3.1 收益目标函数

根据题目，希望商超收益最大化，即要求六种品类蔬菜的 2023 年 7 月 1-7 日总收益最大, 目标函数可写为

$$\max S = \sum_{k=1}^{7} \sum_{i=1}^{\delta} S_{ik}. \tag{12}$$

其中 S 表示商超在 2023 年 7 月 1-7 日的总收益;S_{ik} 表示第 i 个品类蔬菜在第 k 天带来的收益。

由于蔬菜商品根据成本加成定价法确定销售价格，商超的收益即蔬菜商品加成部分。每种品类蔬菜的收益可表示为

$$S_{ik} = A_{ik} - B_{ik} = B_{ik}\alpha_{ik}. \tag{13}$$

若使商超未来一周内总收益最大，只需保证每种品类蔬菜每天的的收益最大即可，即要求 S_{ik} 最大。

5.3.2 损耗量预测

本文将根据以往损耗量预测 2023 年 7 月 1-7 日的每个品类蔬菜的损耗量。由于可能存在突发因素的影响，部分数据反映是存在不同年份在相同时间的损耗量相差较大的情况。因此为了预测的准确性，减小无关因素的影响，我们利用临近时间的数据作为参考进行预测。

本文最终选取了 2023 年 6 月 1 日-30 日的流水数据作为参考。附件 4 数据是由近期盘点周期的数据得到，因此本文该数据具有普适性，可用于前期销售中损耗的计算。基于上文，本文假设过去处于损耗量进行打折销售且均可售完的理想状态，损耗量即为为损耗率与当天总销量的乘积。据此，本文计算出过去一个月各品类蔬菜的总损耗量，并利用如下公式计算未来一周的损耗率。

$$\beta_i^{'} = \frac{\sum_{k=1}^{30} n_{ijk}^{'}}{\sum_{k=1}^{30} N_{ik}} \tag{14}$$

从而得到未来一周的各品类蔬菜的损耗率为

表 2　各品类蔬菜的损耗率

分类名称	花叶类	花菜类	水声根茎类	茄类	辣椒类	食用菌
损耗率	9.366%	10.536%	8.796%	6.285%	1.778%	16.062%

5.3.3 确定损耗商品价格折扣

各蔬菜品类未来一周定价仍采用以往策略，对损耗蔬菜进行打折处理。本文将根据历史数据确定各品类蔬菜的折扣。

通过分析附件 2 不同品类蔬菜的打折前后的销售单价变化，本文发现同一天的折扣相同，但不同时间不同品类蔬菜的折扣不尽相同。本文选取出现频率最高的折扣作为参考，将 2023 年 7 月 1-7 日的各品类蔬菜的价格折扣设定如下。

表 3　各品类蔬菜的价格折扣

分类名称	花叶类	花菜类	水生根茎类	茄类	辣椒类	食用菌
价格折扣	0.6	0.5	0.5	1	0.6	0.6

5.3.4 基于 SARIMA 预测未来需求量

本文将根据以往总销量预测各品类蔬菜的未来需求量。根据问题一的各品类间的销售量的分布规律，我们可知各品类蔬菜的销售量可能具有季节性并含有长期趋势效应，因此本文决定采用 SARIMA(Seasonal Autoregressive Integrated Moving Average，季节性差分自回归滑动平均模型) 对 2023 年 7 月 1-7 日的需求量进行预测。

•ADF 检验：

由于 SARIMA 模型通常用于处理平稳的时间序列[4]，在进行预测前，我们需要对各品类蔬菜关于日销售量的时间序列进行平稳性检验。本文采用 ADF 检验法，通过判断一个时间序列是否具有单位根来检验是否平稳。ADF 检验的原假设是时间序列具有单位根即非平稳性，备择假设是时间序列不具有单位根即平稳性。

经过求解得到各品类蔬菜的 ADF 检验的 p 值如下所示。可以看出，六类蔬菜的 ADF 检验 p 值均小于 0.1，说明高于 90% 的把握拒绝原假设，序列均平稳，无需进行差分操作。

表 4　各品类蔬菜的 ADF 检验的 p 值

分类名称	花叶类	花菜类	水生根茎类	茄类	辣椒类	食用菌
p 值	0.0555600	0.0244632	0.0263780	0.0008465	0.0051945	0.0066066

•ARIMA 模型求解：

该模型是由 ARIMA 模型的基础上改进而来，通过引入季节性差分即将时间序列与其某个固定滞后期的值相减，来去除季节性变动。通过引入季节性自回归（SAR）和移动平均（SMA）的部分，同时考虑了季节性差分和非季节性差分，从而使得 SARIMA 模型能够适应具有季节性模式的数据。

首先，对各品类蔬菜关于总销量的时间序列 $\{N_t\}$ 进行 D 次季节差分来去序列季节性，一次季节差分可表示为

$$\Delta s \cdot N_t = (1 - L^s)N_t = N_t - N_{t-s}. \tag{15}$$

其中 s 表示季节周期的长度，根据问题一的季节性分析 s 取值为 12 个月；Δs 表示季节

差分算子；L 表示滞后算子。

从而经过 D 次季节差分, 建立关于周期为 s 的 P 阶自回归 Q 阶移动平均季节时间序列模型为

$$A_P(L^s)\Delta s^D N_t = B_Q(L^s)u_t. \tag{16}$$

其中 P 表示季节性自回归阶数，$A_P(L^s)$ 表示季节自回归多项式，Q 表示季节性移动平均阶数，$B_Q(L^s)$ 表示季节移动平均多项式，u_t 表示经过季节差分处理后的时间序列。

然后，再进行 d 次非季节差分来去序列趋势性得到

$$\varphi_p(L)\Delta^d u_t = \Phi_q(L)v_t. \tag{17}$$

其中 p 表示自回归阶数，$\varphi_p(L)$ 表示非季节自回归多项式，q 表示移动平均阶数，$\Phi_q(L)$ 表示非季节移动平均多项式，v_t 是误差项，是一个高斯白噪声序列。

最后，得到差分后的序列，并嵌入外部回归器，建立如下 SARIMA 模型

$$\varphi_p\left(L\right)A_p\left(L^s\right)\left(\Delta^d\Delta s^D N_t\right) = \Phi_q\left(L\right)B_Q\left(L^s\right)v_t. \tag{18}$$

其一般表示式可写作

$$\text{SARIMA}(p,d,p)\times(P,D,Q)_s. \tag{19}$$

•ACF 和 PACF 图像确定模型参数

本文通过利用 $\{N_t\}$ 的时间序列自相关函数图 (ACF) 与偏自相关函数图 (PACF) 确定 SARIMA 模型的移动平均阶数 q 和自回归阶数 p. 根据结果，最终将时间序列的 SARIMA 的基本模型确定为 $\text{SARIMA}(1,1,1)\times(1,1,0)_{12}$.

• 模型灵敏度检验

由于现实情况受到多方面的外部因素影响, 导致本文预测模型得到的预测值和真实值不可避免地存在偏差。考虑到 SARIMA 的预测结果将影响最大收益策略模型的效果, 因此我们引入误差区间和风险概率，表示误差在该区间内，模型仍能给出优化的策略, 并且随着误差的增大，模型效果会随之下降，若误差超过此区间，模型给出的策略将存在亏本或者不能优化的风险。

本文取 2023 年 4 月 1 日至 2023 年 6 月 30 日的数据作为验证集，采用滑动窗口的方式，每次预测七天的数据，通过计算预测利润与真实利润的残差作为衡量预测误差的指标，接下来使用 95% 的置信区间作为初始误差区间，计算这段时间内误差超过区间的概率作为风险概率。然后逐步缩小置信区间大小，得到对应的风险概率最终计算得到的结果如下表所示。

表 5　（2023 年 6 月 24 日至 6 月 30 日）模型灵敏度分析图

品类	95% 的置信区间	70% 的置信区间	60% 的置信区间	40% 的置信区间
水生根茎类	0.1429	0.2857	0.1429	0.8571
花叶类	0.1429	0.2857	0.2857	1.0000
花菜类	0.2857	0.2857	0.1429	0.5714
茄类	0.1429	0.2857	0.2857	0.8571
辣椒类	0	0.1429	0.2857	0.5714
食用菌	0.1429	0.1429	0.2857	1

由上表可知，在置信区间较小的时候，模型的风险概率较小。综上所述，本文建立的最大收益策略模型效果较好，能为商超提供优化收益的策略。

- **预测结果分析**

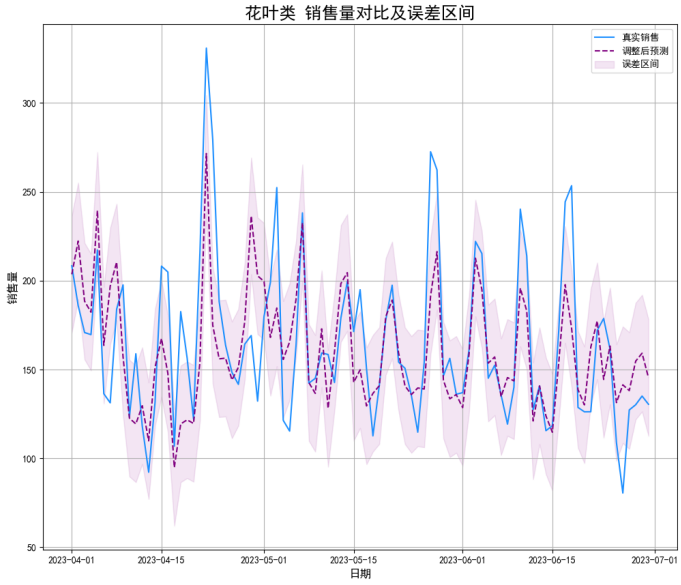

图 10　花叶类需求量预测图

最终，本文预测出了 2023 年 7 月 1 日至 7 月 7 日的各品类蔬菜的总需求量，其中花叶类的预测结果如上图所示，其他品类的预测结果见附件 A。

5.3.5 不同阶段的收益分析

通过上述对销售总量与成本加成定价的关系的分析，加成比例受日补货量变化影响，而加成比例决定收益大小。于是，本文对日补货量的改变对收益的影响展开研究，并依据收益变化情况分为三个阶段进行分析。

(1) 当 $H_{ik} \leq N_{ik}^0$ 时，出现供不应求情况，所有蔬菜商品均可售出，此时商超的收益为

$$S_{ik} = \left[\bar{a}_{ik}\left(1 - \bar{\beta}_{ik} + \bar{\beta}_{ik} \times \gamma_i\right) - \bar{b}_{ik}\right] \times H_{ik}. \tag{20}$$

在此范围内，增加日补货量会填补消费需求，商超收益增大。

(2) 当 $N_{ik}^0 < H_{ik} \leq \frac{N_{ik}^0}{1-\bar{\beta}_{ik}}$ 时，供应可满足需求，考虑到新鲜蔬菜相比已损耗蔬菜带来的收益更大，商超会优先销售新鲜蔬菜，但在该阶段仅供应新鲜蔬菜无法满足需求，会有已损耗蔬菜打折销售的情况，此时商超的收益为

$$S_{ik} = \left[\bar{a}_{ik}\left(1 - \bar{\beta}_{ik}\right)\left(1 - \gamma_i\right) - \bar{b}_{ik}\right] \times H_{ik} + N_{ik}^0 \bar{a}_{ik}\gamma_i. \tag{21}$$

在此范围内，再增加日补货量，新鲜蔬菜会顶替已损耗的蔬菜销售，尽管收益改变相比上一阶段较小，但仍会继续增加。

(3) 当 $H_{ik} > \frac{N_{ik}^0}{1-\bar{\beta}_{ik}}$ 时，仅销售新鲜蔬菜即可满足需求，此时商超的收益为

$$S_{ik} = \bar{a}_{ik}N_{ik}^0 - \bar{b}_{ik} \times H_{ik}. \tag{22}$$

在此范围内，继续增加日补货量，徒增无法售出多余的蔬菜量，商超收益减小。

综上分析，收益随日补货量的变化趋势如下图所示。当日各品类供应量为 $\frac{N_{ik}^0}{1-\bar{\beta}_{ik}}$ 时，商超将获得最大收益。

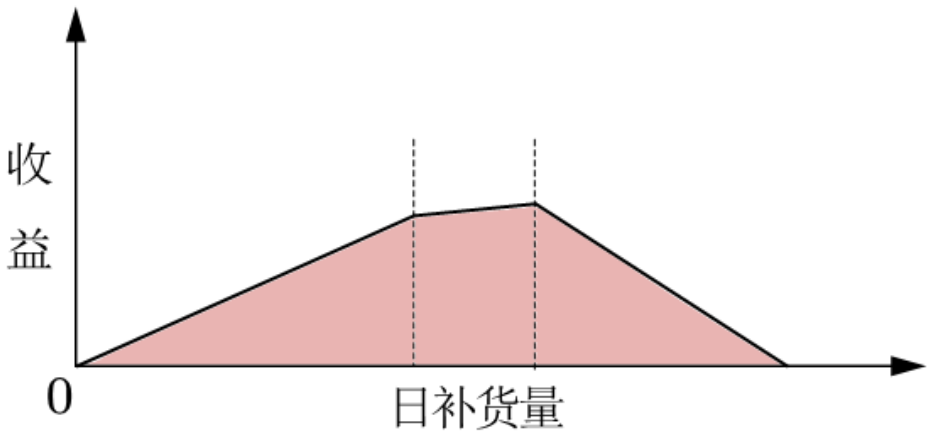

图 11　收益随日补货量的变化趋势图

最终，可以计算出 2023 年 7 月 1 日至 7 月 7 日的每个品类蔬菜的日补货量和最优加成比例，以及根据预测值计算得到的总收益，2023 年 7 月 1 日的品类补货定价策略信息如下表，其他详细结果见附件 B。

表 6　品类补货定价策略表 (7 月 1 日)

	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌
α(加成比例)	0.00041	0.25599	0.34253	0.52782	0.59058	0.32371
7 月 1 日补货量	14.4369	146.7712	23.8718	23.1992	82.1595	48.6912
各品类收益	16.3423	183.2859	84.8041	55.2195	171.9067	99.2694

由表的结果可知，在 7 月 1 日的总收益为 **610.8277** 元，但由于问题二没有考虑到具体单品以及打折对需求量的影响，补货定价策略还可以进一步的优化改进。

6 问题三模型的建立与求解

6.1 问题三分析

在问题二的基础上，问题三要求以单品为单位制定使收益最大化的补货定价策略。我们进行更进一步更细致的规划。首先，考虑到消费者低价购买偏好，我们在模型中引入价格敏感系数和需求奖励度，并设置蔬菜单品是否补货的 0-1 变量。然后，根据题目加入可售单品数、单品陈列数约束，并考虑到问题二结果，设置最佳日补货量约束。经分析，收益由各单品的日补货量和单位补货量收益共同决定，据此，通过贪心算法，建立双层结构，求得最优补货定价策略。

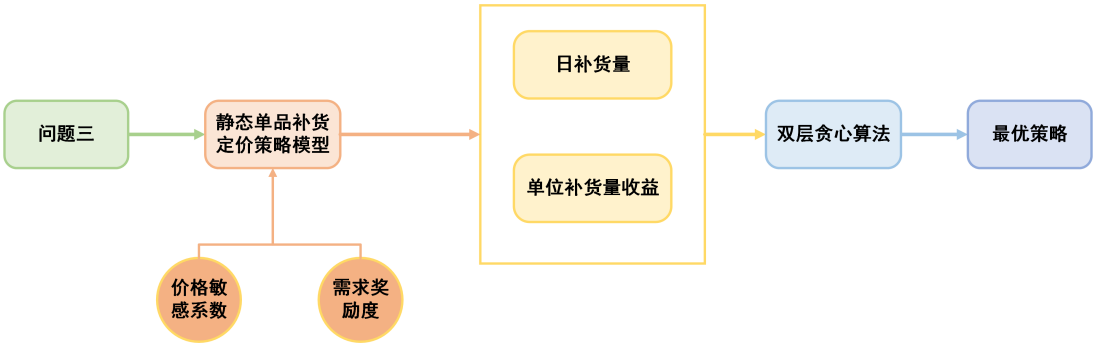

图 12　问题三思路图

6.2 静态单品补货定价策略模型

6.2.1 决策变量

(1) 若当天对某单品进行补货，则认为该单品可售；若当天某单品未补货，假设没有库存，则认为该单品不可售。为表示某单品蔬菜当天是否可售，本文设置 0-1 变量，

其具体含义如下：

$$x_{ij}=\begin{cases}0，第i品类的第j个单品蔬菜不可售\\1，第i品类的第j个单品蔬菜可售\end{cases} \tag{23}$$

(2) 考虑到本题与问题二不同，补货时是以单品为单位，考虑得更加具体化，因此我们在规划销售时也可以更加详细化，加入消费者价格对价格变化的敏感程度的考虑[9]。

当价格较高时，人们的购买量会下降，从而导致想需求量下跌；当较低价格时，人们的购买意愿会增大，从而会导致需求量上涨。对此，本文引入价格敏感系数 w_{ij}。举例说明，当价格敏感系数为 0.7 时，需求量变为原先的 0.7。

经查资料并结合数据分析，本文将不同商品销售单价对应的消费者价格敏感系数规定如下。

表 7　价格敏感系数规定

销售单价 (元/千克)	价格敏感系数
0∼5	1
6∼14	0.9
15 及以上	0.7

(3) 当对损耗的商品进行打折时，考虑到会存在人们低价购买偏好的的情况，总体需求量可能会由于价格折扣而增加。于是本文将引入需求奖励度 z。需求奖励度会受折扣的大小的影响，一般来说折扣越大，需求奖励度越大。经查阅资料，将需求奖励度表示如下。

$$z_{ij}=1-\gamma_{ij}. \tag{24}$$

其中 z_{ij} 表示第 i 个品类的第 j 个单品的需求奖励度。

6.2.2 目标函数

本文要求 7 月 1 日收益最大化，目标函数可表示为

$$\max S'=A'-B' \tag{25}$$

定价：

在上式中，总售价 A' 考虑原价和折扣价两部分，可表示为

$$A^{'}=\sum_{i=1}^{\delta}\sum_{j=1}^{m_i^{'}}A_{ij}=A^{1'}+A^{2'}. \tag{26}$$

其中 $A^{1'}$ 表示以原价销售的蔬菜商品的总售价，$A^{2'}$ 表示打折销售的蔬菜商品的总售价。

对新鲜商品按原价进行销售，并考虑价格敏感系数，则 $A^{1'}$ 可表示为

$$A^{1'} = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i'} H_{ij} x_{ij} \left[\left(1 - \beta_{ij} \right) w_{ij} \right] a_{ij} \tag{27}$$

综合考虑价格敏感系数和需求奖励度，存在部分商品因定价过高，销售量大幅减小，商超为减少损失，对此部分商品，将与损耗商品一同进行打折销售。则 $A^{2'}$ 可表示为

$$A^{2'} = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i'} H_{ij} x_{ij} \gamma_{ij} z_{ij} \left[\left(1 - \beta_{ij} \right) \left(1 - w_{ij} \right) + \beta_{ij} \right]. \tag{28}$$

成本：

总成本 B 可表示为

$$B' = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i'} B_{ij}' = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i'} b_{ij} H_{ij} x_{ij}. \tag{29}$$

6.2.3 约束条件

• 日补货量约束：

联系问题二，定价实际上主要是由加成比例决定的，而加成比例受日补货量影响。因此，本文需考虑日补货量约束。结合问题二结果，当各品类的日补货量满足如下约束时，既可满足市场对各品类蔬菜商品需求，也可满足的收益最大化。

$$\sum_{j=1}^{m_i'} H_{ij} x_{ij} = \frac{N_{i\max}^{0}}{1 - \bar{\beta}_{ik}} \tag{30}$$

• 可售单品总数约束:

根据题目要求，用于销售的单品总数需要控制在 27 至 33 个范围内，故可作如下约束:

$$27 \leq \sum_{i=1}^{\delta} \sum_{j=1}^{m_i'} x_{ij} \leq 33 \tag{31}$$

其中 m_i' 表示在 2023 年 6 月 24-33 日的可售品种中第 i 个品类包含的单品数量。

• 单品陈列量约束:

根据题目要求，每个蔬菜单品的订购量要求不能大于最小陈列量即 2.5 千克, 故可作如下约束:

$$H_{ij} x_{ij} \geq 2.5 x_{ij} \tag{32}$$

6.2.4 模型总结

综上，单价补货定价策略模型为

$$\max S' = A' - B'. \tag{33}$$

$$\begin{cases} A' = A^{1'} + A^{2'} \\ A^{1'} = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i^{'}} H_{ij} x_{ij} \left[\left(1 - \beta_{ij} \right) w_{ij} \right] a_{ij} \\ A^{2'} = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i^{'}} H_{ij} x_{ij} \gamma_{ij} z_{ij} \left[\left(1 - \beta_{ij} \right) \left(1 - w_{ij} \right) + \beta_{ij} \right] \\ B^{'} = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i^{'}} b_{ij} H_{ij} x_{ij} \\ \sum_{j=1}^{m_i^{'}} H_{ij} x_{ij} = \frac{N_{i\max}^{0}}{1-\beta_{ik}} \\ 27 \leq \sum_{i=1}^{\delta} \sum_{j=1}^{m_i^{'}} x_{ij} \leq 33 \\ H_{ij} x_{ij} \geq 2.5 x_{ij}. \end{cases} \tag{34}$$

6.3 双层贪心算法求解

6.3.1 题目条件分析

基于上述模型，可将收益目标函数化简如下

$$S = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i^{'}} H_{ij} x_{ij} \left\{ \left(1 - \beta_{ij} \right) w_{ij} a_{ij} + \gamma_{ij} z_{ij} \left[\left(1 - \beta_{ij} \right) \left(1 - w_{ij} \right) + \beta_{ij} \right] - b_{ij} \right\}. \tag{35}$$

假设

$$S = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i^{'}} H_{ij} x_{ij} D_{ij} \tag{36}$$

其中 D_{ij} 表示 i 个品类的第 j 个单品的表示单位补货量收益。

本题要求收益最大化，收益主要是由日补货量和定价共同决定的。但若将定价作为决策变量进行规划，容易出现最终求得的定价不切实际的情况，相比以往定价情况该日发生突变。现实看来，这种情况是不能存在的，所以不能对定价进行决策规划。

根据题目是基于 2023 年 6 月 24-30 日预测 7 月 1 日的数据，因此本文将 6 月 24-30 日的单品销售单价的平均值作为 7 月 1 日的定价，并将出现频率最大的折扣作为对用的单品的折扣。

在已知单品定价和折扣的情况下，价格敏感系数与需求奖励度取值确定，从而每个单品的单位补货量收益固定。需要进行决策的变量仅有每种单品的日补货量，当日补货量为 0 时指该单品不可售。

6.3.2 双层贪心算法求解

本文采用贪心算法决策每种单品的日补货量。贪心算法的原理是在每一步选择中都采取当前状态下的最佳选择，即通过局部最优选择来构建整体最优解。

本文的贪心满意目标为收益最大化。通过上述分析，收益由每个单品的日补货量和单位日补货量收益共同决定，对此，架构双层贪心如下。

外层贪心：

整体上，经计算，发现各单品的单位日补货量收益有正负之分，即销售某种商品可能会给商超带来利润，也可能带来亏损。基于贪心，优先选择为商超产生正收益的单品补货，跳过产生亏损的单品。经统计，2023 年 6 月 24-30 日的可售品种中涵盖 49 种单品。经过外层贪心后，共筛选得到 33 中正收益单品，也满足可售单品总数约束。并且，该 33 种单品中涵盖 6 个品类，因此也保证了商超蔬菜品类的多样性。

内层贪心：

由于日补货量约束，每种品类的日补货量固定。基于贪心，在满足单品陈列量约束条件下，优先选择一个蔬菜品类中单位收益大的单品补货。尽管可能会存在一个品类中的单品数量不均衡的情况，但消费者可以选择购买该品类的销售总量大的其他单品来代替，一定程度上也可以满足需求。

6.4 结果分析

基于双层贪心算法，本文初步筛除了 15 个会产生负面收益的单品，对剩余的 33 个会产生正面收益的单品蔬菜进行补货，并且剩余单品满足品类的多样性，保证每种品类蔬菜都有补货。然后在满足其他单品陈列量的约束条件下对产生正面收益最高的单品分配该品类的最大日补货量额度。7 月 1 日正面收益较高的几个单品的补货定价策略信息如下，详细结果可见附件 D。

表 8　单品补货定价策略 (部分)

单品	定价 (元/千克)	价格敏感度	打折	打折需求奖励度	日补货量 (kg)
红莲藕带	9.20	0.9	-	0	12.1
木耳菜 (份)	27.30	0.7	6 折	0.4	101.3
西兰花	12.63	0.7	-	0	21.6
紫茄子 (1)	31.50	0.7	-	0	11.7
小米椒 (份)	5.80	0.9	6 折	0.4	59.9
白玉菇 (袋)	25.55	0.7	6 折	0.4	27.5

最终算得 7 月 1 日的总收益为 **1109.8847** 元，与问题二的收益结果相比有一定的增加，进一步说明单品补货定价策略对提高收益的价值。

7 问题四模型的建立与求解

7.1 问题四分析

问题四要求给出其他相关数据来更好的制定蔬菜商品的补货定价决策。联系实际，本问题增加了对平均每天不同时段商超的人流量和单品购买次数、平均每天不同时段单品的剩余比例、不同保鲜方式的成本信息和损耗率信息、每个单品近期的供应上限四项数据的考虑。在问题三动态策略模型的基础上，增加单品供应上限约束，考虑损耗率和成本与保鲜方式的函数关系，据此建立动态策略模型，以求解最优补货定价决策。

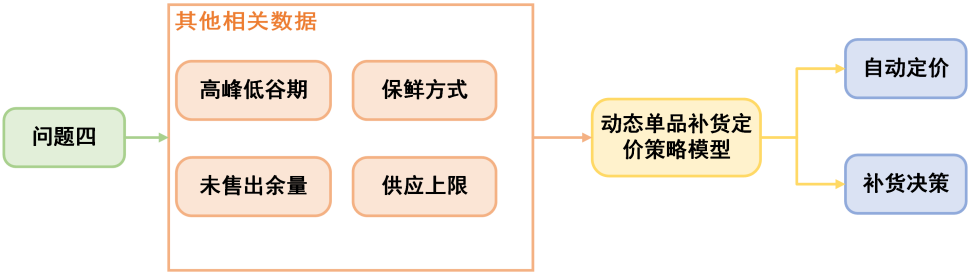

图 13　问题四思路图

7.2 数据采集建议

平均每天不同时段商超的人流量和单品购买次数

针对不同时段的人流量和单品购买次数信息可以分析购买高峰期和低谷期，可以进一步对当日的定价分时段实现动态调整。

平均每天不同时段单品的剩余比例

根据单品的剩余情况，可以进行实现定价的动态规划，当剩余比例在某个时段之后不再发生变化，可以认为剩余比例为未售出的余量，针对这种情况可以根据时段进行价格的动态调整。

不同保鲜方式的成本信息和损耗率信息

针对每种不同变质特性的蔬菜在不同时间的补货售卖过程中采取不同的保鲜方式可能会导致不同损耗率和成本，比如低温保鲜和洒水保鲜往往会增加成本，但是对应的蔬菜损耗率也会降低。

每个单品近期的供应上限

基于问题三的结果分析，由于贪心策略里并没有对供应量上限的约束，故每个单品的供应量存在过量分配的问题，但根据实际情况每个单品并不能无限供应，若每个单品近期的和供应上限信息已知，便可对策略规划策略进行改进。

7.3 动态单品补货定价策略模型

基于上面的信息数据，可以对问题三的静态策略规划模型进行优化，建立动态策略规划模型[9]，改进公式如下。

$$S'' = \sum_{i=1}^{\delta} \sum_{j=1}^{m_i} H_{ij} x_{ij} \left\{ (1-\beta_{ij})\, w_{ij} a_{ij} + \int_0^T r_{ij}(t)\, z_{ij} \left[(1-\beta_{ij})(1-w_{ij}) + \beta_{ij} \right] dt - b_{ij} \right\} \tag{37}$$

$$b_{ij} = f(\mu_{ij}) \tag{38}$$

$$\beta_{ij} = g(\mu_{ij}) \tag{39}$$

$$2.5 x_{ij} \leq H_{ij} x_{ij} \leq H_{ij\,\max} x_{ij} \tag{40}$$

r_{ij} 此时是随时间变化的动态折扣，μ_{ij} 表示第 i 个品类第 j 个单品对应的保鲜方式，b_{ij} 和 β_{ij} 受保鲜方式的影响，$H_{ij\,\max}$ 则是每个单品的补货上限。

7.4 结果分析

在引入了动态折扣、保险方式和补货上限这 3 个条件后，策略的模型变得更加灵活，考虑更加全面，所求得的补货定价策略也更接近真实情况。商超通过采集了上面的建议的数据，可以更准确的去进行补货定价，为商场的经营提供了更大的帮助，

8 模型总结与评价

8.1 模型总结

问题一从统计角度分析销量间分布规律，针对相关情况采用 DTW 多尺度相关性分析，并采用 Apriori 算法挖掘关联关系。问题二采用 SARIMA 算法预测销售总量，然后根据销售总量与成本加成定价关系分阶段建立最大收益策略模型最大化收益。问题三建立静态单品补货定价策略，设计双层贪心算法求解最优策略。问题四提出数据采集建议，并假设数据集更完善的情况下建立动态策略优化模型灵活解决定价补货问题。

8.2 模型优点

1) 问题一采用多角度分析的方法分析品类和单品之间的关联性，并对关系做出量化处理，为后续的定价补货模型提供数据支撑，使策略模型更贴近真实情况。

2) 问题二的预测模型结合问题一的分析结论，引入品类间关系和外部影响因素，使预测结果更贴近真实情况。

3) 问题二分阶段考虑供需关系，为预测模型存在的误差提供容错空间，经验证，分阶段的最大收益策略模型在预测存在误差风险的条件下仍能得出较好收益最大划策略。

4) 问题四的动态策略优化模型进一步验证了数据采集建议的合理性，并为模型推广到更复杂情况做出考量。

8.3 模型的缺点及改进

1) 问题二和问题三忽略商品的日补货上限，这导致模型求解得到的策略可能偏离市场需求，但问题四中提出建议并作出假设，可以进一步完善和优化模型。

2) 由于数据量较大，模型所涉及算法耗时较长，后续可以通过设计启发式算法，对算法做出约束，从而提高算法效率。

8.4 模型推广

本文基于某商超的数据分析，建立静态单品补货定价策略模型，并通过第四问进一步完善模型，得到动态策略优化模型，若能收集到更完善和多维化数据集，模型将能在复杂的实际情况下提出建设性意见。并且本文建立的模型尽可能多的考虑实际影响因素和外部异常影响，得到的策略更加贴合实际，具有一定地参考价值。经后续不断完善和推广，模型得到的策略，可为解决商超与消费者间供需不平衡问题提供具有可解释性的建议，也可以更好的满足市场需求，同时优化商超的收益。

参考文献

[1] Herbon A , Khmelnitsky E .Optimal dynamic pricing and ordering of a perishable product under additive effects of price and time on demand[J].European Journal of Operational Research, 2016, 260(2):546-556.

[2] [1]Li J, Yin C, Wang H, et al. Mining Algorithm of Relatively Important Nodes Based on Edge Importance Greedy Strategy[J]. Applied Sciences, 2022, 12(12): 6099.

[3] Mashud M. Designing an Application for Analyzing Consumer Spending Patterns Using the Frequent Pattern Growth Algorithm[J]. Jurnal Penelitian Pos dan Informatika, 2019, 9(2): 151-159.

[4] Sirisha U M, Belavagi M C, Attigeri G. Profit prediction using Arima, Sarima and LSTM models in time series forecasting: A Comparison[J]. IEEE Access, 2022(10): 124715-124727.

[5] Hussein E,Thomas L,Pierre G, et al. Constrained DTW preserving shapelets for explainable time-Series clustering[J]. Pattern Recognition,2023,143: 109804.

[6] 闻卉, 许明辉, 陶建平. 考虑绿色度的生鲜农产品供应链的销售模式与定价策略 [J]. 武汉大学学报 (理学版),2020,66(05):495-504.

[7] 张泽公, 龙彦天. 数据挖掘在“薄利多销”销售策略中对折扣力度与商品销售额以及利润率的关系的研究 [J]. 商场现代化,2020(10):4-6.

[8] 陈智超, 张晓林. 动态价格下生鲜农产品供应链合作博弈分析 [J]. 天津农业科学, 2023, 29(2):74-79.

[9] 毕文杰, 周玉冰. 基于深度强化学习的生鲜产品联合库存控制与动态定价研究 [J]. 计算机应用研究,2022,39(09):2660-2664.

附录 A　问题二各品类需求量预测结果

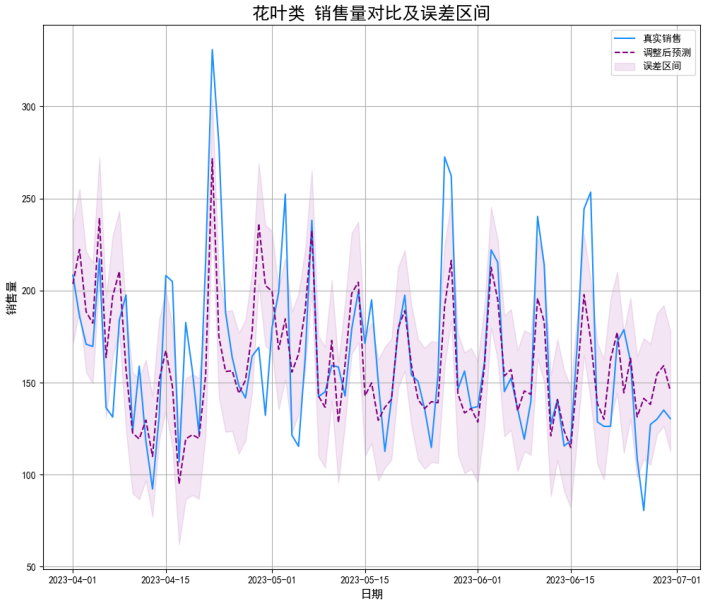

图 14　花叶类需求量预测图

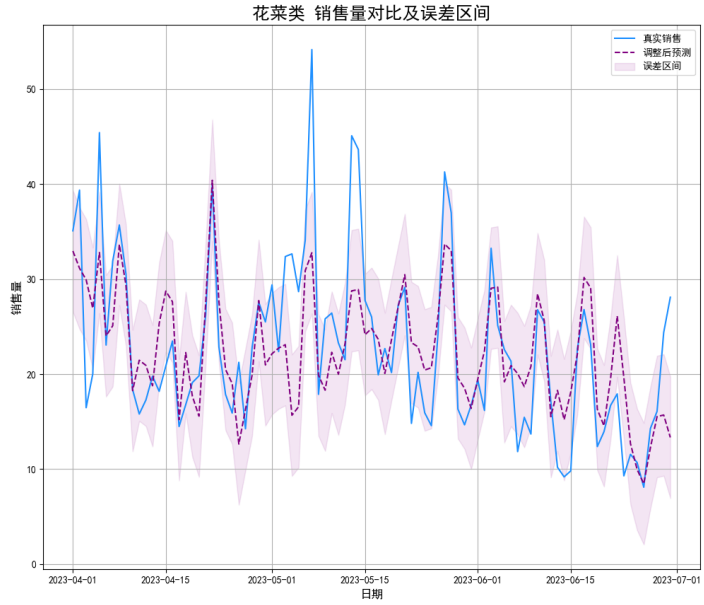

图 15　花菜类需求量预测图

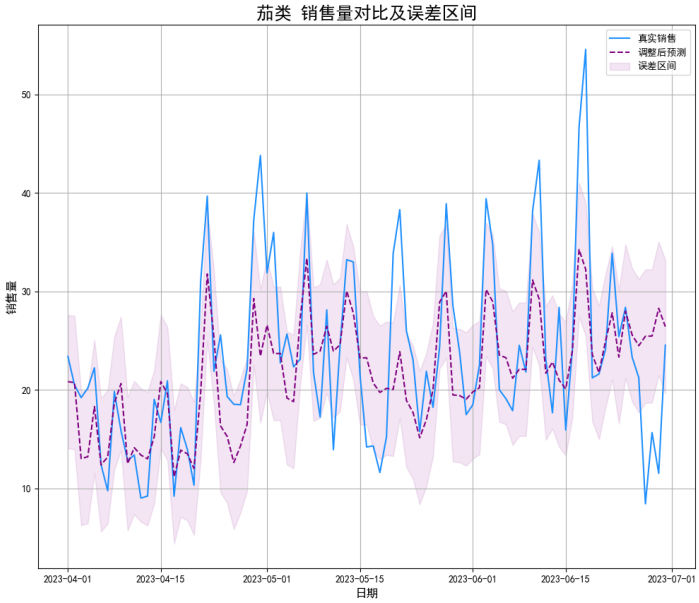

图 16　茄类需求量预测图

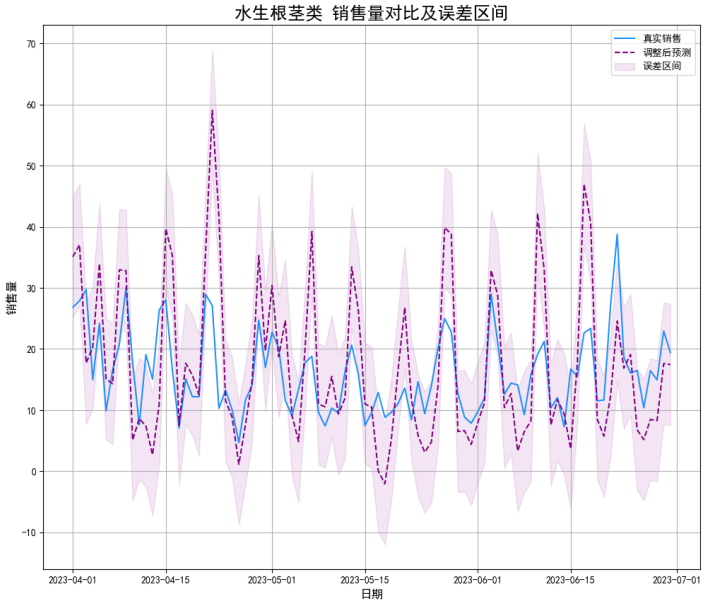

图 17　水生根茎类需求量预测图

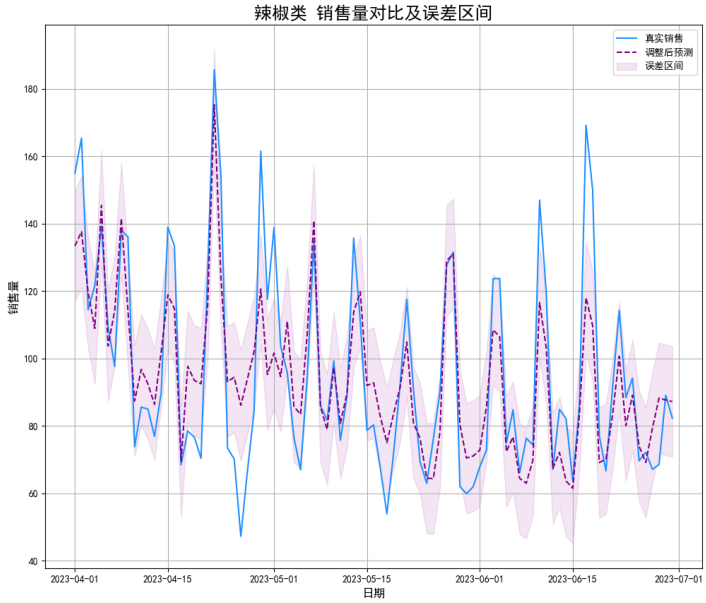

图 18　辣椒类需求量预测图

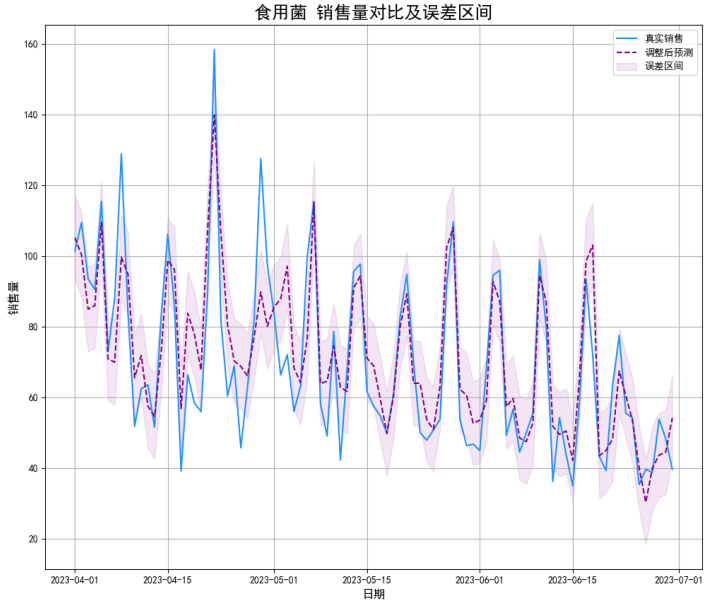

图 19　食用菌需求量预测图

附录 B　问题二品类补货定价策略表

	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌
α(加成比例)	0.00041	0.25599	0.34253	0.52782	0.59058	0.32371
7 月 1 日补货量	14.4369	146.7712	23.8718	23.1992	82.1595	48.6912
7 月 2 日补货量	16.1842	156.7872	19.2676	22.1445	78.3736	54.1423
7 月 3 日补货量	17.8307	151.3970	19.0757	21.6787	79.7099	56.9773
7 月 4 日补货量	15.5960	145.9873	21.4592	21.7020	81.3168	52.9486
7 月 5 日补货量	13.7828	152.4035	19.3834	20.6994	75.9813	51.1189
7 月 6 日补货量	18.4952	145.4142	17.6481	21.9264	80.4915	53.4021
7 月 7 日补货量	17.8810	154.9391	16.2827	21.9973	82.4378	54.5803
平均收益	18.4684	187.9779	69.5212	52.1432	167.5289	108.3046

附录 C　问题 3 单品补货定价策略信息表

单品	定价 (元/千克)	价格敏感度	折扣	需求奖励度	日补货量 (kg)
红莲藕带	9.20	0.9	-	0	12.1
木耳菜 (份)	27.30	0.7	6 折	0.4	101.3
菠菜 (份)	9.24	0.9	6 折	0.4	2.5
云南油麦菜 (份)	4.50	0.9	7 折	0.3	2.5
云南生菜 (份)	4.69	0.9	6 折	0.4	2.5
小青菜 (1)	5.20	0.9	-	0	2.5
红薯尖	5.37	0.9	-	0	2.5
奶白菜	5.60	0.9	-	0	2.5
苋菜	3.74	0.9	8.5 折	0.15	2.5
娃娃菜	6.80	0.9	6 折	0.4	2.5
上海青	8.00	0.9	-	0	2.5
木耳菜	6.00	0.9	6 折	0.4	2.5
竹叶菜	3.74	0.9	8.5 折	0.15	2.5
云南油麦菜	16.80	0.7	-	0	2.5
西兰花	12.63	0.7	-	0	21.6
圆茄子 (2)	7.70	0.9	-	0	2.5
长线茄	12.00	0.7	-	0	2.5
紫茄子 (2)	6.00	0.9	-	0	2.5
青茄子 (1)	7.00	0.9	-	0	2.5

紫茄子 (1)	31.50	0.7	-	0	11.7
姜蒜小米椒组合装 (小份)	4.80	0.9	-	0	2.5
七彩椒 (2)	24.07	0.7	7 折	0.3	2.5
芜湖青椒 (1)	5.20	0.9	-	0	2.5
小皱皮 (份)	2.80	1	6 折	0.4	2.5
小米椒 (份)	5.80	0.9	6 折	0.4	59.9
青红杭椒组合装 (份)	6.77	0.9	6 折	0.4	2.5
螺丝椒 (份)	5.04	0.9	6 折	0.4	2.5
蟹味菇与白玉菇双拼 (盒)	13.53	0.7	6 折	0.4	2.5
虫草花 (份)	5.32	0.9	-	0	2.5
金针菇 (盒)	2.00	1	6 折	0.4	2.5
白玉菇 (袋)	25.55	0.7	6 折	0.4	27.5
双孢菇 (盒)	5.50	0.9	6 折	0.4	2.5
海鲜菇 (包)	3.00	1	6 折	0.4	2.5
七彩椒 (2)	24.07	0.7	7 折	0.3	2.5
青线椒 (份)	15.05	0.7	-	0	0
芜湖青椒 (1)	5.20	0.9	-	0	2.5
小皱皮 (份)	2.80	1	6 折	0.4	2.5
小米椒 (份)	5.80	0.9	6 折	0.4	59.9
青红杭椒组合装 (份)	6.77	0.9	6 折	0.4	2.5
螺丝椒 (份)	5.04	0.9	6 折	0.4	2.5
红椒 (2)	20.00	0.7	6 折	0.4	0
蟹味菇与白玉菇双拼 (盒)	13.53	0.7	6 折	0.4	2.5
西峡花菇 (1)	24.00	0.7	-	0	0
虫草花 (份)	5.32	0.9	-	0	2.5
金针菇 (盒)	2.00	1	6 折	0.4	2.5
白玉菇 (袋)	25.55	0.7	6 折	0.4	27.5
双孢菇 (盒)	5.50	0.9	6 折	0.4	2.5
海鲜菇 (包)	3.00	1	6 折	0.4	2.5

附录 D　问题一源代码

4.1　数据预处理程序

```
# 读取附件1中的数据
df = pd.read_excel('E:/Desktop/附件1.xlsx', sheet_name='Sheet1')

# 定义分类函数
def classify_item(item):
    """
    将单品名称归类，并生成唯一ID
    """
    # 如果单品名称中存在括号，则以括号前的字符串为分类标准
    if '(' in item:
        category = item[:item.find('(')].strip()
    else:
        category = item.strip()
    # 生成唯一的ID
    return f'{category}_{len(category)}'

# 对单品名称进行分类，并生成唯一ID
df['id'] = df['单品名称'].apply(classify_item)

# 保存结果至新的 Excel 文件
df.to_excel('E:/Desktop/附件1_合并版.xlsx', index=False)
import pandas as pd

# 读取Excel文件
data = pd.read_excel('E:/Desktop/分类结果.xlsx')

# 统计每个商品的取值范围
min_max_values = data.groupby('单品编码')['销量(千克)'].agg(['min', 'max']).reset_index()

# 将统计结果与原始数据合并
data = data.merge(min_max_values, on='单品编码')

# 按天分组并计算总销量
daily_sales = data.groupby('销售日期')['销量(千克)'].sum().reset_index()

# 创建关联规则矩阵
matrix = pd.get_dummies(data[['销售日期', '单品编码']], columns=['销售日期', '单品编码'])
matrix = matrix.groupby(level=0, axis=1).sum()

# 使用Apriori算法进行关联性分析
from mlxtend.frequent_patterns import apriori, association_rules

frequent_itemsets = apriori(matrix, min_support=0.01, use_colnames=True)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)

# 输出结果
print("销量:")
```

```
print(data[['单品编码', '销量(千克)']])
print("\n每天的总销量:")
print(daily_sales)
print("\n关联规则:")
print(rules)
import pandas as pd

# 读取Excel数据
df = pd.read_excel('E:/Desktop/附件2.xlsx')

# 删除销售单价(元/千克)这一列
df.drop('销售单价(元/千克)', axis=1, inplace=True)

# 合并数据并累加销售（千克）
df_merged = df.groupby(['销售日期', '单品编码', '销售类型', '是否打折销售']).sum().reset_index()

# 按时间排序
df_merged.sort_values(by='销售日期', inplace=True)
import pandas as pd

def filter_and_save_excel(input_file, output_file1, output_file2):
    # 读取 Excel 文件
    df = pd.read_excel(input_file)

    # 统计每列数据中数值为0的占比
    zero_percentages = (df == 0).mean() * 100

    # 筛选出占比大于20%的列
    filtered_columns = zero_percentages[zero_percentages > 50].index

    # 从原始数据中删除这些列
    df_filtered = df.drop(columns=filtered_columns)

    # 创建一个新的DataFrame保存结果
    result_df = pd.DataFrame({'列名': zero_percentages.index, '数值为0的占比':
        zero_percentages.values})

    # 将结果保存在Excel文件中
#     result_df.to_excel(output_file1, index=False)

    # 将筛选过后的数据保存在新的Excel文件中
    df_filtered.to_excel(output_file2, index=False)

# 设置输入输出文件路径
input_file = 'E:/Desktop/p2/单品_随时间变化有缺失_合并版.xlsx'
output_file1 = 'E:/Desktop/p2/单品_缺失值占比情况_合并版.xlsx'
output_file2 = 'E:/Desktop/p2/单品_去除过大缺失值_同时有地区无地区.xlsx'
```

```
# 调用函数进行处理和保存
filter_and_save_excel(input_file, output_file1, output_file2)

import pandas as pd

# 读取附件1数据
df1 = pd.read_excel('E:/Desktop/p2/附件1.xlsx')

# 读取附件2数据
df2 = pd.read_excel('E:/Desktop/p2/附件2.xlsx')

# 获取附件1中的单品编码列表
codes_1 = df1['单品编码'].tolist()

# 获取附件2中的单品编码列表
codes_2 = df2['单品编码'].tolist()

# 找到在附件1中存在但在附件2中不存在的单品编码
missing_codes = set(codes_1) - set(codes_2)

# 根据单品编码筛选附件1中对应的单品名称
missing_names = df1[df1['单品编码'].isin(missing_codes)]['单品名称']

# 输出缺失的单品编码及对应的单品名称
for code, name in zip(missing_codes, missing_names):
    print("单品编码：", code, " 单品名称：", name)

####类别的数据预处理#####
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

data = pd.read_excel('E:/Desktop/类别相关性用.xlsx')

# 将数据进行透视，计算每个商品的销量
basket = data.pivot_table(index=['销售日期'], columns='分类名称', values='销量(千克)',
    aggfunc='sum').fillna(0)
```

4.2 APIORI 关联规则挖掘程序

```
# # 读取附件1_合并版数据
# df1 = pd.read_excel('E:/Desktop/附件1.xlsx')
```

```
# # 读取附件2数据
# df2 = pd.read_excel('E:/Desktop/附件2.xlsx')

# # 根据单品编码在附件1中查找对应的id
# df2['id'] = df2['单品编码'].map(df1.set_index('单品编码')['单品名称'])

# # 数据处理
# df_merged = df2.groupby(['id', '销售日期'])['销量(千克)'].sum().reset_index()

# # 保存处理后的数据到问题一附件2.xlsx
# df_merged.to_excel('E:/Desktop/问题一附件2_考虑地区.xlsx', index=False)


import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

def generate_association_rules(input_file_path, output_file_path, min_support=0.2,
    correlation_threshold=0.5):
    # 从Excel读取数据
    data = pd.read_excel(input_file_path)

    # 数据预处理：将同一个类别的销售量映射到0至1的区间内
    data['销量(千克)'] = data.groupby('id')['销量(千克)'].transform(lambda x: (x - x.min()) /
        (x.max() - x.min()))

    # 将数据进行透视，计算每个商品的销量
    basket = data.pivot_table(index=['销售日期'], columns='id', values='销量(千克)',
        aggfunc='sum').fillna(0)

    # 根据设定的阈值将销量转换为二进制变量（大于阈值置为1，小于等于阈值置为0）
    def encode_units(x, threshold):
        return 1 if x > threshold else 0

    threshold = 0.001
    basket_sets = basket.applymap(lambda x: encode_units(x, threshold))

    # 使用Apriori算法挖掘频繁项集
    frequent_itemsets = apriori(basket_sets, min_support=min_support, use_colnames=True,
        max_len=2)

    # 根据频繁项集生成关联规则
    rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

    # 计算销售量相关系数矩阵
    sales_correlation = basket_sets.corr()
```

```
    # 筛选出相关系数大于阈值的商品对
    related_items = []
    for item1 in sales_correlation.columns:
        for item2 in sales_correlation.columns:
            if item1 != item2 and sales_correlation.loc[item1, item2] > correlation_threshold:
                related_items.append((item1, item2))

    # 遍历关联规则，添加销量关系
    rules['sales_relation'] = ''
    for i, rule in rules.iterrows():
        antecedents = set(rule['antecedents'])
        consequents = set(rule['consequents'])
        for item1, item2 in related_items:
            if item1 in antecedents and item2 in consequents:
                rules.at[i, 'sales_relation'] = 'Related'
                break # 只需要判断是否存在一个相关性即可

    # 将频繁二项集表示为集合
    frequent_itemsets['itemsets'] = frequent_itemsets['itemsets'].apply(lambda x: set(x))

    # 保存更新后的关联规则至Excel文件
    with pd.ExcelWriter(output_file_path) as writer:
        frequent_itemsets.to_excel(writer, sheet_name='二项频繁项集', index=False)
        rules.to_excel(writer, sheet_name='二项关联规则', index=False)

generate_association_rules(input_file_path='E:/Desktop/问题一附件2_考虑地区2.xlsx',
                           output_file_path='E:/Desktop/p1__地区_单品类别关联规则_with_set_改进.xlsx',
                           min_support=0.6,
                           correlation_threshold=-2)
```

4.3 DTW 多尺度分析程序

```
import numpy as np
import pywt
import matplotlib.pyplot as plt

def calculate_correlation(input_file):
    # 读取 Excel 文件
    df = pd.read_excel(input_file)

    # 获取除时间列外的所有数据列
    data_columns = df.columns[1:10]

    # 初始化相关性矩阵和列名
    correlation_matrix = np.zeros((len(data_columns), len(data_columns)))
```

```
    column_names = []

    for i in range(len(data_columns)):
        for j in range(i+1, len(data_columns)):
            # 获取两列数据进行相关性计算
            column1 = df[data_columns[i]].values
            column2 = df[data_columns[j]].values

            # 仅保留每年4月到10月的数据
            dates = pd.to_datetime(df['销售日期'])
            mask = (dates.dt.month >= 4) & (dates.dt.month <= 10)
            column1 = column1[mask]
            column2 = column2[mask]

            # 使用DWT进行多尺度相关性计算
            coeffs1 = pywt.wavedec(column1, wavelet='db1', level=5)
            coeffs2 = pywt.wavedec(column2, wavelet='db1', level=5)
            corr = np.corrcoef(coeffs1[0], coeffs2[0])[0, 1]
            correlation_matrix[i, j] = corr

    # 获取数据列的名称
    column_names = list(data_columns)

    return correlation_matrix, column_names

def visualize_correlation(correlation_matrix, column_names):
    # 绘制相关性矩阵图
    plt.imshow(correlation_matrix, cmap='RdBu', vmin=-1.0, vmax=1.0)
    plt.colorbar()

    # 设置横纵坐标标签
    plt.xticks(range(len(column_names)), column_names, rotation=90)
    plt.yticks(range(len(column_names)), column_names)

    # 在每个格子中显示数值大小
    for i in range(len(column_names)):
        for j in range(len(column_names)):
            plt.text(j, i, f"{correlation_matrix[i, j]:.2f}", ha='center', va='center')

    plt.title('部分单品的DWT多尺度相关性矩阵示意图',fontsize=15)
    plt.xlabel('类别',fontsize=12)
    plt.ylabel('类别',fontsize=12)
    plt.savefig("E:/Desktop/DWT多尺度相关性矩阵示意图_单品.png")
    plt.show()

def save_correlation_to_excel(correlation_matrix, column_names, output_file):
    # 创建 DataFrame
```

```
    df = pd.DataFrame(correlation_matrix, columns=column_names, index=column_names)

    # 保存到 Excel 文件
    df.to_excel(output_file)

def get_products_with_high_correlation(correlation_matrix, column_names, threshold):
    # 获取相关性大于阈值或小于阈值的单品
    products = []
    for i in range(len(column_names)):
        for j in range(i+1, len(column_names)):
            corr = correlation_matrix[i, j]
            if corr > threshold or corr < -threshold:
                products.append((column_names[i], column_names[j], corr))

    return products

# 设置输入文件路径
input_file = 'E:/Desktop/p2/单品_随时间变化.xlsx'
# 设置输出文件路径
output_file = 'E:/Desktop/p2/单品_随时间变化_相关性矩阵.xlsx'

# 计算相关性矩阵和列名
correlation_matrix, column_names = calculate_correlation(input_file)

# 可视化相关性矩阵
visualize_correlation(correlation_matrix, column_names)

# 保存相关性矩阵到 Excel 文件
save_correlation_to_excel(correlation_matrix, column_names, output_file)

# 获取相关性大于0.7或小于-0.7的单品
threshold = 0.7
high_correlation_products = get_products_with_high_correlation(correlation_matrix,
    column_names, threshold)

# 输出相关性大于0.5或小于-0.5的单品
print(f"相关性大于{threshold}或小于{-threshold}的单品：")
for product in high_correlation_products:
    print(f"{product[0]} 和 {product[1]}，相关性：{product[2]:.2f}")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
# 显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

```
def load_data(filepath):
    return pd.read_excel(filepath)

def compute_dtw(series_a, series_b, window):
    n, m = len(series_a), len(series_b)
    dtw_matrix = float('inf') * np.ones((n, m))
    dtw_matrix[0, 0] = abs(series_a[0] - series_b[0]) # Compute Euclidean distance directly
    for i in range(1, n):
        dtw_matrix[i, 0] = dtw_matrix[i-1, 0] + abs(series_a[i] - series_b[0])
    for j in range(1, m):
        dtw_matrix[0, j] = dtw_matrix[0, j-1] + abs(series_a[0] - series_b[j])
    for i in range(1, n):
        for j in range(max(1, i - window), min(m, i + window)):
            cost = abs(series_a[i] - series_b[j])
            dtw_matrix[i, j] = cost + min(dtw_matrix[i-1, j], dtw_matrix[i, j-1],
                dtw_matrix[i-1, j-1])
    return dtw_matrix[-1, -1]

def calculate_dtw_distances(data, window):
    categories = data.columns[1:]
    dtw_distances = pd.DataFrame(index=categories, columns=categories)
    for category1 in categories:
        for category2 in categories:
            distance = compute_dtw(data[category1].values, data[category2].values, window)
            dtw_distances.at[category1, category2] = distance
    return dtw_distances

def plot_dendrogram(dtw_distances, title):
    linked = linkage(dtw_distances, 'ward')
    plt.figure(figsize=(12, 7))
    dendrogram(linked, orientation='top', labels=dtw_distances.index.tolist(),
        distance_sort='descending', show_leaf_counts=True)
    plt.title(title)
    plt.savefig("E:/Desktop/p2/部分单品DTW分析结果图.png")
    plt.show()

if __name__ == "__main__":
    data_path = "E:/Desktop/p2/单品_随时间变化.xlsx" # Please replace with your file path
    data = load_data(data_path)
    dtw_distances_30days = calculate_dtw_distances(data, window=30)
    plot_dendrogram(dtw_distances_30days, '部分单品基于DTW的聚类树图 (时间窗口: 30天)')
```

4.4 单品季节性初步分析程序

```
import matplotlib.pyplot as plt
#显示中文
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

def plot_sales_trends_2_years(file_path, veg_names):
    """
    Plot sales trends for specified vegetables in a 3x2 grid for both 2021 and 2022.

    Args:
    - file_path (str): Path to the Excel file containing monthly sales data.
    - veg_names (list): List of vegetable names to plot.
    """
    # Load the aggregated monthly sales dataset
    monthly_sales_data = pd.read_excel(file_path)

    # Filter data for the years 2021 and 2022
    data_2021_2022 = monthly_sales_data[monthly_sales_data['年'].isin([2021, 2022])]

    # Create a dataframe with all possible combinations of months, vegetables, and years
    all_months = list(range(1, 13)) # Months from 1 to 12
    all_vegs = data_2021_2022['单品名称'].unique()
    all_years = [2021, 2022]
    all_combinations = [(year, month, veg) for year in all_years for month in all_months for
        veg in all_vegs]
    all_combinations_df = pd.DataFrame(all_combinations, columns=['年', '月', '单品名称'])

    # Merge the all_combinations_df with the data to fill missing months with 0 sales
    merged_data = pd.merge(all_combinations_df, data_2021_2022, on=['年', '月', '单品名称'],
        how='left')
    merged_data['销量(千克)'].fillna(0, inplace=True)

    # Plot seasonal trends for the specified vegetables in a 3x2 grid
    plt.figure(figsize=(25, 15))
    for idx, veg in enumerate(veg_names, 1):
        plt.subplot(2, 3, idx)

        for year in all_years:
            veg_data = merged_data[(merged_data['单品名称'] == veg) & (merged_data['年'] == year)]
            plt.plot(veg_data['月'], veg_data['销量(千克)'], marker='o', label=f"{veg} {year}")

        plt.title(veg,fontsize=30)
        plt.xlabel('月份',fontsize=22)
        plt.ylabel('销售量(kg)',fontsize=22)
        #调节刻度坐标的大小
        plt.xticks(fontsize=20)
        plt.yticks(fontsize=20)
```

```
        plt.legend(fontsize=20)
        plt.tight_layout()
    plt.savefig("E:/Desktop/单品季节性分析.png")
    plt.show()

# Replace '/path/to/your/file.xlsx' with your file path
# And ["胡萝卜", "白萝卜", ...] with your desired vegetable names
# desired_vegs = ["红薯尖", "西兰花", "净藕(1)", "奶白菜", "小白菜", "西峡香菇(1)"]
desired_vegs = ["红薯尖", "菠菜", "小白菜","西兰花", "净藕(1)", "西峡香菇(1)"]
plot_sales_trends_2_years("E:/Desktop/工作簿9.xlsx", desired_vegs)
```

4.5 类别季节性初步分析程序

```
import numpy as np
import matplotlib.pyplot as plt

#显示中文
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# 读取数据
monthly_sales_data = pd.read_excel("E:/Desktop/工作簿5.xlsx", header=1)

# 重命名列，以使其更清晰
monthly_sales_data.columns = ['年', '月', '品类', '销量']

# 过滤数据，只保留2021年和2022年的数据
filtered_monthly_sales = monthly_sales_data[monthly_sales_data['年'].isin([2021, 2022])]

# 定义季节映射
seasons_mapping = {
    1: "冬季", 2: "冬季", 3: "春季", 4: "春季", 5: "春季",
    6: "夏季", 7: "夏季", 8: "夏季", 9: "秋季", 10: "秋季", 11: "秋季", 12: "冬季"
}

# 将月份映射到相应的季节
filtered_monthly_sales['季节'] = filtered_monthly_sales['月'].map(seasons_mapping)

# 按年份、品类和季节分组，计算每个季节的平均销售量
seasonal_sales_avg = filtered_monthly_sales.groupby(['年', '品类',
    '季节'])['销量'].mean().reset_index()

# 可视化
plt.figure(figsize=(15, 10))
```

```
# colors = ['darkcyan', 'mediumturquoise'] # 设置颜色列表
colors = ['paleturquoise', 'c']
categories = ['水生根茎类', '花叶类', '花菜类', '茄类', '辣椒类', '食用菌']
for idx, category in enumerate(categories, 1):
    plt.subplot(2, 3, idx)

    for year, color in zip([2021,2022], colors): # 使用zip函数将年份和颜色一一对应
        tick_label=['春季','夏季','秋季','冬季']
        xpos = np.arange(4)
        yearly_data = seasonal_sales_avg[(seasonal_sales_avg['年'] == year) &
            (seasonal_sales_avg['品类'] == category)]
        x = yearly_data['季节']
        width = 0.35
        if year==2021:
            plt.bar(xpos, yearly_data['销量'], width, alpha=0.7, label=f'{year}', color=color)
        if year==2022:
            plt.bar(xpos+width, yearly_data['销量'], width, alpha=0.7, label=f'{year}',
                color=color)

    plt.title(category,fontsize=18)
    plt.xlabel("季节",fontsize=15)
    plt.ylabel("平均销售量 (kg)",fontsize=15)
    plt.xticks(xpos+width/2,tick_label,fontsize=12)#显示x坐标轴的标签,即tick_label,调整位置，使其落在两个直方图中间
    plt.legend()
    plt.tight_layout()

plt.savefig("E:/Desktop/季节性分析条形图.png")
plt.show()
```

4.6 缺失统计程序

```
# 读取附件1数据
df1 = pd.read_excel('E:/Desktop/p2/附件1.xlsx')

# 读取附件2数据
df2 = pd.read_excel('E:/Desktop/p2/附件2.xlsx')

# 获取附件1中的单品编码列表
codes_1 = df1['单品编码'].tolist()

# 获取附件2中的单品编码列表
codes_2 = df2['单品编码'].tolist()

# 找到在附件1中存在但在附件2中不存在的单品编码
```

```
missing_codes = set(codes_1) - set(codes_2)

# 根据单品编码筛选附件1中对应的单品名称
missing_names = df1[df1['单品编码'].isin(missing_codes)]['单品名称']

# 输出缺失的单品编码及对应的单品名称
for code, name in zip(missing_codes, missing_names):
    print("单品编码: ", code, " 单品名称: ", name)
```

4.7 斯皮尔曼秩相关性分析程序

```
import pandas as pd
import seaborn as sns
#显示中文
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

# 读取Excel数据
data = pd.read_excel('E:/Desktop/类别相关性.xlsx')

# 计算斯皮尔曼秩相关系数并保存结果到Excel中
spearman_corr = data.corr(method='spearman')
# spearman_corr.to_excel("Spearman_corr.xlsx", index=False)

# 绘制相关性热力图
sns.heatmap(spearman_corr, annot=True, cmap='coolwarm')
```

附录 E 问题二源代码

5.1 箱型图异常值分析程序

```
import matplotlib.pyplot as plt
import seaborn as sns

# 从Excel文件读取数据
data = pd.read_excel('E:/Desktop/p2/类别_随时间变化.xlsx')

# 创建一个空的DataFrame，用于存储每一列的数据
plot_data = pd.DataFrame()

# 遍历每一列（除了销售日期列）
for col in data.columns:
    if col != '销售日期':
```

```
        # 提取当前列数据
        column_data = data[col]

        # 将当前列数据存储到plot_data中
        plot_data[col] = column_data

# 设置浅色的色系
sns.set_palette("pastel")

# 绘制每一列的小提琴图
plt.figure(figsize=(12, 8))
sns.violinplot(data=plot_data, scale='width')
plt.title('不同品类的小提琴图')
plt.ylabel('销量')
plt.xticks(rotation=45) # 旋转x轴标签，以免重叠
plt.savefig("E:/Desktop/不同品类的小提琴图.png")
plt.show()
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 从Excel文件读取数据
data = pd.read_excel('E:/Desktop/p2/类别_随时间变化.xlsx')

# 创建一个空的DataFrame，用于存储每一列的数据
plot_data = pd.DataFrame()

# 遍历每一列（除了销售日期列）
for col in data.columns:
    if col != '销售日期':
        # 提取当前列数据
        column_data = data[col]

        # 将当前列数据存储到plot_data中
        plot_data[col] = column_data

# 绘制每一列的小提琴图
plt.figure(figsize=(12, 8))
sns.violinplot(data=plot_data)
plt.title('各列数据的小提琴图')
plt.ylabel('值')
plt.xticks(rotation=45) # 旋转x轴标签，以免重叠
plt.show()
```

5.2 多变量时间序列预测程序

```
import pandas as pd
import numpy as np
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import coint
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_percentage_error
import matplotlib.pyplot as plt
import holidays

# 显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 读取数据
data = pd.read_excel('E:/Desktop/p2/类别_随时间变化.xlsx')

# 将销售日期列转换为datetime类型
data['销售日期'] = pd.to_datetime(data['销售日期'])

# 滑动窗口预测
window_size = 30
step_size = 7

# 重新划分训练集和测试集
train_data = data[data['销售日期'] < pd.Timestamp(2023, 7, 1)]
test_data = data[data['销售日期'] >= pd.Timestamp(2023, 7, 1)]

def sarima_forecast(category):
    # 获取训练集和测试集对应品类的销量数据
    train_sales = train_data[category].values
    test_sales = test_data[category].values

    # 获取公共节假日信息
    public_holidays = holidays.China(years=range(train_data['销售日期'].dt.year.min(),
        test_data['销售日期'].dt.year.max()+1))

    # 将节假日信息与销售数据匹配
    train_holidays = [1 if date in public_holidays else 0 for date in train_data['销售日期']]
    test_holidays = [1 if date in public_holidays else 0 for date in test_data['销售日期']]

    # 品类相关性分析
    related_categories = categories.copy() # 复制品类列表
    related_categories.remove(category) # 移除当前品类

    # 存储相关性结果
    coint_results = []
```

```
# 计算品类之间的协整性分析
for related_category in related_categories:
    related_train_sales = train_data[related_category].values
    _, pvalue, _ = coint(train_sales, related_train_sales)
    coint_results.append(pvalue)

# 根据p值判断哪些品类之间存在显著相关性
correlated_categories = [related_categories[i] for i, pvalue in enumerate(coint_results) if
     pvalue < 0.1]

# 如果存在显著相关性，则将其他品类的销量作为额外特征变量输入到SARIMA模型中
if correlated_categories:
    exog_train_data = train_data[correlated_categories].values
    exog_test_data = test_data[correlated_categories].values

    # 添加时间序列的季节性变化趋势作为额外特征变量
    train_seasonality = pd.Series(train_data['销售日期'].dt.quarter)
    exog_train_data = np.column_stack((exog_train_data, train_seasonality))

    test_seasonality = pd.Series(test_data['销售日期'].dt.quarter)
    exog_test_data = np.column_stack((exog_test_data, test_seasonality))

    # 训练SARIMA模型，同时传入额外的特征变量和节假日信息
    model = SARIMAX(train_sales, exog=exog_train_data, order=(1, 1, 1), seasonal_order=(2,
        1, 1, 12))
    model_fit = model.fit()

    # 初始化预测结果数组
    predictions = np.zeros(len(test_sales))

    # 滑动窗口预测
    for i in range(0, len(test_sales), step_size):
        end = min(i + window_size, len(test_sales))
        forecast_length = end - i

        # 滑动窗口内的训练数据和特征变量
        window_train_sales = np.concatenate((train_sales, test_sales[:i]))
        window_exog_train = np.concatenate((exog_train_data, exog_test_data[:i]))

        # 滑动窗口内的测试数据和特征变量
        window_test_sales = test_sales[i:end]
        window_exog_test = exog_test_data[i:end]

        # 预测测试集中的销量，同时传入滑动窗口内的特征变量和节假日信息
        window_model = SARIMAX(window_train_sales, exog=window_exog_train, order=(1, 1, 1),
            seasonal_order=(2, 1, 1, 12))
```

```
            window_model_fit = window_model.filter(model_fit.params)
            window_predictions = window_model_fit.forecast(forecast_length,
                exog=window_exog_test)

            # 更新预测结果数组
            predictions[i:end] = window_predictions

            # 更新训练数据和特征变量
            train_sales = np.concatenate((train_sales, window_test_sales))
            exog_train_data = np.concatenate((exog_train_data, window_exog_test))

    else:
        # 没有相关性的情况下，使用普通的SARIMA模型进行预测
        model = SARIMAX(train_sales, order=(1, 1, 1), seasonal_order=(2, 1, 1, 12))
        model_fit = model.fit()

        # 初始化预测结果数组
        predictions = np.zeros(len(test_sales))

       # 滑动窗口预测
        for i in range(0, len(test_sales), step_size):
            end = min(i + window_size, len(test_sales))
            forecast_length = end - i

            # 滑动窗口内的训练数据
            window_train_sales = np.concatenate((train_sales, test_sales[:i]))

            # 滑动窗口内的测试数据
            window_test_sales = test_sales[i:end]

            # 预测测试集中的销量
            window_model = SARIMAX(window_train_sales, order=(1, 1, 1), seasonal_order=(2, 1, 1,
                12))
            window_model_fit = window_model.filter(model_fit.params)
            window_predictions = window_model_fit.forecast(forecast_length)

            # 更新预测结果数组
            predictions[i:end] = window_predictions

            # 更新训练数据
            train_sales = np.concatenate((train_sales, window_test_sales))

        # 添加预测值矫正的功能
        for category in categories:
            # 计算下四分位数和上四分位数
            lower_quartile = np.percentile(predictions[category], 25)
            upper_quartile = np.percentile(predictions[category], 75)
```

```
            # 计算四分位距离
            iqr = upper_quartile - lower_quartile

            # 定义下界和上界
            lower_bound = lower_quartile - 1.5 * iqr
            upper_bound = upper_quartile + 1.5 * iqr

            # 矫正极端值
            for i in range(len(predictions[category])):
                if predictions[category][i] > upper_bound:
                    predictions[category][i] = upper_bound
                    # 寻找相邻的两个极端值对应的时间段
                    start_idx = max(0, i - window_size)
                    end_idx = min(len(predictions[category]), i + window_size + 1)
                    extreme_values = test_data[category].values[start_idx:end_idx]
                    # 计算要缩放的比例
                    scale_ratio = upper_bound / np.max(extreme_values)
                    # 缩放该时间段内的预测销量
                    predictions[category][start_idx:end_idx] = extreme_values * scale_ratio

                elif predictions[category][i] < lower_bound:
                    predictions[category][i] = lower_bound
                    # 寻找相邻的两个极端值对应的时间段
                    start_idx = max(0, i - window_size)
                    end_idx = min(len(predictions[category]), i + window_size + 1)
                    extreme_values = test_data[category].values[start_idx:end_idx]
                    # 计算要缩放的比例
                    scale_ratio = lower_bound / np.min(extreme_values)
                    # 缩放该时间段内的预测销量
                    predictions[category][start_idx:end_idx] = extreme_values * scale_ratio

    return predictions

# 预测每个品类的销量
categories = ['水生根茎类', '花叶类', '花菜类', '茄类', '辣椒类', '食用菌']
sales_ratios = [0.06, 0.42, 0.10, 0.05, 0.19, 0.15] # 给定的销量占比

predicted_sales = {}
confidence_intervals = {}

for category in categories:
    predicted_sales[category] = sarima_forecast(category)

    # 计算每次预测的置信区间
    window_size = len(predicted_sales[category])
    residuals = test_data[category].values - predicted_sales[category]
```

```
    std_dev = np.std(residuals)
    confidence_interval = (predicted_sales[category] - 1.96 * std_dev / np.sqrt(window_size),
                           predicted_sales[category] + 1.96 * std_dev / np.sqrt(window_size))
    confidence_intervals[category] = confidence_interval

# 计算总销量预测
total_predicted_sales = sum(predicted_sales.values())

# 设置允许的浮动百分比
float_percentage = 0.02

# 计算每个品类销量占比的上下界
upper_ratios = [ratio * (1 + float_percentage) for ratio in sales_ratios]
lower_ratios = [ratio * (1 - float_percentage) for ratio in sales_ratios]

# 按照给定的占比调整预测，并确保预测落在上下界之内
adjusted_sales = {}
for category, ratio, upper, lower in zip(categories, sales_ratios, upper_ratios, lower_ratios):
    proposed_sales = total_predicted_sales * ratio
    adjusted_sales[category] = np.clip(proposed_sales, total_predicted_sales * lower,
        total_predicted_sales * upper)


# 评估调整后的预测
rmse_scores = []
rmape_scores = []

for category in categories:
    predicted_sales[category] = sarima_forecast(category)

    # 计算每次预测的置信区间
    window_size = len(predicted_sales[category])
    residuals = test_data[category].values - predicted_sales[category]
    std_dev = np.std(residuals)
    confidence_interval = (predicted_sales[category] - 1.96 * std_dev / np.sqrt(window_size),
                           predicted_sales[category] + 1.96 * std_dev / np.sqrt(window_size))
    confidence_intervals[category] = confidence_interval

# 计算总销量预测
total_predicted_sales = sum(predicted_sales.values())

# 设置允许的浮动百分比
float_percentage = 0.02

# 计算每个品类销量占比的上下界
upper_ratios = [ratio * (1 + float_percentage) for ratio in sales_ratios]
lower_ratios = [ratio * (1 - float_percentage) for ratio in sales_ratios]
```

```
# 按照给定的占比调整预测，并确保预测落在上下界之内
adjusted_sales = {}
for category, ratio, upper, lower in zip(categories, sales_ratios, upper_ratios, lower_ratios):
    proposed_sales = total_predicted_sales * ratio
    adjusted_sales[category] = np.clip(proposed_sales, total_predicted_sales * lower,
        total_predicted_sales * upper)


# 评估调整后的预测
rmse_scores = []
rmape_scores = []

for category in categories:
    rmse = np.sqrt(mean_squared_error(test_data[category].values, adjusted_sales[category]))
    rmape = mean_absolute_percentage_error(test_data[category].values, adjusted_sales[category])
    rmse_scores.append(rmse)
    rmape_scores.append(rmape)
    print(category + ': RMSE =', rmse, ', RMAPE =', rmape)
def calculate_r_squared(true_values, predicted_values):
    mean_true = np.mean(true_values)
    ss_total = np.sum((true_values - mean_true) ** 2)
    ss_residual = np.sum((true_values - predicted_values) ** 2)
    r_squared = 1 - (ss_residual / ss_total)
    return r_squared

def calculate_probability_in_interval(true_values, lower_bound, upper_bound):
    count_within_interval = np.sum((true_values >= lower_bound) & (true_values <= upper_bound))
    probability = count_within_interval / len(true_values)
    return probability

r_squared_values = {}
probabilities_in_interval = {}
errors = {}

for category in categories:
    residuals = test_data[category].values - predicted_sales[category]
    std_dev = np.std(residuals)
    print(std_dev)
    error_interval = (predicted_sales[category] - std_dev, predicted_sales[category] + std_dev)
    r_squared_values[category] = calculate_r_squared(test_data[category],
        predicted_sales[category])
    probabilities_in_interval[category] = calculate_probability_in_interval(test_data[category],
                                                                            error_interval[0],
                                                                            error_interval[1])
    errors[category] = residuals
```

```
    plt.figure(figsize=(12, 10))
    plt.plot(test_data['销售日期'], test_data[category], label='真实销售', color='dodgerblue')
    plt.plot(test_data['销售日期'], predicted_sales[category], label='调整后预测',
        color='purple', linestyle='--')

    # 绘制误差区间
    lower_bound, upper_bound = error_interval
    plt.fill_between(test_data['销售日期'], lower_bound, upper_bound, color='darkmagenta',
        alpha=0.1, label='误差区间')

    plt.xlabel('日期', fontsize=12)
    plt.ylabel('销售量', fontsize=12)
    plt.title(category + ' 销售量对比及误差区间', fontsize=18)
    plt.legend()
    plt.grid(True)
    plt.savefig("E:/Desktop/" + category + " 销售量对比及误差区间图.png")
    plt.show()

    print(f"在品类 '{category}' 上的 R 方为: {r_squared_values[category]}")
#     print(f"每天真实值落入误差区间内的概率为: {probabilities_in_interval[category]}")
```

5.3 误差区间与绘图程序

```
for category in categories:
    residuals = test_data[category].values - predicted_sales[category]
    std_dev = np.std(residuals)
    print(std_dev)
    error_interval = (predicted_sales[category] - std_dev, predicted_sales[category] + std_dev)
    r_squared_values[category] = calculate_r_squared(test_data[category],
        predicted_sales[category])
    probabilities_in_interval[category] = calculate_probability_in_interval(test_data[category],
                                                                             error_interval[0],
                                                                             error_interval[1])
    errors[category] = residuals

    plt.figure(figsize=(12, 10))
    plt.plot(test_data['销售日期'], test_data[category], label='真实销售', color='dodgerblue')
    plt.plot(test_data['销售日期'], predicted_sales[category], label='调整后预测',
        color='purple', linestyle='--')

    # 绘制误差区间
    lower_bound, upper_bound = error_interval
    plt.fill_between(test_data['销售日期'], lower_bound, upper_bound, color='darkmagenta',
        alpha=0.1, label='误差区间')
```

```
    plt.xlabel('日期', fontsize=12)
    plt.ylabel('销售量', fontsize=12)
    plt.title(category + ' 销售量对比及误差区间', fontsize=18)
    plt.legend()
    plt.grid(True)
    plt.savefig("E:/Desktop/" + category + " 销售量对比及误差区间图.png")
    plt.show()
```

5.4 灵敏度分析程序

```
lower_bound_probabilities = {}
upper_bound_probabilities = {}

start_date = '2023-06-24'
end_date = '2023-06-30'

date_range = (test_data['销售日期'] >= start_date) & (test_data['销售日期'] <= end_date)

for category in categories:
    lower_bound = error_interval[0]
    upper_bound = error_interval[1]

    lower_bound_count = np.sum((test_data[category] < lower_bound) & date_range)
    upper_bound_count = np.sum((test_data[category] > upper_bound) & date_range)
    within_interval_count = np.sum(date_range) - (lower_bound_count + upper_bound_count)

    lower_bound_probability = lower_bound_count / np.sum(date_range)
    upper_bound_probability = upper_bound_count / np.sum(date_range)
    within_interval_probability = within_interval_count / np.sum(date_range)

    lower_bound_probabilities[category] = lower_bound_probability
    upper_bound_probabilities[category] = upper_bound_probability

    print(f"在品类 '{category}' 上的 R 方为: {r_squared_values[category]}")
    print(f"在2023年6月24日到6月30日期间，每天真实值落入误差区间内的概率为:
        {within_interval_probability}")
    print(f"在2023年6月24日到6月30日期间，真实值小于下界的概率为: {lower_bound_probability}")
    print(f"在2023年6月24日到6月30日期间，真实值大于上界的概率为: {upper_bound_probability}")
```

附录 F 问题三源代码

```
import pandas as pd
```

```python
# Step 1: Load the initial provided data
initial_data = pd.read_excel("E:/Desktop/6.24开始的.xlsx")

# Step 2: Filter out the non-discounted sales
no_discount_data = initial_data[initial_data['是否打折销售'] == '否']

# Step 3 & 4: Calculate the number of days each product was sold in the 7-day period
daily_sales = no_discount_data.groupby(['单品编码',
    '销售日期']).size().reset_index(name='count')
days_sold = daily_sales.groupby('单品编码').agg({'count': 'count'}).reset_index()

# Step 5, 6 & 7: Calculate the adjusted average wholesale and sales price
grouped_data = no_discount_data.groupby('单品编码').agg({
    '小分类名称': 'first',
    '分类名称': 'first',
    '批发价格(元/千克)': 'mean',
    '销售价格': 'mean'
}).reset_index()
merged_data = grouped_data.merge(days_sold, on='单品编码')
merged_data['近7天平均批发价格'] = merged_data['批发价格(元/千克)'] * 7 / merged_data['count']
merged_data['近7天平均销售价格'] = merged_data['销售价格'] * 7 / merged_data['count']

# Step 8: Clean up the data (drop unnecessary columns and rename)
final_data = merged_data.drop(columns=['count', '批发价格(元/千克)', '销售价格'])
final_data.rename(columns={'小分类名称': '品类'}, inplace=True)

# Step 9: Assign the sensitivity based on the average sales price
def assign_sensitivity(price):
    if price <= 6:
        return 1
    elif 6 < price <= 15:
        return 0.8
    else:
        return 0.6
final_data['价格敏感度'] = final_data['近7天平均销售价格'].apply(assign_sensitivity)

# Step 10: Load the loss rate data from the second sheet of the provided Excel file
loss_rate_data = pd.read_excel("/mnt/data/附件4.xlsx", sheet_name=1)

# Step 11: Merge the data based on the product code to fill in the loss rate
final_data_with_loss_rate = final_data.merge(loss_rate_data[['单品编码', '损耗率(%)']],
    on='单品编码', how='left')

# Step 12: Sort the data by '分类名称'
sorted_data = final_data_with_loss_rate.sort_values(by='分类名称')

# Step 13: Save the processed data to an Excel file
```

```
sorted_data.to_excel("E:/Desktop/final_processed_data.xlsx", index=False)
```

附录 G　问题四源代码