基于时间序列联动分析的补货与定价策略研究

摘　要

在生鲜商超中，由于蔬菜类商品的保质期短，为确保商品的品质以及商超的利润最大化，制定合理的蔬菜类商品的补货和定价策略十分重要。本文利用时间序列分解和联动分析探究了蔬菜商品的销售规律以及相互关系，并以此为基础建立非线性规划模型，通过遗传算法求解来确定最优的补货和定价策略。

针对问题一，本文首先对数据进行了预处理以及探索性分析，发现大部分蔬菜单品的分布都具有**季节性差异**。根据这个特征，利用 **STL 分解**将各蔬菜品类或单品的销售量时间序列分解为趋势项、季节项和残差项，进而分析出其销量分布规律。对于各蔬菜品类及单品销售量的相互关系，本文先用**格兰杰因果检验**筛选出存在相互影响的蔬菜品类组合，再用**时间序列的联动分析模型**进一步细化品类组合的相互影响：从**横向时间角度**发现花叶类-花菜类组合存在**相互替代关系**，即两者销售量集中分布在不同季节；从**纵向组内角度**发现花叶类-辣椒类、花叶类-食用菌类等共 6 对组合存在**互补关系**，即组合内部蔬菜销量主要分布在相同季节。最后，本文对各单品间的相关性进行分析，发现大芥兰与菊花油菜、云南生菜与云南油麦菜、竹叶菜和红薯尖等蔬菜单品组合的**相关性**较强。

针对问题二，对于各蔬菜品类的销售总量与成本加成定价的关系，本文先将成本加成定价转为成本加成定价比，然后从**短期趋势、长期波动和周内波动**三个角度进行分析，将一周中的成本定价比与季节项作为自变量，对平均销量进行**多元线性回归**，确定出各蔬菜品类总销量与成本加成定价的关系。对于未来一周最优的补货和定价策略，本文用 **GBDT 算法**并结合蔬菜品类间的**互补替代关系**预测出各蔬菜品类的最大销售量，进而确定补货量的范围。接下来，综合考虑库存以及销量与成本加成定价比关系等约束条件，以最大化收益作为目标函数，建立了**补货与定价决策模型**，再通过**遗传算法**求解出最优的补货和定价策略。其中，**花叶类的补货量最多，花菜类相对较少**，未来一周的**总补货量为 2444.19 千克，预计最大收益为 10021 元**。

针对问题三，本文先从附件 2 中提取 6 月 24 日至 30 日的可销售单品，筛选出 49 种蔬菜单品。在有限的销售空间下，选取**平均销量大、波动变化小、关联度强**的蔬菜单品有利于商超收益最大化。故首先通过平均销量和销量波动两个指标综合确定蔬菜单品的**优先度**，再根据它们的相关性强弱进行**层次聚类，**确定**最佳的销售组合，**进而利用互补替代关系的影响对优先度进行修正，筛选出排名前 27 到 33 的蔬菜单品。最后基于问题二得到的 7 月 1 日的销售量，综合考虑最小陈列量，构建新的补货与决策模型，并通过**遗传算法逐步求解**，求解得到当**蔬菜单品数为 30** 时的补货定价策略具有最大收益，**总订货量为 307.987 千克，预计最大收益为 747.482 元。**

针对问题四，本文从**外部因素**和**内部因素**进行分析。宏观的外部因素包括行业的相关政策、通货膨胀、竞争对手以及供应链。微观的内部因素包括商超所在地区消费者的总体消费水平以及消费者对于商超的反馈。最后在文中详细讲述了这些数据对商超确定补货和定价策略等问题的帮助。

最后，本文对补货与定价决策模型关于成本加成定价比的波动比例θ的灵敏度进行分析，检验了模型的稳定性，分析得到模型具有一定的抗干扰性。

关键词：STL 分解　时间序列联动分析　GBDT 算法　非线性规划　遗传算法

一、 问题重述

1.1 问题背景

随着近几年人们生活水平的提高，对新鲜食品的需求也随之涨高，于是，大量的生鲜商超开始络绎不绝地涌现。在这类商超中，蔬菜类商品保质期短，较易腐烂，且不能隔日再售。为保证商品的品相以及商超利润，商超会根据每款商品历史的销售情况以及需求量来确定补货和定价策略。

对于每日的补货，因为蔬菜类商品的种类之多以及产地之广，商家无法提前确定具体的单品以及进货价格。对于蔬菜的定价，由于存在运损和品相变差的产品，商家需要采用“成本加成定价”方法对这类产品进行打折促销。因此，进行准确的市场分析对补货和定价策略的制定来讲十分重要。在市场需求方面，蔬菜类产品销量与时间存在一定关系；在市场供给方面，4 月份至 10 月份的蔬菜类品种较为丰富，而由于商超的空间限制与蔬菜种类多的矛盾，商超则需要制定合理的蔬菜种类销售组合。

1.2 数据集

附件 1 给出了 6 个蔬菜类的商品信息；附件 2 给出了销售的明细流水数据；附件 3 给出了蔬菜类各商品的批发价格；附件 4 给出了蔬菜类各商品的近期损耗率。

1.3 问题提出

问题一：结合不同蔬菜品类或不同蔬菜单品间的潜在联系，分析出各品类蔬菜及各蔬菜单品的销售量分布规律及相互关系。

问题二：商超在补货时以蔬菜品类为单位，为追求商超的最大化利益，分析出各蔬菜品类的销售总量和成本加成定价之间的关系，确定出各蔬菜品类未来一周（即 2023 年 7 月 1 日到 7 月 7 日）的每日补货总量和定价策略。

问题三: 由于商超中蔬菜类商品的销售空间存在限制，可销售的蔬菜单品总数需控制在 27 至 33 种，同时每种单品在订购时需满足最小陈列 2.5 千克的要求。根据 2023 年 6 月 24 日至 30 日的可销售单品品种，基于上述的要求，在保证商超的最大化利益的前提下，制定出 7 月 1 日的蔬菜单品单日补货和定价策略。

问题四：为了制定更好的蔬菜类商品补货及定价策略，给出商超还可以收集的其他方面数据，并分析出这些数据的对解决上述问题的作用。

二、 问题分析

2.1 问题一的分析

对于问题一，要求分析出各蔬菜单品的销售分布规律及相互联系。考虑到题目所提供的数据量较大，无法直观清晰地观察出规律，本问首先对数据进行预处理并通过数据可视化对其进行探索性分析。为了分析各类蔬菜单品的销售规律，可以利用时间序列的 STL 分解方法，将每个研究样本的销售量随时间变化的曲线分解为趋势项、季节项和残差项，以此来研究销量的趋势、周期以及波动等分布规律。而对于各蔬菜销售的相互关系，首先我们对各蔬菜品类的销售量序列进行格兰杰因果分析，筛选出有一定因果关系的蔬菜品类组合；其次，为了确定这些组合内两种单品之间是相互替代或者是互补的关系，本文通过时间序列的联动分析来量化不同蔬菜样本的销售量之间的相关程度，进而得出不同蔬菜样本销售量之间的相互关系。

2.2 问题二的分析

对于问题二，第一小问要求分析各蔬菜品类总销量与成本加成定价的关系，第二小问中要求确定未来一周每天的补货和定价策略。对于第一小问，首先可以将问

题转化为分析一周中每一天各蔬菜品类总销量与成本加成定价比的关系。接下来，由附件中的成本价和批发价确定成本加成定价比。对于各蔬菜品类的总销量，可以从短期趋势、长期的波动以及周内的波动特征三个角度来分析，将一周中的成本定价比与季节项作为自变量，对平均销量进行多元线性回归，由此来确定出各蔬菜品类总销量与成本加成定价的关系。对于第二小问，首先为了确定补货量的范围，可以基于 GBDT 算法对各蔬菜品类的最大销售量进行预测。再以库存的范围以及销量与成本加成定价比作为约束条件，以最大收益作为目标函数，建立基于遗传算法的补货与订价决策模型求解出最优的补货和定价策略。

2.3 问题三的分析

根据题意，首先可以从附件 2 中把从 2023 年 6 月 24 日至 30 日的可销售单品品种筛选出来。为了收益的最大化，考虑历史的销售数据，故通过这段时间内的平均销量和销量波动两个指标综合确定蔬菜单品筛选的优先度。再综合考虑蔬菜单品间的互补替代关系的影响，根据蔬菜单品间的相关性强弱进行层次聚类，划分互补替代关系，再对优先度进行修正，用修正后的优先度筛选出排名前 27 到 33 的蔬菜单品。最后，结合问题二得到的 7 月 1 日销售量以及蔬菜单品最小陈列 2.5 千克的要求，构建新的补货与决策模型，并通过遗传算法逐步求解出最优补货和定价策略。

2.4 问题四的分析

本问要求找出商超还可以收集的数据以制定更好的补货和定价策略，可以从外部因素和内部因素进行分析。宏观的外部因素包括行业的相关政策、通货膨胀、竞争对手以及供应链。微观的内部因素包括商超所在地区消费者的总体消费水平以及消费者对于商超的反馈。最后，再根据这些数据对解决补货策略和定价策略的制定问题的帮助。

三、 模型假设

1. 假设不同蔬菜销售量的白噪声是互不相关的；
2. 假设不同蔬菜单品之间的相互关系是双向的；
3. 假设蔬菜品相随时间的变化服从伯努利分布；
4. 假设品相差和损耗的蔬菜全都打折出售；
5. 假设短期内蔬菜的批发价变化可以忽略。

四、 符号说明

符号	说明
Y_t	蔬菜单品 t 的时间序列
T_t	蔬菜单品 t 的时间序列趋势项
S_t	蔬菜单品 t 的时间序列季节项
R_t	蔬菜单品 t 的时间序列残差项
B_i	第 i 笔订单的定价
A_i	第 i 笔订单的批发价
w_i	成本加成定价比
ρ_{jx_k}	第 j 类蔬菜品类的第 x 天的第 k 笔订单销量
Q_{jm}	第 j 类蔬菜品类一周中第 m 天的总销量
$\widehat{Q_{ij}}$	第 i 类蔬菜第 j 天的最大预测销售量

五、 数据的预处理和探索性分析

5.1 数据的预处理

附件 2 中给出了销售流水数据，对于存在部分单品的退货记录，将其与之前销售记录相互抵消，即将此销售记录去除。另外，在附件 2 中存在某些月份中缺失了某几天的数据，利用同期的数据进行等比例替换。

5.2 数据的探索性分析

由于附件 1 中给出的 6 个蔬菜类产品信息以及附件 2 给出的销售流水数据的数据量较大，为使信息更加直观以及便于之后对各蔬菜单品的销售量分布规律的分析，我们首先对数据进行预处理并通过数据可视化对其进行探索性分析。

5.2.1 各个蔬菜品类下单品数目的描述性统计.

通过附件 1 中的 6 个蔬菜类产品信息，统计得到各个蔬菜类下单品数目，结果如表 1 所示。

表 1 各蔬菜品类下单品数目统计

品类	花叶类	食用菌	辣椒类	水生根茎类	茄类	花菜类
单品数	100	72	45	19	10	5

由表 1 可以看出，不同蔬菜品类下的单品数量具有较大的差异性。其中，花叶类蔬菜下的单品数目最多，共 100 种，而花菜类蔬菜下的单品数目最少，仅有 5 种。

5.2.2 需求侧：各蔬菜单品销量的统计性分析

我们对各蔬菜单品 2020 年到 2022 年三年的销售总量进行了统计，如图 1 所示。

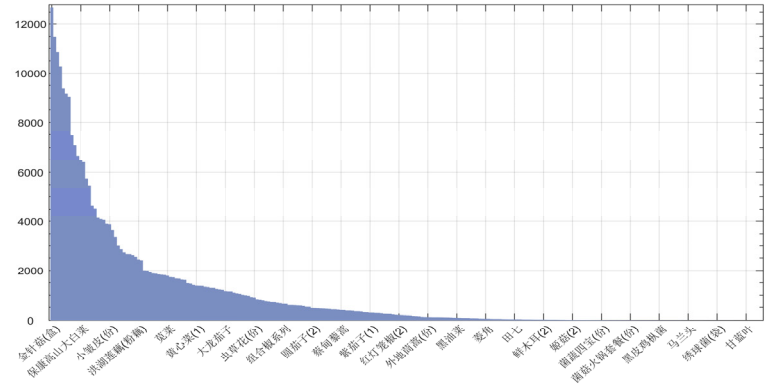


图 1 各蔬菜单品总销量统计

由图 1 可看出，各蔬菜单品间销量分布具有不均匀性，部分蔬菜单品的销量几乎为 0，主要销量集中在部分的蔬菜单品，在商超进行补货时可作为一个重要的参考依据。

5.2.3 供给侧：不同月份下蔬菜单品的数目分布

在市场供给方面，通过对附件 1 和附件 2 数据进行分析，我们绘制出了 21 年和 22 年的蔬菜单品数目随月份的变化，如图 2 所示。

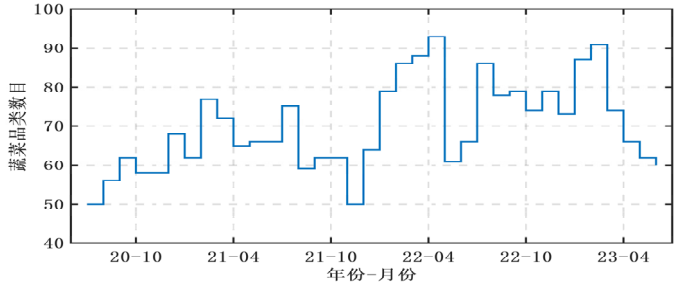


图 2 蔬菜单品数目随月份的变化

由图 2 相同月份的蔬菜单品数目具有相似的分布规律可以看出，大部分蔬菜单品的分布都具有季节性差异，具体表现为蔬菜品类的数目主要分布在 3 到 5 月和 9 到 11 月。

六、 问题一模型的建立和求解

6.1 各蔬菜品类及单品的销售分布规律

6.1.1 时间序列 STL 分解

结合前文对产品的销量和时间关系的分析，我们得到了每款蔬菜单品销量的时间变化序列。由于这些时间序列数据的复杂性，我们难以直观地发现其规律，而通过时间序列的 STL 分解， 我们可以将时间序列分解为趋势项、季节项和残差项来直观地分析其分布规律。

STL 分解包括内循环和外循环两个部分，内循环主要是分解计算时间序列的趋势项和季节项两个分量，而外循环主要用于调节鲁棒性的权重[1]。对于一个时间序列 Y_t，可分解为：

$$Y_t = T_t + S_t + R_t, \left(t = 1, 2, \cdots N\right) \tag{1}$$

其中，T_t、S_t 和 R_t 分别表示时间序列的趋势项、季节项和残差项。N 表示蔬菜品类或单品的数目。

6.1.2 销售分布规律结果及分析

通过上述时间序列的 STL 分解，得到各个品类蔬菜的三年的销量时间序列的趋势项、季节项和残差项，具体结果在附件中展示，由于篇幅限制，此处仅展示茄类的销量时间序列分解结果，如图 3 所示。

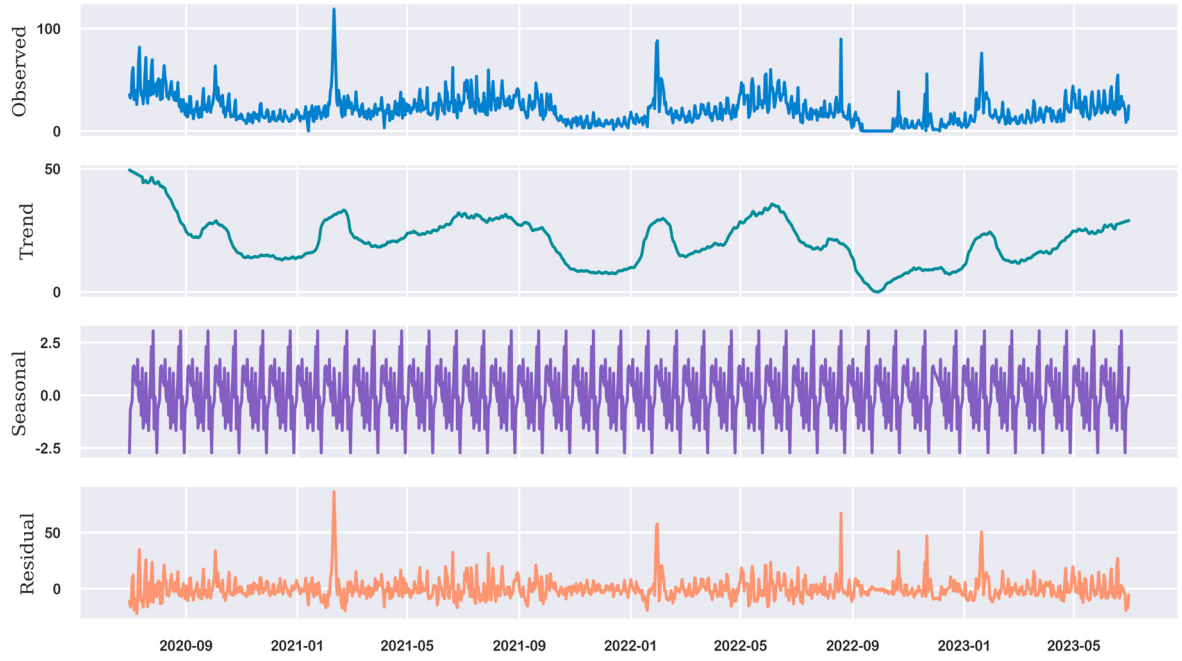


图 3 茄类销量时间序列分解结果

从分解后的整体趋势中可观察到，茄类的销量在 1 月份和 2 月份有比较明显的上升趋势，3 月份和 4 月份销量有所下降，而从 5 月份开始又有小幅平稳的上升趋势。同时，其销量有逐年减少的趋势，总体销量保持在 50 千克以内。图中第三段紫色的曲线为季节项子序列，即周期项，第四段橙色的曲线为残差项，即不规则波动。

而对于花叶类，根据其分解出的趋势项可以发现，其销量在每年 9 月份时会迎来较大的增长，销量集中分布于春季与秋季，总体的销量在 200 千克左右。对于花菜类，其销量在 2022 年 9 月份呈现出较大幅度的增长，其总体销量在 50 千克左右。对于水生根茎类，其销量在每年 3 月到 8 月销量较低，在每年 9 月到第二年的 1 月份销量较高，总体销量在 25~75 千克之间。对于辣椒类，其在 2022 年 9 月到 2023 年销量较大，在每年的一月份会迎来小幅度的增长，总体销量在 50~150 千克之间。最后，对于食用菌类，在每年 3 月到 8 月的销量低迷，主要销量分布在每年的 9 月到第二年的 2 月，总体销量在 50~150 千克之间。

另外，对各蔬菜单品，我们选取了各蔬菜品类中销量最多的蔬菜单品作为代表，分析其时间序列的趋势项子序列，如图 4 所示。

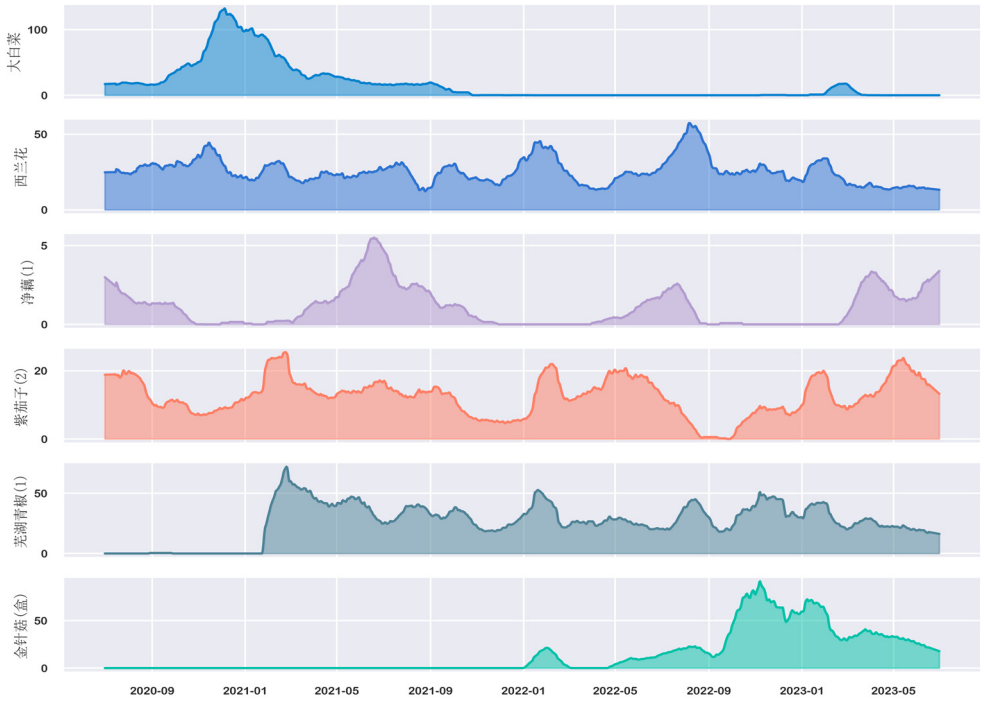


图 4 蔬菜单品代表的销量时间序列趋势项

由图 4 可以看出，花叶类单品代表大白菜在销量集中于 2020 年 9 月至 2021 年 3 月，而 22 年的销量较为惨淡；花菜类代表西兰花的销量随时间分布较为均匀，但随月份的变化波动较大；水生根茎类代表净藕(1)的销量集中分布于每年 3 月份到隔年的 6 月份，即春季的销量较多；受冬季气温较低的影响，茄类的代表紫茄子(2)的销量在每年 12 月份会有比较明显的下降趋势；辣椒类的代表芜湖青椒(1)在 2020 年 7 月到 12 月销量几乎为零，在 2021 年初销量大幅上升后恢复到稳定波动的趋势；食用菌类的代表金针菇主要销量集中于 2022 年的 6 月份到 12 月份的夏季和秋季。

6.2 基于联动分析的销售量相互关系研究

6.2.1 格兰杰因果检验

基于 6.1 中得到的各蔬菜品类销售量的时间序列，首先对各个蔬菜品类销量时间序列之间进行格兰杰因果检验[2]，筛选出存在相互影响的蔬菜品类组合，以便于后文对其相互关系的量化分析。

首先，运用 ADF 检验方法检验各时间序列的平稳性[3]，从而避免数据的“伪回归”现象。对于不平稳的时间序列，进行差分处理直到通过平稳性检验。在得到平稳的时间序列后，利用格兰杰因果检验模型检验各蔬菜品类间是否存在相互影响的

因果关系。根据 AIC 准则确定最优滞后阶数，最后确定出存在相互影响因果关系的蔬菜品类组合如下表所示。

表 2 存在因果关系的蔬菜品类组合

花菜 $\rightarrow$ 花叶	花菜 $\rightarrow$ 水生根茎	食用菌 $\rightarrow$ 水生根茎	茄类 $\leftrightarrow$ 水生根茎
水生根茎 $\leftrightarrow$ 花叶	茄类 $\leftrightarrow$ 花菜	辣椒 $\leftrightarrow$ 茄类	辣椒 $\rightarrow$ 水生根茎
茄类 $\leftrightarrow$ 花叶	辣椒 $\leftrightarrow$ 花叶	食用菌 $\leftrightarrow$ 花叶	花菜 $\rightarrow$ 食用菌

表 2 中，“A $\rightarrow$ B”表示 A 类蔬菜可以引起 B 类蔬菜的变化，而 “A $\leftrightarrow$ B” 则表示 A 和 B 两类蔬菜会相互引起对方的销量变化。

6.2.2 基于因果检验的时间序列联动分析

根据上文筛选出的会相互影响的蔬菜品类组合，为了细化其相互影响关系，我们从**横向时间的角度**分析不同品类蔬菜的**相互替代关系**，从**纵向组内的角度**分析不同品类蔬菜的**互补关系**，接下来对这些蔬菜品类的销售量时间序列之间进行联动分析[4]。具体的步骤如下：

步骤 1：归一化。对时间序列曲线 $Y_t=\{y_{t1},\ldots,y_{tn}\}$ 进行 $z-score$ 的归一化，其中，n 表示 2020 年 7 月份到 2023 年 6 月份总的天数，归一化具体计算公式如下：

$$\mu_t=\frac{\sum_{i=1}^{n}y_{ti}}{n},\quad \delta_t=\sqrt{\frac{\sum_{i=1}^{n}\left(y_{ti}-\mu\right)^2}{n}} \tag{2}$$

得到归一化后新的时间序列 $Y_t'=\{y_{t1}',\ldots,y_{tn}'\}$ 为：

$$Y_t'=\frac{Y_t-\mu_t}{\delta_t} \tag{3}$$

步骤 2：特征放大。对归一化后的时间序列曲线进行特征放大，定义函数 $f_{\alpha,\beta}(x)$ 如下式所示：

$$f_{\alpha,\beta}(x)=\begin{cases}e^{\alpha\min(x,\beta)}-1,(x\geq 0)\\ -e^{\alpha\min(|x|,\beta)}+1,(x<0)\end{cases} \tag{4}$$

由此可以得到时间序列的特征放大特征曲线 $\hat{Y}_t=\left\{f\left(y_{t1}'\right),\ldots,f\left(y_{tn}'\right)\right\}$。

步骤 3：相关性分析。在各蔬菜单品中，对于蔬菜单品 k 和蔬菜单品 g 的时间序列特征放大曲线 $\hat{Y}_k=\left\{f\left(y_{k1}'\right),\ldots,f\left(y_{kn}'\right)\right\}$ 和 $\hat{Y}_g=\left\{f\left(y_{g1}'\right),\ldots,f\left(y_{gn}'\right)\right\}$，为便于后文的表述，将其分别记为 $K=\{k_1,\ldots,k_n\}$ 和 $G=\{g_1,\ldots,g_n\}$。首先计算两者的相关性，若存在相关性，再分析两条曲线的波动先后顺序，波动的方向一致或相反。令：

$$K_s=\begin{cases}\{0,\cdots,0,k_1,\cdots,k_{n-s}\},(s\geq 0)\\ \{k_{n-s},\cdots,k_1,0,\cdots,0\},(s<0)\end{cases} \tag{5}$$

在式 (5) 中，$|s|$ 表示 0 的个数，且 $-n<s<n$。值得注意的是，当 $s=0$ 时，满足：

$$K_0=\{k_1,\cdots,k_s\}=K \tag{6}$$

定义K_s与G的内积为$R(K_s,G)=K_s\cdot G$，相关性的具体计算式如下：

$$CC(K_s,G)=\frac{R(K_s,G)}{\sqrt{R(K_s,G)\cdot R(G,G)}} \tag{7}$$

由于两条时间序列曲线间的波动方向可能一致也有可能相反，所以需要考虑相关性大于零和小于零两种情况。即：

$$\begin{aligned}\min CC&=\min_{-n<s<n} CC(K_s,G)\\ \max CC&=\max_{-n<s<n} CC(K_s,G)\end{aligned} \tag{8}$$

其中，最大值与最小值的指标为：

$$\begin{aligned}s_1&=\arg\min_{-n<s<n} CC(K_s,G)\\ s_2&=\arg\max_{-n<s<n} CC(K_s,G)\end{aligned} \tag{9}$$

设定元组$FCC(K,G)$为：

$$FCC(K,G)=\begin{cases}(\min CC,s_1),(|\max CC|<|\min CC|)\\ (\max CC,s_2),(|\max CC|\geq|\min CC|)\end{cases} \tag{10}$$

该元组中代表着三个信息：两个时间序列的显著相关性、发生波动的先后顺序以及波动的方向。首先，$FCC(K,G)$限制在$[-1,1]$这个区间，当其值越接近于 1 或者是-1 时，说明K和G之间的相关性越强。对于发生波动的先后顺序的判断，$s>0$时说明G先于K发生波动，反之则K先于G发生波动。对于波动方向的判断，$FCC(K,G)$大于 0 时，说明波动方向相同，即两种蔬菜单品的销售量时间序列曲线正相关，反之则为负相关。

6.2.3 各蔬菜品类及单品销售量相互关系结果的分析

对于各品类，根据它们的时间序列之间波动先后的差值，即两个时间序列发生波动的先后相差天数，当相差天数大于 7 天且小于 14 天时则将这两种蔬菜单品视为相互替代的关系，但相差天数小于 7 天时则将这两种蔬菜单品视为互补的关系。图 5 展示了具有互补和相互替代关系的蔬菜单品组合以及组内蔬菜单品间的相关性大小。

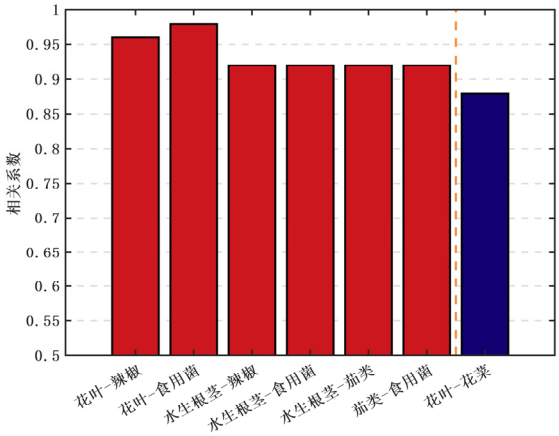


图 5 互补或相互替代的蔬菜品类及相关性大小

由图 5 中可以看出，用红色标注的是存在互补关系的蔬菜品类组合，包括了花叶和辣椒、花叶和食用菌、水生根茎和食用菌、水生根茎和茄类以及茄类和食用菌，这些蔬菜组合的销售量在时间维度上相对集中分布，同时由纵向所表示的相关性可以发现，这些蔬菜单品的相关性都比较强。用蓝色标注的是存在相互替代关系的蔬菜单品组合，即花叶和花菜，这两种蔬菜品类的销售量在时间维度上分布相隔较远，即花叶的销售量集中于某些月份，而花菜的销售量集中于另外一些月份。

另外，对于各蔬菜单品，我们选取其中相关性较强的一组单品，其相关性大小如图 6 所示：

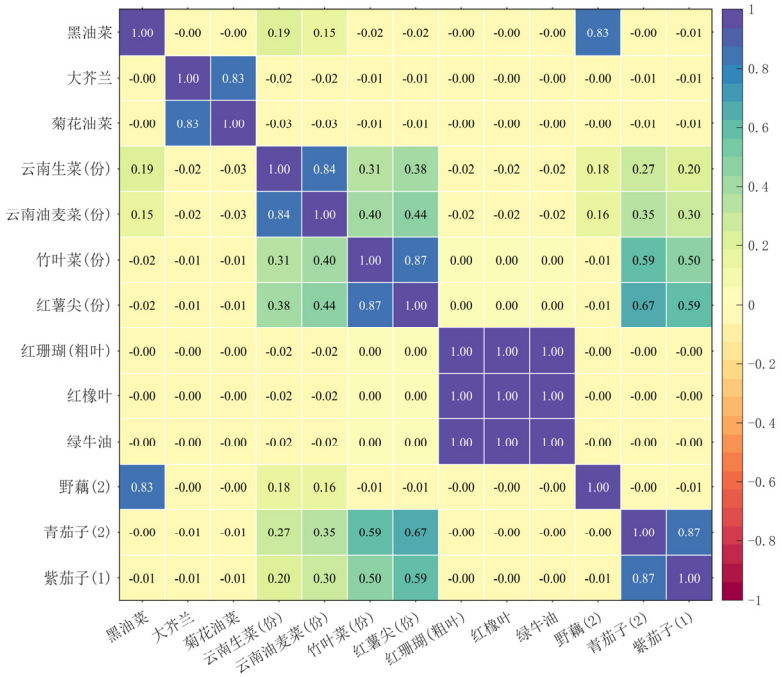


图 6 各蔬菜单品相关性大小

由图 6 的各单品相关性中可以看出，大芥兰与菊花油菜、云南生菜与云南油麦菜、竹叶菜和红薯尖、红珊瑚与红橡叶与绿牛油、青茄子与紫茄子这种蔬菜单品组合的相关性较强。

七、 问题二模型的建立和求解

7.1 各蔬菜品类总销量与成本加成定价比关系研究

7.1.1 成本加成定价比的分析

第一小问要求分析各蔬菜品类总销量与成本加成定价的关系。首先，可以将成本加成定价等价为成本加成定价比 w_i 来研究，其具体转换公式如下：

$$B_i = A_i \cdot (1 + w_i) \tag{11}$$

其中，B_i 表示第 i 笔订单的定价，可以从附件 2 中获得，A_i 表示第 i 笔订单的批发价，可以从附件 3 中获得。

对于第 j 类蔬菜品类，研究其第 x 天的平均成本定价比，具体的计算式如下：

$$\overline{w_{jx}} = \frac{\sum_{k=1}^{n} \rho_{jx_1} w_{jx_k}}{\sum_{k=1}^{n} \rho_{jx_k}} \tag{12}$$

其中，ρ_{jx_k} 和 w_{jx_k} 分别表示第 j 类蔬菜品类的第 x 天的第 k 笔订单销量和成本加成定价比。

7.1.2 单周各蔬菜品类总销量的影响因素分析

对于蔬菜品类 j 每一周的总销量，我们从以下几个角度考虑其影响因素：

1. 短期趋势角度：考虑到不同的季节月份对蔬菜品类的销量存在影响，则可以从近若干天的销量变化趋势来分析，将蔬菜品类 j 的变化趋势的影响记为：

$$f\left(Q_{j(m-1)},Q_{j(m-2)},\cdots,Q_{j(m-v)}\right) \tag{13}$$

其中，$Q_{j(m-v)}$ 表示蔬菜品类 j 的往前第 v 天总销量。

2. 长期波动角度：对于长期的波动，基于问题一，我们可以从以周为时间单位的时间序列的季节项的周期波动和残差项不规则波动两个角度视为影响销量变化的长期因素，记为 ε_j。

3. 周内波动特征：一周之内的波动特征可以从两个角度来分析，一个是每日的成本加成定价比 $\overline{w'_{jm}}$ 对销量的影响以及问题一求解出的每周内时间序列的周期项波动，记为 S_{jm}。

7.1.3 各蔬菜品类总销量与成本加成定价比关系分析

基于上述的角度，将第 j 类蔬菜品类一周中第 m 天的总销量 Q_{jm} 写为：

$$Q_{jm}=\overline{Q_{jm}}+f\left(Q_{j(m-1)},Q_{j(m-2)},\cdots,Q_{j(m-v)}\right)+\varepsilon_j \tag{14}$$

设 $\overline{w'_{jm}}$ 为第 j 类蔬菜品类一周中第 m 天的平均成本加成定价比，对一周中每天的总平均销量 $\overline{Q_{jm}}$ 用时间序列的季节项 S_{jm} 以及成本加成定价比 $\overline{w'_{jm}}$ 进行多元线性回归，具体计算式如下：

$$\overline{Q_{jm}}=\beta_{0j}+\beta_{1j}\overline{w'_{jm}}+\beta_{2j}S_{jm} \tag{15}$$

其中，β_{0j}、β_{1j} 和 β_{2j} 为第 j 个蔬菜品类的回归系数。得到六类蔬菜品类的拟合系数以及 R^2 (拟合准确性)如下表所示：

表 3 六类蔬菜品类的回归系数以及 R^2 (拟合准确性)

品类	花叶类	花菜类	水生根茎类	茄类	辣椒类	食用菌类
β_0	182.696	38.4825	37.3884	20.6799	83.9565	70.3919
β_1	0.331755	0.015051	0.023747	-0.007184	0.593782	-0.412205
β_2	1.011438	1.019519	0.994415	1.015009	1.000676	1.002297
R^2	0.999932	0.999909	0.999997	0.999817	0.999990	0.999993

由表 3 中可以看出，各品类蔬菜的拟合准确性均趋近于 1，拟合准确性较高。其中，茄类、水生根茎类和花菜类的 β_1 绝对值较小，即说明成本加成定价比对其销量的影响程度较小，属于季节性销售类蔬菜品类。辣椒类、花叶类和食用菌类的 β_1 绝对值较大，说明成本加成定价比对它们其销量的影响程度较大，并且对于食用菌，其 β_1 为负值，即表示其成本加成定价比对销量影响较小，而主要以季节性趋势为主。

7.2 未来一周的补货和定价策略的制定

本小问要制定未来一周最优的补货和定价策略来让商超的利益最大化，而收益取决于每日的销售量以及定价，以此出发来制定最优的补货和定价决策。

7.2.1 基于 GBDT 算法的最大销售量预测

在制定补货策略时，根据前面的假设，每日的补货量若过多则会导致当天无法售出造成亏损，每日的补货量若较少则会导致收益无法最大化。因此我们需要对每种蔬菜品类的最大销售量进行预测以确定一个合理的补货量范围。

本文基于 GBDT 算法对最大销售量进行预测。GBDT 算法是集成算法的一种，其主要思想是通过集成多棵决策树形成一个强学习器，并用自适应梯度下降法更新每个弱学习器的权重，以达到增强预测准确性的目的[5]。预测模型函数可表示为：

$$F\left(Q_j;q_1\right)=\sum_{t=0}^{T}\alpha_t h_t\left(Q_j;q_{1t}\right)=\sum_{t=0}^{T}f_t\left(Q_j;q_{1t}\right) \tag{16}$$

其中，Q_j 表示输入样本，即第 j 类蔬菜的历史销量数据；h_t 表示 t 棵回归树；α_t 表示第 t 棵回归树的权重；q_{1t} 表示第 t 棵回归树的参数；f_t 表示第 t 棵回归树的值；T 表示预设置的回归树的数目。

对于第 N 个样本，求解最优预测模型可转化为求最小的 L_1 损失函数。最后得到六种蔬菜品类最大销售量的预测，预测的拟合优度如下表所示：

表 4 最大销售量拟合优度 R^2

品类	花叶类	花菜类	水生根茎类	茄类	辣椒类	食用菌类
训练集 R^2	0.99322	0.98717	0.99319	0.99135	0.99384	0.99352
测试集 R^2	0.60119	0.77374	0.78583	0.74616	0.53956	0.64094

由表 4 中可以看出，训练集的拟合优度很高，同时测试集的拟合优度也达到了较高的水平，但是这种算法不能完全体现时间尺度上不同品类之间的相互关系，所以本文基于问题一得到的蔬菜品类间关系对预测结果进行进一步调整：对于存在互补关系的两种蔬菜品类，它们的销量变化趋势应该是相似的，即蔬菜的销售季节应该相同，所以一种蔬菜的畅销的同时另一种蔬菜的销量也会随之上升。因此，我们将一种蔬菜的销量变化反映到另一种蔬菜的销量变化上，当 A 品类蔬菜多销售 a kg 时，B 品类蔬菜同时间的销量也会上涨 b kg，a 和 b 成正比，相关性越强的组合这一比率应该更高；对于存在替代关系的两种蔬菜品类则相反。

通过以上方式的调整，可以得到新的最大销售量与成本加成定价比的关系式，

$$\hat{Q}_{jm}=\beta_{0j}+\beta_{1j}w_{jm}+\beta_{2j}S_{jm}+(1+\rho_j)F\left(Q_j;q_1\right) \tag{17}$$

其中，ρ_j 表示第 j 类蔬菜的调整比例。

7.2.2 基于遗传算法的补货与定价决策模型

A. 约束条件

在约束方面，可以从两个角度出发，一方面当天补货量过大造成的亏损和补货量较少带来的收益减少，另一方面是销售量与成本加成定价比之间的关系。

从补货量的角度，I_{ij} 表示第 i 类蔬菜第 j 天的库存量，其具体计算式如下：

$$I_{ij}=\begin{cases}Q_{ij}-\hat{Q}_{ij}, Q_{ij}>\hat{Q}_{ij}\\ 0, Q_{ij}\le\hat{Q}_{ij}\end{cases} \tag{18}$$

其中，Q_{ij} 和 $\widehat{Q_{ij}}$ 为第 i 类蔬菜第 j 天的补货量和预测的最大销售量 。

从销售量与成本加成定价比之间的关系，满足：

$$\widehat{Q}_{ij}=\beta_{0j}+\beta_{1j}w_{ij}+\beta_{2j}S_{ij}+(1+\rho_j)F\left(Q_{ij};q_1\right) \tag{19}$$

其中，w_{ij} 为第 i 类蔬菜第 j 天的成本加成定价比，为决策变量，满足：

$$w_{ij}\in\left[(1-\theta)\overline{w}_{ij},(1+\theta)\overline{w}_{ij}\right] \tag{20}$$

其中，θ 为规定的成本加成定价比在其平均值的波动比例。

B. 目标函数

本问要制定未来一周最优的补货和定价策略使得商超的利益最大化。从商超的利益角度出发，收益取决于每日的销售量以及定价，未来一周的收益 Z 如下：

$$Z=\sum_{i}^{n}\sum_{j}^{7}Z_{ij}=\sum_{i}^{n}\sum_{j}^{7}\left(Z_{1ij}+Z_{2ij}\right) \tag{21}$$

其中，Z_{1ij} 表示第 i 类蔬菜第 j 天的无损耗部分的收益，具体计算式如下：

$$Z_{1ij}=Q_{ij}\left(1-L_{ij}\right)A_{ij}\left\{w_{ij}\left[\left(1-p_2\right)+p_2\left(1-p_1\right)\right]+\left[\left(1+w_{ij}\right)\lambda-1\right]p_2p_1\right\} \tag{22}$$

在式 (22) 中，L_{ij} 表示在第 i 类蔬菜第 j 天的运输中损耗的比例。p_2 表示销售量分布于晚市的概率，此时有些蔬菜的品相可能有所下降，故确定打折的概率 p_1 以及打折率 λ 来进行打折促销，p_1 和 p_2 可以从附件 2 中统计出来。

在式 (21) 中，Z_{2ij} 表示第 i 类蔬菜第 j 天的损耗部分的收益减去无法隔日再售的库存的亏损后的实际收益，具体计算式如下：

$$Z_{2ij}=Q_{ij}\cdot L_{ij}\cdot\left[\left(1+w_{ij}\right)\lambda-1\right]-I_{ij}\cdot A_{ij} \tag{23}$$

总的最大收益目标函数可以写为：

$$\max Z=\max\sum_{i}^{n}\sum_{j}^{7}\left(Z_{1ij}+Z_{2ij}\right) \tag{24}$$

综上所述，得到基于非线性规划的补货和定价模型如下：

$$\max Z=\max\sum_{i}^{n}\sum_{j}^{7}\left(Z_{1ij}+Z_{2ij}\right)$$

$$s.t.\begin{cases}\widehat{Q}_{ij}=\beta_{0j}+\beta_{1j}w_{ij}+\beta_{2j}S_{ij}+(1+\rho_j)F\left(Q_{ij};q_1\right)\\ I_{ij}=\begin{cases}Q_{ij}-\widehat{Q}_{ij},Q_{ij}>\widehat{Q}_{ij}\\ 0,Q_{ij}\le\widehat{Q}_{ij}\end{cases}\\ w_{ij}\in\left[(1-\theta)\overline{w}_{ij},(1+\theta)\overline{w}_{ij}\right]\end{cases} \tag{25}$$

7.2.3 模型求解及结果分析

对上述非线性规划模型，利用遗传算法[6]进行求解。遗传算法包括选择、交叉和变异三个基本操作。

1) **选择操作：**将个体适应度评估较优的个体挑选出来，以直接遗传或是配对交叉产生新个体的方式遗传到下一代。

2) **交叉操作：**为了组合有益的基因，将种群中两个父代个体根据交叉率随机对换某些基因，产生出新的基因组合。

3) **变异操作：**变异操作是改变种群中某一个体的某点基因值以产生更优的个体。变异操作可以让遗传算法具备局部的随机搜索能力，从而加强交叉操作向最优解收敛的速度。

当最优群体与个体的适应度不再上升时，迭代次数达到预设值时算法终止。得到未来一周（即 2023 年 7 月 1 日到 7 月 7 日）的每日各蔬菜品类补货量如图 7 所示，各蔬菜品类的成本加成定价比如表 5 所示。

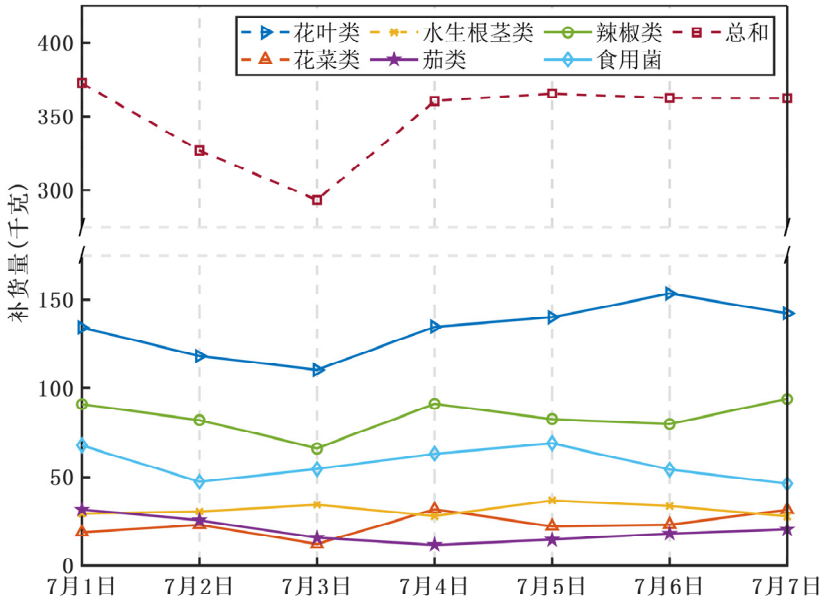


图 7 未来一周各蔬菜品类补货量策略

由图 7 中分析可得，在未来一周中，花叶类蔬菜需要的补货量较多，约占总补货量的三分之一，且其补货量呈现先下降再上升再下降的趋势，这个趋势与总销量趋势相近。而与花叶类存在相互替代关系的花菜类，由于两者的销量集中季节不同，花菜类的补货量远远小于花叶类。另外，辣椒类和食用菌类蔬菜补货量也较多。对总补货量，其也呈现了一个先下降再逐渐上升的变化趋势，未来一周平均每日的补货量大致为 350 千克。

表 5 未来一周各蔬菜品类成本加成定价比

品类	7 月 1 日	7 月 2 日	7 月 3 日	7 月 4 日	7 月 5 日	7 月 6 日	7 月 7 日
花叶类	0. 843	0. 858	0. 853	0. 871	1. 141	0. 901	0. 835
花菜类	0. 609	0. 606	0. 588	0. 613	0. 609	2. 401	0. 622
水生根茎类	0. 535	0. 57	0. 534	0. 574	1. 12	0. 534	0. 573
茄类	0. 68	0. 677	0. 631	0. 64	1. 88	0. 65	0. 651
辣椒类	0. 832	0. 846	0. 906	0. 852	0. 855	0. 815	0. 815
食用菌类	0. 711	0. 694	0. 725	0. 704	0. 672	0. 703	0. 736

由表 5 可得，花叶类的成本加成定价比基本大于花菜类，除去某一天的较高成本加成定价比，花叶类的成本加成定价比基本稳定为 0.86，花菜类为 0.61，水生根

茎类为 0.55，茄类为 0.66，辣椒类为 0.82，食用菌类为 0.71。

最后，根据此最优补货和定价策略，计算出该周的总订货量为 2444.19 千克，最大收益约为 10021 元。

八、 问题三的模型建立和求解

8.1 销售品种的筛选

8.1.1 蔬菜单品优先度

依据题意，要求从 2023 年 6 月 24 日至 30 日的可销售单品品种筛选出 27 到 33 种来进行销售，从附件 2 中可以提取出这段时间内共有 49 种蔬菜单品。为考虑收益的最大化，同时为在补货时考虑历史的销售数据，商超应优先选择这段时间内**平均销量大**，且**销量波动小**的蔬菜单品。定义蔬菜单品的优先度ϕ_i：

$$\phi_i = \alpha\mu_i - (1-\alpha)\sigma_i^2 \tag{26}$$

其中，μ_i和σ_i^2表示第i 种蔬菜单品在这段时间内的平均销量以及方差。选取合适的α作为加权系数以综合评估蔬菜单品的优先度。

8.1.2 基于层次聚类的优先度修正模型

基于问题一，为了综合考虑蔬菜单品间互补替代关系的影响以确定最佳的销售组合，我们依据蔬菜单品间的相关性强弱对蔬菜单品进行层次聚类，具体的步骤如下：

1. 相关系数

在对变量进行聚类分析之前，应首先确定变量的相似度量，记蔬菜单品 j 的时间序列Y_j的取值$\left(y_{1j}, y_{2j}, \ldots, y_{nj}\right)^T \in R^n$，用$Y_j$和$Y_k$两条序列的样本相关系数作为它们的相似度量，具体计算公式如下。

$$r_{jk} = \frac{\sum_{i=1}^{n}\left(y_{ij}-\bar{y}_j\right)\left(y_{ik}-\bar{y}_k\right)}{\left[\sum_{i=1}^{n}\left(y_{ij}-\bar{y}_j\right)^2\sum_{i=1}^{n}\left(y_{ik}-\bar{y}_k\right)^2\right]^{\frac{1}{2}}} \tag{27}$$

2. 最短距离

定义两种变量之间距离如下式所示：

$$d_{jk} = 1 - \left|r_{jk}\right| \tag{28}$$

$$R\left(G_1, G_2\right) = \min\left\{d_{jk}\right\}, x_j \in G_1, x_k \in G_2 \tag{29}$$

其中，$R\left(G_1, G_2\right)$与两类单品中的相似度最大的两条序列的相似度量值有关。

3. 最佳销售组合分析

通过手肘法确定最优聚类簇数$K=4$，得到聚类结果如下图所示：

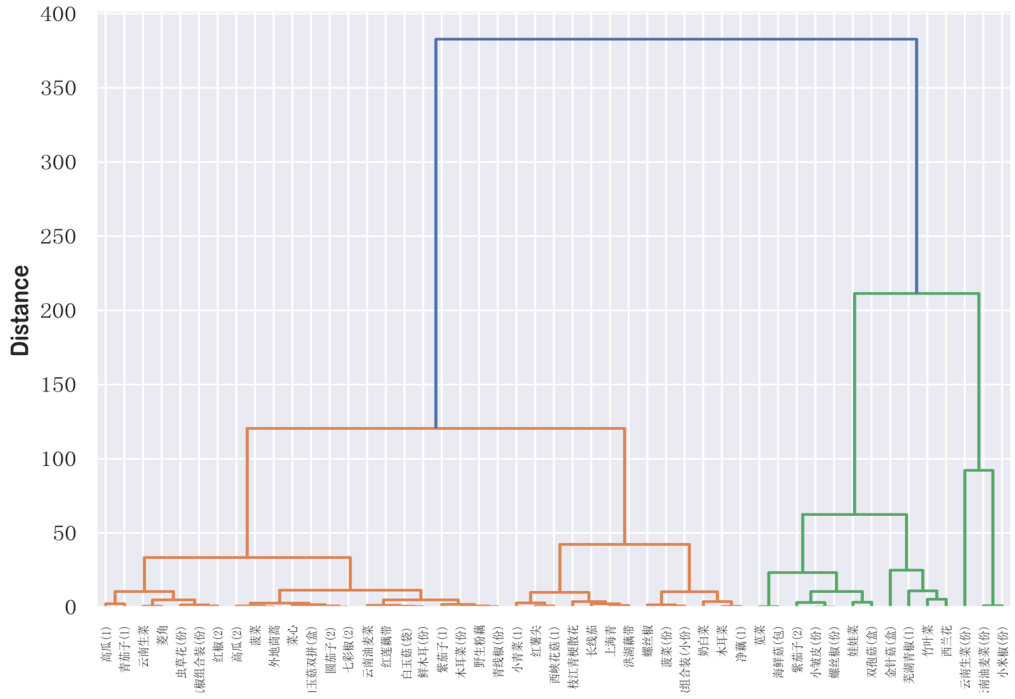


图 8 聚类树状图

由图中可以看出，聚类结果为 4 类，对同一类中的各蔬菜单品，其相关性较强，体现为互补的关系；对于不同类之间的蔬菜单品，其相关性相对较弱，体现为相互替代关系。与问题二类似，通过以上相互关系将蔬菜单品优先度 ϕ_i 修正为：

$$\phi_i^* = (1+\rho_i')\phi_i \tag{30}$$

根据修正后的蔬菜单品优先度，初步筛选出前 27 至 33 种蔬菜单品如下表所示。

表 6 初步筛选结果

排名	单品名称	排名	单品名称	排名	单品名称	排名	单品名称	排名	单品名称	排名	单品名称
1	云南生菜	**7**	螺丝椒	**13**	紫茄子(2)	**19**	姜蒜小米椒	**25**	莴菜	**31**	螺丝椒
2	小米椒	**8**	双孢菇	**14**	菜心	**20**	海鲜菇	**26**	青茄子(1)	**32**	木耳菜
3	云南油麦菜	**9**	云南油麦菜	**15**	云南生菜	**21**	外地茼蒿	**27**	红薯尖	**33**	长线茄
4	金针菇	**10**	菠菜	**16**	竹叶菜	**22**	上海青	**28**	净藕(1)		
5	小皱皮	**11**	西兰花	**17**	圆茄子(2)	**23**	菠菜	**29**	奶白菜		
6	芜湖青椒(1)	**12**	野生粉藕	**18**	娃娃菜	**24**	白玉菇	**30**	西峡花菇(1)		

在表 6 中，所筛选出的蔬菜单品覆盖了 6 种蔬菜品类，再对比这些蔬菜单品销售量的均值和方差，可以发现，排名越靠前的蔬菜单品销量越高、方差越小，依然符合前文提出的优先关系。

8.2 基于遗传算法的补货与定价决策模型

8.2.1 约束条件

首先，决策变量包括第 i 种蔬菜单品在第 j 天的补货量 q_{ij} 以及第 i 种蔬菜单品在第 j 天的成本加成定价比 w_{ij}。由附件 2 中数据，可以统计出初步筛选出的蔬菜单品

在这段时间内的最大日销售量$q_{i\max}$以及最小日销售量$q_{i\min}$。考虑到题目要求最小陈列量不小于2.5千克，则补货量q_{ij}满足约束：

$$\begin{cases}\max\{2.5,q_{i\min}\}\le q_{ij}\le q_{i\max},q_{i\max}\ge 2.5\\ q_{ij}=0,q_{i\max}<2.5\end{cases} \tag{31}$$

另外，基于问题二求解的每种蔬菜品类最大销售量Q_j，第i种蔬菜单品的补货量q_i满足：

$$\sum_{i\in S_j} q_i \le Q_j \tag{32}$$

即第j类蔬菜品类中蔬菜单品的销售量之和不会超过该蔬菜品类的最大销售量。

8.2.2 目标函数

结合问题二中建立的非线性规划模型，为求得商超的最大收益，对于未来一天的收益Z，其可以写为：

$$Z=\sum_{i}^{n} Z_i=\sum_{i}^{n}\left(Z_{1i}+Z_{2i}\right) \tag{33}$$

其中，Z_{1i}表示第i类蔬菜的无损耗部分的收益，具体计算式如下：

$$Z_{1i}=q_i\left(1-L_i\right)A_i\left\{w_i\left[\left(1-p_2\right)+p_2\left(1-p_1\right)\right]+\left[\left(1+w_i\right)\lambda-1\right]p_2p_1\right\} \tag{34}$$

在式23中，L_i表示在第i类蔬菜运输中损耗的比例。p_2表示销售量分布于晚间的概率，此时有些蔬菜的品相可能有所下降，故确定打折的概率p_1以及打折率λ来进行打折促销，p_1和p_2可以从附件2中统计出来。

在式22中，Z_{2i}表示第i类蔬菜的损耗部分的收益减去无法隔日再售的库存的亏损后的实际收益，具体计算式如下：

$$Z_{2i}=q_i\cdot L_i\cdot\left[\left(1+w_i\right)\lambda-1\right]-I_i\cdot A_i \tag{35}$$

总的最大收益目标函数可以写为：

$$\max Z=\max\sum_{i}^{n}\left(Z_{1i}+Z_{2i}\right) \tag{36}$$

综上所述，结合问题二，得到基于非线性规划的补货和定价模型如下：

$$\begin{aligned}&\max Z=\max\sum_{i}^{n}\left(Z_{1i}+Z_{2i}\right)\\ &s.t.\begin{cases}\sum_{i\in S_j} q_i\le Q_j\\ \begin{cases}\max\{2.5,q_{i\min}\}\le q_i\le q_{i\max},q_{i\max}\ge 2.5\\ q_i=0,q_{i\max}<2.5\end{cases}\\ w_i\in\left[\left(1-\theta\right)\overline{w}_i,\left(1+\theta\right)\overline{w}_i\right]\end{cases}\end{aligned} \tag{37}$$

8.3 模型求解及结果分析

对上述建立的非线性规划模型进行结合前面筛选出的 33 种蔬菜单品通过遗传算法进行逐步求解，即逐步计算当筛选 27、28、……、33 种的蔬菜单品，得到每一步的最大利润如下表所示：

表 7 逐步求解结果

选取种数	27	28	29	30	31	32	33
单日最大利润/元	675.075	682.173	673.064	747.482	730.403	717.409	710.854

由表 7 中，可以看出，当选择 30 种蔬菜单品时，获得的单日最大利润取到最大值 747.482 元。由此确定出这 30 种蔬菜单品的补货和定价策略如下表所示。

表 8 补货与定价策略

单品名称	补货量/kg	成本加成定价比	单品名称	补货量/kg	成本加成定价比
云南生菜	10.64	0.27	娃娃菜	17.76	0.43
小米椒	32.92	2.30	姜蒜小米椒	8.66	1.14
云南油麦菜	21.04	0.59	海鲜菇	12.51	0.49
金针菇	15.46	0.29	外地茼蒿	3.44	0.65
小皱皮	13.30	0.91	上海青	4.99	1.11
芜湖青椒(1)	6.08	0.67	菠菜	13.84	0.30
螺丝椒	16.29	0.51	苋菜	8.22	0.78
双孢菇	12.10	0.67	青茄子(1)	3.04	0.36
云南油麦菜	4.03	0.79	红薯尖	6.95	0.74
菠菜	2.52	-0.36	净藕(1)	6.76	0.40
西兰花	14.85	0.76	奶白菜	8.23	1.04
紫茄子(2)	12.89	0.73	西峡花菇(1)	5.65	0.75
菜心	4.10	0.34	螺丝椒	7.40	0.72
云南生菜	6.95	0.32	木耳菜	6.66	0.96
竹叶菜	14.34	0.86	长线茄	6.40	0.88

通过对表 8 进行分析，所列出蔬菜单品覆盖了六种蔬菜品类，并根据统计，其中花叶类的补货量最多，为 133.72 千克，水生根茎的补货量最少，为 6.76 千克。

九、 问题四的求解

9.1 相关数据来源

本问要求找出商超还可以收集的数据以制定更好的补货和定价策略，影响补货和定价策略的因素有很多，可以从外部因素和内部因素[7]进行分析，如下图所示。

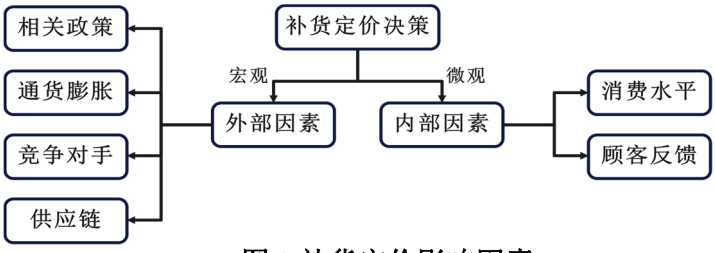


图 9 补货定价影响因素

外部因素从宏观角度出发，包括了行业的相关政策数据、通货膨胀数据、竞争对手数据以及供应链的数据。内部因素从微观角度出发，包含了商超所在地区消费者的总体消费水平以及消费者对于商超的反馈调查数据。

9.2 数据作用分析

9.2.1 外部因素

1) 相关政策

相关的政策数据中包含了商超在进行补货和定价过程中所必须遵守的有关蔬菜商品质量、卫生、安全等合规性要求和法规，以及行业标准等信息。对于补货策略，若涉及跨境贸易，行业相关政策数据可以帮助商超了解蔬菜进口出口的法规、税收以及相关许可证需求。

对于定价策略，部分地区可能存在蔬菜价格控制政策，商超需要根据这些政策确定合理的定价范围，以确保销售的合规性。

2) 通货膨胀

对于补货策略，通货膨胀可能导致某些蔬菜品类的成本上升较大，商超需要考虑这些受通货膨胀影响较大的单品来适度地调整补货量。

对于定价策略，通货膨胀会导致蔬菜的采购和运输成本以及商超的运营成本上升，商超需要综合考虑这些成本重新确定定价策略。

3) 竞争对手

竞争对手的数据包括竞争对手的定价策略、促销策略、蔬菜品类的市场份额、蔬菜的品种以及质量等数据。对于补货策略，了解竞争对手的蔬菜品种数量可以帮助商超决定是否要拓展或是减少自己的蔬菜品种组合，优化自己单品补货策略。

对于定价策略，了解竞争对手的实时定价策略，当对手进行降价时，商超可以适当调整价格来保持竞争力。而分析竞争对手的促销策略，例如折扣、捆绑销售或是赠品等可以帮助商超确定是否要进行类似的促销活动来吸引顾客。

4) 供应链

供应链数据包含了供应商的稳定性、供应链的成本结构、供应商的绩效等数据。可以根据供应链数据提供不同从供应商采购的成本，不同供应商的运输成本以及不同供应商的绩效(包括交货准时性和质量)来综合选择较好的供应商来进行补货。

9.2.2 内部因素

1) 消费水平

对于补货策略，高消费水平地区更加容易促销高端蔬菜品种，商超可以引入更多高端蔬菜品种，增多高端品种的占比，将其陈列在更显眼的位置。

对于定价策略，高消费水平的地区可以接受较高的成本加成定价比，而在低消费水平地区则要更加谨慎地确定成本加成定价比大小以及上升幅度，并可以多进行打折促销活动来刺激消费。

2) 顾客反馈

顾客反馈数据可以提供有关顾客需求、满意度和偏好等关键信息，有助于商超更好地制定补货和定价策略。对于补货策略，顾客反馈可以告诉商超哪些蔬菜品类或是单品最受欢迎，商超可以以此增多相关品类或者是单品的补货量或是根据顾客喜好来调整蔬菜菜品的陈列布局，确保畅销的蔬菜可以有更多的陈列空间。另外、根据顾客对蔬菜质量的满意度可以帮助商超选择合适的供应商。对于定价策略，根据顾客对蔬菜价格的感知和接受程度，可以帮助商超制定合理的定价以及适当的打折力度。同时，可以调查顾客对促销活动的偏好来制定更好的促销策略。

十、 灵敏度分析

本文对问题二的补货与定价决策模型进行敏感度分析，综合考虑最大销售量的因素，通过调节成本加成定价比的波动比例θ，分析其在 0.05~0.1 范围内调节对未来一周总收益的影响。针对问题二设定的非线性规划模型，分别代入θ 进行求解，得到结果如图 10 所示。

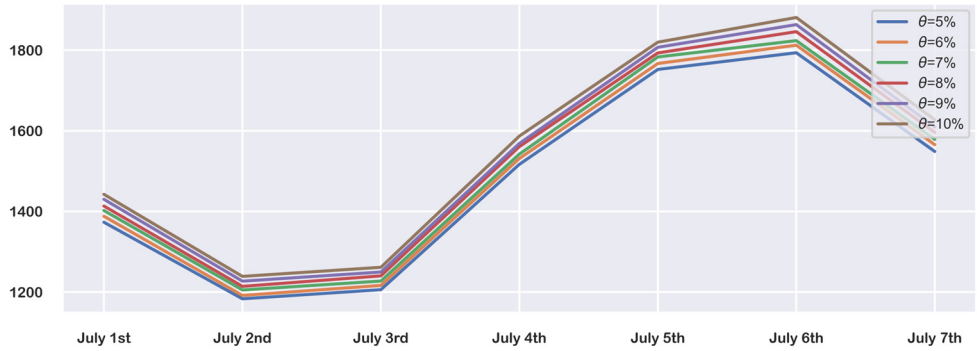


图 10 总收益灵敏度分析

从图 10 可以发现，未来一周的收益随着成本加成定价比的波动比例θ的增加而上浮。再计算每种情况的收益关于平均收益的均方误差百分比，得到结果如图 11 所示。

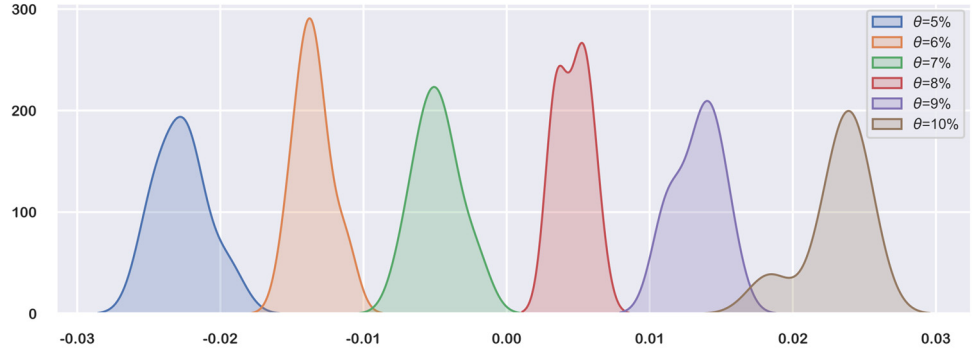


图 11 均方误差百分比核密度曲线

观察图 11 中曲线，可以发现均方误差百分比都在-3%到+3%范围内，由此可以认为成本加成定价比的波动比例θ对模型的结果灵敏度较低，模型具有一定的抗干扰性。

十一、 模型的评价、改进与推广

11.1 模型的优点

1. 将一种蔬菜单品的销售量看作时间序列，用 STL 时间序列分解模型将其分解，体现销售量在不同时间维度下的变化趋势与分布规律，同时可以促进其它模型的准确性和合理性；

2. 采用联动分析即计算序列间的偏相关系数，能够反映不同时间维度上两者相互关系的紧密程度，并将其划分为多个类别，层层递进，模型的泛化能力得到显著提高；

3. 建立多重模型，在初步模型结果的基础之上进行规划，并采用遗传算法替代大规模非线性规划求解，结果具有全局收敛性，更符合实际情况。

11.2 模型的缺点

1. 在计算偏自相关系数时，采用补零的方法使两条序列长度相等，这可能会弱化两条序列之间原有的相关关系；

2. 在对单品补货定价进行决策时，优先决定单品的补货顺序，这样得到的结果可能不是全局最优解，仅是当前先验条件下的一个最优解。

11.3 模型的改进

1. 对销量和成本加成定价关系的研究中，本文选择的自变量仅是部分因素，而剩余的因素影响使用机器学习的方法进行预测，这样可能不能很好地确定两者的关系。因此，可以考虑从数据中提取更多特征，进行非线性函数形式的拟合，以此得到更为准确的销售量预测。

2. 对单品补货定价进行决策时，可以将单品是否补货与补货量的多少同时当作决策变量，通过建立标准的非线性约束规划模型求解更为准确的补货计划和定价策略。

11.4 模型的推广

本文提出的基于时间序列分解的分布规律与相互关系研究模型，可以用于其它长序列的因素分析中，如土壤中成分含量变化、区域生物多样性变化等时序研究中；另外，本文通过相关性将研究对象进行分类，并基于相互关系对模型结果进行调整，可以用于表现未知关系，可以用于研究生态学、气象学或经济学方面的灰箱问题。

参考文献

[1]赵建东,朱丹,刘佳欣.基于时间序列分解与门控循环单元的地铁换乘客流预测[J].华南理工大学学报(自然科学版),2022,50(05):22-31.

[2]曹永福.格兰杰因果性检验评述[J]世界经济统计研究[J],2005.52(2):16-21.

[3]张伟琦.实体经济部门杠杆率与房地产价格的关系研究[D].吉林大学,2023.

[4] Su Y, Zhao Y, Xia W, et al. CoFlux: robustly correlating KPIs by fluctuations for service troubleshooting[C]//Proceedings of the International Symposium on Quality of Service. 2019: 1-10.

[5]郎庆凯,王兴勋,王月香等.基于 GBDT 和 SVM 的光伏发电出力预测研究[J].上海电力大学学报,2023,39(03):275-280.

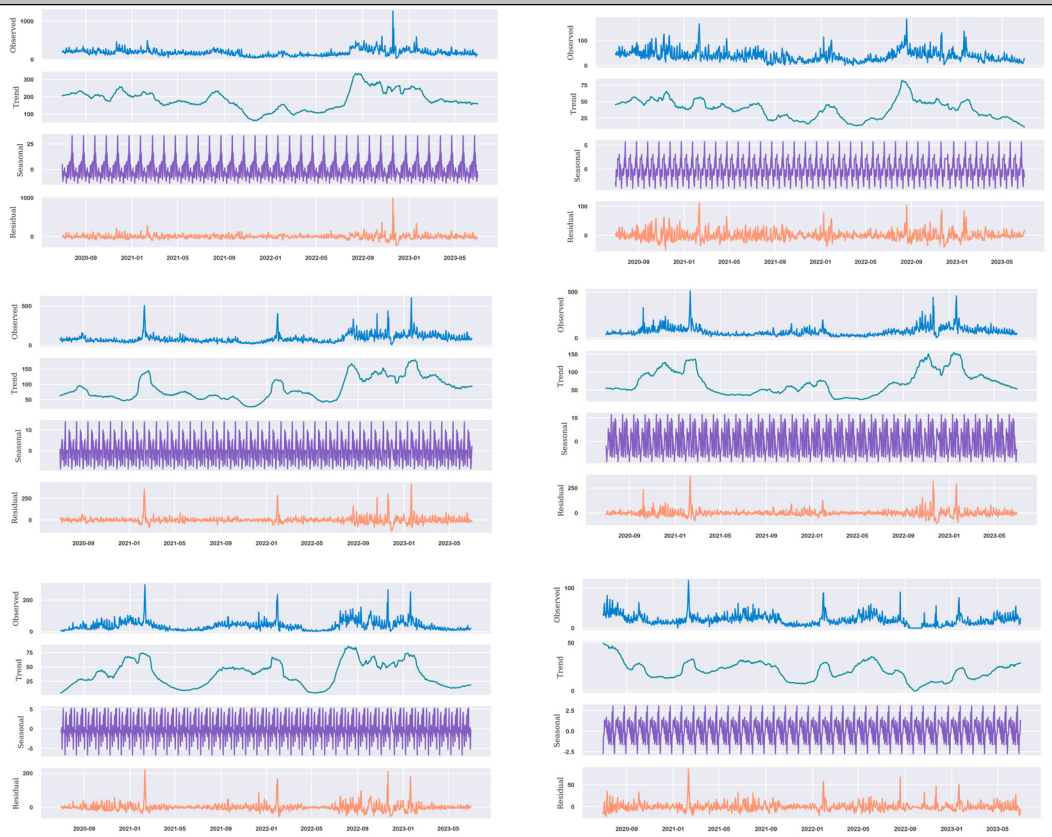
[6]金玲,刘晓丽,李鹏飞等.遗传算法综述[J].科学中国人,2015(27):230.

[7]王艳.中小型连锁超市定价的影响因素及其定价策略探析[J].兰州工业学院学报,2014,21(03):99-102.

附录

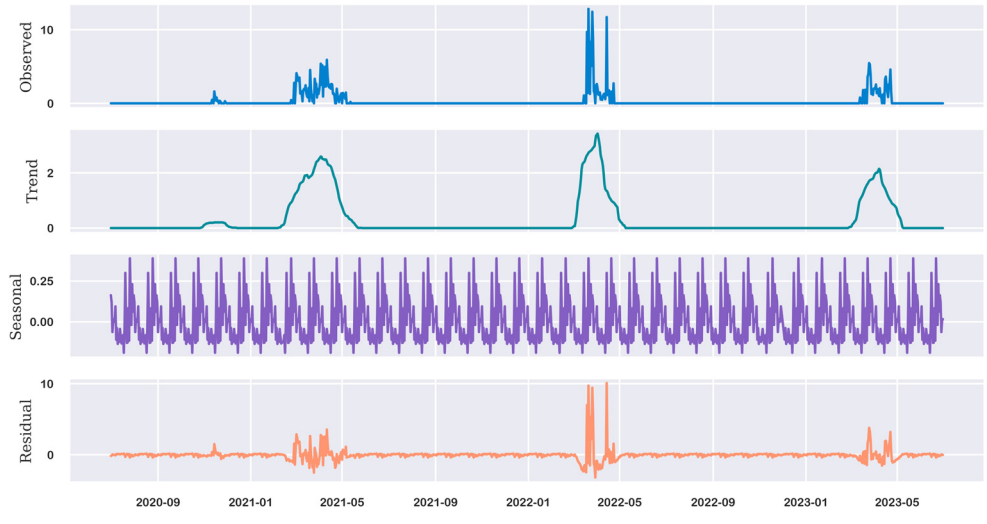
附录 1

介绍：时间序列分解—品类 1~6



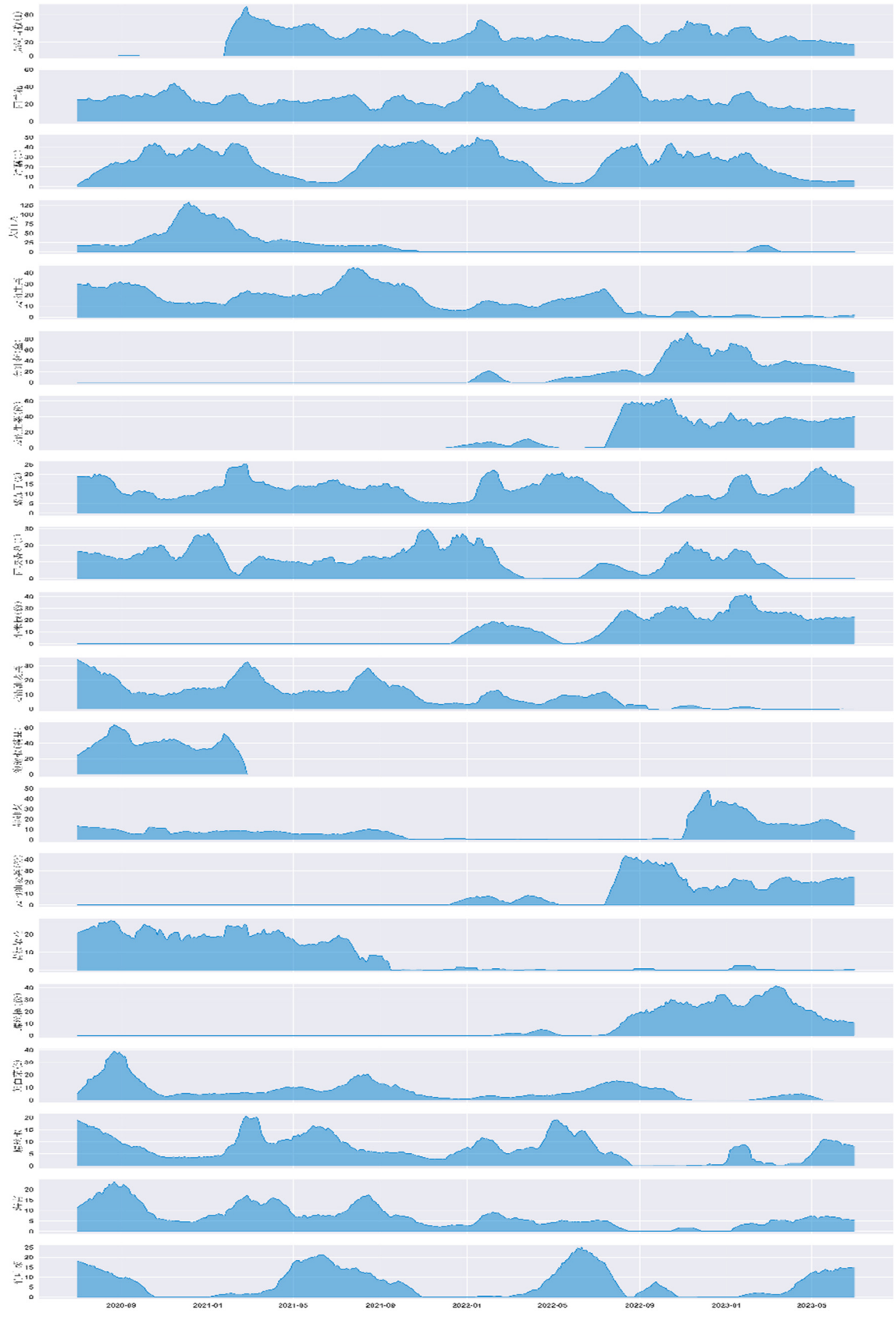
附录 2

介绍：时间序列分解—单品（其中一个）



附录 3

介绍：时间序列分解—单品（趋势）



附录 4

介绍： 单品-统计分布数据

单品	月季分布	总销量
0 芜湖青椒(1)	1-12 月	0 253478.979
1 西兰花	1-12 月	1 247835.052
2 净藕(1)	6 月-12 月，1-4 月	2 244344.960
3 大白菜	1-5 月	3 172684.962
4 云南生菜	1-12 月	4 143194.149
5 金针菇(盒)	9-12 月， 1-5 月	5 140364.000
6 云南生菜(份)	8-12 月， 1-5 月	6 128925.000
7 紫茄子(2)	1-12 月	7 122418.009
8 西峡香菇(1)	8-12 月， 1 月	8 107282.043
9 小米椒(份)	8 月-12 月，1-5 月	9 97497.000
10 云南油麦菜	1-12 月	10 92748.276
11 泡泡椒(精品)	8-12 月，1 月	11 87328.125
12 娃娃菜	10-12 月，1-5 月	12 80838.000
13 云南油麦菜(份)	9-12 月，1-5 月	13 79632.000
14 青梗散花	1-12 月	14 75544.074
15 螺丝椒(份)	9-12 月，1-5 月	15 74115.000
16 黄白菜(2)	5-9 月	16 71891.910
17 螺丝椒	1-9 月	17 70129.629
18 上海青	5-9 月	18 68460.804
19 竹叶菜	5-9 月	19 65166.876
20 奶白菜(份)	9-12 月	20 97034.000
21 保康高山大白菜	10 月-12 月，1 月	21 90786.304
22 菠菜(份)	9-12 月	22 88788.000
23 洪湖莲藕(粉藕)	9-12 月， 1-5 月	23 84728.000
24 枝江青梗散花	8-12 月， 1-2 月	24 81501.994
25 奶白菜	8-10 月	25 81428.312
26 菠菜	9-12 月，1-2 月	26 73030.454
27 小皱皮(份)	9-12 月，1-5 月	27 72450.000
28 红薯尖	4-9 月	28 72232.062
29 苋菜	3-8 月	29 71400.854
30 枝江红菜苔	1-3 月	30 69474.356
31 金针菇(1)	8-12 月，1-5 月	31 65764.580
32 甜白菜	8-10 月	32 65593.654
33 菜心	8-9 月	33 62954.038
34 双孢菇(盒)	1-7 月	34 59206.000
35 茼蒿	9-12 月，1 月	35 57542.268
36 小青菜(份)	8-12 月，1 月	36 56798.000
37 牛首油菜	9-12 月，1 月	37 53711.924
38 青茄子(1)	6-8 月	38 49234.682
39 红椒(1)	2-3 月	39 48410.250

附录 5

介绍：P1.ipynb | 用于数据处理分析与问题一的求解

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.font_manager import FontProperties
import warnings
warnings.filterwarnings('ignore')

# Set up Seaborn style
sns.set(style="darkgrid")
# Set font style
font_prop1 = FontProperties(fname="C:\Windows\Fonts\simsun.ttc")   # 宋体
font_prop2 = FontProperties(fname="C:\Windows\Fonts\simkai.ttf")   # 楷体
Efont_prop = FontProperties(fname="C:\Windows\Fonts\ARLRDBD.TTF")
label_prop = FontProperties(family='serif', size=9, weight='normal')
legend_font = FontProperties(family='serif', size=7, weight='normal')
plt.rcParams['axes.unicode_minus'] = False
# import dataset
names = pd.read_excel("data\附件 1.xlsx")
print(names.head())

data = pd.read_excel("data\附件 2.xlsx")
print(data.head())
def WeekSplit(starttime, endtime):
    starttime = datetime.strptime(starttime, "%Y-%m-%d")
    endtime = datetime.strptime(endtime, "%Y-%m-%d")
    interval = timedelta(days=7)
    current_time = starttime
    time_periods = []

    while current_time <= endtime:
        end_of_period = current_time + interval - timedelta(days=1)
        time_periods.append((current_time.strftime("%Y-%m-%d"), end_of_period.strftime("%Y-%m-%d")))
        current_time += interval

    return time_periods
# weekdata
data["销售日期"] = pd.to_datetime(data["销售日期"])
months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
Results = np.zeros((3, 12, 250, 2))  # year, month, kind, salesdata
Periods = WeekSplit("2020-07-01", "2023-06-30")
A, B, C, D, E, F = [], [], [], [], [], []
for i in range(len(Periods)):
    starttime = Periods[i][0]
    endtime = Periods[i][1]
    index = np.array((data["销售日期"] <= endtime) & (data["销售日期"] >= starttime))
    TD = data.iloc[index, :]
    a, b, c, d, e, f = 0, 0, 0, 0, 0, 0
    for v in range(len(TD["单品编码"])):
        index2 = np.array(names["单品编码"] == TD.iloc[v, 2])
        if names.iloc[index2, 2].values == 1011010101:
            a += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010201:
            b += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010402:
            c += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010501:
            d += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010504:
            e += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010801:
            f += TD.iloc[v, 3]
    A.append(a)
    B.append(b)
    C.append(c)
    D.append(d)
    E.append(e)
    F.append(f)


Time = pd.to_datetime([u[0] for u in Periods])
```

```
A, B, C, D, E, F = np.array(A), np.array(B), np.array(C), np.array(D), np.array(E), np.array(F)

WDF = pd.DataFrame({"时间":Time.values,"花叶类":A,"花菜类":B,
                    "水生根茎类":C,"茄类":D,
                    "辣椒类":E,"食用菌":F})
WDF.to_excel("data/全年周销量数据.xlsx", index=False)
WDF = WDF.set_index("时间")
print(WDF.head())

# day data
data["销售日期"] = pd.to_datetime(data["销售日期"])
months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
Results = np.zeros((3, 12, 250, 2))  # year, month, kind, salesdata
uv = data["销售日期"].unique()
A, B, C, D, E, F = [], [], [], [], [], []
for i in range(len(uv)):
    TD = data.iloc[np.array(data["销售日期"] == uv[i]), :]
    a, b, c, d, e, f = 0, 0, 0, 0, 0, 0
    for v in range(len(TD["单品编码"])):
        index2 = np.array(names["单品编码"] == TD.iloc[v, 2])
        if names.iloc[index2, 2].values == 1011010101:
            a += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010201:
            b += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010402:
            c += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010501:
            d += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010504:
            e += TD.iloc[v, 3]
        elif names.iloc[index2, 2].values == 1011010801:
            f += TD.iloc[v, 3]
    A.append(a)
    B.append(b)
    C.append(c)
    D.append(d)
    E.append(e)
    F.append(f)

print(np.array(A))
Time = pd.to_datetime(uv)
A, B, C, D, E, F = np.array(A), np.array(B), np.array(C), np.array(D), np.array(E), np.array(F)
D20 = (Time <= "2020-12-31") | (Time >= "2023-01-01")
D21 = (Time <= "2021-12-31") & (Time >= "2021-01-01")
D22 = (Time <= "2022-12-31") & (Time >= "2022-01-01")
DF = pd.DataFrame({"时间":Time.values,"花叶类":A,"花菜类":B,
                    "水生根茎类":C,"茄类":D,
                    "辣椒类":E,"食用菌":F})
DF.to_excel("data/全年日销量数据.xlsx", index=False)
DF = DF.set_index("时间")
print(DF)
kind = "花叶类"
DDF = DF.copy()
Season = np.zeros((7, 6))
for i in range(6):
    result_mul = seasonal_decompose(DDF.iloc[:, i],
                                    model='addictive',
                                    period=7,
                                    extrapolate_trend='freq')
    Season[:, i] = result_mul.seasonal[:7]
DF = pd.DataFrame(Season)
DF.to_excel("data/季节性数据.xlsx", index=False)

plt.rcParams.update({'figure.figsize': (10, 10), 'figure.dpi': 600})

plt.figure(figsize=(10, 8), dpi=600)
plt.subplot(611)
plt.plot(result_mul.observed, color='#0081CF')
plt.ylabel('Observed', fontproperties=label_prop, fontsize=9)
plt.yticks(fontproperties=Efont_prop, fontsize=7)
plt.xticks(fontproperties=Efont_prop, fontsize=7)
plt.gca().set_xticklabels([])

plt.subplot(612)
plt.plot(result_mul.trend, color='#008E9B')
plt.yticks(fontproperties=Efont_prop, fontsize=7)
plt.xticks(fontproperties=Efont_prop, fontsize=7)
```

```
plt.ylabel('Trend', fontproperties=label_prop, fontsize=9)
plt.gca().set_xticklabels([])

plt.subplot(613)
plt.plot(result_mul.seasonal, color='#845EC2')
plt.yticks(fontproperties=Efont_prop, fontsize=7)
plt.xticks(fontproperties=Efont_prop, fontsize=7)
plt.ylabel('Seasonal', fontproperties=label_prop, fontsize=9)
plt.gca().set_xticklabels([])

plt.subplot(614)
plt.plot(result_mul.resid, color='#FF9671')
plt.yticks(fontproperties=Efont_prop, fontsize=7)
plt.xticks(fontproperties=Efont_prop, fontsize=7)
plt.ylabel('Residual', fontproperties=label_prop, fontsize=9)

plt.subplot(616)
plt.plot(result_mul.seasonal[:7], color='#FF9671')
plt.yticks(fontproperties=Efont_prop, fontsize=7)
plt.xticks(fontproperties=Efont_prop, fontsize=7)
plt.ylabel('Single Trend', fontproperties=label_prop, fontsize=9)
import pandas as pd
import math

class CorrelationMeasurement():
    coTHR = 0.8

    # def __init__(self):
    #     self.coTHR = 0.8

    def __init__(self, *args):
        if len(args) == 1:
            self.coTHR = args
        else:
            self.coTHR = 0.8
    def normalize(self, x):
        x = (x - np.mean(x))/np.std(x)
        return x
    def amplification(self, a, b, x):
        temp_result = []
        for v in x:
            if v >= 0:
                temp = math.pow(math.exp(1), a * min(v, b)) - 1
                temp_result.append(temp)
            else:
                temp = -math.pow(math.exp(1), a * min(-v, b)) + 1
                temp_result.append(temp)
        result = pd.Series(temp_result)
        return result

    def R(self, Gs, H):
        summ = 0
        for index in range(len(Gs)):
            summ += Gs[index] * H[index]
        return summ

    def CC(self, G, H, Gs):
        T =self.R(Gs, H) / math.sqrt(self.R(G, G) * self.R(H, H))
        return T

    def FCC(self, afx, afy):
        maxCC = 0
        minCC = 0
        s1 = 0
        s2 = 0
        l = len(afx)
        s = 0
        for i in range(-s, s + 1):
            Gs = [0] * l
            if i < 0:
                for j in range(l - abs(i)):
                    Gs[j] = afx[j - i]
            else:
                for j in range(l - abs(i)):
                    Gs[j + i] = afx[j]

            tempcc = self.CC(afx, afy, Gs)
            temparg = i
            if tempcc > maxCC:
```

```
                maxCC = tempcc
                s2 = temparg
            if tempcc < minCC:
                minCC = tempcc
                s1 = temparg

        if abs(maxCC) < abs(minCC):
            return minCC, s1
        else:
            return maxCC, s2

    def correlation_measurement(self, afx_set, afy_set):
        afx_set, afy_set = self.normalize(afx_set), self.normalize(afy_set)
        result_set = []
        for afx in afx_set:
            for afy in afy_set:
                result_set.append(self.FCC(self.amplification(1, 20, afx), self.amplification(1, 20, afy)))

        maxv = 0
        minv = 0
        for result in result_set:
            maxv = max(maxv, result[0])
            minv = min(minv, result[0])

        if abs(maxv) > abs(minv):
            ccV, shiftV = max(result_set)
        else:
            ccV, shiftV = min(result_set)
        print(ccV)
        print(shiftV)
        if abs(ccV) >= self.coTHR:

            if shiftV == 0:
                if ccV >= 0:
                    print('x<-+->y')
                else:
                    print('x<--->y')
            elif shiftV < 0:
                if ccV >= 0:
                    print('x-+->y')
                else:
                    print('x--->y')
            else:
                if ccV >= 0:
                    print('x<-+-y')
                else:
                    print('x<---y')
        else:
            print('x=||=y')
        return 0


afx_set = DF.iloc[:, 3]
afx_set = [afx_set.values.tolist()]
for item in [1, 2, 3, 4, 5]:
    cm = CorrelationMeasurement()
    afy_set = [DF.iloc[:, item].values.tolist()]
    cm.correlation_measurement(afx_set=afx_set, afy_set=afy_set)
def RValue(x, y):
    def amplification(a, b, x):
        temp_result = []
        for v in x:
            if v >= 0:
                temp = math.pow(math.exp(1), a * min(v, b)) - 1
                temp_result.append(temp)
            else:
                temp = -math.pow(math.exp(1), a * min(-v, b)) + 1
                temp_result.append(temp)
        result = pd.Series(temp_result)
        return result
    def normalize(x):
        x = (x - np.mean(x))/np.std(x)
        return x
    def R(G, H):
        summ = 0
        for index in range(len(G)):
            summ += G[index] * H[index]
```

```
        return summ

    def CC(G, H):
        T = R(G, H) / math.sqrt(R(G, G) * R(H, H))
        return T
    return CC(amplification(1, 20, normalize(x)), amplification(1, 20, normalize(y)))
R = np.zeros((5, 5))
for i in range(5):
    for j in range(5):
        R[i, j] = RValue(DF.iloc[:, i], DF.iloc[:, j])
print(R)
Ndata = pd.read_excel("data/单品日-周-月数据.xlsx", sheet_name="Sheet1")
print(Ndata.shape)
print(Ndata.head())
Tsum = Ndata.iloc[:, 1:].sum(1)
Ndata["总和"] = Tsum
sort_Ndata = Ndata.sort_values(by="总和", ascending=False).reset_index(drop=True)

plt.figure(figsize=(30, 70), dpi=600)
colors = ["#0081CF", "#2C73D2","#B39CD0", "#FF8066", "#4E8397", "#00C2A8"]
Labels = ['大白菜','西兰花', '净藕(1)','紫茄子(2)','芜湖青椒(1)','金针菇(盒)']
Num = 40
for i in range(20, Num):
    DDF = pd.DataFrame({"时间": Time.values, str(i): sort_Ndata.iloc[i, 1:-1]})
    DDF = DDF.set_index("时间")
    result_mul = seasonal_decompose(DDF[str(i)],
                                    model='addictive',
                                    period=30,
                                    extrapolate_trend='freq')

    plt.subplot(Num,1, i+1)
    #sns.kdeplot(result_mul.trend.values, shade=True, color=colors[i])
    plt.plot(result_mul.trend, color="#0081CF")
    plt.fill_between(result_mul.trend.index,result_mul.trend, color="#0081CF", alpha=0.5)
    plt.ylabel(sort_Ndata.iloc[i, 0], fontproperties=font_prop1, fontsize=20)
    plt.yticks(fontproperties=Efont_prop, fontsize=15)
    plt.xticks(fontproperties=Efont_prop, fontsize=15)
    if i < Num-1:
        plt.gca().set_xticklabels([])
plt.savefig('1.4.png', dpi=600)
```

附录 6

介绍：P2-P3.ipynb | 用于问题二和问题三的求解（核心代码）

```
#!/usr/bin/env python
# coding: utf-8

# In[977]:


# import dataset
names = pd.read_excel("data\附件 1.xlsx")
print(names.head())

data = pd.read_excel("data\附件 2.xlsx")
print(data.head())

prices = pd.read_excel("data\附件 3.xlsx")
print(prices.head())


# In[1046]:


import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.font_manager import FontProperties
import warnings
warnings.filterwarnings('ignore')

# Set up Seaborn style
sns.set(style="darkgrid")
# Set font style
font_prop1 = FontProperties(fname="C:\Windows\Fonts\simsun.ttc", weight='bold')
font_prop2 = FontProperties(fname="C:\Windows\Fonts\simkai.ttf", weight='bold')
Efont_prop = FontProperties(fname="C:\Windows\Fonts\ARLRDBD.TTF", weight='bold')
label_prop = FontProperties(family='serif', size=9, weight='normal')
legend_font = FontProperties(family='serif', size=6, weight='normal')
plt.rcParams['axes.unicode_minus'] = False


# In[978]:


# 获取每日各类蔬菜的批发价格
prices["日期"] = pd.to_datetime(prices["日期"])
uv = prices["日期"].unique()
uv = uv.astype('datetime64[D]')
DPrices = {}
for i in range(len(uv)):
    TD = prices.iloc[np.array(prices["日期"] == uv[i]), :]
    DPrices[str(uv[i])] = {}
    cards = TD["单品编码"].unique()
    for j in range(len(cards)):
        DPrices[str(uv[i])][cards[j]] = TD.iloc[np.array(TD["单品编码"] == cards[j]), 2].mean()
print(DPrices["2020-07-01"])

data["销售日期"] = pd.to_datetime(data["销售日期"])
uv = data["销售日期"].unique()
uv = uv.astype('datetime64[D]')
DSales = {}
for i in range(len(uv)):
    TD = data.iloc[np.array(data["销售日期"] == uv[i]), :]
    DSales[str(uv[i])] = {}
    cards = TD["单品编码"].unique()
    for j in range(len(cards)):
        DSales[str(uv[i])][cards[j]] = TD.iloc[np.array(TD["单品编码"] == cards[j]), 4].mean()
print(DSales["2020-07-01"])


# In[979]:


```

```
73.
74. # 获取每日各类蔬菜的加价比例
75. uv = data["销售日期"].unique()
76. uv = uv.astype('datetime64[D]')
77. A, B, C, D, E, F = [], [], [], [], [], []
78. for i in range(len(uv)):
79.     TD = data.iloc[np.array(data["销售日期"] == uv[i]), :]
80.     a, b, c, d, e, f = 0, 0, 0, 0, 0, 0
81.     sa, sb, sc, sd, se, sf = 0, 0, 0, 0, 0, 0
82.     for v in range(len(TD["单品编码"])):
83.         index2 = np.array(names["单品编码"] == TD.iloc[v, 2])
84.         rate = (TD.iloc[v, 4] - DPrices[str(uv[i])][TD.iloc[v, 2]])/DPrices[str(uv[i])][TD.iloc[v, 2]]
85.         if names.iloc[index2, 2].values == 1011010101:
86.             a += TD.iloc[v, 3]*rate
87.             sa += TD.iloc[v, 3]
88.         elif names.iloc[index2, 2].values == 1011010201:
89.             b += TD.iloc[v, 3]*rate
90.             sb += TD.iloc[v, 3]
91.         elif names.iloc[index2, 2].values == 1011010402:
92.             c += TD.iloc[v, 3]*rate
93.             sc += TD.iloc[v, 3]
94.         elif names.iloc[index2, 2].values == 1011010501:
95.             d += TD.iloc[v, 3]*rate
96.             sd += TD.iloc[v, 3]
97.         elif names.iloc[index2, 2].values == 1011010504:
98.             e += TD.iloc[v, 3]*rate
99.             se += TD.iloc[v, 3]
100.        elif names.iloc[index2, 2].values == 1011010801:
101.            f += TD.iloc[v, 3]*rate
102.            sf += TD.iloc[v, 3]
103.    A.append(0) if sa == 0 else A.append(a/sa)
104.    B.append(0) if sb == 0 else B.append(b/sb)
105.    C.append(0) if sc == 0 else C.append(c/sc)
106.    D.append(0) if sd == 0 else D.append(d/sd)
107.    E.append(0) if se == 0 else E.append(e/se)
108.    F.append(0) if sf == 0 else F.append(f/sf)
109.
110. print(np.array(A))
111.
112.
113. # In[982]:
114.
115.
116. # 导入数据
117. A, B, C, D, E, F = np.array(A), np.array(B), np.array(C), np.array(D), np.array(E), np.array(F)
118. DF = pd.DataFrame({"花叶类":A,"花菜类":B,
119.                    "水生根茎类":C,"茄类":D,
120.                    "辣椒类":E,"食用菌":F})
121. DF.to_excel("data/日加价比例数据.xlsx", index=False)
122. SalesD = pd.read_excel("data/全年日销量数据.xlsx")
123. print(SalesD.shape)
124.
125.
126. # In[1058]:
127.
128.
129. from sklearn.metrics import mean_squared_error, r2_score
130. from sklearn.linear_model import LinearRegression
131.
132. # 多元线性回归模型
133. def Regression(sA, salesA, season):
134.     plt.figure(figsize=(10, 8), dpi=600)
135.     plt.subplot(311)
136.     plt.plot(sA, color='#0081CF')
137.     plt.title('Observed', fontproperties=label_prop, fontsize=10)
138.     plt.yticks(fontproperties=Efont_prop, fontsize=7)
139.     plt.xticks(fontproperties=Efont_prop, fontsize=7)
140.
141.     plt.subplot(312)
142.     plt.plot(salesA, color='#008E9B')
143.     plt.yticks(fontproperties=Efont_prop, fontsize=7)
144.     plt.xticks(fontproperties=Efont_prop, fontsize=7)
145.     plt.ylabel('Trend', fontproperties=label_prop, fontsize=9)
146.
147.     plt.subplot(313)
148.     plt.plot(season, color='#008E9B')
149.     plt.yticks(fontproperties=Efont_prop, fontsize=7)
150.     plt.xticks(fontproperties=Efont_prop, fontsize=7)
```

```
    plt.ylabel('Trend', fontproperties=label_prop, fontsize=9)

    x = pd.DataFrame({"成本加价比例": sA, "季节分量": season})
    y = np.array(salesA)

    model = LinearRegression()
    model.fit(np.array(x), y)
    y_pred = model.predict(x)

    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    print("均方误差 (MSE):", mse)
    print("R^2 分数 (拟合准确性):", r2)

    coefficients = model.coef_
    coefficients = np.append(coefficients, model.intercept_)

    print("回归系数、截距:", coefficients)

    return coefficients

Num = 7
Season = pd.read_excel("data/季节性数据.xlsx")
print(Season.head())
SalesRestD = np.zeros((1085, 6))
Coff = np.zeros((6, 3))
sA = np.zeros((6, 7))
for i in range(6):
    sA[i, :] = np.array(DF.iloc[:, i]).reshape((-1, Num)).mean(0)
    salesA = np.array(SalesD.iloc[1:, i+1]).reshape((-1, Num)).mean(0)
    Coff[i, :] = Regression(sA[i, :], salesA, Season.iloc[:, i])
    SalesRest = np.array(SalesD.iloc[1:, i+1]).reshape(-1, Num)
    SalesRest -= salesA
    SalesRestD[:, i] = np.array(SalesRest.reshape((-1, 1)))[:, 0]

Data = pd.DataFrame(SalesRestD)
Data["时间"] = SalesD["时间"]
Data = Data.set_index("时间")
Data.to_excel("data/P2 数据.xlsx")
191.

# In[986]:


print(SalesRestD)
Data = pd.DataFrame(SalesRestD)
Data["时间"] = SalesD["时间"]
Data = Data.set_index("时间")
Data.to_excel("data/随机分量数据.xlsx")
print(SalesD.head())
print(SalesD.iloc[-14:, 1:].mean())


# In[987]:


SD = np.array(SalesD.iloc[:, 1:])
LinearPredict = np.zeros((7, 6))
PriceRate = np.zeros((7, 6))
Daysum = np.zeros((7, 1))
Seven = pd.read_excel("data/7 天预测.xlsx", header=None)
Seven = np.array(Seven)
Prices = np.array([4.44645836396608, 5.77153576129882, 8.08550377155173, 5.68973684210526,
          7.69230883735510, 6.14664332470695])
SpendRate = np.array([11.1577000000000, 9.16800000000000, 9.44052631578948,6.75100000000000,
             8.00422222222222,8.29750000000000])/100


# In[990]:


class Gene:

    # Set parameters for Genetic Algorithm
    GENE_SIZE = 20
    POP_SIZE = 200
    CROSSOVER_RATE = 1
```

```
    MUTATION_RATE = 0.02
    N_GENERATIONS = 200

    def __init__(self, n, m, bounds):
        self.n = n
        self.m = m
        # Set boundary
        self.bounds = bounds
        # save results
        self.Cresults = np.zeros((N_GENERATIONS, 2))
        self.BestPop = np.random.randint(2, size=(POP_SIZE, GENE_SIZE*self.m))

    def Correlation(self, SalesDay):
        # 蔬菜品类的互补和替代关系
        alpha = 0.1
        Delta = np.array((SD[-7:, :].mean(0) - SD[-14:, :].mean(0)) * alpha)
        Influence = np.zeros((6, 1))
        Relation = {"0": [[1], [0, 4, 5]], "1": [[0], [1]],
                    "2": [[], [2, 4, 5]], "3": [[], [2, 3, 5]],
                    "4": [[], [0, 2, 4]], "5": [[], [0, 2, 3, 5]]}
        for i in range(6):
            for u in Relation[str(i)][0]:
                SalesDay[u] += Delta[i]
            for u in Relation[str(i)][1]:
                SalesDay[u] -= Delta[i]
        return SalesDay

    def Fx(self, x):
        P = 0.0539
        beta = 0.5
        LinearPredict[self.n, :] = x * Coff[:, 0] + Coff[:, 1] * Season.iloc[self.n, :] + Coff[:, 2] + Seven[self.n, :]
        LinearPredict[self.n, :] = self.Correlation(LinearPredict[self.n, :])
        LinearPredict[self.n, LinearPredict[self.n, :] < 0] = 0
        Q1 = Prices*LinearPredict[self.n, :]*(1-SpendRate)*(x*(1-P)+((1+x)*beta-1)*P)
        Q2 = Prices*LinearPredict[self.n, :]*SpendRate*((1+x)*beta-1)
        return (Q1+Q2).sum()

    def TranslateDNA(self, pop):
        FacDNA = np.zeros((POP_SIZE, self.m))
        Gene_Size = GENE_SIZE
        for i in range(POP_SIZE):
            for j in range(self.m):
                Temp = pop[i, j::self.m]
                FacDNA[i, j] = Temp.dot(2**np.arange(Gene_Size)[::-1])/float(2**Gene_Size-1)*(self.bounds[j, 1]-self.bounds[j, 0])+self.bounds[j, 0]
        return FacDNA

    def Mutation(self, child, Mutation_Rate=0.003):
        if np.random.rand() < Mutation_Rate:
            mutate_point = np.random.randint(0, GENE_SIZE)
            child[mutate_point] = child[mutate_point] ^ 1

    def CrossoverMutation(self, pop, Crossover_Rate=0.8):
        for father in pop:
            child = father
            if np.random.rand() < Crossover_Rate:
                mother = pop[np.random.randint(POP_SIZE)]
                cross_points = np.random.randint(low=0, high=GENE_SIZE*self.m)
                child[cross_points:] = mother[cross_points:]
            self.Mutation(child, MUTATION_RATE)
            np.append(pop, child)
        return pop
    def GetFitness(self, pop):
        x = self.TranslateDNA(pop)

        pred = np.zeros((len(x[:, 0]), 1))
        for i in range(len(x[:, 0])):
            pred[i] = Fx(x[i, :])
        return (pred - np.min(pred)) + 1e-3
    def SelectPop(self, pop, fitness):
        idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
                               p=(fitness[:, 0]) / (fitness[:, 0].sum()) )
        return pop[idx]
    def Print_Info(self, pop):
        fitness = self.GetFitness(pop)
        max_fitness_index = np.argmax(fitness)
        x = self.TranslateDNA(pop)
        PriceRate[self.n, :] = x[max_fitness_index, :]
```

```
        Daysum[self.n, :] = self.Fx(x[max_fitness_index, :])
        return self.Cresults

    def Check_Info(self, pop):
        fitness = self.GetFitness(pop)
        max_fitness_index = np.argmax(fitness)
        x = self.TranslateDNA(pop)

        return fitness[max_fitness_index], self.Fx(x[max_fitness_index, :])

    def GeneticAlgorithm(self):
        pop = np.random.randint(2, size=(POP_SIZE, GENE_SIZE*self.m))
        for i in range(N_GENERATIONS):
            pop = np.array(self.CrossoverMutation(pop, CROSSOVER_RATE))
            fitness = self.GetFitness(pop)
            pop = self.SelectPop(pop, fitness)
            self.Cresults[i, :] = self.Check_Info(pop)
        self.BestPop = pop
        return self.Print_Info(pop)


# In[1016]:


alpha = 0
for i in range(7):
    bounds = np.array([[l, u] for l, u in zip(0.9*(1+alpha)*sA[:, i], 1.1*(1+alpha)*sA[:, i])])
    GPredict = Gene(i, 6, bounds)
    Data = GPredict.GeneticAlgorithm()
    plt.figure(figsize=(10, 8), dpi=600)
    plt.subplot(211)
    plt.plot(Data[:, 0], color='#0081CF')
    plt.title('Observed', fontproperties=label_prop, fontsize=10)
    plt.yticks(fontproperties=Efont_prop, fontsize=7)
    plt.xticks(fontproperties=Efont_prop, fontsize=7)

    plt.subplot(212)
    plt.plot(Data[:, 1], color='#008E9B')
    plt.yticks(fontproperties=Efont_prop, fontsize=7)
    plt.xticks(fontproperties=Efont_prop, fontsize=7)
    plt.ylabel('Trend', fontproperties=label_prop, fontsize=9)


# In[1017]:


print("最优周获取利润: ", Daysum.sum())
Names = ["花叶类", "花菜类", "水生根茎类","茄类",
         "辣椒类", "食用菌"]
for i in range(7):
    print("第 %d 天，总补货量为: %.3f 千克，总利润为: %.3f 元
    " % (i+1, LinearPredict[i, :].sum(), Daysum[i]))
    print("-----------------第 %d 天补货计划----------------" % (i+1))
    for j in range(6):
        print("<%s> 菜品当日补货量为: %.3f 千克" % (Names[j], LinearPredict[i, j]))
    print("-----------------------------------------------------")
    print("-----------------第 %d 天定价策略----------------" % (i+1))
    for j in range(6):
        print("<%s> 菜品当日加成定价比为: %.3f" % (Names[j], PriceRate[i, j]))
    print("-----------------------------------------------------")
Kindsum = np.array(LinearPredict.sum(0))
print("-----------------7 月 1 日-7 月 7 日补货计划汇总---------------")
for i in range(6):
    print("<%s> 菜品周最优补货量为: %.3f 千克" % (Names[i], LinearPredict[:, i].sum()))
print("-------------------------------------------------------------")


# In[1066]:


# 模型灵敏度分析
DaySum[:, 3] = Daysum.reshape((7, ))
plt.figure(figsize=(8, 6), dpi=600)
plt.subplot(211)
for i in range(6):
    plt.plot(DaySum[:, i], label=r'$\theta$='+str(i+5)+'%')
plt.yticks(fontproperties=Efont_prop, fontsize=7)
x_labels = ['July 1st', 'July 2nd', 'July 3rd', 'July 4th', 'July 5th', 'July 6th', 'July 7th']
```

```
plt.xticks(range(7), x_labels)
plt.xticks(fontproperties=Efont_prop, fontsize=7)
plt.legend(loc='upper right', fontsize=7, fancybox=True)

SM = (DaySum - DaySum.mean(1).reshape((7, 1)))/DaySum.mean(1).reshape((7, 1))
plt.subplot(212)
for i in range(6):
    sns.kdeplot(SM[:, i], shade=True, label=r'$\theta$='+str(i+5)+'%')
plt.ylabel('')
plt.xticks(fontproperties=Efont_prop, fontsize=7)
plt.yticks(fontproperties=Efont_prop, fontsize=7)
plt.legend(loc='upper right', fontsize=7, fancybox=True)
plt.savefig('灵敏度分析.png', dpi=600)


# In[1059]:


# 问题三的求解
# import dataset
SpendRate = pd.read_excel("data/附件 4.xlsx", sheet_name="Sheet1")
SpendRate = SpendRate.set_index("单品编码")
print(SpendRate.head())
R = pd.read_excel("data/r.xlsx", header=None)

Ndata = pd.read_excel("data/单品日-周-月数据.xlsx", sheet_name="Sheet1")
print(Ndata.head())
KonwD = Ndata.iloc[:, -7:]


# In[1060]:


# preprocess data
KnowD = KonwD.iloc[np.array(KonwD.sum(1) > 0), :]
# 总销售量
KnowD["总销量"] = KnowD.sum(1)
KnowD["波动性"] = KnowD.apply(np.std, axis=1)
KnowD["联动性"] = np.array(R.sum(1))
KnowD["联动性"] = -1 + 2*(KnowD["联动性"] - KnowD["联动性"].min()) / (KnowD["联动性"].max() - KnowD["联动性"].min())
423.
KnowD["单品编码"] = Ndata.iloc[KnowD.index, 0]
KnowD["单品名称"] = Ndata.iloc[KnowD.index, 1]
KnowD = KnowD.set_index("单品编码")
KnowD["损耗率"] = SpendRate.loc[KnowD.index, "损耗率(%)"]
print(KnowD.head())
print("6 月 24 日-6 月 30 日销售的单品总数：%d" % len(KnowD.index))


# In[1061]:


import scipy.cluster.hierarchy as sch
X = KnowD[["总销量", "波动性"]]

# 计算距离矩阵
plt.figure(figsize=(8, 6), dpi=600)
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'), labels=np.array(KnowD["单品名称"]))
plt.title("不同单品聚类树状图", fontproperties=font_prop1)
plt.ylabel("Distance", fontproperties=Efont_prop)
plt.yticks(fontproperties=font_prop1)
plt.xticks(fontproperties=font_prop1, rotation=90, fontsize=6)
plt.savefig('单品聚类树状图.png', dpi=600)

lamb = 0.65
scaler = StandardScaler()
M = scaler.fit_transform(np.array(KnowD["总销量"]).reshape(-1, 1))
N = scaler.fit_transform(np.array(KnowD["波动性"]).reshape(-1, 1))

Z = lamb * M - (1-lamb) * N + M * np.array(KnowD["联动性"]).reshape(-1, 1)
KnowD["Scores"] = (Z - Z.min()) / (Z.max() - Z.min())

Sort_KnowD = KnowD.sort_values(by="Scores", ascending=False)
print(Sort_KnowD.head())
Sort_KnowD.to_excel("data/归一化数据.xlsx")
```

```
458.
459.
460. # In[1062]:
461.
462.
463. # 获取各单品批发价
464. Sort_KnowD["批发价"] = np.zeros((49, 1))
465. for i in range(24, 31):
466.     date = "2023-06-" + str(i)
467.     for k, v in DPrices[date].items():
468.         try:
469.             Sort_KnowD.loc[int(k), "批发价"] += v
470.         except:
471.             continue
472.
473. Sort_KnowD["批发价"] = Sort_KnowD["批发价"]/7
474. # 获取各单品售价
475. Sort_KnowD["售价"] = np.zeros((49, 1))
476. for i in range(24, 31):
477.     date = "2023-06-" + str(i)
478.     for k, v in DSales[date].items():
479.         try:
480.             Sort_KnowD.loc[int(k), "售价"] += v
481.         except:
482.             continue
483.
484. Sort_KnowD["售价"] = Sort_KnowD["售价"]/7
485. Sort_KnowD["成本加价比"] = (Sort_KnowD["售价"] - Sort_KnowD["批发价"])/Sort_KnowD["批发价"]
486. Sort_KnowD["最大销量"] = Sort_KnowD.iloc[:, :6].max(1)
487. Sort_KnowD.drop(Sort_KnowD[Sort_KnowD["最大销量"] < 2.5].index, inplace=True)
488. print(Sort_KnowD.head())
489.
490.
491. # In[1063]:
492.
493.
494. class GENE:
495.
496.     # Set parameters for Genetic Algorithm
497.     GENE_SIZE = 40    # Gene length
498.     POP_SIZE = 200     # Population size
499.     CROSSOVER_RATE = 1   # Crossover rate
500.     MUTATION_RATE = 0.02   # Mutation rate
501.     N_GENERATIONS = 200   # Maximum generations
502.
503.     def __init__(self, m, e):
504.         self.m = m
505.         # save results
506.         self.Cresults = np.zeros((N_GENERATIONS, 2))
507.         self.BestPop = np.random.randint(2, size=(POP_SIZE, GENE_SIZE*m))
508.         self.A, self.B, self.C, self.D, self.E, self.F = [], [], [], [], [], []
509.         for v in range(len(Sort_KnowD.index[:m])):
510.             index2 = np.array(names["单品编码"] == Sort_KnowD.index[v])
511.             if names.iloc[index2, 2].values == 1011010101:
512.                 self.A.append(v)
513.             elif names.iloc[index2, 2].values == 1011010201:
514.                 self.B.append(v)
515.             elif names.iloc[index2, 2].values == 1011010402:
516.                 self.C.append(v)
517.             elif names.iloc[index2, 2].values == 1011010501:
518.                 self.D.append(v)
519.             elif names.iloc[index2, 2].values == 1011010504:
520.                 self.E.append(v)
521.             elif names.iloc[index2, 2].values == 1011010801:
522.                 self.F.append(v)
523.         # Set boundary
524.         self.Nbounds = np.array([[2.5, u] for u in Sort_KnowD["最大销量"]])
525.         self.Wbounds = np.array([[l, u] if l > 0 else [u, l] for l, u in zip(0.8*(1-e)*Sort_KnowD["成本
    加价比"], 1.4*(1+e)*Sort_KnowD["成本加价比"])])
526.
527.     # define object function
528.     def Fx(self, x):
529.         P = 0.0539
530.         beta = 0.7
531.         N, W = x[:, 0], x[:, 1]
532.         Q1 = N * Sort_KnowD.iloc[:self.m, -4]*(1-0.01*Sort_KnowD.iloc[:self.m, -6])*(W*(1-
    P)+((1+W)*beta-1)*P)
```

```
        Q2 = N * Sort_KnowD.iloc[:self.m, -4]*0.01*Sort_KnowD.iloc[:self.m, -6]*((1+W)*beta-1)
        return (Q1+Q2).sum()

    def TranslateDNA(self, pop):
        FacDNA = np.zeros((POP_SIZE, self.m, 2))
        Gene_Size = GENE_SIZE / 2
        for i in range(POP_SIZE):
            for j in range(self.m):
                Temp = pop[i, j::self.m]
                Earn, Rate = Temp[0::2], Temp[1::2]
                FacDNA[i, j, 0] = Earn.dot(2**np.arange(Gene_Size)[::-1])/float(2**Gene_Size-1)*(self.Nbounds[j, 1]-self.Nbounds[j, 0])+self.Nbounds[j, 0]
                FacDNA[i, j, 1] = Rate.dot(2**np.arange(Gene_Size)[::-1])/float(2**Gene_Size-1)*(self.Wbounds[j, 1]-self.Wbounds[j, 0])+self.Wbounds[j, 0]
        return FacDNA
    def Mutation(self, child, Mutation_Rate=0.003):
        # Mutation with Mutation_Rate probability
        if np.random.rand() < Mutation_Rate:
            # Randomly generate an mutation location
            mutate_point = np.random.randint(0, GENE_SIZE)
            # Inverts the binary bit of the mutation point
            child[mutate_point] = child[mutate_point] ^ 1
    def CrossoverMutation(self, pop, Crossover_Rate=0.8):

        # Traverse each individual in the population, taking that individual as the father
        for father in pop:
            child = father    # The child first gets all the genes of the father

            # Crossover occurs with a certain probability when producing offspring
            if np.random.rand() < Crossover_Rate:
                # Another individual is selected in the population and that individual is taken as the mother
                mother = pop[np.random.randint(POP_SIZE)]
                # Randomly generate an intersection
                cross_points = np.random.randint(low=0, high=GENE_SIZE*self.m)
                # The child gets the mother's genes located behind the intersection
                child[cross_points:] = mother[cross_points:]

            # Each child has a certain chance of mutating
            self.Mutation(child, MUTATION_RATE)
            np.append(pop, child)

        return pop
    def GetFitness(self, pop):
        x = self.TranslateDNA(pop)
        # 检查约束条件
        pred = np.zeros((len(x[:, 0, 0]), 1))
        for i in range(len(x[:, 0, 0])):
            if (x[i, self.A, 0].sum()-134.278 > 0):
                pred[i] = 0
            elif (x[i, self.B, 0].sum()-18.789 > 0):
                pred[i] = 0
            elif (x[i, self.C, 0].sum()-29.257 > 0):
                pred[i] = 0
            elif (x[i, self.D, 0].sum()-31.610 > 0):
                pred[i] = 0
            elif (x[i, self.E, 0].sum()-90.873 > 0):
                pred[i] = 0
            elif (x[i, self.F, 0].sum()-67.892 > 0):
                pred[i] = 0
            else:
                pred[i] = self.Fx(x[i, :, :])   # Calculate objective value
        return (pred - np.min(pred)) + 1e-3
    # nature selection according to pop's fitness
    def SelectPop(self, pop, fitness):
        idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
                               p=(fitness[:, 0]) / (fitness[:, 0].sum()) )
        return pop[idx]
    def Print_Info(self, pop, SumD):
        fitness = self.GetFitness(pop)
        max_fitness_index = np.argmax(fitness)
        x = self.TranslateDNA(pop)
        if(x[max_fitness_index, self.A, 0].sum()-134.278 <= 0):
            print("<%s> 满足补货限制条件" % Names[0])
        if((x[max_fitness_index, self.B, 0].sum()-18.789) <= 0):
            print("<%s> 满足补货限制条件" % Names[1])
        if (x[max_fitness_index, self.C, 0].sum()-29.257 <= 0):
            print("<%s> 满足补货限制条件" % Names[2])
```

```
        if (x[max_fitness_index, self.D, 0].sum()-31.610 <= 0):
            print("<%s> 满足补货限制条件" % Names[3])
        if (x[max_fitness_index, self.E, 0].sum()-90.873 <= 0):
            print("<%s> 满足补货限制条件" % Names[4])
        if (x[max_fitness_index, self.F, 0].sum()-67.892 <= 0):
            print("<%s> 满足补货限制条件" % Names[5])

        SumD[:, 0] = x[max_fitness_index, :, 0]
        SumD[:, 1] = x[max_fitness_index, :, 1]
        obj = self.Fx(x[max_fitness_index, :, :])
        print("Optimal value:", obj)
        return obj

    def Check_Info(self, pop):
        fitness = self.GetFitness(pop)
        max_fitness_index = np.argmax(fitness)
        x = self.TranslateDNA(pop)

        return fitness[max_fitness_index], self.Fx(x[max_fitness_index, :, :])

    def GeneticAlgorithm(self, SumD):
        pop = np.random.randint(2, size=(POP_SIZE, GENE_SIZE*self.m))
        for i in range(N_GENERATIONS):
            pop = np.array(self.CrossoverMutation(pop, CROSSOVER_RATE))
            fitness = self.GetFitness(pop)
            pop = self.SelectPop(pop, fitness)
            self.Cresults[i, :] = self.Check_Info(pop)
        self.BestPop = pop
        return self.Print_Info(pop, SumD)


# In[1067]:


m = 30
SalesRate = np.zeros((30, 20))
PRate = np.zeros((30, 20))
Obj = np.zeros((20, 1))
for i in range(1):
    SumD = np.zeros((m, 2))
    Gpredict = GENE(m, i*0.01 + 0.01)
    Obj[i] = Gpredict.GeneticAlgorithm(SumD)
    print("当选取 %d 种单品蔬菜时，单日预计获得最大利润为: %.3f 元" % (m, Obj[i]))
    Output = pd.DataFrame({"单品名称": Sort_KnowD.iloc[:m, -7].values, "补货计划": SumD[:, 0], "定价策略": SumD[:, 1]})
    Output = Output.set_index("单品名称")
    print("当选取 %d 种单品蔬菜时，单日总补货量为: %.3f 千克" % (m, Output.iloc[:, 0].sum()))
    print("当选取 %d 种单品蔬菜时，<%s> 单日总补货量为: %.3f 千克" % (m, Names[0], Output.iloc[Gpredict.A, 0].sum()))
    print("当选取 %d 种单品蔬菜时，<%s> 单日总补货量为: %.3f 千克" % (m, Names[1], Output.iloc[Gpredict.B, 0].sum()))
    print("当选取 %d 种单品蔬菜时，<%s> 单日总补货量为: %.3f 千克" % (m, Names[2], Output.iloc[Gpredict.C, 0].sum()))
    print("当选取 %d 种单品蔬菜时，<%s> 单日总补货量为: %.3f 千克" % (m, Names[3], Output.iloc[Gpredict.D, 0].sum()))
    print("当选取 %d 种单品蔬菜时，<%s> 单日总补货量为: %.3f 千克" % (m, Names[4], Output.iloc[Gpredict.E, 0].sum()))
    print("当选取 %d 种单品蔬菜时，<%s> 单日总补货量为: %.3f 千克" % (m, Names[5], Output.iloc[Gpredict.F, 0].sum()))
    print(Output)
    SalesRate[:, i] = np.array(Output.iloc[:, 0])
    PRate[:, i] = np.array(Output.iloc[:, 1])

```

附录 7

介绍：C.m | 用于问题二 GBDT 算法预测的求解

```
%%  清空环境变量
warning off             % 关闭报警信息
close all               % 关闭开启的图窗
clear                   % 清空变量
clc                     % 清空命令行

%%  导入数据（时间序列的单列数据）
result = readmatrix('P2data.xlsx');
result = result(2:end,2); % 2:7
```

```
%%  数据分析
num_samples = length(result);  % 样本个数
kim = 10;                      % 延时步长（kim 个历史数据作为自变量）
zim =  1;                      % 跨 zim 个时间点进行预测

%%  构造数据集
for i = 1: num_samples - kim - zim + 1
    res(i, :) = [reshape(result(i: i + kim - 1), 1, kim), result(i + kim + zim - 1)];
end

%%  划分训练集和测试集
temp = 1: 1: length(result)-kim;

P_train = res(temp(1: ceil(0.8*length(temp))), 1: kim)';
T_train = res(temp(1: ceil(0.8*length(temp))), kim+1)';
M = size(P_train, 2);

P_test = res(temp(ceil(0.8*length(temp))+1: end), 1: kim)';
T_test = res(temp(ceil(0.8*length(temp))+1: end), kim+1)';
N = size(P_test, 2);

%%  数据归一化
[p_train, ps_input] = mapminmax(P_train, 0, 1);
p_test = mapminmax('apply', P_test, ps_input);

[t_train, ps_output] = mapminmax(T_train, 0, 1);
t_test = mapminmax('apply', T_test, ps_output);

%%  转置以适应模型
p_train = p_train'; p_test = p_test';
t_train = t_train'; t_test = t_test';

%% GBDT 梯度提升决策树
% 设置参数
MaxDepth = 7; % 最大树深度
maxgen = 300; % 迭代次数/弱分类器个数
alpha = 0.5; % 最大学习率

% 初始化弱分类器
Mdl{maxgen} = [];
Mdl{1} = fitrtree(p_train,t_train,'MaxNumSplits',MaxDepth);
prediction1(:,1) = predict(Mdl{1},p_train);
prediction2(:,1) = predict(Mdl{1},p_test);
rou = 0.1; % 初始学习率
prediction1(:,1) = prediction1(:,1)*rou';
prediction2(:,1) = prediction2(:,1)*rou';
yhat(:,1) = t_train-prediction1(:,1);
Loss = zeros(1,maxgen);
Loss(1) = sum(yhat(:,1).^2)/2;
for i = 2:maxgen
    Mdl{i} = fitrtree(p_train,yhat(:,i-1),'MaxNumSplits',MaxDepth);
    prediction1(:,i) = predict(Mdl{i},p_train);
    prediction2(:,i) = predict(Mdl{i},p_test);
    cnt = 1;
    Loss(i) = sum((t_train-prediction1*[rou,alpha]').^2)/2;
    temp2 = alpha;
    for j = 0:0.001:alpha
        temp1 = prediction1*[rou,j]';
        if sum((t_train-temp1).^2)<Loss(i)
            Loss(i) = sum((t_train-temp1).^2)/2;
            temp2 = j;
        end
    end
    % prediction1(:,i) = temp2*prediction1(:,i);
    % prediction2(:,i) = temp2*prediction2(:,i);
    rou = [rou,temp2];
    yhat(:,i) = t_train-prediction1*rou';
end

%%  仿真测试
t_sim1 = prediction1*rou';
t_sim2 = prediction2*rou';
k=7;
for i = 1:k
    prediction = zeros(1,maxgen);
    temp1 = result(end-kim+1:end);
```

```
    temp2 = mapminmax('apply',temp1,ps_input);
    for j = 1:maxgen
        prediction(j) = predict(Mdl{j},temp2');
        % prediction(j) = prediction(j)*rou(j);
    end
    temp3 = prediction*rou';
    result = [result;mapminmax('reverse',temp3,ps_output)];
end
figure
plot(1:1085,result(1:1085),'b-')
hold on
plot(1086:1092,result(1086:1092),'r-')
result(1086:1092)
%%  数据反归一化
T_sim1 = mapminmax('reverse', t_sim1, ps_output);
T_sim2 = mapminmax('reverse', t_sim2, ps_output);

T_sim1 = T_sim1';
T_sim2 = T_sim2';

%%  均方根误差
error1 = sqrt(sum((T_sim1 - T_train).^2) ./ M);
error2 = sqrt(sum((T_sim2 - T_test ).^2) ./ N);

%%  绘图
figure
plot(1: M, T_train, 'r-*', 1: M, T_sim1, 'b-o', 'LineWidth', 1)
legend('真实值','预测值')
xlabel('预测样本')
ylabel('预测结果')
string = {'训练集预测结果对比'; ['RMSE=' num2str(error1)]};
title(string)
xlim([1, M])
grid

figure
plot(1: N, T_test, 'r-*', 1: N, T_sim2, 'b-o', 'LineWidth', 1)
legend('真实值','预测值')
xlabel('预测样本')
ylabel('预测结果')
string = {'测试集预测结果对比';['RMSE=' num2str(error2)]};
title(string)
xlim([1, N])
grid

%%  相关指标计算
%  R2
R1 = corrcoef(T_train,T_sim1);
R2 = corrcoef(T_test,T_sim2);

disp(['训练集数据的 R2 为：', num2str(R1(2))])
disp(['测试集数据的 R2 为：', num2str(R2(2))])

%  MAE
mae1 = mae(T_train,T_sim1);
mae2 = mae(T_test,T_sim2);

disp(['训练集数据的 MAE 为：', num2str(mae1)])
disp(['测试集数据的 MAE 为：', num2str(mae2)])

%  MSE
mse1 = mse(T_train,T_sim1);
mse2 = mse(T_test,T_sim2);

disp(['训练集数据的 MSE 为：', num2str(mse1)])
disp(['测试集数据的 MSE 为：', num2str(mse2)])

% RMSE
rmse1 = sqrt(sum((T_sim1-T_train).^2/M));
rmse2 = sqrt(sum((T_sim2-T_test).^2/N));

disp(['训练集数据的 RMSE 为：', num2str(rmse1)])
disp(['测试集数据的 RMSE 为：', num2str(rmse2)])

%  SSR
ssr1 = sum((T_sim1-mean(T_train)).^2);
ssr2 = sum((T_sim2-mean(T_test)).^2);
```

```
164.
165. %  SST
166. sst1 = sum((T_train-mean(T_train)).^2);
167. sst2 = sum((T_test-mean(T_test)).^2);
168.
169. % R2
170. R1 = ssr1/sst1;
171. R2 = ssr2/sst2;
172.
173. disp(['训练集数据的 R^2 为: ', num2str(R1)])
174. disp(['测试集数据的 R^2 为: ', num2str(R2)])
```