基于数据挖掘的商超补货与定价策略模型

摘　要

本文通过对商超历史销售流水数据进行数据挖掘，一方面分析了蔬菜品类和单品之间的关联和分布情况，另一方面还**提取到了与定价相关的指标**，并根据这些指标信息构建规划模型，最终实现了商超利益最大化的补货与定价策略，这对于实际商超经营者来说具有很重要的参考意义。

首先，从已给附件中得知单品数目较多，其中的部分可能只是商超的一些尝试行为，如果全部考虑会导致分析过程较复杂，且并无实际意义。因此我们**筛去**过去销量较少（占比 0.5%以下），对后续问题分析无意义的（从未在七月时间点左右销售过），且价格并不具备暴利性（部分蔬菜稀有，价格高）的单品，最终剩下 132 个单品，简化了分析的计算难度。

针对问题一，先使用 **Shapiro-Wilk 检验**品类销量符合正态分布，并使用**斯皮尔曼相关性分析**各品类日销量和月销量之间的关联，两者相关性情况基本一致，品类销量之间关系为正相关。仅茄类的情况存在差异，我们认为可能时由于茄类本身的单品较少，且销量总体最少，导致日销量存在较大波动。使用箱型图反映单品销量的分布情况，另外使用**互信息度**分析单品的销售分布关系，**Apriori 算法**量化单品间的购买协同性，购买协同性为消费者消费特征，侧面体现单品间关联。

针对问题二，从数据中挖掘出各指标数据，根据其各自特性，选择适合的模型进行构建。即使用**高斯过程回归**得到销量波动函数，借助**最小二乘回归树**预测未来的批发价格（$\mathbf{R^2>0.99}$），另外使用**多元线性回归**拟合定价与销量和批发价格的情况并进行修正，而对于打折力度和损耗率，其变化波动不大，则使用已有数据的均值进行量化。以最大化利润为目标，各品类销量波动上下限为限制条件，得到了收益最大化的方案。

针对问题三，先对处于此时间段的 46 个单品进行筛选，得到在 7 月 1 日的销量可能超过 2.5kg 的单品共 31 个单品，进一步绘制其销量曲线进一步排除，得到最后 30 个单品，符合题目要求，进而与问题二的各品类在 7 月 1 日的销量波动结合，建立约束条件，即各品类中的单品销量之和应处于问题二求出的上下限内，且其本身的销量应该大于 2.5kg 和求得的定价关系式，使用**禁忌搜索算法**模型进行求解出 30 个单品各自的销量定价以及商超的最大利润，得到收益最大化方案，收益 708.7 元，与第二问所得到的结果 732.9 较为相近。

针对第四问，我们从二三问的指标信息建立可知，从附件二的销量流水数据提取到的平均批发价格与实际真实的批发价格存在出入，原因是根据销量情况**分权平均**导致结果偏高，因而对多元线性回归得到的结果进行了比例修正。所以我们认为还需采集**批发量**的数据，才能使得上述模型的指标更加符合实际情况。

本文模型的主要创新点有：引入了原创指标和函数，即损耗率，打折力度，批发价，销售量，定价与销量的回归函数；使用了高斯过程回归得到销售量的波动函数，允许误差的存在进行规划求解最优策略；使用禁忌搜索智能算法寻得最优策略。

关键词：Shapiro-Wilk 检验，斯皮尔曼相关性分析，互信息度，Apriori 算法，高斯过程回归，最小二乘回归树，多元线性回归，禁忌搜索算法

一、 问题重述

1.1 问题背景

随着生鲜产品逐渐成为消费者的刚需,我国生鲜市场已步入刚需期。在这一过程中,以生鲜商超为代表的新零售模式已成为生鲜销售的主导力量。然而,生鲜商超仍需确立最优的店铺营收模式,以保证长期稳定发展[1]。因此，商超需要对单品情况进行分析，有偏好性的补充价格稳定销售量高的单品，另外有一定概率补货量小却利益较大的品类以尽可能的获取更大的利润。

1.2 问题设立

问题的设立是层层递进的，并且指向同一核心目的——如何在未知单品种类和批发价格的情况下对第二天的补货量和定价制定策略使得利益最大。

1.问题一首先要求对蔬菜品类和单品的关系进行分析，并了解其销量的分布规律

2.问题二要求以品类为单位，对后续一周的补货量和定价制定策略使得利益最大化

3.问题三聚焦于最近一周的商品情况，在给定品种的情况下限制单品的选择范围，且每个单品的购入量有最低限制吗，指定 7 月 1 日的补货和定价策略。

4.问题四希望根据上述模型的建立分析后，对其不足进行分析，并考虑还需要哪些数据可以更好的完善上述模型，使得补货和定价策略更准确。

二、 问题分析

2.1 总体分析

本题是一个有关商超销售流水数据进行数据挖掘的问题，并通过挖掘到的信息进行补货和定价策略的制定。要求分析蔬菜单品之间的关系，从供应商来看，有季节性的相关变化情况；从商超角度看，有销量的变化相关关系；从消费者角度，有蔬菜购买时的协同性[3]。因此，面对 246 种商品性质不一的单品，由于供货商提供蔬菜品类的未知性与波动性，商超需要对单品情况进行分析，有偏好性的补充价格稳定销售量高的单品，另外有一定概率补货量小却利益较大的品类。

2.2 问题一的分析

问题一的目标在于分析蔬菜品类，单品之间的相关性和分布规律，从中得到蔬菜可能存在的捆绑销售，进而可能帮助商超决策利益更大的补货定价策略。因此本文的分析过程如图一，即分别从品类的日销量，月销量分析可能存在的相关性。对于单品，本文使用互信息度衡量其单品间的依赖程度，另外使用 Apriori（关联分析算法）挖掘出消费者的购买信息中可能存在的购买协同性[2]。

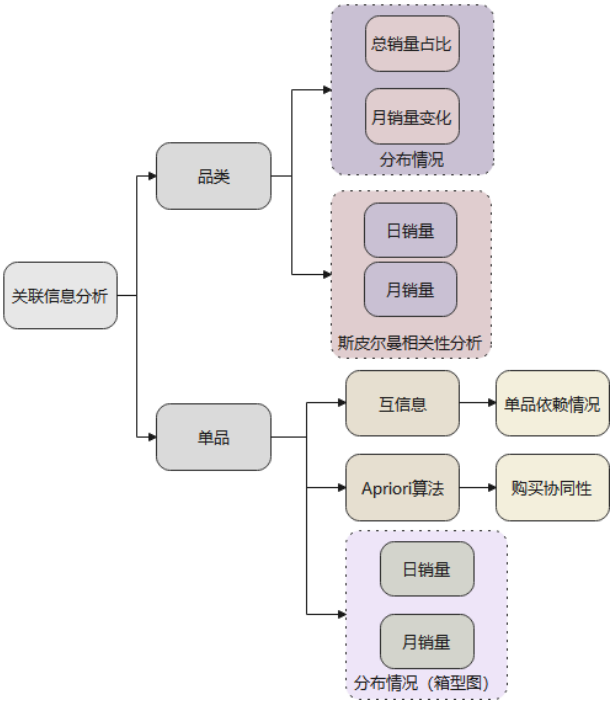


图 1 问题一的工作

2.3 问题二的分析

问题二目的是根据已有信息对未来一周的情况进行策略制定，力求最大利润，本文尽可能的对数据信息进行挖掘如图二，并对各数据情况分析使用契合的模型，最大的优点在于使用高斯过程回归拟合以往数据进而分析得到 7 月 1 日到 7 月 7 日一周内每天的销量需求偏差函数，其波动情况较为完美的体现数据分布的情况，是符合实际的。

进而在分析出的信息中，构建信息间的关系得到利润函数，构建规划模型，使用 python 进行求解，得到各品类每一天的补货量和定价策略。

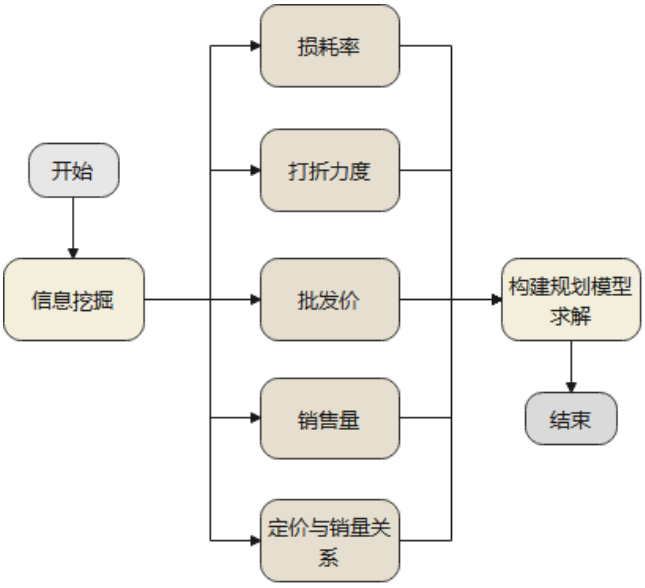


图 2 问题二的思路

2.4 问题三的分析

先对处于此时间段的 46 个单品进行筛选，得到在 7 月 1 日的销量可能超过 2.5 的单品共 30 个单品，符合题目要求，进而与问题二的各品类在 7 月 1 日的销量波动结合，建立限制条件，即各品类中的单品销量之和应处于问题二求出的上下限内，且其本身的销量应该大于 2.5kg，建立凸优化模型使用禁忌搜索算法进行求解。

2.5 问题四的分析

针对问题四，需要根据上述问题搭建的模型进行分析和评价，考虑什么数据的采集会使得模型更加真实可靠，符合实际情况。最终认为采集其它成本，每日批发量，商超自身可容纳的容量上限的数据可以对已建立的模型进行扩展，使其更具有实际意义。

三、 模型假设

1. 假设超市的数据是真实可信的。
2. 假设超市每日蔬菜均售卖完，即无余量，使用打折手段售卖过量蔬菜
3. 假设蔬菜损耗率每日波动较小
4. 假设超市的员工成本等已包含在蔬菜进价成本中，即利润为销售额与成本价的差值
5. 假设流水数据中个人的商品扫码是连续的

四、 符号说明

表 1 符号表

符号	说明
p(x,y)	X 和 Y 的联合概率分布函数
I(X;Y)	互信息度
S_i	第i蔬菜品类的平均损耗率
G_i	第i蔬菜品类的打折力度
m	为品类总数
n	品类所包括的单品总数
P_{ij}	该单品的当日定价
P_{ij}'	该单品打折时的售价
O_i	第i蔬菜品类的平均损耗率
$G_i(z_{i,t})$	销售量波动函数
x_i,	销售量的偏差函数
y_i	第i蔬菜品类的平均定价

五、 数据预处理

5.1 异常值检测

经检测，给出的附件数据无缺失值，其中蔬菜单品供应符合实际情况，即季节性供应。

5.2 数据筛除

本题的重点在于批发策略与定价的最大利益规划，而对于商品是否值得批发，销量是一个重要的考虑量。所以对于附件 2 的数据，由于曾销售的单品数目过多，为了聚焦于更有价值的商品，我们决定对三年内总销量少于 1000Kg，并未在 6,7,8 月份销售过，且价格为正常物价的商品数据进行剔除，这样做的合理性如下：

- 在三年的时间跨度上累积销量并为超过 1000Kg，说明其远不如销量大的蔬菜重要；
- 后续需要规划决策七月初的商品的规划，考虑到每年季节气候导致蔬菜供应可能存在的波动，我们设定第二个限制，保障在七月份可能销售的单品不被筛除；
- 部分蔬菜单品可能由于自身的稀有情况，导致产量较少，且较昂贵（通常利润较高），因此将非正常物价的单品设置限制保留下来；

通过以上的操作，我们将以下表 2 的共计 114 个蔬菜单品（只展示部分，整体见支撑文件），共计删除数据 20522 条销售数据。

表 2 删除的单品统计

单品编码	总销量(千克)	6, 7, 8 月销售次数
水果辣椒(橙色)	0.415	0
红橡叶	0.419	0
紫白菜(2)	0.615	0
芥兰	0.671	1
红珊瑚(粗叶)	0.682	0
甘蓝叶	0.943	0
……	……	……
蔡甸藜蒿	939.575	0
虫草花(袋)	978	0

5.3 数据融合

由于附件一和附件二的蔬菜品类并未建立映射，我们将附件一中的蔬菜品类映射到附件二中，方便后续分析品类相关数据的处理。

六、 模型的建立与求解

6.1 问题一模型的建立与求解

6.1.1 品类之间的关系

对已经预处理后的附件二中的流水数据按品类进行日销量，总销量的统计，得到

图 3 的各品类的总销售量比例图。可以知道的是花叶类蔬菜共有大致有 100 种单品，其总销量占据了所有销量的 42.2%，而茄类只有 10 种单品，销售量也只有总的 4.8%。其中最值得注意的是花菜类仅 5 种单品，其销售量却占比 8.9%，侧面体现了其单品销售量较多。

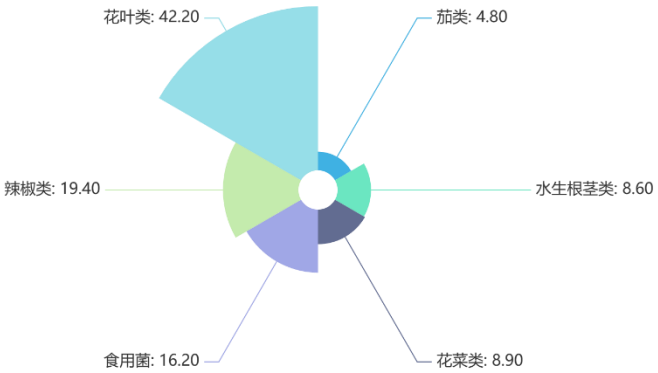


图 3 各品类总销量占比

然后对品类进行相关性分析，如果符合正太分布，则适合选取斯皮尔曼相关性分析，若不符合正太分布，则选取其他的相关性分析如灰色相关分析等。所以先对数据分布使用 python 进行 Shapiro-Wilk 检验。

H_0:样本所来自的总体分布服从正态分布

H_1:样本所来自的总体分布不服从正态分布

当计算得到的显著性概率 p 值>0.05 时，无法拒绝 H0，即代表样本总体服从正太分布，计算结果如表 3 所示，所有品类的日销售量分布均通过检验，符合正太分布。

表 3 各品类的 Shapiro-Wilk 检验

指标	**花叶类**	**水生根茎类**	**花菜类**	**茄类**	**辣椒类**	**食用菌**
Shapiro-Wilk	0.98	0.95	0.94	0.97	0.94	0.94
p 值	0.85	0.14	0.09	0.72	0.06	0.06
显著性	**显著**	**显著**	**显著**	**显著**	**显著**	**显著**

后续对数据进行斯皮尔曼相关性分析，其使用单调方程评价两个统计变量的相关性。Spearman（斯皮尔曼）相关系数（秩相关系数），又称斯皮尔曼阶数。其方法是采用单调性公式来衡量两组间的相关关系。定义式：

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{1}$$

其中，r_s表示斯皮尔曼相关系数，n表示样本大小，d_i表示两个变量的排名之差。如果两个变量的排名差异较大，则d_i的值也会较大，反之则较小。r_s的取值范围为[-1，+1]，其中，-1 表示完全的负相关，0 表示无相关，+1 表示完全的正相关。计算得到相关系数热力图如所示。

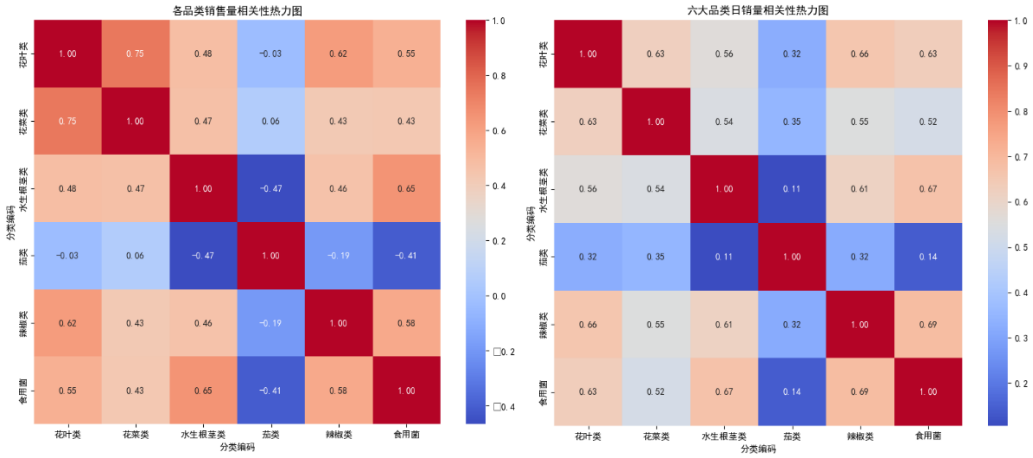


图 4 品类 spearman 相关系数（左日销量，右月销量）

考虑到日销量的波动性较大，于是我们再次对日销量进行汇总得到月销量，并对其进行斯皮尔曼相关性分析，得到月销量的热力图如图 4 的右图。根据日销量与月销量的相关性热力图对比可知，两者相关性情况基本一致。可以得知茄类与其他品类在日销量上为无关或者负相关，而月销量的相关性情况下，基本所有品类都是正相关。仅茄类的情况存在差异。我们认为可能时由于茄类本身的单品较少，且销量总体最少，导致日销量存在较大波动。

仅仅分析销量数据分布的相关情况，不足以体现其在时间上变化的相关情况，因为日销量的波动性较大，因此我们统计月销量的数据更具有统计意义，得到 20 年 7 月到 23 年 6 月的月销分布情况如图 5。

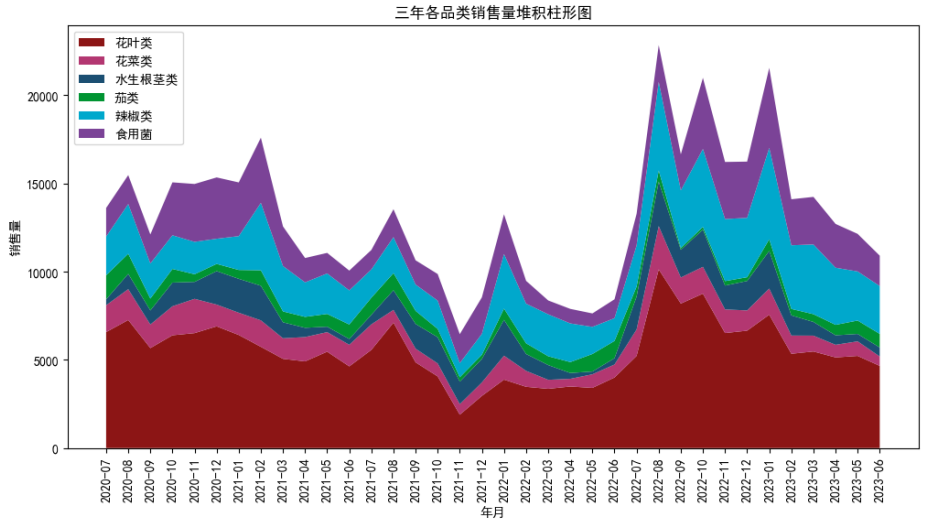


图 5 品类月销量分布

由各品类月销量与时间的时序情况得知：总体销售情况在 21 年 8 月后开始跌落，至 21 年 11 月达到谷底，了解过去情况我们可以得知时由于该时间点德尔塔毒株肆虐，导致全国的市场恐慌，经济停滞。而后续的增长和跌落正好对应着该特殊点结束后的报复性消费和消费热情退热。再后续的销量高峰正对应着疫情防控的放松，大家恢复正常生活后的经济复苏。

6.1.2 蔬菜单品间的关系

蔬菜单品经过前文的数据预处理进行晒除后仍具有较多的种类（132 个），先对其根据品类分别绘制日销量和月销量的箱形图分布情况如

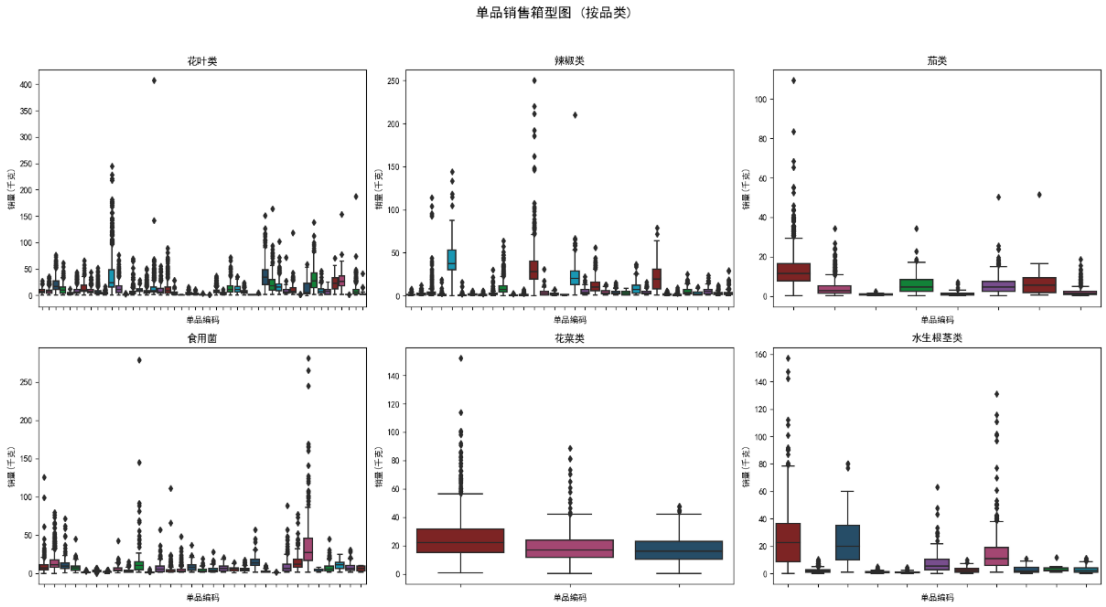


图 6 单品日销量分布

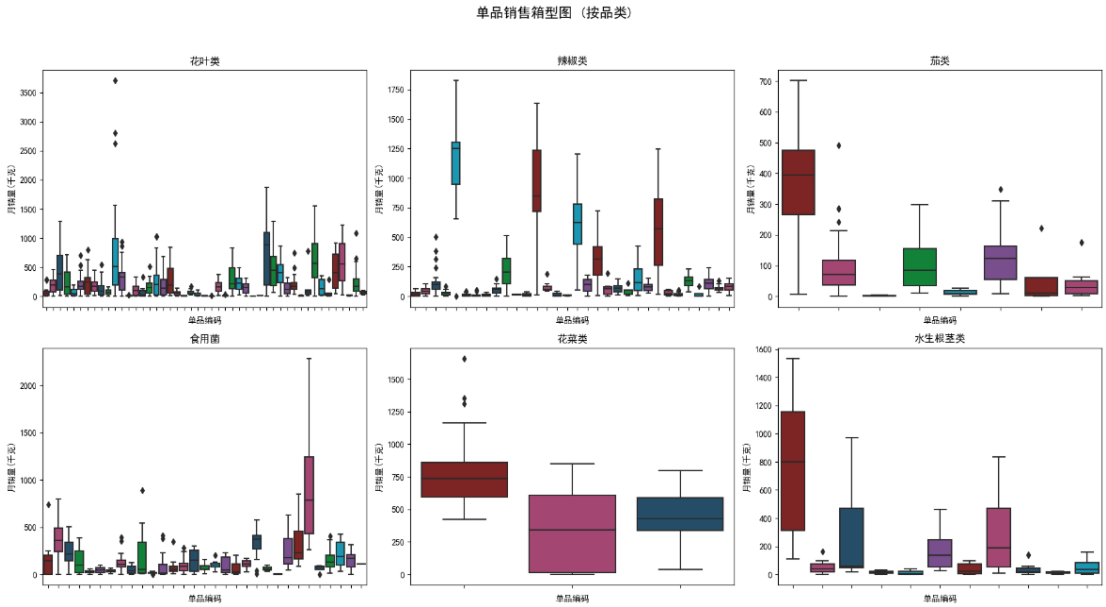


图 7 单品月销量分布

由上图分析可知，花叶类的总销量主要由 1/6 的单品贡献，辣椒类仅 7 种单品销

量较大，茄类的销量主要集中在其某一单品上，花菜类仅剩下三个单品，是由于在数据预处理时已经将其销量不多的种类晒除。可以直观的了解到各单品的销量分类情况，也客观体现了其部分的重要程度。

分析单品间的相关性时，对日销量的相关依赖情况使用互信息度进行度量，互信息是信息论的度量，可以衡量两个变量的依赖程度。对离散或连续变量皆适用，且不受线性关系的限制，较高的互信息值表示较强的关联[5]。

两个变量的互信息可以定义为：

$$I(X;Y)=\sum_{y\in Y}\sum_{x\in X}p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right) \tag{2}$$

其中 p(x,y)是 X 和 Y 的联合概率分布函数，而 p(x)和 p(y)分别是 X 和 Y 的边缘概率分布函数。在连续随机变量的情形下，求和被替换成了二重定积分：

$$I(X;Y)=\int_Y\int_X p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)dxdy \tag{3}$$

其中 p(x,y)当前是 X 和 Y 的联合概率密度函数，而 p(x)和 p(y)分别是 X 和 Y 的边缘概率密度函数。平均互信息 I(X;Y)是一个确定的量。如果对数以 2 为基底，互信息的单位是 bit。

使用 python 进行计算后得到单品间的互信息度如下表 4（展示部分，附录无法放下，详细见支撑文件）

表 4 单品信息度结果

单品 1	单品 2	互信息度（bit）
云南生菜(份)	云南油麦菜(份)	0.722149672
杏鲍菇(1)	金针菇(1)	0.605331524
青线椒	红椒(1)	0.561014147
大白菜	杏鲍菇(1)	0.575189985
大白菜	金针菇(1)	0.566359246
云南生菜	云南油麦菜	0.551502572

绘制互信息度最高的单品对，云南生菜(份)和云南油麦菜(份)的日销量折线图如图 8 可以直观观察到两者的变化情况具有较高的一致性（波动与涨幅）。

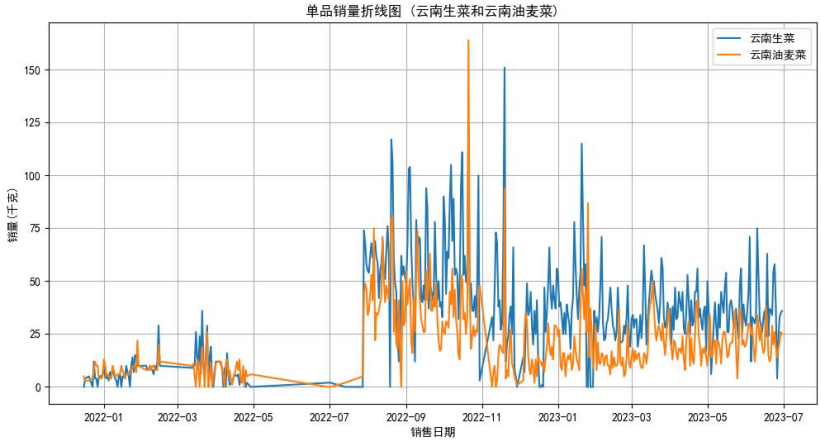


图 8 最高互信息度单品相关情况

6.1.3 数据挖掘-购买协同性

之前的信息提取分析局限于商超，而当前所做的，是处于消费者的角度进行购买行为的分析，分析消费者购买蔬菜单品时，对于其他商品的偏好性分析。

对此本文采用 Apriori（关联分析算法）进行，关联分析即关联挖掘，其在信息载体中，查找存在于项目集合或对象集合之间的频繁模式、关联、相关性或因果结构。属于无监督学习。基于频繁项集理论的递归方法，采用逐层搜索的迭代方法挖掘出所有频繁项集，直至找到最高阶频繁项集即止，最后通过对获得的频繁项集进行计算得到强关联规则[2]。

在计算之前，我们需要从原始数据中提取出每个消费者的购买情况数据，本文了解到收银台扫码枪的工作原理，且允许人工操作的误差，大概两次扫码时间间隔小于 1 秒表明这两次扫码的商品为同一个人的购买物。另外通过计算附件二中的扫码时间间隔得到　的直方统计图与累计频率统计图，由于日常中扫码间隔大于 5 秒的情况基本为不同人的物品扫码，所以设置时间间隔统计上限为 5 秒。

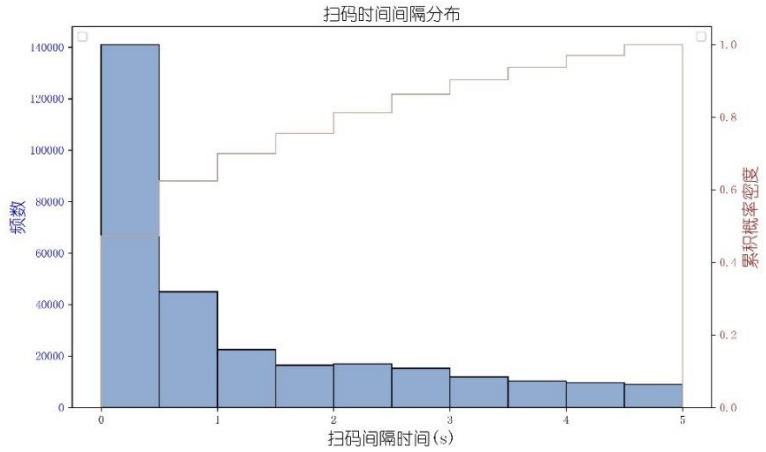


图 9 扫码时间间隔分布

根据论文[4]的研究可知，居民消费行为特征中，78%的居民会购买两件及以上的商品，适用于本文的判断，即扫码间隔时间计数应占总体比 78%左右，据可知 1s 的分位点大致位于 76%左右，该结论也与前文的估测一致，证明了其定义的合理性。

通过设置时间间隔限制，对于扫码时间间隔超过 1s 的，本文认为其不在是同一人购买的物品。所有根据此逻辑使用 python 对消费者的购买情况进行数据提取。设置最小支持度为 0.05，搜索项集及其支持度，剪枝去掉低于阈值 0.05 的项集，得到频繁项集，并依次迭代下去。

具体操作流程如下:

(一)算法首先简单扫描所有的数据,数据中的每一个化学成分皆为候选 1 项集的成员,即构成初始集合 C1,并计算每一项的支持度。

(二)随后,对 C1 中各项集的支持度与预先设定的最小支持度阈值进行比较,保留支持度大于或等于阈值的项集,得到 1 项频繁集 L1。

(三)然后,扫描全部数据,将 L1 中各项与其自身连接,生成候选 2 项集 C2,并计算每一项的支持度。进行剪枝处理,即剔除 C2 中非频繁集的子集。

(四)接着,对 C2 中各项集的支持度与最小支持度阈值进行比较,保留支持度大于或等于阈值的项集,得到 2 项频繁集 L2。

(五)后续重复第(三)步的操作,不断循环,直至得到最大频繁项集。本题中,我们规定

最小支持度阈值为 0.05,最终得到最大频繁项集为 2 项集。

(六)最后,由频繁项集生成关联规则。

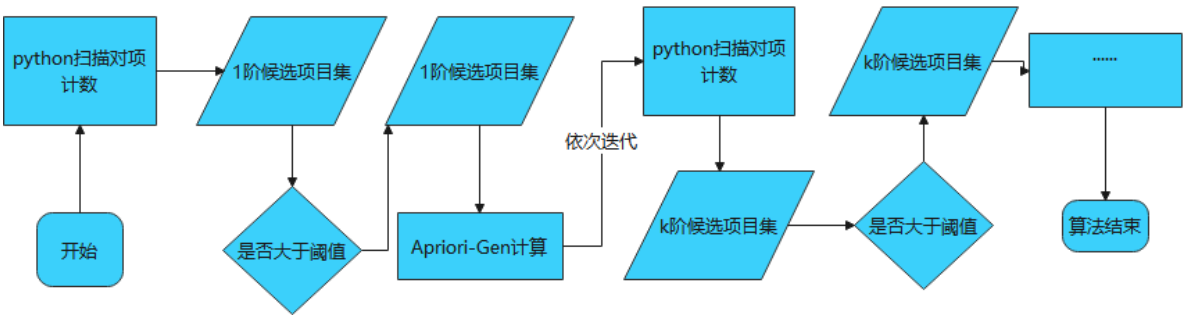


图 10Apriori 算法流程

以下是我们得到的各蔬菜单品之间的支持度,置信度,提升度值:

表 5Apriori 算法求解结果

antecedents	consequents	antecedent support	consequent support	support(支持度)	confidence(置信度)	lift
西峡香菇(1)	云南生菜	0.1698	0.1378	0.0256	0.1506	1.0930
云南生菜	西兰花	0.1378	0.2115	0.0304	0.2208	1.0436
净藕(1)	云南生菜	0.1451	0.1378	0.0203	0.1397	1.0142
云南生菜	芜湖青椒(1)	0.1378	0.2392	0.0303	0.2199	0.9193
.....	.....	.....	.....	.....	.....	.....
净藕(1)	西兰花	0.1451	0.2115	0.0334	0.2303	1.0889
西兰花	芜湖青椒(1)	0.2115	0.2392	0.0505	0.2389	0.9991

分析以上的数据，得到消费者购买蔬菜单品的协同偏好，这将有利于后续我们对进货和定价策略更加全面合理的制定，使得模型分析更加符合实际情况。

6. 2 问题二模型的建立与求解

6. 2. 1 建模前的准备

根据日常经验和题干中的相关背景信息，可得知利润的影响因素的选取原则应为：系统性，独立性，可测性和科学性。

根据此原则，分析可能从数据中得到的影响因素，本文从数据中挖掘到损耗率，打折力度，批发价，销售量，定价与销量的信息。

- 损耗率代表着运输货物损耗的部分，这部分会进行打折销售，对利润有影响
- 打折力度即代表着一般蔬菜品类打折程度的大小，对不同蔬菜有区分度
- 批发价即成本部分，根据“成本加成定价”原则，其会影响定价
- 销售量为市场需求，本文认为市场需求是应该处于一个波动的范围内

- 定价与销量的关系分析，从此可以以销量需求确定定价策略

6.2.2 相关数据的挖掘

a. 损耗率

损耗率的计算，是基于附件四与附件一的数据进行品类的归纳，求得平均损耗率。

$$S_i = \frac{\sum_{j=1}^{n} S_{ij}}{n} \tag{4}$$

其中S_i为第i蔬菜品类的平均损耗率，$i=1,\cdots,m$，m为品类总数，即为6。S_{ij}为某一蔬菜单品最近的损耗率，n为某一品类所包括的单品总数。计算得到每一蔬菜品类的损耗率分别为 10.2803%，14.142%，11.9747%，7.122%，8.5153，8.13097%。

b. 打折力度

打折力度的计算，是基于该品类所包含的单品在打折销售时的价格为定价的百分比的平均值。

$$\begin{cases} G_i = \dfrac{\sum_{j=1}^{n} G_{ij}}{n} \\ G_{ij} = \dfrac{P_{ij} - P_{ij}'}{P_{ij}} \times 100 \end{cases} \tag{5}$$

其中G_i为第i蔬菜品类的打折力度，$i=1,\cdots,m$，m为品类总数，即为6。G_{ij}为某一蔬菜单品的单日打折力度，n为某一品类当日打折的单品总数。P_{ij}为该单品的当日定价，P_{ij}'为该单品打折时的售价。计算得到每一类蔬菜的平均打折力度为69.9937%，67.5847%，71.8194%，81.3046%，67.2934%，67.256%。

c. 批发价

根据附件三可以得到以往每日批发进价，根据文献[4]的调查发现，蔬菜批发进价的变化与每日蔬菜供应量有关，此供应量取决于菜农。而实际生活中，蔬菜授粉与成熟总是几天一批一批为周期，因此本文决定根据所有品类过去三年的每日批发价进行单序列时间序列预测。

最小二乘回归树，是一种基于梯度提升树的回归方法[6]，不同于神经网络等黑箱模型，这种树模型的解释性较高，其目的为使平方误差最小的树结构模型，其通过依次对各特征的取值进行遍历，并计算出当前每一个可能的切分点误差，最后选择切分误差最小的点，并将输入空间切分为两部分，递归上述步骤，直到切分结束，得到最终模型。

本文先对每个品类各自的每日平均批发价进行计算

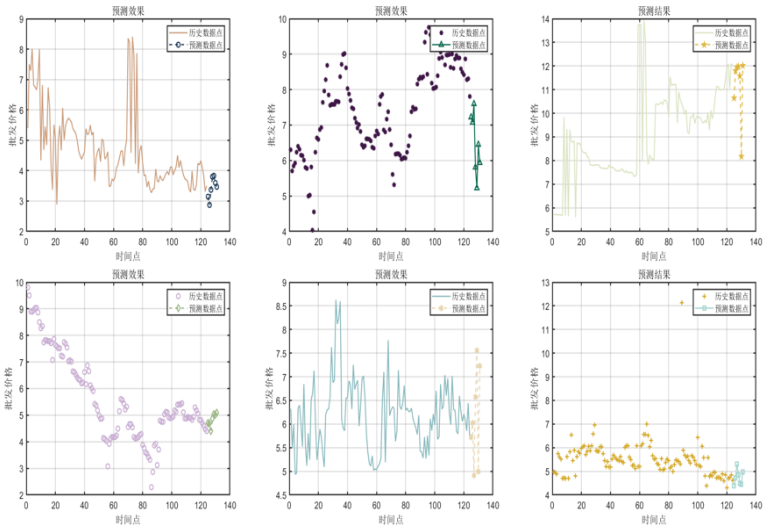
$$O_i = \frac{\sum_{j=1}^{n} O_{ij}}{n} \tag{6}$$

其中O_i为第i蔬菜品类的平均损耗率，$i=1,\cdots,m$，m为品类总数，即为6。O_{ij}

为某一蔬菜单品最近的批发价格，n 为某一品类所包括的单品总数。

将 O_i 时间序列构造进行训练集合的构建，使用连续历史数据的十个点，对后续一个周的批发价格进行预测。使用 matlab 的工具箱[10]进行计算，可以直观的得到预测结果如

表 6 预测后续一周的批发价格



模型的详细信息如表 7 所示，可以得知每一蔬菜品类的拟合优度皆在 0.99 以上，说明模型具有良好的回归效果。

表 7 最小二乘回归树结果

LSBoost	花菜类	花叶类	辣椒类	茄类	食用菌类	水生根茎
MAE	0.056682	0.019202	0.033005	0.026707	0.030952	0.025744
MSE	0.0061384	0.00066009	0.0019882	0.0013495	0.0016642	0.0011848
RMSE	2.417	0.025692	0.04459	0.036736	0.040794	0.034421
R^2	0.99973	0.99987	0.99996	0.99988	0.99969	0.99988
未来 7 天预测批发价格	10.6448	3.1431	5.7174	4.6973	4.3784	7.2234
	11.7968	2.8713	6.0211	4.6759	4.7261	7.0769
	11.9516	3.3694	4.9116	4.4119	5.3043	7.5989
	11.9764	3.7993	6.5723	4.8605	4.8524	5.8097
	11.5712	3.8319	7.5563	5.0261	4.4978	5.2242
	8.1742	3.5956	4.9955	5.0159	4.4557	6.4515
	12.0158	3.4676	7.2284	5.0966	4.9757	5.9376

d. 销售量

作为补货策略的一大影响因素，考虑到销售量可能存在每年的相似规律，且考虑到近期的销量情况，本文同时考虑季节因素和近期影响，使用每年的 6 月 24 到 7 月 8 日的销量数据进行高斯过程回归分析（其合理性在于考虑到销量的波动性），使用高斯过程回归的置信区间表征销量波动误差。高斯过程回归是使用高斯过程对数据进行回归分析的非参数模型。

令随机向量 $X=[x_1,x_2,...,x_n]$ 服从多元高斯分布 $X\sim N(\mu,\Sigma)$，其中：$X_1=[x_1,...,x_m]$ 为已经观测到的变量，即之前我们提到的往年的销量数据，

$X_2=[x_{m+1},...,x_n]$为未知数据，即今年 7 月 1 日到 7 月 7 日的销量预测数据。

$$\begin{cases} X=\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \\ \mu=\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \\ \Sigma=\begin{bmatrix} \Sigma_{11}, & \Sigma_{12} \\ \Sigma_{21}, & \Sigma_{22} \end{bmatrix} \end{cases} \tag{7}$$

最终得到销售量的波动函数为

$$\begin{cases} x_i=G_i(z_{i,t}) \\ G_i(z_{i,t})-N(\mu_i(z_{i,t}),{\sigma_i}^2(z_{i,t})) \end{cases} \tag{8}$$

其中,x_i表示为销售量的偏差函数,偏差函数通过高斯过程回归对往年这个时间段的销量拟合得到,能够有效反映供货商的供贷特点。使用 matlab 工具箱[10]进行模型拟合，得到销量波动情况如图 11 所示，可以看到其直观的体现了各品类日销量的可能波动情况。

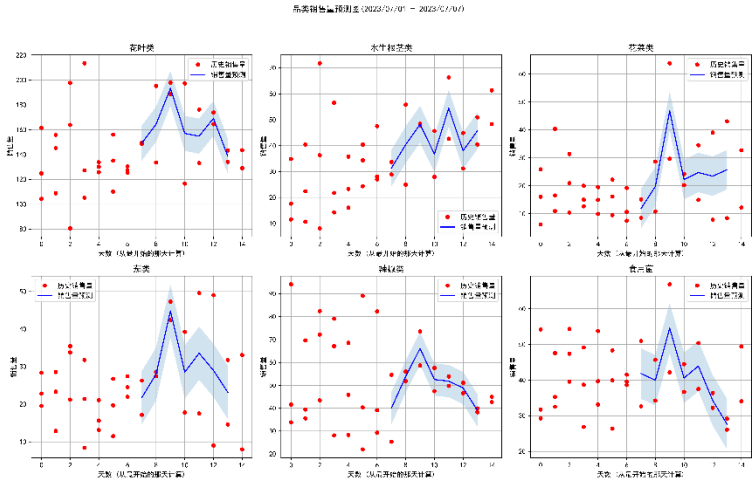


图 11 销量波动情况

根据偏差函数我们可以得到预测销售量上下波动 95%的置信区间上下限如表 8 所示。、

表 8 销量波动上下限

日期	花叶类		花菜类		辣椒类		茄类		食用菌类		水生根茎类	
	上限	下限	上限	下限	上限	下限	上限	下限	上限	下限	上限	下限
7 月 1 日	163.3	135.0	45.4	17.2	25.9	2.3	35.8	7.6	54.1	25.9	55.9	27.7
7 月 2 日	178.5	150.3	54.5	26.3	33.8	5.6	42.1	13.9	68.0	39.8	54.1	25.9
7 月 3 日	207.4	179.2	62.3	34.1	60.9	32.7	59.0	30.7	80.2	52.0	68.6	40.4

7月4日	171.1	142.9	50.9	22.7	36.3	8.1	42.6	14.4	66.6	38.4	54.7	26.5
7月5日	168.9	140.7	68.6	40.4	38.8	10.6	47.7	19.4	65.9	37.7	58.0	29.8
7月6日	183.3	155.0	52.2	23.9	37.5	9.3	43.1	14.9	62.9	34.7	48.4	20.2
7月7日	152.8	124.6	59.9	31.6	39.8	11.6	37.3	9.0	53.2	24.9	41.8	13.5

e. 定价与销量

基与题干的“成本加成定价”法，本文不仅只考虑销售总量与成本加成定价的关系，我们认为成本加成定价还应与批发价相关联。建立二元线性回归方程，使用个性化 matalb 工具箱[10]得到各蔬菜品类加成定价与销量和批发价之间的关系如表 9。

表 9 品类定价拟合函数

蔬菜品类	拟合方程	R^2
花叶类	$y=-0.0637x_i+0.9326O_i$	0.91682
花菜类	$y=0.015x_i+0.9708O_i$	0.93457
辣椒类	$y=0.0185x_i+0.9832O_i$	0.95816
茄类	$y=-0.0329x_i+0.9151O_i$	0.84948
食用菌类	$y=-0.0405x_i+0.9411O_i$	0.90597
水生根茎	$y=0.0196x_i+0.9315O_i$	0.85502

而从此结果我们可以明确的知道其可能存在错误，因为其定价为实际的批发价格的 95%左右，且有三类甚至与销量成反比，这是明显的错误。我们推测其原因可能是由于从数据二并未给出批发量，所以我们单从销售流水中推导品类批发价格，会导致推导出的平均批发价格必然比实际批发价格高，才导致拟合方程呈现如此的状态，因此，我们对其进行修正，根据少量数据的计算，得到的修正结果如下。

表 10 修正后的拟合方程

蔬菜品类	拟合方程
花叶类	$y=-0.00637x_i+1.3326O_i$
花菜类	$y=0.015x_i+1.2708O_i$
辣椒类	$y=0.0185x_i+1.1832O_i$

茄类	$y=-0.00329x_i+1.4151O_i$
食用菌类	$y=-0.00405x_i+1.3411O_i$
水生根茎	$y=0.0196x_i+1.2315O_i$

根据表 10 可知，所有品类的定价皆大程度上与成本价成一定比例，成本价对其影响更大，这也符合题干的“成本加成定价”策略。另外可以发现的是，花菜类，茄类，食用菌类与销量之间的关系为反比，可能是由于根据以往数据可知其销量占比并不大，即市场对其的需求较小，如果补货过多，会导致其售价反而降低。而花叶类，辣椒类，水生根茎类由于其种类多样，成正比的可能原因是定义售价时考虑了销量的占比程度进行非均等的平均售价集合的影响。

6.2.3 最优策略模型的建立

根据前文相关数据的挖掘，已经获取到了构建规划决策模型的所有变量，因此此段开始建立规划模型。由于每一天的数据都已获取，因此其使用的是相同的限制，文中展示其通用模型。

a. 决策变量

对于每一天的情况，我们有六个决策变量$x_1,\cdots,x_6$，即每一日六个品类的补货量。

b. 目标函数

最大化总利润，其中总利润是每日中各品类的利润之和，利润由以下公式计算

$$h_{ij}=\sum_{i=1}^{m}\left(y_{ij}\cdot x_{ij}\cdot(1-S_i)+y_{ij}\cdot G_i\cdot x_{ij}\cdot S_i-x_{ij}\cdot O_i\right) \tag{9}$$

其中，h_{ij}是总利润,O_i是已知的批发价，S_i是已知的损耗率，G_i是已知的打折率,x_{ij}为当日的向量，其中$j=1,\cdots,7$为天数。

c. 限制条件

1. 品类当日的总体销售量受到上限约束，本文选取前一周中的日总销量的最大值作为上限，其合理性为：在当周的销售情况较低迷，销售量呈现轻微的下降趋势，应该设置上限。

$$\sum_i x_{ij}\le 365 \tag{10}$$

2. 单个品类单日销售量受到前文使用高斯过程回归预测的上下限约束

$$\max G_i(z_{i,t})\ge x_{ij}\ge \min G_i(z_{i,t}) \tag{11}$$

3. 另外还需符合前文分析得到的定价与批发价格和销量之间的关系

$$y_{ij}=a\cdot x_{ij}+b\cdot O_{ij} \tag{12}$$

其中 a 和 b 是已知相关系数。

4. 所有决策变量y_{ij}和x_{ij}必须是非负数

$$x_{ij}\ge 0 \tag{13}$$

所以最终模型为

$$
\begin{aligned}
Max\ h_{ij} = \sum_{i=1}^{m} & \left(y_{ij} \cdot x_{ij} \cdot (1 - S_i) + y_{ij} \cdot G_i \cdot x_{ij} \cdot S_i - x_{ij} \cdot O_i \right) \\
s.t. \quad & \sum_i x_{ij} \leq 365 \\
& \max G_i(z_{i,t}) \geq x_{ij} \geq \min G_i(z_{i,t}) \\
& y_{ij} = a \cdot x_{ij} + b \cdot O_{ij} \\
& x_{ij} \geq 0
\end{aligned}
\tag{14}
$$

使用 python 的 linprog 函数进行求解，得到最终 7 月 1 日到 7 月 7 日一周的补货量和定价的最优策略如下表所示。

表 11 最大利润定价策略

日期	花叶类		花菜类		辣椒类		茄类		食用菌类		水生根茎类		
	进货量	定价	进货量	定价	进货量	定价	进货量	定价	进货量	定价	进货量	定价	最优总收益
7 月 1 日	163.27	13.15	17.18	4.25	10.20	6.95	7.58	6.62	25.86	5.77	55.91	13.35	732.90
7 月 2 日	168.56	14.65	26.28	4.04	5.62	7.23	13.89	6.57	39.78	6.18	25.86	10.77	635.95
7 月 3 日	191.17	14.71	34.09	4.79	32.65	6.42	24.70	6.16	46.98	6.92	35.40	12.18	779.85
7 月 4 日	169.94	14.88	22.68	5.17	8.11	7.93	14.39	6.83	38.41	6.35	26.47	9.26	610.60
7 月 5 日	142.07	14.51	40.36	5.47	10.62	9.14	19.43	7.05	37.69	5.88	29.83	8.81	606.97
7 月 6 日	176.98	9.77	23.91	4.93	9.33	6.08	14.87	7.05	34.72	5.83	20.19	9.55	590.22
7 月 7 日	152.78	15.04	31.60	4.88	39.85	9.29	9.04	7.18	24.95	6.57	21.79	10.34	589.87

此策略得到的总收益与过去一周（6 月 24 到 6 月 30）的收益相差不大，一定程度上可以说明策略的合理性。

6.3 问题三模型的建立与求解

针对问题三，其与问题二相似，但重点聚焦在单天的各单品的补货策略，我们决定问题三中单品的打折力度，定价与批发价格和销量函数表达式根据其所属品类于问题二中得到的信息匹配（同一个品类具有类似的销售策略和定价策略），损耗率则直接索引附件四中的数据，而批发价则使用前一个周的平均批发价格。

6.3.1 单品的筛选

读取了文件筛选了 2023/6/24-2023/6/30 的数据，共统计出 49 种单品。将这些数据按单品统计每天销售量，去掉这七天中 3 天以上销量为 0 或者 7 天中销量小于 2.5 的天数大于 5 天的单品得到了 31 种单品，将青茄子和虫草花单品的 2023/6/24-2023/6/30 日销售量数据进行构图，从中得到了时间与其这七日销量的折现统计情况，根据此判断出青茄子由于一直处于递减趋势需要被淘汰，而虫草花由于处于波动需要保留进规划中，最后共留下 30 种单品进行规划求解。

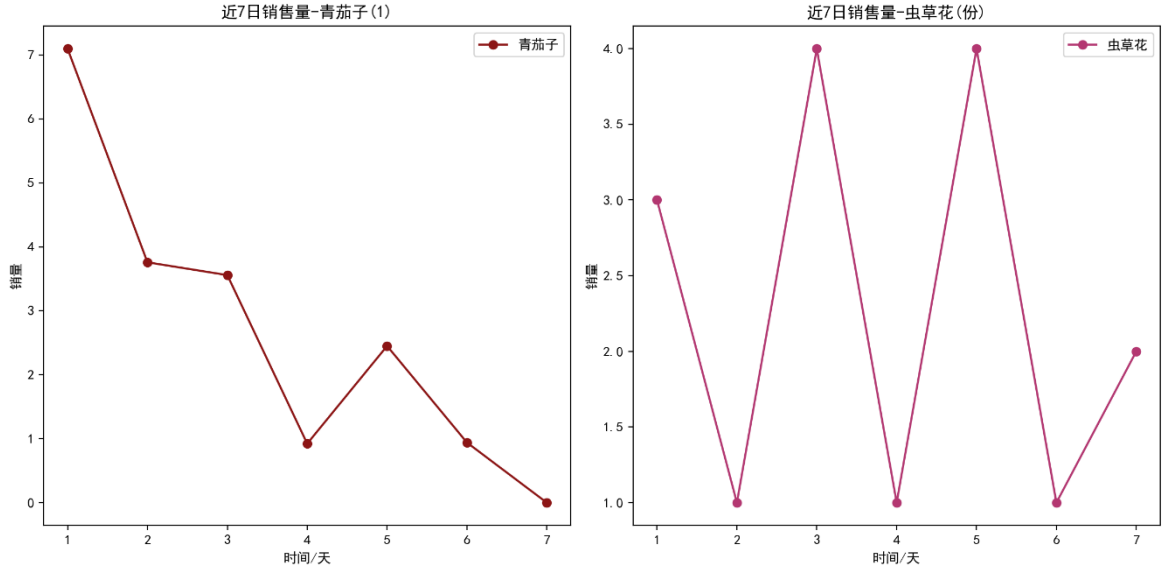


图 12 筛选策略

6.3.2 凸优化模型的建立

1) 决策变量

30 个决策变量$x_1,\cdots,x_{30}$，即 7 月 1 日 30 个单品的补货量。

2) 目标函数

要最大化的目标函数是 30 个单品的利润之和，可以表示为

$$\sum_{i=1}^{30}(y_i\cdot(1-S_i)\cdot x_i+y_i\cdot S_i\cdot x_i\cdot G_i-x_i\cdot O_i) \tag{15}$$

其中的变量在问题二中皆已阐述过，只不过此处为蔬菜单品的相关指标。

3) 限制条件

每个单品至少补货 2.5Kg。

$$x_i>2.5 \tag{16}$$

每个品类的销售量（即其包含的单品销售总量）应该在问题二所求的总销量波动上下限内。

$$\max G_i(z_{i,t})\geq\sum x_i\geq\min G_i(z_{i,t}) \tag{17}$$

其中的$(x_1\cdots x_{11})$为花叶类单品，$(x_{12}\cdots x_{18})$为辣椒类单品，$(x_{19}\cdots x_{23})$为食用菌类，$(x_{24}\cdots x_{27})$为水生根茎类，x_{28}，x_{29}茄类，x_{30}为花菜类。

当日的总体销售量受到上限约束，本文选取前一周中的日总销量的最大值作为上限，其合理性为：在当周的销售情况较低迷，销售量呈现轻微的下降趋势，应该设置上限。

$$\sum_i x_{ij}\leq 365 \tag{18}$$

另外还需符合前文分析得到的定价与批发价格和销量之间的关系，单品的定价关系由其所属品类决定。

$$y_i=a\cdot x_i+b\cdot O_i \tag{19}$$

6.3.3 最终模型

所以最终模型为

$$MAX\sum_{i=1}^{30}(y_i\cdot(1-S_i)\cdot x_i+y_i\cdot S_i\cdot x_i\cdot G_i-x_i\cdot O_i)$$

$$\begin{aligned}
&s.t.\ x_i>2.5\\
&135<\sum(x_1\cdots x_{11})<163.3\\
&2.3<\sum(x_{12}\cdots x_{18})<25.9\\
&25.9<\sum(x_{19}\cdots x_{23})<54.1\\
&27.7<\sum(x_{24}\cdots x_{27})<55.9\\
&7.6<x_{28}+x_{29}<35.8\\
&17.2<x_{30}<45.4\\
&\sum x_i<365\\
&i=1\cdots30
\end{aligned}\tag{20}$$

由于决策变量的数目较大，且搜索范围广，如果只是基础的 linprog 求解很容易陷入局部最优，因此我们决定使用智能算法进行寻优。禁忌搜索是一种策略性的随机搜索算法，它从一个初始解开始，针对特定目标函数的最大变化，选择一系列特定的搜索方向（即移动）。为了防止陷入局部最优解，禁忌搜索采用一种智能的"记忆"技术，记录已经探索过的优化路径，并基于这些信息来指导下一步的搜索方向。这个方法旨在提高搜索的多样性，以获得更好的全局解。

使用 python 进行禁忌搜索算法的流程如图 13 所示。

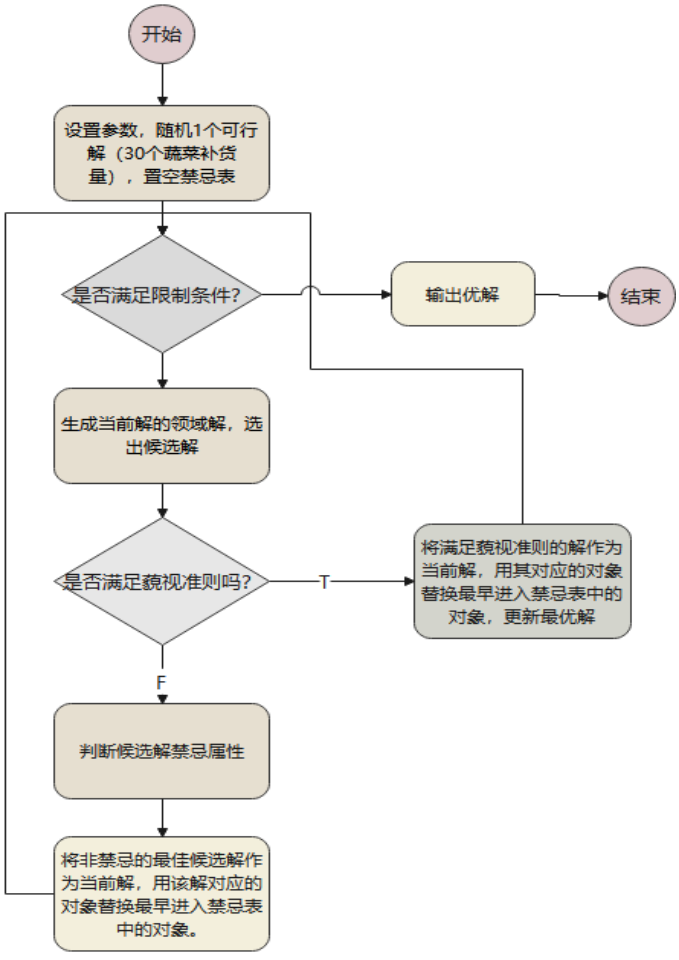


图 13 禁忌搜索流程图

最终决策得到的 7 月 1 日补货与定价策略如表 12，得到最大收益为 708.7 元，与第二问结果较为相近。

表 12 最优补货和定价策略

单品	西峡花菇(1)	苋菜	竹叶菜	上海青	木耳菜	紫茄子(2)	西兰花	净藕(1)	高瓜(1)	娃娃菜
销量	19.00	15.00	8.00	10.00	23.00	11.00	14.00	10.00	6.00	20.00
定价	20.84	3.00	3.04	5.43	4.13	5.27	10.15	13.43	14.33	6.18
单品	红薯尖	螺丝椒	菱角	奶白菜	芜湖青椒(1)	长线茄	小青菜(1)	云南生菜(份)	云南油麦菜(份)	菠菜(份)
销量	3.20	7.52	9.17	2.53	3.38	6.98	2.83	3.60	2.86	4.10
定价	4.14	9.06	11.34	3.33	4.28	9.84	3.73	4.73	3.75	5.33
单品	小米椒(份)	虫草花(份)	小皱皮(份)	螺丝椒(份)	姜蒜小米椒组合装(小份)	双孢菇(盒)	青红杭椒组合装(份)	洪湖藕带	金针菇(盒)	海鲜菇(包)
销量	2.14	2.66	1.54	3.28	2.44	3.40	3.23	18.00	1.45	1.95
定价	2.67	3.50	2.14	4.17	2.98	4.52	3.95	22.48	1.91	2.58

6.4 问题四模型的建立和求解

问题四的重心在于根据问题二，三的模型，分析还需要采集什么数据可以使得策略制定更加的合理真实。我们认为还需采集其它成本，每日批发量，商超自身可容纳的容量上限。

6.4.1 其它成本

定价的设置并非只关注于进价成本，还应该有其它的经营成本，根据以上模型假设后，我们认定利润为销售额与蔬菜成本的差值，但实际商超运营过程中还伴随着很多的其他成本，这些我们并未考虑在模型，所以使得我们的模型是理想化的。因此可以收集到每日的其它成本a，对定价拟合方程进行优化。

$$y = f(x_i, O_i, a) \tag{21}$$

使得定价与销售量，批发价，其他成本之前可能扩展为更加复杂的关系，实现对模型的延伸。

6.4.2 每日批发量

从问题二建立对定价与销量和批发价的多元线性回归模型可以得知，因为没有每

日批发量的数据，导致我们以销量占比作为权重对批发价格获取品类的平均批发价格与实际的平均批发价格是有出入的，并经过我们简单的推断，所求得的均价是高于实际的，所以我们对得到的回归方程修正。

因此从存在的问题可以得知，每日批发量数据的采集，可以使得我们对一个品类的均价计算更加符合实际。另外这部分还有助于对市场需求更加精确的预测，因为所给的数据并未告诉我们是否留有余量以及余量有多少，这一部分未售卖，将导致亏损。

6.4.3 商超容量上限

所给数据并为给出商超的容量上限，而实际的商品补货还需考虑到容纳空间上限的情况，我们所建立的模型的销量上限皆是高斯过程回归的置信上限，这只是最极端的波动情况，但实际应该还要考虑到商超自身容量的上限。

七、 模型的评价、改进与推广

7.1 模型的优点

(1)根据销售流水数据挖掘出了较多与利润相关的影响因素，有较强的创新性。
(2)多方面的因素考虑，所预测结果更加符合实际情况。
(3)引入了波动函数的概念，对未来销量进行了科学合理的预测。
(4)使用了最小二乘树算法对时间序列进行拟合，不同于神经网络黑盒模型，树模型解释性更高。

7.2 模型的缺点

(1)提取品类的平均批发价格加权存在小问题，在正文模型中已修正，修正后结果良好，但缺乏修正逻辑支持。
(2)多个指标的获取涉及太多模型，有些复杂。
(3)模型未考虑单日内因为品相变差导致的打折情况

7.3 模型的推广

(1)可以收集问题四中提到的数据，对模型进一步优化
(2)另外我们的模型不仅仅只使用与商超策略的分析，还可以用在运货公司等的决策分析。
(3)考虑一天内的品相变化导致的售卖策略改变，进一步完善模型。

八、 参考文献

[1] 李晓璐,周曙光.我国生鲜商超零售业发展问题研究[J].商业经济研究,2021(23):35-37.

[2] 王朝. 基于 Apriori 的 S 公司产品质量诊断与分析[D].河北科技大学,2022.DOI:10.27107/d.cnki.ghbku.2022.000660.

[3] 高永亮.基于消费心理与行为的超市市场营销策略探析[J].现代经济信息,2019(08):165.

[4] 陆鸣,夏倩,吴军.无锡市蔬菜价格特征、问题及对策[J].长江蔬菜,2018(22):76-78.

[5] Pavlo G,Andrej A,Pavel B, et al. Mutual information prediction for strongly correlated systems[J]. Chemical Physics Letters,2023,813.

[6] 王如冰,蔡喜运.基于梯度提升回归树的有机污染物生物-沉积物积累因子预测模型 [J/OL]. 生 态 毒 理 学 报 :1-13[2023-09-09].http://kns.cnki.net/kcms/detail/11.5470.X.20230808.1359.002.html

[7] Apriori（关联分析算法）_apriori 算法_古城白衣少年 i 的博客-CSDN 博客

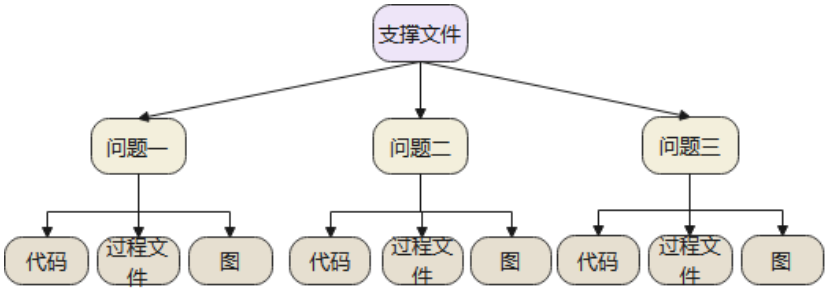
[8] https://www.zhihu.com/question/20888182

[9] 李伟才,张莉佳,张东凯.居民大型超市购物行为特征统计分析——以石家庄市为例 [J]. 江 西 科 学 ,2015,33(04):453-457.DOI:10.13990/j.issn1001-3679.2015.04.002.

[10]一键实现百种高效算法|轻松解决评价、降维、聚类、回归、分类、时序预测、多输入多输出问题|一键导出代码_哔哩哔哩_bilibili

附录

支撑文件结构：



我们的代码主要为 jupyter 环境编写的，部分图在单元格的运行结果中。

附录 1

Python 代码：绘制品类总销量图

```
import pandas as pd
import matplotlib.pyplot as plt

# 读取附件 1 的数据
attachment1_df = pd.read_excel('附件 1.xlsx')

# 创建商品编码和分类编码的映射关系
category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分类编码']))

# 读取附件 2 的数据
attachment2_df = pd.read_excel('附件 2.xlsx')

# 初始化一个字典来存储每个品类的总销量
total_sales_by_category = {}

# 遍历附件 2 中的每一行，计算总销量
for index, row in attachment2_df.iterrows():
    product_code = row['单品编码']
    sales = row['销量(千克)']

    # 查找商品编码对应的分类编码
    if product_code in category_mapping:
        category_code = category_mapping[product_code]

        # 将销量添加到对应的分类中
        if category_code in total_sales_by_category:
            total_sales_by_category[category_code] += sales
        else:
```

```
            total_sales_by_category[category_code] = sales

# 转换字典为 DataFrame
total_sales_df = pd.DataFrame(list(total_sales_by_category.items()), columns=[
'分类编码', '总销量'])

# 绘制销售量分布图
plt.figure(figsize=(12, 6))
total_sales_df.plot(kind='bar', x='分类编码', y='总销量', rot=0)
plt.xlabel('分类编码')
plt.ylabel('总销量')
plt.title('各品类总销量分布图')
    plt.show()
```

附录 2

Python:生成折线堆叠图

```
import pandas as pd
    import matplotlib.pyplot as plt

    attachment1_df = pd.read_excel('附件 1.xlsx')

    category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分
类编码']))

    attachment2_df = pd.read_excel('附件 2.xlsx')

    attachment2_df['销售日期'] = pd.to_datetime(attachment2_df['销售日期'])

    attachment2_df['  分  类  编  码  '] = attachment2_df['  单  品  编  码
'].map(category_mapping)

    monthly_sales = attachment2_df.groupby(['分类编码', pd.Grouper(key='销售日期
', freq='M')])['销量(千克)'].sum().reset_index()

    monthly_sales['年月'] = monthly_sales['销售日期'].dt.strftime('%Y-%m')

    pivot_table = monthly_sales.pivot(index='  年  月  ', columns='  分  类  编  码
', values='销量(千克)').fillna(0)

    # 设置颜色和品类标签
    colors = ['#8c1515', '#b33771', '#1b4f72', '#009432', '#00a8cc', '#7b4397'
]
```

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

    # 绘制堆积折线图
    plt.figure(figsize=(12, 6))
    stacks = plt.stackplot(pivot_table.index, *[pivot_table[column] for column
 in pivot_table.columns], labels=categories, colors=colors)
    plt.xlabel('年月')
    plt.ylabel('销售量')
    plt.title('三年各品类销售量堆积柱形图')
    plt.legend(loc='upper left', labels=categories)
    plt.xticks(rotation=90)
plt.show()
```

附录 3

Python：绘制日销量的相关系数热力图

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

attachment1_df = pd.read_excel('附件 1.xlsx')

# 创建映射关系
category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分类编
码']))

sales_data = pd.DataFrame()

attachment2_df = pd.read_excel('附件 2.xlsx')
attachment2_df['销售日期'] = pd.to_datetime(attachment2_df['销售日期'])

attachment2_df['分类编码'] = attachment2_df['单品编码'].map(category_mapping)

# 计算每天的销售量
daily_sales = attachment2_df.groupby(['分类编码', pd.Grouper(key='销售日期
', freq='D')])['销量(千克)'].sum().reset_index()

# 为了计算品类之间的相关性，将数据重新排列成以分类编码为列的形式
sales_pivot = daily_sales.pivot(index='销售日期', columns='分类编码', values='销
量(千克)')

# 计算品类之间的相关性系数
correlation_matrix = sales_pivot.corr()
```

```
# 将分类编码标签改为对应的名字
category_names = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']
correlation_matrix = correlation_matrix.rename(columns=dict(zip(correlation_ma
trix.columns, category_names)),
                                              index=dict(zip(correlation_matr
ix.index, category_names)))

# 绘制相关性热力图
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('六大品类日销量相关性热力图')
plt.show()
```

附录 4

Python：绘制品类月销量的相关系数热力图

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

attachment1_df = pd.read_excel('附件 1.xlsx')

# 创建商品编码和分类编码的映射关系
category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分类编
码']))

sales_data = pd.DataFrame()

attachment2_df = pd.read_excel('附件 2.xlsx')

# 将销售日期列转换为日期格式
attachment2_df['销售日期'] = pd.to_datetime(attachment2_df['销售日期'])

# 遍历附件 2 中的每一行，根据映射关系添加分类编码列
attachment2_df['分类编码'] = attachment2_df['单品编码'].map(category_mapping)

monthly_sales = attachment2_df.groupby(['分类编码', pd.Grouper(key='销售日期
', freq='M')])['销量(千克)'].sum().reset_index()

sales_pivot = monthly_sales.pivot(index='销售日期', columns='分类编码
', values='销量(千克)')

# 计算品类之间的相关性系数
```

```
correlation_matrix = sales_pivot.corr()
category_names = ['花叶类', '花菜类', '水生根茎类', '茄类', '辣椒类', '食用菌']
correlation_matrix = correlation_matrix.rename(columns=dict(zip(correlation_ma
trix.columns, category_names)),
                                              index=dict(zip(correlation_matr
ix.index, category_names)))

# 绘制相关性热力图
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('各品类销售量相关性热力图')
plt.show()
```

附录 5

Python:将品类月销量存为 xlsx

```
import pandas as pd

attachment1_df = pd.read_excel('附件 1.xlsx')

category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分类编
码']))

# 初始化一个空的 DataFrame 来存储销售数据
sales_data = pd.DataFrame()
attachment2_df = pd.read_excel('附件 2.xlsx')

# 将销售日期列转换为日期格式
attachment2_df['销售日期'] = pd.to_datetime(attachment2_df['销售日期'])

# 遍历附件 2 中的每一行，根据映射关系添加分类编码列
attachment2_df['分类编码'] = attachment2_df['单品编码'].map(category_mapping)

monthly_sales = attachment2_df.groupby(['分类编码', pd.Grouper(key='销售日期
', freq='M')])['销量(千克)'].sum().reset_index()

# 创建一个字典来存储每个品类的销售数据
category_sales_data = {}

# 遍历每个品类
for category_code in monthly_sales['分类编码'].unique():
    category_sales = monthly_sales[monthly_sales['分类编码'] == category_code]
    category_name = f'品类{category_code}'
    category_sales_data[category_name] = category_sales
```

```
# 创建一个 Excel 文件
with pd.ExcelWriter('品类销售数据.xlsx') as writer:
    # 将每个品类的销售数据写入 Excel
    for category_name, category_sales in category_sales_data.items():
        category_sales.to_excel(writer, sheet_name=category_name, index=False)
print("数据已存储到'品类销售数据.xlsx'")
```

附录 6

Python:绘制单品日销量的箱型图

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

# 读取单品销售数据
single_day_sales_df = pd.read_excel('single_day_sales_filtered.xlsx')

attachment1_df = pd.read_excel('附件 1.xlsx')
category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分类编码']))

single_day_sales_df['分类编码'] = single_day_sales_df['单品编码'].map(category_mapping)

# 莫兰迪配色方案
marrsala = "#8c1515"
rosin = "#b33771"
oceanblue = "#1b4f72"
emerald = "#009432"
aqua = "#00a8cc"
mauve = "#7b4397"

# 获取不同品类的唯一编码
unique_categories = single_day_sales_df['分类编码'].unique()

# 创建 6 张子图，每张子图表示一个品类
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))
fig.suptitle('单品销售箱型图 (按品类)', fontsize=16)

for i, category_code in enumerate(unique_categories):
    row = i // 3
    col = i % 3
    ax = axes[row, col]

    # 筛选特定品类的数据
```

```
    category_data = single_day_sales_df[single_day_sales_df['分类编码'] ==
category_code]

    # 绘制箱型图
    sns.boxplot(data=category_data, x='单品编码', y='销量(千克)', ax=ax,
palette=[marrsala, rosin, oceanblue, emerald, aqua, mauve])
    ax.set_xticklabels([])
    # 设置标题和标签
    ax.set_title(name2[i])
    #ax.set_xlabel('单品编码')
    ax.set_ylabel('销量(千克)')

# 调整子图布局
plt.tight_layout(rect=[0, 0, 1, 0.95])
# 显示图形
plt.show()
```

附录 7

Python:单品月销量箱型图绘制

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

single_day_sales_df = pd.read_excel('single_day_sales_filtered.xlsx')

attachment1_df = pd.read_excel('附件 1.xlsx')
category_mapping = dict(zip(attachment1_df['单品编码'], attachment1_df['分类编码']))

single_day_sales_df['分类编码'] = single_day_sales_df['单品编码'].map(category_mapping)

single_day_sales_df['销售日期'] = pd.to_datetime(single_day_sales_df['销售日期'])

single_day_sales_df['月份'] = single_day_sales_df['销售日期'].dt.to_period('M')

unique_categories = single_day_sales_df['分类编码'].unique()

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))
fig.suptitle('单品销售箱型图 (按品类)', fontsize=16)

for i, category_code in enumerate(unique_categories):
    row = i // 3
    col = i % 3
```

```
    ax = axes[row, col]

    category_data = single_day_sales_df[single_day_sales_df['分类编码'] ==
category_code]

    monthly_sales = category_data.groupby(['单品编码', '月份'])['销量(千
克)'].sum().reset_index()

    # 绘制箱型图
    sns.boxplot(data=monthly_sales, x='单品编码', y='销量(千克)', ax=ax,
palette=[marrsala, rosin, oceanblue, emerald, aqua, mauve])

    # 设置标题和标签
    ax.set_title(name2[i])
    ax.set_xticklabels([])
    ax.set_ylabel('月销量(千克)')
    #ax.set_xticklabels(ax.get_xticklabels(), rotation=45)  # 旋转 x 轴标签

# 调整子图布局
plt.tight_layout(rect=[0, 0, 1, 0.95])

# 显示图形
plt.show()
```

附录 8

Python:计算单品间的互相信息度

```
import pandas as pd
from sklearn.feature_selection import mutual_info_regression
data = pd.read_excel('single_day_sales_filtered.xlsx')
data['销售日期'] = pd.to_datetime(data['销售日期'])
unique_product_codes = data['单品编码'].unique()

date_range = pd.date_range(start=data['销售日期'].min(), end=data['销售日期'].max())

df = pd.DataFrame({'销售日期': date_range})
for product_code in unique_product_codes:
    df[product_code] = 0.0
for _, row in data.iterrows():
    df.loc[df['销售日期'] == row['销售日期'], row['单品编码']] = row['销量(千克)']

mutual_info = {}
for product_code1 in unique_product_codes:
    for product_code2 in unique_product_codes:
```

```
        if product_code1 != product_code2:
            X = df[product_code1].values.reshape(-1, 1)
            y = df[product_code2].values
            mi = mutual_info_regression(X, y)[0]
            mutual_info[(product_code1, product_code2)] = mi

# 打印互信息值
for (product_code1, product_code2), mi in mutual_info.items():
    if mi>0.5:
        print(f'互信息值 ({product_code1}, {product_code2}): {mi}')
```

附录 9

Python：绘制互信息度最大的两单品日销量折线图

```
import pandas as pd
import matplotlib.pyplot as plt

# 读取数据文件
data = pd.read_excel('single_day_sales_filtered.xlsx')

# 将销售日期列转换为日期格式
data['销售日期'] = pd.to_datetime(data['销售日期'])

# 提取单品 102900011030059 和 102900011030097 的销量数据
product_code1 = 102900011030059
product_code2 = 102900011030097
product_name1 = '云南生菜'
product_name2 = '云南油麦菜'
# 创建一个 DataFrame 来存储这两个单品的销量数据
df = data[data['单品编码'].isin([product_code1, product_code2])]
df = df.pivot(index='销售日期', columns='单品编码', values='销量(千克)').fillna(0)

# 计算销量数据之间的相关性（可以使用互信息值或皮尔逊相关系数）
correlation = df[product_code1].corr(df[product_code2])

# 打印相关性值
print(f'相关性值 ({product_name1}, {product_name2}): {correlation}')

# 绘制销量折线图
plt.figure(figsize=(12, 6))
plt.plot(df.index, df[product_code1], label=product_name1)
plt.plot(df.index, df[product_code2], label=product_name2)
plt.xlabel('销售日期')
plt.ylabel('销量(千克)')
plt.title(f'单品销量折线图 ({product_name1}和{product_name2})')
```

```
plt.legend()
plt.grid(True)
plt.show()
```

附录 11

Python：扫码时间间隔的分布图

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
matplotlib.rc("font",family='YouYuan')
# 你的数据
data = pd.read_excel('./time_intervals.xlsx')
data=np.array(data)
fig, ax1 = plt.subplots(figsize=(10, 6))
bins = np.arange(0, 5.5, 0.5)
color=np.array([146,172,209])/255
ax1.hist(data, bins=bins, edgecolor='k',density=False,color=color)
# 添加标题和标签
ax1.set_xlabel('扫码间隔时间(s)',fontsize=15)
ax1.set_ylabel('频数', color='b',fontsize=15)
ax1.tick_params(axis='y', labelcolor='b')
color=np.array([188,169,162])/255
ax2=ax1.twinx()
ax2.hist(data, bins=bins, density=True, cumulative=True, histtype='step', colo
r=color)
# 添加标题和标签
ax2.set_ylabel('累积概率密度', color='brown',fontsize=15)
ax2.tick_params(axis='y', labelcolor='brown')
# 显示图例
#plt.legend()
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
plt.title('扫码时间间隔分布',fontsize=15)
# 显示直方图
plt.show()
plt.savefig('time_intervals.jpg',dpi=600)
```

附录 12

Python:正态检验

```
    import pandas as pd
    import numpy as np
    from scipy import stats
```

```
    name = ['花叶类','水生根茎类','花菜类','茄类','辣椒类','食用菌']
    data1 = pd.read_excel(r'C:\Users\ 杨 锋 \Desktop\C 题 \ 品 类 销 售 数
据.xlsx',sheet_name='品类1011010101')
    data2 = pd.read_excel(r'C:\Users\ 杨 锋 \Desktop\C 题 \ 品 类 销 售 数
据.xlsx',sheet_name='品类1011010201')
    data3 = pd.read_excel(r'C:\Users\ 杨 锋 \Desktop\C 题 \ 品 类 销 售 数
据.xlsx',sheet_name='品类1011010402')
    data4 = pd.read_excel(r'C:\Users\ 杨 锋 \Desktop\C 题 \ 品 类 销 售 数
据.xlsx',sheet_name='品类1011010501')
    data5 = pd.read_excel(r'C:\Users\ 杨 锋 \Desktop\C 题 \ 品 类 销 售 数
据.xlsx',sheet_name='品类1011010504')
    data6 = pd.read_excel(r'C:\Users\ 杨 锋 \Desktop\C 题 \ 品 类 销 售 数
据.xlsx',sheet_name='品类1011010801')
    p_value_list = []
    statistic_list = []
    sales_data = data1['销量(千克)']
    statistic, p_value = stats.shapiro(sales_data)
    p_value_list.append(p_value)
    statistic_list.append(statistic)
    print("Shapiro-Wilk 检验统计量：", statistic)
    print("p 值：", p_value)
    alpha = 0.05

    if p_value > alpha:
        print("花叶类 p 值大于显著性水平，销售量数据可能符合正态分布")
    else:
        print("花叶类 p 值小于显著性水平，销售量数据不符合正态分布")
    sales_data = data2['销量(千克)']
    statistic, p_value = stats.shapiro(sales_data)
    p_value_list.append(p_value)
    statistic_list.append(statistic)
    print("Shapiro-Wilk 检验统计量：", statistic)
    print("p 值：", p_value)
    alpha = 0.05

    if p_value > alpha:
        print("花菜类 p 值大于显著性水平，销售量数据可能符合正态分布")
    else:
        print("花菜类 p 值小于显著性水平，销售量数据不符合正态分布")

    sales_data = data3['销量(千克)']
    statistic, p_value = stats.shapiro(sales_data)
    p_value_list.append(p_value)
    statistic_list.append(statistic)
```

```
print("Shapiro-Wilk 检验统计量：", statistic)
print("p 值：", p_value)
alpha = 0.05
if p_value > alpha:
    print("水生根茎类 p 值大于显著性水平，销售量数据可能符合正态分布")
else:
    print("水生根茎类 p 值小于显著性水平，销售量数据不符合正态分布")

sales_data = data4['销量(千克)']
statistic, p_value = stats.shapiro(sales_data)
p_value_list.append(p_value)
statistic_list.append(statistic)
print("Shapiro-Wilk 检验统计量：", statistic)
print("p 值：", p_value)
alpha = 0.05
if p_value > alpha:
    print("茄类 p 值大于显著性水平，销售量数据可能符合正态分布")
else:
    print("茄类 p 值小于显著性水平，销售量数据不符合正态分布")

sales_data = data5['销量(千克)']
statistic, p_value = stats.shapiro(sales_data)
p_value_list.append(p_value)
statistic_list.append(statistic)
print("Shapiro-Wilk 检验统计量：", statistic)
print("p 值：", p_value)
alpha = 0.05
if p_value > alpha:
    print("辣椒类 p 值大于显著性水平，销售量数据可能符合正态分布")
else:
    print("辣椒类 p 值小于显著性水平，销售量数据不符合正态分布")

sales_data = data6['销量(千克)']
statistic, p_value = stats.shapiro(sales_data)
p_value_list.append(p_value)
statistic_list.append(statistic)
print("Shapiro-Wilk 检验统计量：", statistic)
print("p 值：", p_value)
alpha = 0.05
if p_value > alpha:
    print("食用菌 p 值大于显著性水平，销售量数据可能符合正态分布")
else:
 print("食用菌 p 值小于显著性水平，销售量数据不符合正态分布")
```

附录 13

Python:导出消费者购买物品信息

```
import pandas as pd
import numpy as np
data = pd.read_excel(r'C:\Users\杨锋\Desktop\C 题\附件 2_filtered.xlsx')#附件
2_filtered
original_index = data.index
data=data[data['销售类型']=='销售']
data = data.reset_index(drop=True)
person_buy=[[]]
k=0
i=0
while i<len(data)-1:
    person_buy[k].append(data['单品编码'][i])
    diff=pd.to_datetime(data['扫码销售时间'][i+1])-pd.to_datetime(data['扫码销售
时间'][i])
    diff=diff.total_seconds()
    while(diff<1):
        i+=1
        if(data['单品编码'][i] in person_buy[k]):
            break
        person_buy[k].append(data['单品编码'][i])
        diff = pd.to_datetime(data['扫码销售时间
'][i + 1]) - pd.to_datetime(data['扫码销售时间'][i])
        diff = diff.total_seconds()
    else:
        i+=1
    k+=1
    person_buy.append([])
with open('output.txt', 'w') as f:
    for sublist in person_buy:
            f.write(f'{", ".join(map(str, sublist))}\n')
```

附录 14

Python:Apriori 算法计算

```
import pandas as pd
import numpy as np
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

lines = []
with open('output.txt', 'r') as file:
```

```
    for line in file:
        line_list = line.strip().split()
        if len(line_list) > 1:
            lines.append(line_list)
new_lines = [[item.strip(',') for item in sub_list] for sub_list in lines]

transactions = new_lines
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.02, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="lift")

print("频繁项集:")
print(frequent_itemsets)
print("\n 关联规则:")
print(rules)

frequent_itemsets.to_excel('frequent_itemsets.xlsx', index=False)
    rules.to_excel('association_rules.xlsx', index=False)
```

附录 15

Python:打折力度的计算

```
# 创建每一天的总销售额、总进货价、总销售量和平均打折率的 DataFrame
daily_sales_profit = pd.DataFrame(columns=['日期', '品类编码', '总销售额', '总进货价', '
总销售量', '平均打折率'])

# 遍历附件 2 中的数据，按日期和品类编码计算总销售额、总进货价、总销售量和平均打折率
for date, group in df2.groupby(['销售日期', '单品编码']):
    sales_date, product_code = date
    sales_quantity = group['销量'].sum()

    # 获取品类编码
    category_code = product_category_mapping.get(product_code, None)

    if category_code is not None:
        # 获取进价
        cost_price = date_product_cost_mapping.get(sales_date, {}).get(product_code, 0)

        # 计算销售额和进货价
        sales_amount = group['销售额'].sum()
        total_cost_price = sales_quantity * cost_price
```

```
        # 计算打折率
        discount_prices = group[group['是否打折销售'] == '是']['销售单价(元/千
克)'].tolist()
        regular_prices = group[group['是否打折销售'] == '否']['销售单价(元/千
克)'].tolist()

        if discount_prices:
            avg_discount_rate = np.mean([discount / regular for discount, regular in
zip(discount_prices, regular_prices) if discount != 1.0])
        else:
            avg_discount_rate = 1.0  # 如果没有打折，则打折率为 1

        # 将结果添加到 DataFrame 中
        daily_sales_profit = pd.concat([daily_sales_profit, pd.DataFrame({'日期':
[sales_date], '品类编码': [category_code],
                                                                          '总销售额':
[sales_amount], '总进货价': [total_cost_price],
                                                                          '总销售量':
[sales_quantity], '平均打折率': [avg_discount_rate]})],
                                        ignore_index=True)

# 合并数据按品类编码计算总销售额、总进货价、总销售量和平均打折率
category_sales_profit = daily_sales_profit.groupby(['日期', '品类编码'])[['总销售额', '总
进货价', '总销售量', '平均打折率']].agg({'总销售额':'sum', '总进货价':'sum', '总销售量
':'sum', '平均打折率': lambda x: np.mean([value for value in x if value !=
1.0])}).reset_index()
```

附录 16

Python：平均批发价格的计算

```
# 创建每一天的平均批发价格的 DataFrame
daily_avg_wholesale_price = pd.DataFrame(columns=['日期', '品类编码', '平均批发
价格'])

# 遍历附件 3 中的数据，按日期和品类编码计算平均批发价格
for date, group in df3.groupby(['日期', '单品编码']):
    wholesale_date, product_code = date
    wholesale_price = group['批发价格'].mean()

    # 获取品类编码
    category_code = product_category_mapping.get(product_code, None)

    if category_code is not None:
        # 将结果添加到 DataFrame 中
```

```
        daily_avg_wholesale_price = pd.concat([daily_avg_wholesale_price,
pd.DataFrame({'日期': [wholesale_date], '品类编码': [category_code],

       '平均批发价格': [wholesale_price]})],
                                              ignore_index=True)

# 合并数据按品类编码计算平均批发价格
category_avg_wholesale_price = daily_avg_wholesale_price.groupby(['日期', '品类
编码'])['平均批发价格']].mean().reset_index()
```

附录 17

Python:高斯过程回归得到销售量

```
import pandas as pd
import numpy as np
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

data = pd.read_excel(r'C:\Users\杨锋\Desktop\C 题\每天总销售额和总进货
价 (1).xlsx')
selected_categories = [1011010101, 1011010201, 1011010402, 1011010501, 1011010
504, 1011010801]
filtered_data = data[data['品类编码'].isin(selected_categories)]
name = ['花叶类','水生根茎类','花菜类','茄类','辣椒类','食用菌']
filtered_data['日期'] = pd.to_datetime(filtered_data['日期'])

data_after_start_date = data[
    (data['日期'].dt.month == 6) &
    (data['日期'].dt.day >= 24) &
    (data['日期'].dt.year.isin([2021, 2022, 2023])) |
    (data['日期'].dt.month == 7) &
    (data['日期'].dt.day <= 8) &
    (data['日期'].dt.year.isin([2021, 2022, 2023]))
]
data_after_start_date=data_after_start_date.reset_index(drop=True)
filtered_data = data_after_start_date
filtered_data['日期'] = filtered_data['日期'].dt.strftime('%m-%d')

category_data_dict = {}
for category in selected_categories:
    category_data_dict[category] = filtered_data[filtered_data['品类编码
'] == category]
```

```
date_range = pd.date_range(start='2023-07-01', end='2023-07-07', freq='D')
date_range = date_range.strftime('%m-%d')

kernel = C(1.0, (1e-3, 1e3)) * RBF(1.0, (1e-2, 1e2))
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10)

date_str = '06-24'
date1 = datetime.strptime(date_str, '%m-%d')
for i, (category, category_data) in enumerate(category_data_dict.items()):
    category_data['日期'] = pd.to_datetime(category_data['日期
'], format='%m-%d')
    X = (category_data['日期'] - date1).dt.days.values.reshape(-1, 1)
    y = category_data['总销售量'].values
    days = X
    sales = y
    gp.fit(days, sales)
    date_range = pd.to_datetime(date_range, format='%m-%d')
    X_pred = (date_range - date1).days.to_numpy().reshape(-1, 1)
    y_pred, y_std = gp.predict(X_pred, return_std=True)
    print("预测销量:", y_pred)
    print("标准差:", y_std)

low = []
high = []
fig, axs = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('品类销售量预测图(2023/07/01 - 2023/07/07)')

date_str = '06-24'
date1 = datetime.strptime(date_str, '%m-%d')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
for i, (category, category_data) in enumerate(category_data_dict.items()):
    category_data['日期'] = pd.to_datetime(category_data['日期
'], format='%m-%d')
    X = (category_data['日期'] - date1).dt.days.values.reshape(-1, 1)
    y = category_data['总销售量'].values
    days = X
    sales = y
    gp.fit(days, sales)
    date_range = pd.to_datetime(date_range, format='%m-%d')
    X_pred = (date_range - date1).days.to_numpy().reshape(-1, 1)
    y_pred, sigma = gp.predict(X_pred, return_std=True)

    row = i // 3
```

```
    col = i % 3
    axs[row, col].set_title(f'{name[i]}')
    axs[row, col].plot(X, y, 'r.', markersize=10, label='历史销售量')
    axs[row, col].plot(X_pred, y_pred, 'b-', label='销售量预测')
    if i == 0:
        axs[row, col].fill_between(X_pred[:, 0], y_pred - 10**6.5*sigma, y_pre
d + 10**6.5*sigma, alpha=0.2)
        low.append(y_pred - 10**6.5*sigma)
        high.append(y_pred + 10**6.5*sigma)
    else:
        axs[row, col].fill_between(X_pred[:, 0], y_pred - 10**6*sigma, y_pred
+ 10**6*sigma, alpha=0.2)
        low.append(y_pred - 10**6.0*sigma)
        high.append(y_pred + 10**6.0*sigma)
    axs[row, col].set_xlabel('天数 (从最开始的那天计算)')
    axs[row, col].set_ylabel('销售量')
    axs[row, col].legend()
    axs[row, col].grid(True)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.savefig('gaussian_process.png', dpi=600)
plt.show()

data = [
    [163.26997124, 178.54926811, 207.40864311, 171.11176811, 168.89301811, 183
.26801811, 152.77583061],
    [135.04643501, 150.32573189, 179.18510689, 142.88823189,140.66948189, 155.
04448189, 124.55229439],
    [45.43638886, 54.53404511, 62.34849824, 50.94029511, 68.61217011, 52.16685
761, 59.85435761],
    [17.18079864, 26.27845489, 34.09290801, 22.68470489, 40.35657989, 23.91126
739, 31.59876739],
    [25.91254936, 33.84614311, 60.87739311, 36.33051811, 38.84614311, 37.54926
811, 39.84614311],
    [-
2.31098686,  5.62260689, 32.65385689,  8.10698189, 10.62260689, 9.32573189, 11
.62260689],
    [35.83873261, 42.1463498 , 58.95592011, 42.64342011, 47.69029511, 43.12779
511, 37.29967011],
    [ 7.58314239, 13.89075957, 30.70032989, 14.38782989, 19.43470489, 14.87220
489,  9.04407989],
    [54.08051811, 68.00239311, 80.20551811, 66.63520561, 65.91645561, 62.94770
561, 53.17036186],
```

```
    [25.85698189, 39.77885689, 51.98198189, 38.41166939, 37.69291939, 34.72416939, 24.94682564],
    [55.90864311, 54.08833061, 68.62739311, 54.68989311, 58.04926811, 48.40864311, 41.75629936],
    [27.68510689, 25.86479439, 40.40385689, 26.46635689, 29.82573189, 20.18510689, 13.53276314],
]

df = pd.DataFrame(data).T
df.columns = ['品类 1high','品类 1low', '品类 2high','品类 2low', '品类 3high','品类 3low', '品类 4high','品类 4low', '品类 5high','品类 5low', '品类 6high','品类 6low']
    df.to_excel('output.xlsx', index=False)
```

附录 18

Matlab：使用 b 站 lvy 的自制工具箱导出的代码，多元线性回归分析定价与销售量和批发价的关系

```
clc;
clear;
close all;

load('新数据导入.mat')

data_path_str_new = G_out_data_new.data_path_str;  % 读取新数据的路径

data1_new = readtable(data_path_str_new, 'VariableNamingRule', 'preserve'); % 读取新数据
data2_new = data1_new(:, 2:end);
data_new = table2array(data1_new(:, 2:end));
data_biao_new = data2_new.Properties.VariableNames;  % 新数据特征的名称

A_data1_new = data_new;
data_biao1_new = data_biao_new;
select_feature_num_new = G_out_data_new.select_feature_num1;   % 新特征选择的个数

data_select_new = A_data1_new;
feature_need_last_new = 1:size(A_data1_new, 2) - 1;
```

```
%% 新数据划分
x_feature_label_new = data_select_new(:, 1:end-1);    % x 特征
y_feature_label_new = data_select_new(:, end);         % y 标签
index_label1_new = 1:(size(x_feature_label_new, 1));
index_label_new = G_out_data_new.spilt_label_data;  % 新数据索引
if isempty(index_label_new)
    index_label_new = index_label1_new;
end
spilt_ri_new = G_out_data_new.spilt_rio;  % 划分比例 训练集:验证集:测试集
train_num_new = round(spilt_ri_new(1) / (sum(spilt_ri_new)) *
size(x_feature_label_new, 1));          % 训练集个数
vaild_num_new = round((spilt_ri_new(1) + spilt_ri_new(2)) /
(sum(spilt_ri_new)) * size(x_feature_label_new, 1)); % 验证集个数
% 训练集，验证集，测试集
train_x_feature_label_new =
x_feature_label_new(index_label_new(1:train_num_new), :);
train_y_feature_label_new =
y_feature_label_new(index_label_new(1:train_num_new), :);
vaild_x_feature_label_new =
x_feature_label_new(index_label_new(train_num_new+1:vaild_num_new), :);
vaild_y_feature_label_new =
y_feature_label_new(index_label_new(train_num_new+1:vaild_num_new), :);
test_x_feature_label_new =
x_feature_label_new(index_label_new(vaild_num_new+1:end), :);
test_y_feature_label_new =
y_feature_label_new(index_label_new(vaild_num_new+1:end), :);
% Zscore 新数据标准化
% 训练集
x_mu_new = mean(train_x_feature_label_new);
x_sig_new = std(train_x_feature_label_new);
train_x_feature_label_norm_new = (train_x_feature_label_new -
x_mu_new) ./ x_sig_new;    % 训练新数据标准化
y_mu_new = mean(train_y_feature_label_new);
y_sig_new = std(train_y_feature_label_new);
train_y_feature_label_norm_new = (train_y_feature_label_new -
y_mu_new) ./ y_sig_new;    % 训练新数据标准化
% 验证集
vaild_x_feature_label_norm_new = (vaild_x_feature_label_new -
x_mu_new) ./ x_sig_new;    % 验证新数据标准化
vaild_y_feature_label_norm_new = (vaild_y_feature_label_new -
y_mu_new) ./ y_sig_new;  % 验证新数据标准化
% 测试集
test_x_feature_label_norm_new = (test_x_feature_label_new -
x_mu_new) ./ x_sig_new;    % 测试新数据标准化
```

```
test_y_feature_label_norm_new = (test_y_feature_label_new -
y_mu_new) ./ y_sig_new;    % 测试新数据标准化

%% 新参数设置
num_pop_new = G_out_data_new.num_pop1;   % 种群数量
num_iter_new = G_out_data_new.num_iter1;   % 种群迭代数
method_mti_new = G_out_data_new.method_mti1;   % 优化方法
BO_iter_new = G_out_data_new.BO_iter;   % 贝叶斯迭代次数
min_batchsize_new = G_out_data_new.min_batchsize;   % 新 batchsize
max_epoch_new = G_out_data_new.max_epoch1;   % 新 maxepoch
hidden_size_new = G_out_data_new.hidden_size1;   % 新 hidden_size

%% 新算法处理块
X_new = ones(size(train_x_feature_label_norm_new, 1), 1);
train_x_feature_label_norm1_new = [X_new,
train_x_feature_label_norm_new];
alpha1_new = 0.05; % 置信区间
disp('多元线性回归预测')
t1_new = clock;
[b_new, bint_new, r_new, rint_new, stats_new] =
regress(train_y_feature_label_norm_new,
train_x_feature_label_norm1_new, alpha1_new);
disp('线性系数：');
disp(b_new);
disp(['R 方系数：', num2str(stats_new(1))]);    % 新 R 方 接近 1 线性相关性越强
disp(['p 检验值：  ', num2str(stats_new(3))])   % p<0.05 默认显著性水平，代表 y
和 X 中的预测变量之间存在显著的线性回归关系
y_train_predict_norm_new = b_new(1) + train_x_feature_label_norm_new *
b_new(2:end);  % 新训练集预测结果
y_vaild_predict_norm_new = b_new(1) + vaild_x_feature_label_norm_new *
b_new(2:end);  % 新验证集预测结果
y_test_predict_norm_new = b_new(1) + test_x_feature_label_norm_new *
b_new(2:end);  % 新测试集预测结果
t2_new = clock;
Mdl_new{1, 1} = b_new(2:end);
Mdl_new{1, 2} = b_new(1);
Time_new = t2_new(3) * 3600 * 24 + t2_new(4) * 3600 + t2_new(5) * 60 +
t2_new(6) - (t1_new(3) * 3600 * 24 + t1_new(4) * 3600 + t1_new(5) * 60
+ t1_new(6));

y_train_predict_new = y_train_predict_norm_new * y_sig_new +
y_mu_new;  % 反标准化操作
y_vaild_predict_new = y_vaild_predict_norm_new * y_sig_new + y_mu_new;
y_test_predict_new = y_test_predict_norm_new * y_sig_new + y_mu_new;
```

```
train_y_new = train_y_feature_label_new;
disp('**************************************************************
*****************************************');
train_MAE_new = sum(abs(y_train_predict_new - train_y_new)) /
length(train_y_new);
disp(['训练集平均绝对误差 MAE: ', num2str(train_MAE_new)]);
train_MAPE_new = sum(abs((y_train_predict_new - train_y_new) ./
train_y_new)) / length(train_y_new);
disp(['训练集平均相对误差 MAPE: ', num2str(train_MAPE_new)]);
train_MSE_new = (sum(((y_train_predict_new - train_y_new)).^2) /
length(train_y_new));
disp(['训练集均方根误差 MSE: ', num2str(train_MSE_new)]);
train_RMSE_new = sqrt(sum(((y_train_predict_new - train_y_new)).^2) /
length(train_y_new));
disp(['训练集均方根误差 RMSE: ', num2str(train_RMSE_new)]);
train_R2_new = 1 - (norm(train_y_new - y_train_predict_new)^2 /
norm(train_y_new - mean(train_y_new))^2);
disp(['训练集均方根误差 R2: ', num2str(train_R2_new)]);

vaild_y_new = vaild_y_feature_label_new;
disp('**************************************************************
*****************************************');
vaild_MAE_new = sum(abs(y_vaild_predict_new - vaild_y_new)) /
length(vaild_y_new);
disp(['验证集平均绝对误差 MAE: ', num2str(vaild_MAE_new)]);
vaild_MAPE_new = sum(abs((y_vaild_predict_new - vaild_y_new) ./
vaild_y_new)) / length(vaild_y_new);
disp(['验证集平均相对误差 MAPE: ', num2str(vaild_MAPE_new)]);
vaild_MSE_new = (sum(((y_vaild_predict_new - vaild_y_new)).^2) /
length(vaild_y_new));
disp(['验证集均方根误差 MSE: ', num2str(vaild_MSE_new)]);
vaild_RMSE_new = sqrt(sum(((y_vaild_predict_new - vaild_y_new)).^2) /
length(vaild_y_new));
disp(['验证集均方根误差 RMSE: ', num2str(vaild_RMSE_new)]);
vaild_R2_new = 1 - (norm(vaild_y_new - y_vaild_predict_new)^2 /
norm(vaild_y_new - mean(vaild_y_new))^2);
disp(['验证集均方根误差 R2:  ', num2str(vaild_R2_new)]);

test_y_new = test_y_feature_label_new;
disp('**************************************************************
*****************************************');
test_MAE_new = sum(abs(y_test_predict_new - test_y_new)) /
length(test_y_new);
```

```
disp(['测试集平均绝对误差 MAE: ', num2str(test_MAE_new)]);
test_MAPE_new = sum(abs((y_test_predict_new - test_y_new) ./
test_y_new)) / length(test_y_new);
disp(['测试集平均相对误差 MAPE: ', num2str(test_MAPE_new)]);
test_MSE_new = (sum(((y_test_predict_new - test_y_new)).^2) /
length(test_y_new));
disp(['测试集均方根误差 MSE: ', num2str(test_MSE_new)]);
test_RMSE_new = sqrt(sum(((y_test_predict_new - test_y_new)).^2) /
length(test_y_new));
disp(['测试集均方根误差 RMSE: ', num2str(test_RMSE_new)]);
test_R2_new = 1 - (norm(test_y_new - y_test_predict_new)^2 /
norm(test_y_new - mean(test_y_new))^2);
disp(['测试集均方根误差 R2: ', num2str(test_R2_new)]);
disp(['算法运行时间 Time: ', num2str(Time_new)]);

%% 新绘图块
color_list_new = G_out_data_new.color_list;   % 颜色数据库
rand_list1_new = G_out_data_new.rand_list1;   % 颜色数据库
Line_Width_new = G_out_data_new.Line_Width;   % 线粗细
makesize_new = G_out_data_new.makesize;   % 标记大小
yang_str2_new = G_out_data_new.yang_str2;   % 符号库
yang_str3_new = G_out_data_new.yang_str3;   % 符号库
kuang_width_new = G_out_data_new.kuang_width;   % 画图展示数据
show_num_new = G_out_data_new.show_num;   % 测试集画图展示数据
show_num1_new = G_out_data_new.show_num1;   % 验证集画图展示数据
show_num2_new = G_out_data_new.show_num2;   % 训练集画图展示数据

FontSize_new = G_out_data_new.FontSize;   % 字体设置
xlabel1_new = G_out_data_new.xlabel1;   %
ylabel1_new = G_out_data_new.ylabel1;   %
title1_new = G_out_data_new.title1;   %
legend1_new = G_out_data_new.legend1;   % 图例
box1_new = G_out_data_new.box1;   % 框
le_kuang_new = G_out_data_new.le_kuang;   % 图例框
grid1_new = G_out_data_new.grid1;   % 网格
figure

XX_new = 1:length(train_y_feature_label_new);
index_show_new = 1:show_num2_new;
plot(gca, XX_new(index_show_new),
train_y_feature_label_new(index_show_new), yang_str2_new{1, 3},
'Color', color_list_new(rand_list1_new(1), :), 'LineWidth',
Line_Width_new(1))
hold(gca, 'on')
```

```
plot(gca, XX_new(index_show_new), y_train_predict_new(index_show_new),
yang_str3_new{1, 1}, 'Color', color_list_new(rand_list1_new(2), :),
'LineWidth', Line_Width_new(2))
hold(gca, 'on')
set(gca, 'FontSize', FontSize_new, 'LineWidth', kuang_width_new)
xlabel(gca, xlabel1_new)
ylabel(gca, ylabel1_new)
title(gca, '新训练集结果')
legend(gca, legend1_new)
box(gca, box1_new)
legend(gca, le_kuang_new) % 图例框消失
grid(gca, grid1_new)

figure

XX_new = 1:length(vaild_y_feature_label_new);
index_show_new = 1:show_num1_new;
plot(gca, XX_new(index_show_new),
vaild_y_feature_label_new(index_show_new), yang_str2_new{1, 3},
'Color', color_list_new(rand_list1_new(1), :), 'LineWidth',
Line_Width_new(1))
hold(gca, 'on')
plot(gca, XX_new(index_show_new), y_vaild_predict_new(index_show_new),
yang_str3_new{1, 1}, 'Color', color_list_new(rand_list1_new(2), :),
'LineWidth', Line_Width_new(2))
hold(gca, 'on')
set(gca, 'FontSize', FontSize_new, 'LineWidth', kuang_width_new)
xlabel(gca, xlabel1_new)
ylabel(gca, ylabel1_new)
title(gca, '新验证集结果')
legend(gca, legend1_new)
box(gca, box1_new)
legend(gca, le_kuang_new) % 图例框消失
grid(gca, grid1_new)

figure

XX_new = 1:length(test_y_feature_label_new);
index_show_new = 1:show_num_new;
plot(gca, XX_new(index_show_new),
test_y_feature_label_new(index_show_new), yang_str2_new{1, 3}, 'Color',
color_list_new(rand_list1_new(1), :), 'LineWidth', Line_Width_new(1))
hold(gca, 'on')
```

```
plot(gca, XX_new(index_show_new), y_test_predict_new(index_show_new),
yang_str3_new{1, 1}, 'Color', color_list_new(rand_list1_new(2), :),
'LineWidth', Line_Width_new(2))
hold(gca, 'on')
set(gca, 'FontSize', FontSize_new, 'LineWidth', kuang_width_new)
xlabel(gca, xlabel1_new)
ylabel(gca, ylabel1_new)
title(gca, '新测试集结果')
legend(gca, legend1_new)
box(gca, box1_new)
legend(gca, le_kuang_new) % 图例框消失
grid(gca, grid1_new)
```

附录 19

Matlab：使用 b 站 lvy 的自制工具箱导出的代码，最小二乘回归树预测批发价格

```
clc;
clear;
close all;

load('MT_09_Sep_2023_18_24_11.mat');

dataPath = G_out_data.data_path_str;  % 读取数据的路径
dataTable = readtable(dataPath, 'VariableNamingRule', 'preserve'); % 读
取数据
data = dataTable(:, 2:end);
testData = table2cell(dataTable(1, 2:end));

for i = 1:length(testData)
    if ischar(testData{1, i}) == 1
        dataType(i) = 1;  % char 类型
    elseif isnumeric(testData{1, i}) == 1
        dataType(i) = 2;  % double 类型
    else
        dataType(i) = 0;  % 其他类型
    end
end

charIndices = find(dataType == 1);
doubleIndices = find(dataType == 2);

% 数值类型数据处理
if length(doubleIndices) >= 1
```

```
    numericalData = table2array(data(:, doubleIndices));
    numericalData2 = numericalData;
    lastNeededIndices = doubleIndices;
else
    lastNeededIndices = doubleIndices;
    numericalData2 = [];
end

% 文本类型数据处理
textData = [];
if length(charIndices) >= 1
    for j = 1:length(charIndices)
        textColumn = table2array(data(:, charIndices(j)));

        uniqueLabels = unique(textColumn);
        for NN = 1:length(uniqueLabels)
            idx = find(ismember(textColumn, uniqueLabels{NN, 1}));
            textData(idx, j) = NN;
        end
    end
end

finalData = [textData, numericalData2];
labelIndices = [charIndices, lastNeededIndices];
data = finalData;

dataTableColumnNames = data.Properties.VariableNames;
for j = 1:length(labelIndices)
    renamedLabels{1, j} = dataTableColumnNames{1, labelIndices(j)};
end

originalData = data;

dataOutput = originalData;
A_data1 = dataOutput;
dataLabels1 = renamedLabels;
selectFeatureNum = G_out_data.select_feature_num;   % 特征选择的个数
predictNum = G_out_data.predict_num_set;   % 预测的点个数

dataSelection = A_data1;
featureSelection = 1:size(A_data1, 2) - 1;

dataSelect1 = dataSelection;
dataSelect1 = dataSelect1;
```

```
% 波形分解
dataSelect1Cell = {dataSelect1};
selectPredictNum = G_out_data.select_predict_num;  % 待预测数
numFeature = G_out_data.num_feature;    % 特征选择量
numSeries = G_out_data.num_series;     % 序列选择
numInputSeries = numSeries;
minBatchSize = G_out_data.min_batchsize;
rollNum = G_out_data.roll_num;
rollNumIn = G_out_data.roll_num_in;
numPop = G_out_data.num_pop;  % 优化种群数
numIter = G_out_data.num_iter;  % 优化迭代数
numBOIter = G_out_data.num_BO_iter;  % 贝叶斯优化迭代次数
maxEpochLC = G_out_data.max_epoch_LC; % 最大轮数
methodMTI = G_out_data.method_mti; % 最大轮数
listCell = G_out_data.list_cell;

% 模型训练
xMuAll = [];
xSigAll = [];
yMuAll = [];
ySigAll =;

for NUM_all = 1:length(dataSelect1Cell)
    dataProcess = dataSelect1Cell{1, NUM_all};
    [xFeatureLabel, yFeatureLabel] = timeseries_process2(dataProcess,
selectPredictNum, numSeries);
    [~, yFeatureLabel1] = timeseries_process2(dataSelect1,
selectPredictNum, numSeries);    % 未分解之前
    indexLabel1 = 1:(size(xFeatureLabel, 1));
    indexLabel = indexLabel1;
    spiltRatios = G_out_data.spilt_ri;
    trainNum = round(spiltRatios(1) / (sum(spiltRatios)) *
size(xFeatureLabel, 1));                    % 训练集个数
    vaildNum = round((spiltRatios(1) + spiltRatios(2)) /
(sum(spiltRatios)) * size(xFeatureLabel, 1)); % 验证集个数

    % 训练集，验证集，测试集
    trainXFeatureLabel = xFeatureLabel(indexLabel(1:trainNum), :);
    trainYFeatureLabel = yFeatureLabel(indexLabel(1:trainNum), :);
    vaildXFeatureLabel = xFeatureLabel(indexLabel(trainNum +
1:vaildNum), :);
    vaildYFeatureLabel = yFeatureLabel(indexLabel(trainNum +
1:vaildNum), :);
```

```
    testXFeatureLabel = xFeatureLabel(indexLabel(vaildNum + 1:end), :);
    testYFeatureLabel = yFeatureLabel(indexLabel(vaildNum + 1:end), :);

    % Zscore 标准化

    % 训练集
    xMu = mean(trainXFeatureLabel);
    xSig = std(trainXFeatureLabel);
    trainXFeatureLabelNorm = (trainXFeatureLabel - xMu) ./ xSig;    % 训
练数据标准化
    yMu = mean(trainYFeatureLabel);
    ySig = std(trainYFeatureLabel);
    trainYFeatureLabelNorm = (trainYFeatureLabel - yMu) ./ ySig;    % 训
练数据标准化
    xMuAll(NUM_all, :) = xMu;
    xSigAll(NUM_all, :) = xSig;
    yMuAll(NUM_all, :) = yMu;
    ySigAll(NUM_all, :) = ySig;

    % 验证集
    vaildXFeatureLabelNorm = (vaildXFeatureLabel - xMu) ./ xSig;    % 训
练数据标准化
    vaildYFeatureLabelNorm = (vaildYFeatureLabel - yMu) ./ ySig;    % 训
练数据标准化

    % 测试集
    testXFeatureLabelNorm = (testXFeatureLabel - xMu) ./ xSig;    % 训练数
据标准化
    testYFeatureLabelNorm = (testYFeatureLabel - yMu) ./ ySig;    % 训练数
据标准化
    yTrainPredictNorm = zeros(size(trainYFeatureLabel, 1),
size(trainYFeatureLabel, 2));
    yVaildPredictNorm = zeros(size(vaildYFeatureLabel, 1),
size(vaildYFeatureLabel, 2));
    yTestPredictNorm = zeros(size(testYFeatureLabel, 1),
size(testYFeatureLabel, 2));

    for N1 = 1:length(listCell)
        Mdl = fitrensemble(trainXFeatureLabelNorm,
trainYFeatureLabelNorm(:, listCell{1, N1}(1)));
        yTrainPredictNorm(:, listCell{1, N1}(1)) = predict(Mdl,
trainXFeatureLabelNorm);
        yVaildPredictNorm(:, listCell{1, N1}(1)) = predict(Mdl,
vaildXFeatureLabelNorm);
```

```
        yTestPredictNorm(:, listCell{1, N1}(1)) = predict(Mdl,
testXFeatureLabelNorm);
        Model{1, N1} = Mdl;
        modelAll{NUM_all, N1} = Mdl;
    end

    trainXFeatureLabelNormRoll = trainXFeatureLabelNorm;
    vaildXFeatureLabelNormRoll = vaildXFeatureLabelNorm;
    testXFeatureLabelNormRoll = testXFeatureLabelNorm;

    if length(listCell) < selectPredictNum
        for N2 = 1:length(listCell)
            rollList = listCell{1, N2};
            for N3 = rollList(2):rollList(end)
                trainXFeatureLabelNormRoll(:, end - numInputSeries +
1:end) = [trainXFeatureLabelNormRoll(:, end - numInputSeries + 2:end),
yTrainPredictNorm(:, N3 - 1)];
                vaildXFeatureLabelNormRoll(:, end - numInputSeries +
1:end) = [vaildXFeatureLabelNormRoll(:, end - numInputSeries + 2:end),
yVaildPredictNorm(:, N3 - 1)];
                testXFeatureLabelNormRoll(:, end - numInputSeries + 1:end)
= [testXFeatureLabelNormRoll(:, end - numInputSeries + 2:end),
yTestPredictNorm(:, N3 - 1)];
                yTrainPredictNorm(:, N3) = predict(Model{1, N2},
trainXFeatureLabelNormRoll);
                yVaildPredictNorm(:, N3) = predict(Model{1, N2},
vaildXFeatureLabelNormRoll);
                yTestPredictNorm(:, N3) = predict(Model{1, N2},
testXFeatureLabelNormRoll);
            end
        end
    end

    yTrainPredictCell{1, NUM_all} = yTrainPredictNorm .* ySig + yMu;  %
反标准化操作
    yVaildPredictCell{1, NUM_all} = yVaildPredictNorm .* ySig + yMu;
    yTestPredictCell{1, NUM_all} = yTestPredictNorm .* ySig + yMu;
end

yTrainPredictFinal = 0;
yVaildPredictFinal = 0;
yTestPredictFinal = 0;

for i = 1:length(dataSelect1Cell)
```

```
    yTrainPredictFinal = yTrainPredictFinal + yTrainPredictCell{1, i};
    yVaildPredictFinal = yVaildPredictFinal + yVaildPredictCell{1, i};
    yTestPredictFinal = yTestPredictFinal + yTestPredictCell{1, i};
end

trainY = yFeatureLabel1(indexLabel(1:trainNum), :);
vaildY = yFeatureLabel1(indexLabel(trainNum + 1:vaildNum), :);
testY = yFeatureLabel1(indexLabel(vaildNum + 1:end), :);

Tvalue = G_out_data.Tvalue;  % 使用的方法

trainMAE = sum(sum(abs(yTrainPredictFinal - trainY))) / size(trainY, 1)
/ size(trainY, 2);
disp([Tvalue, '训练集平均绝对误差 MAE: ', num2str(trainMAE)])

trainMAPE = sum(sum(abs((yTrainPredictFinal - trainY) ./ trainY))) /
size(trainY, 1) / size(trainY, 2);
disp([Tvalue, '训练集平均相对误差 MAPE: ', num2str(trainMAPE)])

trainMSE = (sum(sum(((yTrainPredictFinal - trainY)) .^ 2)) /
size(trainY, 1) / size(trainY, 2));
disp([Tvalue, '训练集均方根误差 MSE: ', num2str(trainMSE)])

trainRMSE = sqrt(sum(sum(((yTrainPredictFinal - trainY)) .^ 2)) /
size(trainY, 1) / size(trainY, 2));
disp([Tvalue, '训练集均方根误差 RMSE: ', num2str(trainRMSE)])

trainR2 = 1 - mean(norm(trainY - yTrainPredictFinal) ^ 2 / norm(trainY
- mean(trainY)) ^ 2);
disp([Tvalue, '训练集均方根误差 R2: ', num2str(trainR2)])
disp('**********************************************************************
******************')

vaildMAE = sum(sum(abs(yVaildPredictFinal - vaildY))) / size(vaildY, 1)
/ size(vaildY, 2);
disp([Tvalue, '验证集平均绝对误差 MAE: ', num2str(vaildMAE)])

vaildMAPE = sum(sum(abs((yVaildPredictFinal - vaildY) ./ vaildY))) /
size(vaildY, 1) / size(vaildY, 2);
disp([Tvalue, '验证集平均相对误差 MAPE: ', num2str(vaildMAPE)])

vaildMSE = (sum(sum(((yVaildPredictFinal - vaildY)) .^ 2)) /
size(vaildY, 1) / size(vaildY, 2));
disp([Tvalue, '验证集均方根误差 MSE: ', num2str(vaildMSE)])
```

```
vaildRMSE = sqrt(sum(sum(((yVaildPredictFinal - vaildY)) .^ 2)) /
size(vaildY, 1) / size(vaildY, 2));
disp([Tvalue, '验证集均方根误差 RMSE: ', num2str(vaildRMSE)])

vaildR2 = 1 - mean(norm(vaildY - yVaildPredictFinal) ^ 2 / norm(vaildY
- mean(vaildY)) ^ 2);
disp([Tvalue, '验证集均方根误差 R2: ', num2str(vaildR2)])
disp('**********************************************************************
******************')

testMAE = sum(sum(abs(yTestPredictFinal - testY))) / size(testY, 1) /
size(testY, 2);
disp([Tvalue, '测试集平均绝对误差 MAE: ', num2str(testMAE)])

testMAPE = sum(sum(abs((yTestPredictFinal - testY) ./ testY))) /
size(testY, 1) / size(testY, 2);
disp([Tvalue, '测试集平均相对误差 MAPE: ', num2str(testMAPE)])

testMSE = (sum(sum(((yTestPredictFinal - testY)) .^ 2)) / size(testY,
1) / size(testY, 2));
disp([Tvalue, '测试集均方根误差 MSE: ', num2str(testMSE)])

testRMSE = sqrt(sum(sum(((yTestPredictFinal - testY)) .^ 2)) /
size(testY, 1) / size(testY, 2));
disp([Tvalue, '测试集均方根误差 RMSE: ', num2str(testRMSE)])

testR2 = 1 - mean(norm(testY - yTestPredictFinal) ^ 2 / norm(testY -
mean(testY)) ^ 2);
disp([Tvalue, '测试集均方根误差 R2: ', num2str(testR2)])
```

附录 20

Python：规划求解每日的最优策略

```
from scipy.optimize import linprog

# 系数
a = [-0.00637, 0.015, 0.0185, -0.00329, -0.00405, 0.0796]
b = [1.3326, 1.2708, 1.1832, 1.4151, 1.3411, 1.2315]
z = [10.6448, 3.1431, 5.7174, 4.6973, 4.3784, 7.2234]
alpha = [0.102803, 0.14142, 0.119747, 0.07122, 0.085153, 0.0813097]
rho = [0.699937, 0.675847, 0.718194, 0.813046, 0.672934, 0.671256]

# 下限和上限
lower_bounds = [135.0464, 17.1808, 0, 7.5831, 25.8570, 27.6851]
```

```
upper_bounds = [163.2700, 45.4364, 25.9125, 35.8387, 54.0805, 55.9086]

# 销售总量上限
total_sales_limit = 280

# 构建线性规划问题
c = [-1 * (a[i] + b[i] * z[i]) for i in range(6)]
A = [[1, 1, 1, 1, 1, 1]]
b_eq = [total_sales_limit]

bounds = [(lower_bounds[i], upper_bounds[i]) for i in range(6)]

# 求解线性规划
result = linprog(c, A_eq=A, b_eq=b_eq, bounds=bounds)

# 检查是否成功找到最优解
if result.success:
    optimal_prices = result.x[:6]
    optimal_sales = result.x[6:]
    print("Optimal Prices:", optimal_prices)
    print("Optimal Sales:", optimal_sales)
else:
    print("No optimal solution found.")
```

附录 21

Python：第三问单品的筛选

```
import pandas as pd

# 读取 Excel 文件
df = pd.read_excel('问题 3.xlsx')

# 选择单品编码、销售日期和销量列
selected_columns = ['单品编码', '销售日期', '销量']
df = df[selected_columns]

# 将销售日期列转换为日期类型
df['销售日期'] = pd.to_datetime(df['销售日期'], format='%Y/%m/%d')

# 筛选出在 6/24 到 6/30 之间的数据
start_date = pd.to_datetime('2023/6/24', format='%Y/%m/%d')
end_date = pd.to_datetime('2023/6/30', format='%Y/%m/%d')
filtered_df = df[(df['销售日期'] >= start_date) & (df['销售日期'] <= end_date)]
```

```
# 创建一个新的 DataFrame，以单品编码为索引，日期为列
pivot_df = filtered_df.pivot_table(index='单品编码', columns='销售日期',
values='销量', aggfunc='sum', fill_value=0)

# 重新排序列，以保证日期顺序正确
date_range = pd.date_range(start_date, end_date)
pivot_df = pivot_df[date_range]

# 显示结果
pivot_df
# 定义函数来检查条件
def filter_single_item(item_row):
    # 统计销量为 0 或小于 2.5 的天数
    zero_sales_count = (item_row == 0).sum()
    below_2_5_sales_count = (item_row < 2.5).sum()

    # 如果 3 天以上销量为 0 或者 7 天中销量小于 2.5 的天数大于 5 天，则返回 False，否则返
回 True
    return not (zero_sales_count >= 3 or below_2_5_sales_count > 5)

# 应用过滤函数到每一行（单品）上
filtered_items = pivot_df.apply(filter_single_item, axis=1)

# 通过索引获取满足条件的单品
selected_items = pivot_df[filtered_items]

# 显示结果
selected_items
# 定义函数来检查条件
def filter_single_item(item_row):
    # 统计销量为 0 或小于 2.5 的天数
    zero_sales_count = (item_row == 0).sum()
    below_2_5_sales_count = (item_row < 2.5).sum()

    # 如果 3 天以上销量为 0 或者 7 天中销量小于 2.5 的天数大于 5 天，则返回 False，否则返
回 True
    return not (zero_sales_count >= 3 or below_2_5_sales_count > 5)

# 应用过滤函数到每一行（单品）上
filtered_items = pivot_df.apply(filter_single_item, axis=1)

# 通过索引获取满足条件的单品
selected_items = pivot_df[filtered_items]
```

```
# 显示结果
selected_items
```

附录 22

Python:禁忌搜索

```
import random

# 基本参数
num_items = 30  # 单品数量
num_days = 7    # 天数
num_categories = 6  # 品类数量

# 单品信息
item_categories = {
    102900005115250: '食用菌',
    102900005115762: '花叶类',
    102900005115786: '花叶类',
    102900005115823: '花叶类',
    102900005115946: '花叶类',
    102900005116257: '茄类',
    102900005116714: '花菜类',
    102900005116899: '水生根茎类',
    102900005118824: '水生根茎类',
    102900005118831: '花叶类',
    102900005119975: '花叶类',
    102900011000328: '辣椒类',
    102900011001691: '水生根茎类',
    102900011008164: '花叶类',
    102900011016701: '辣椒类',
    102900011022764: '茄类',
    102900011023464: '花叶类',
    102900011030059: '花叶类',
    102900011030097: '花叶类',
    102900011030110: '花叶类',
    102900011031100: '辣椒类',
    102900011031926: '食用菌',
    102900011032022: '辣椒类',
    102900011032251: '辣椒类',
    102900011032848: '辣椒类',
    102900011034330: '食用菌',
    102900011034439: '辣椒类',
    102900051000944: '水生根茎类',
    106949711300259: '食用菌',
```

```
    106971533450003: '食用菌'
}

# 品类信息
category_upper_limits = [163.27, 45.4364, 25.9125, 35.8387, 54.0805, 55.9086]
category_lower_limits = [135.0464, 17.1808, 0, 7.5831, 25.857, 27.6851]

# 单品信息
item_prices = {
    102900005115250 :15.6,
    102900005115762 :2.324285714,
    102900005115786 :2.318571429,
    102900005115823 :4.121428571,
    102900005115946 :3.211428571,
    102900005116257 :3.747142857,
    102900005116714 :7.824285714,
    102900005116899 :10.74571429,
    102900005118824 :11.54,
    102900005118831 :4.731428571,
    102900005119975 :3.204285714,
    102900011000328 :7.515714286,
    102900011001691 :9.173333333,
    102900011008164 :2.528333333,
    102900011016701 :3.384285714,
    102900011022764 :6.982857143,
    102900011023464 :2.83,
    102900011030059 :3.604285714,
    102900011030097 :2.864285714,
    102900011030110 :4.098,
    102900011031100 :2.135714286,
    102900011031926 :2.657142857,
    102900011032022 :1.542857143,
    102900011032251 :3.277142857,
    102900011032848 :2.437142857,
    102900011034330 :3.402857143,
    102900011034439 :3.231428571,
    102900051000944 :18,
    106949711300259 :1.452857143,
    106971533450003 :1.954285714
}

item_wear_and_tear = {
    102900005115250 :10.2803/100,  # 转换为小数
    102900005115762 :14.142/100,   # 转换为小数
```

```
    102900005115786 :14.142/100,   # 转换为小数
    102900005115823 :14.142/100,   # 转换为小数
    102900005115946 :14.142/100,   # 转换为小数
    102900005116257 :7.122/100,    # 转换为小数
    102900005116714 :10.2803/100,  # 转换为小数
    102900005116899 :8.13097/100,  # 转换为小数
    102900005118824 :8.13097/100,  # 转换为小数
    102900005118831 :14.142/100,   # 转换为小数
    102900005119975 :14.142/100,   # 转换为小数
    102900011000328 :11.9747/100,  # 转换为小数
    102900011001691 :8.13097/100,  # 转换为小数
    102900011008164 :14.142/100,   # 转换为小数
    102900011016701 :11.9747/100,  # 转换为小数
    102900011022764 :7.122/100,    # 转换为小数
    102900011023464 :14.142/100,   # 转换为小数
    102900011030059 :14.142/100,   # 转换为小数
    102900011030097 :14.142/100,   # 转换为小数
    102900011030110 :14.142/100,   # 转换为小数
    102900011031100 :11.9747/100,  # 转换为小数
    102900011031926 :10.2803/100,  # 转换为小数
    102900011032022 :11.9747/100,  # 转换为小数
    102900011032251 :11.9747/100,  # 转换为小数
    102900011032848 :11.9747/100,  # 转换为小数
    102900011034330 :0.2/100,      # 转换为小数
    102900011034439 :11.9747/100,  # 转换为小数
    102900051000944 :24.05/100,    # 转换为小数
    106949711300259 :0.45/100,     # 转换为小数
    106971533450003 :0/100         # 转换为小数
}
category_info = {
    '花叶类': {'a': -0.00637, 'b': 1.3326, 'ρ': 0.699937},
    '花菜类': {'a': 0.015, 'b': 1.2708, 'ρ': 0.675847},
    '辣椒类': {'a': 0.0185, 'b': 1.1832, 'ρ': 0.718194},
    '茄类': {'a': -0.00329, 'b': 1.4151, 'ρ': 0.813046},
    '食用菌': {'a': -0.00405, 'b': 1.3411, 'ρ': 0.672934},
    '水生根茎类': {'a': 0.0196, 'b': 1.2315, 'ρ': 0.671256}
}

# 定价函数
def calculate_profit(price, quantity, wear_and_tear, cost_price, category):
    category_info_data = category_info[category]
    a = category_info_data['a']
    b = category_info_data['b']
    ρ = category_info_data['ρ']
```

```
    α = wear_and_tear

    y = quantity * ((α * ρ - α + 1) * (a * quantity + b * cost_price) + (α - 1) *
cost_price)

    return y

def check_constraints(solution):
    # 检查每个单品销量不小于 2.5
    #for i, quantity in enumerate(solution):
    #    if quantity < 2.5:
    #        return False
    # 计算每个品类的销量总和
    category_sales = {category: 0 for category in item_categories.values()}
    for i, item in enumerate(item_categories.keys()):
        category = item_categories[item]
        category_sales[category] += solution[i]
    #print(category_sales)
    # 检查每个品类的上下限
    #for category, sales_sum in category_sales.items():
    #    category_index = list(item_categories.values()).index(category)
    #    print(category_index)
    #    if sales_sum > category_lower_limits[category_index]:# or sales_sum >
category_upper_limits[category_index]:
    #        return False

    # 检查总销量不超过 280
    total_sales = sum(solution)
    if total_sales > 365:
        return False
    return True

# 禁忌搜索算法
def tabu_search():
    # 初始化

    best_solution = [0] * num_items
    best_profit = 0
    tabu_list = []
    max_iterations = 100000  # 最大迭代次数
    iteration = 0

    while iteration < max_iterations:
        # 生成邻域解
```

```
        neighbor_solution = best_solution.copy()
        move_item = random.randint(0, num_items - 1)
        neighbor_solution[move_item] += random.randint(-3, 3)  # 随机调整销量

        # 检查约束条件
        if check_constraints(neighbor_solution):
            # 计算邻域解的利润
            neighbor_profit = sum(
                [calculate_profit(item_prices[item], neighbor_solution[i],
item_wear_and_tear[item],
                                  product_prices[item], item_categories[item])
                 for i, item in enumerate(item_categories.keys())])

            # 如果邻域解更优，并且不在禁忌列表中，接受该解
            if neighbor_profit > best_profit and neighbor_solution not in tabu_list:
                best_solution = neighbor_solution
                best_profit = neighbor_profit

                # 更新禁忌列表
                tabu_list.append(neighbor_solution)
                if len(tabu_list) > 10:  # 控制禁忌列表长度
                    tabu_list.pop(0)

        iteration += 1

    return best_solution, best_profit
# 运行禁忌搜索
best_solution, best_profit = tabu_search()
print("最优解（单品销量）：", best_solution)
print("最优利润：", best_profit)
```