基于极坐标变换回归的自动定价与商品补货策略模型

摘要

在商超环境下，蔬菜类商品的**补货**和**定价策略**尤其关键，因蔬菜保鲜期较短，且需求端和供应端通常具有周期性和季节性。凌晨 3:00-4:00 是商家做出补货决策的关键时段，却往往缺乏对当天市场供应和价格的准确掌握。此外，商品的定价通常采用“成本加成定价”的方式，加成率的选择一定程度上会影响到市场需求的大小。

本文根据 2020 年 6 月-2023 年 6 月三年的蔬菜类商品销售和进货情况，建立了基于**极坐标变换回归**得到加成率和销售量为核心关系的**价格测模型**与**基于熵权法和 TOPSIS 评价的销售组合选择模型**，并根据收益最大化原则，优化收益函数。

题目提供数据较为庞大。应对数据进行总体**预处理**。针对数据分类合并，以品类为基准分类，使用 EXCEL 进行标签匹配合并；针对缺失值和异常值的查找，寻找到 5 款未销售单品、1 个月未销售的茄类蔬菜。

针对问题一　基于对数据进行探索性分析，得出蔬菜各品类、个单品销售量在给定时间内的方差、均值、偏度系数、峰度系数得到的各描述性统计量，绘制各品类、各单品销售量随时间变化的**一阶差分图像和自相关系数图像**，得到其周期性与季节性。利用 MATLAB 绘制不同品类、单品销售量之间的散点图，进行线性规律检验；并利用 Spearman 相关系数进行相关性与显著性的分析。

针对问题二　分析生成定价模式的核心部分——加成率，先用 **TCLL 方法**对各品类销量和加成率做**滞后相关性分析**，得到较为明显的最大 TCLL 和对应的滞后步长，因此进一步对加成率和反对数化的销量**作极坐标变换后线性回归**，回归效果解释性好（1、5、6 组 R^2 大于 0.95），解得一隐函数关系。其次，分别使用 **ARIMA** 传统模型和**随机森林**等机器学习方法预测供货价格趋势，构建收益最大的补货和定价策略时，计算不同品类依据销量占比的折损率、采购成本、打折力度的加权平均数，并通过**遗传算法**，优化收益函数，从而得出相应补货与定价策略。在该方案下，超商可在 7 月 1 日至 7 月 7 日之间获取 8922 元总利润。

针对问题三　相比问题二，本题数据量庞大，故而采取先评价、再选择的方式对日补货总量和定价策略进行求解。基于单品销售量及价格具有一定周期性，选择单品的日销量、因运损和品相变差的折损率、售价、采购价格以及单品打折后的折扣率 5 项指标作为权值，运用**熵权法**确定各权值的权重并使用**优劣解距离法 TOPSIS** 对 251 中单品进行评价，从中选出评价得分最高的 33 项，并用时间序列模型预测这些单品在 7 月 1 日的销售量。最后，我们从这 33 项单品中选择补货名单，列出商超盈利与单品定价、预测销售量、进货量的关系式并用一阶差分线性拟合进行求解，得到日补货总量和定价策略。

针对问题四　考虑实现更精准和高效的蔬菜补货策略，商超需深化对**产地信息**的掌握，同时关注蔬菜的**营养价值数据**来精准捕捉消费者需求。物流方面，必须考虑到运输的价格、时间、条件及损坏率，从而找到更经济、环保的物流方案并实时调整补货策略以减少损耗。此外，商家应深入研究**同行业务模式和补货策略**，从中吸取有益经验并不断创新。通过综合多元化数据，商超不仅可以更好地理解市场动态和消费者需求，也能确保商品的新鲜和高品质，从而实现更高质量的补货决策。

综上所述，本文提供了一套基于极坐标变化回归的的蔬菜补货和定价解决方案，有助于商业匹配进货、定价与市场需求，为其创造更多的收益，同时也为消费者提供喜闻乐见的蔬菜商品。

1 问题重述

1.1 问题背景

在生鲜商超中，蔬菜类商品的保鲜期都比较短，同时面临品相下降和售前运输损失的问题，大部分如当日未售出，隔日就无法再售，影响收益。其中，品相与销售时间呈负相关；由于蔬菜类产品在运输过程中的颠簸等情况，会产生 20% - 30% 的运输损失。面对品类下降与产生运输损失的蔬菜类商品，商超采用打折等手段销售。商超进行补货交易时，由于无法得知具体单品和进货价格，因此商家需参考历史销售和可靠的市场需求端与供应端分析，考虑由于品种、产地带来的利润及损耗差异，以制定合理有效的补货策略，最大化商超收益。

定价方面，蔬菜的定价一般采用“成本加成定价”。对于存有历史销售数据相对完整的蔬菜产品，一般采用 $X = C(1 + \Omega)$。X 表示价格，C 表示平均成本，Ω 表示成本加成率。

1.2 问题要求

问题一要求根据附件提供的时间相关的品类与单品销售数据，分析不同品类和单品销售进行分类，并分析二者分布规律及其间可能存在的关联。

问题二要求以品类为单位，分析蔬菜品类销售总量与成本加成定价的关系，并制定 2023 年 7 月 1-7 日一周各蔬菜品类的日补货量和定价策略。

问题三要求在有限的销售空间下，即满足售单量总数 27-33 个，各单品订购量最小陈列量 2.5 千克的约束，单品总量尽量小于问题二中得出的 7 月 1 日补货总量，并根据 2023 年 6 月 24-30 日的可售品种，细化 7 月 1 日满足市场对各类蔬菜需求前提下实现商超最大收益的的单品补货量和定价策略，

问题四要求探索可辅助制定更好蔬菜商品补货和定价策略的相关数据，并对选择的数据如何优化策略提供理由与建议。

2 模型假设、符号说明

2.1 模型假设

- 在预测的时间内，蔬菜需求和供应侧不发生剧变，需求、供应、价格均相对稳定，受季节性因素影响
- 财务记账准确，均为真实结果
- 蔬菜类商品的定价策略遵循“成本加成定价”方法，而且可以通过优化定价策略来影响需求
- 蔬菜销售量能作为反映当地市场需求的重要因素
- 蔬菜损耗率变化较小，可以用附件四数据替代
- 蔬菜运损和品相变化是商超决定打折的直接原因
- 存在商超赠送情况，即打折力度为 100%。
- 滞销情况只存在一天，第二日会将前一日产品全部卖出或送出

.

符号	符号说明	单位
B_{ij}	采购价格矩阵	元/kg
W_{ij}	补货重量矩阵	kg
SP	销售量	kg
d	品类折损率	/
D_i	第 i 品类的打折力度	%
α_i	第 i 号单品的损耗率	%
$Profit$	收益优化函数	元
S	第 i 品类销售量	kg
Ω_i	第 i 品类的加成率	%
dis_i	单品 i 打折力度	%
b_i	单品 i 平均采购价	%

表 1: 符号说明

2.2 符号说明

3 问题分析

3.1 问题一

问题一的核心目标在于提供描述各品类和单品销售量分布规律及相互关系的模型。要求模型针对给定的品类和单品，反映销量在品类与单品的维度上随时间的分布规律，探索品各类和单品销售量的相关关系。

为更直观反映规律和相关性，一方面，选择在已有的数据中除去退货量；另一方面，时间覆盖范围较大，在短时间内的品类和单品销售量波动不具有长期代表性；因此以“一周”为单位，**建立时间序列模型**分析特定品类和单品**在一周内的销量总和**与时间的关系。针对品类与单品的相关性，根据销量分品类进行 Spearman 相关性分析；针对单品与单品的相关性，通过绘制单品与销量、价格、重量的时间波动图，观察得到结果。这既简化模型，也使模型拟合求得的规律较为准确。

3.2 问题二

第一问要求以品类为单位，分析蔬菜品类销售总量与成本加成定价的关系。第二问要求制定 2023 年 7 月 1-7 日一周各蔬菜品类的日补货量和定价策略。第一小问在于分析销售总量和成本加成定价的关系，首先考虑使用 Pearson 检验，若结果不突出，考虑对时间序列进行滑动，分析滞后的相关性。成本定价的关键因素是加成率，由于销量数据左偏，故考虑采用反对数化处理，后对两者尝试回归方法，以便进一步预测。

第二小问，则设置多个影响收益的因素，基于历史销量数据进行加权平均，获取打折力度、折损率、平均生产成本等数据，根据第一小问提供的加成率与销量数据的关系，优化收益目标函数，提供合理的 7 月 1 日-7 月 7 日 W 补货数矩阵和对应的 Ω 矩阵

3.3 问题三

这个问题的主要目标是在约束条件下制定单品的补货计划和定价策略，以满足市场需求并最大化商超的收益。该问题是一个规划问题，约束条件多样，分析各约束条件可知不同约束变量：1. 单品数量；2. 单品采

购额；约束条件则有：1. 27-33 个单品数量；2. 单品采购额需和最小采购额以上；3. 市场需求作为上界软约束。优化函数为：商超在 7 月 1 日的收益额函数

$$Profit_i = \sum_i ((P_i - B_i)(W_i - D_i) - D_i B_i)$$

考虑到选择单品标准难以确定，故考虑使用熵权法对三年日平均销量、平均采购价、损耗率、平均售价处理，确定权重矩阵，后使用 TOPSIS 方法择选可行单品。

3.4 问题四

要求探索可辅助制定更好蔬菜商品补货和定价策略的相关数据，核心在于探索除附件材料提供信息，即商家销售基本信息与蔬菜类别信息，对于模型建立的准确性和合理性的有提高效果的因素。考虑到题目中提及路途损失，因此产地和销售地信息可能会对蔬菜的品相、价格产生影响。题目中只提及因品相损失的打折活动，这一商业模式较为单一，如果提供实施不同商业模式的商家交易情况与市场的平均水平，则能尝试设计提高收益的新交易模式：如打包销售、短视频销售的模式，这对补货的进行和需求的分析也有新的指导作用。

4 数据预处理

数据特征：

（1）数据覆盖时间范围长，具有较为明显的周期性与季节性。

（2）具有时间分析序列数据的常见问题：数据空白多，数据噪声多。

（3）蔬菜品类和单品数多，数据类别复杂，数据维度较大

综合（1）（2）（3），结合相关文献（引用文献要写在最后面），数据预处理：对空白值处理，使用 EXCEL 进行整理，赋予不同品类和单品序号，以便进一步的时间序列分析与预测。

4.1 分类序号

将 6 个品类的蔬菜分配序号，对应 {1,2,3,4,5,6}，销售、退货分配 {1，-1}，打折、未打折分配 {1,0} 各单品分配 1 251 的序号，具体编号分配见支撑材料。

花叶类 =1 **花菜类** =2 **水生根茎类** =3 **茄类** =4 **辣椒类** =5 **食用菌** =6

销售 =1；**退货** =-1；

打折 =1；**未打折** =0；利用 EXCEL 将附件一、附件二、附件三按照单品名称、销售日期为条件合并处理，得到具有销售日期、打折情况、销售情况、单品序号（新）、品类序号（新）的表单。

4.2 缺失值处理

部分菜品在部分时间点不存在数据点，在绘制销售额、销量、售卖价格的时间波动图时，会产生难以解释的直线。对于此类空白数据值，使用 0 替代。合理性如下：考虑商超对财务统计精确的假设和商超在购买前不知单品种类，可认为没有记录的品类单品为未购入与消费。辣椒类商品在 2022 年 9 月 12 日-2022 年 10 月 14 日没有记录信息，认为商超在连续一个月内未进补货，由于时间较长，涉及类别较多，故选择剔除。

经过 EXCEL 数据筛选，发现经过编号的单品中的 21 号、106 号、151 号、164 号、199 号单品虽然有采购记录，但没有销售，在品类较多的情况下，影响小，故在后续分析中剔除。

4.3 赠送商品的情况

经过 python 的操作，同时挑选出有赠送行为的商品：黄花菜、洪山菜苔、猪肚菇 (盒)、黑皮鸡枞菌 (盒)、襄甜红菜苔 (袋)、双孢菇 (份)、紫螺丝椒、黑牛肝菌、双孢菇 (份)、菌菇火锅套餐 (份)、洪山菜薹珍品手提袋、洪山菜薹莲藕拼装礼盒、白蒿（即 29, 48, 65, 66, 70, 71, 156, 196, 211, 213, 217, 230 号商品）。在后续单品打折力度时，被视作 0。

5 问题一的模型建立与求解

5.1 品类销量的分布规律

5.1.1 描述性统计

在下表中，给出了各品类在 2020 年 7 月 1 日至 2023 年 6 月 30 日销售量的中位数、平均数、范围，并给出了销售量分布的偏度和峰度。由此，可以知道这些品类的大致市场欢迎度，**各品类销售量的分布不具有正态性，峰度较低**。[1]

类别	范围统计	平均值统计	标准差统计	偏度		峰度	
				统计	标准误差	统计	标准误差
花叶类	2912.71	1265.3475	465.1914	0.823	0.194	1.87	0.385
花菜类	667.96	266.1769	116.8368	0.768	0.194	1.186	0.385
水生根茎类	840.46	258.6468	167.3719	0.865	0.194	0.516	0.385
茄类	326.98	146.6805	70.5951	0.488	0.196	-0.244	0.39
辣椒类	1347.78	583.7268	179.4377	1.190	0.194	1.363	0.385
食用菌	1334.07	484.9155	255.0301	1.146	0.194	1.507	0.385

表 2: 2020 年 7 月 1 日至 2023 年 6 月 30 日描述性统计结果

5.1.2 时间序列图分析——以周为统计单位

由于数据时间覆盖范围较广，为了简化模型并更准确地体现销量随时间变化趋势，我们先以各品类在一**周内的销量总和**[2]作为因变量制作时间序列图。其中，**纵坐标的绝对值描述品类销额随时间的变化幅度**，纵坐标为正值时，该品类当前时间段的销售量超过前一周的销售额，反之则低于前一周的销售额。故而在给定的两个时刻点，该品类的**销售额差异表现为以给定时刻作为上下限，差异函数作为被积函数**的定积分。

下图描述了**各品类销售量随时间的变化**

[1]偏度系数大于 0，数据右偏，即数据的尾部偏向右侧（正偏）；偏度系数小于 0，数据左偏，即数据的尾部偏向左侧（负偏）。峰度系数小于 3，表示数据的峰度较低，尖峰较平缓，称为 Platykurtic

[2]利用 Python 拆分附件二，将同一品类在周天内的销量求和。具体代码见附件

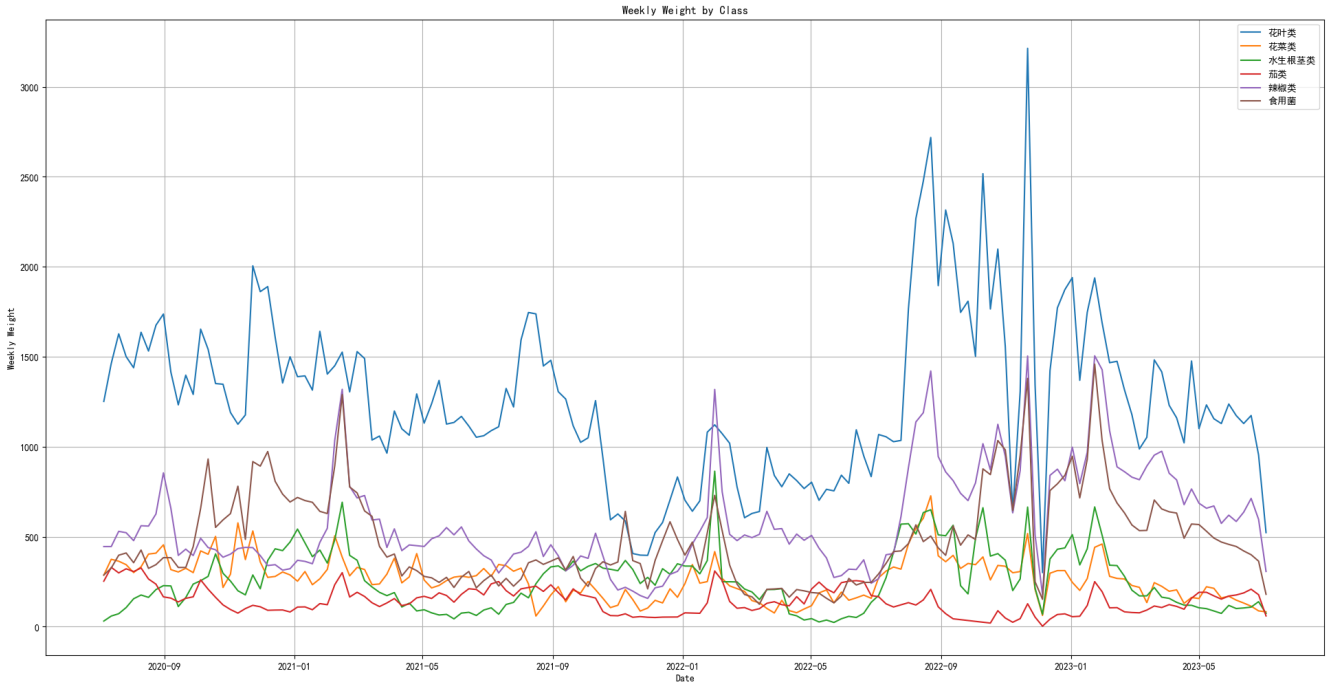
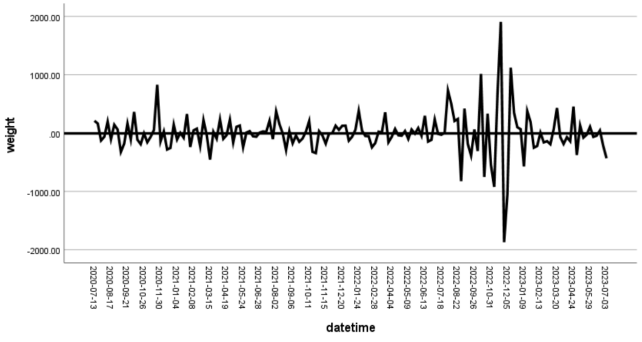
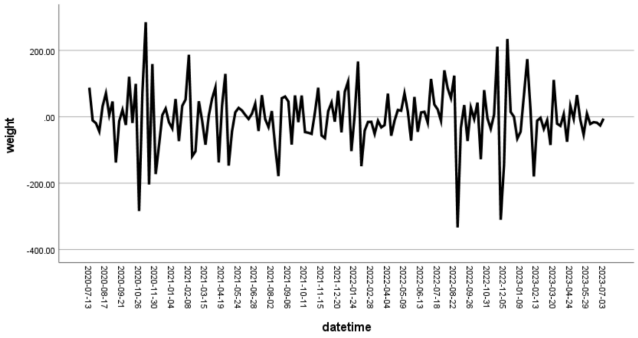


图 1: 各品类销售量随时间的变化——以周销售量为纵坐标

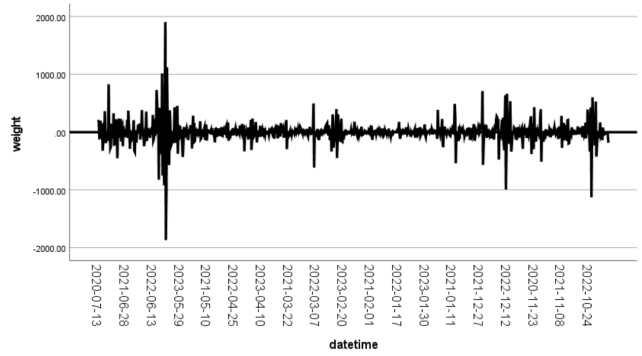
将对各**品类销售量进行一阶差分**的结果作为纵坐标，以时间（以天为单位）作为横坐标，作时间序列图。如图 2:



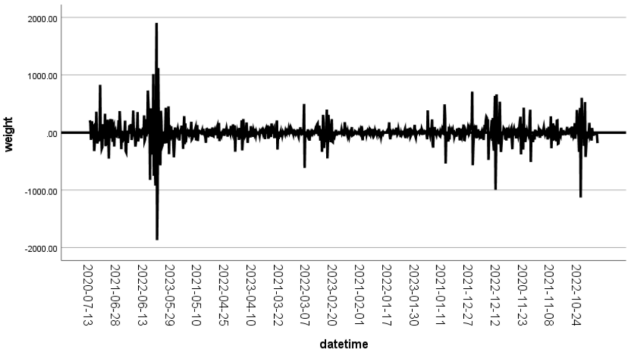
(a) 花叶类



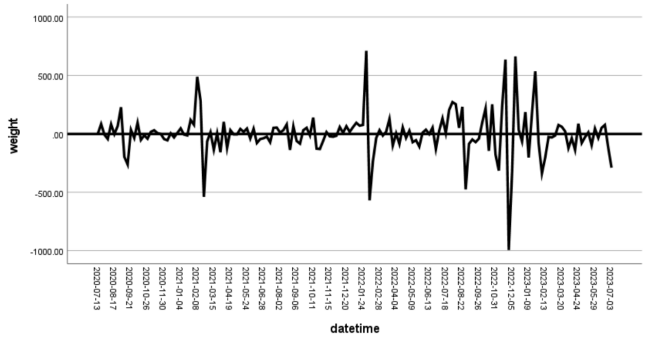
(b) 花菜类



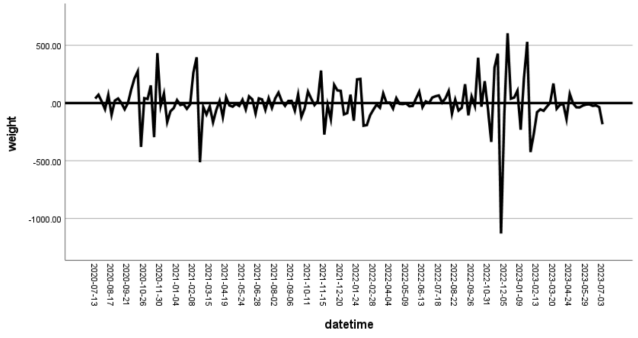
(c) 水生根茎类



(d) 茄类



(e) 辣椒类



(f) 食用菌

图 2: 各品类销售量的**差异值**随时间的变化

结合图 1、图 2 的走向和对时间序列图的分析解释，花菜类的销售量随时间变化呈现出上下波动较为明显，花叶类等另外五个品类的销售量随着时间变化的幅度则相对较小。但在较短一段时间[3]过后，时间段内的总销售量都基本保持平稳。在春节前后，这些品类纵坐标值呈现明显的先上升后下降趋势，品类销售量再急剧增加后急剧减少。考虑到近年疫情等经济社会原因，在春节这些品类的销售量变化也可能相对平缓。

，**在以一周为计时单位的统计情况下，各品类的销售量图像存在不明显的周期性**。故而接下来，**以一天为单位，对时间做任意的 100 天连续抽样，计算自相关系数分析**进一步分析各品类销售量的周期性。

5.1.3 基于自回归系数分析的自相关性检验——以天为单位

以自回归系数为纵坐标，时间（天）为横坐标，分别作出这六个品类的自回归系数图，用于可视化描述各品类销售量的自相关性和周期性。其中，class1-6 的编号分别对应图二中的 a-f 六种品类。由图 3 可知，对于 6 个品类，自回归系数都在抽样的时间滞后上显示出明显的周期性，这通常表明时间序列数据中存在明显的季节性或周期性结构。同时，自回归系数的大小还存在以下意义：

(1): 自回归系数为正，表示时间序列在相邻时刻之间有正相关性，即过去的值对未来的值有正向影响；反之则为负向影响

(2): 自回归系数的绝对值越大，表示时间序列在相邻时刻的相关性越强。

由上述分析，我们可以较为准确地得到**各品类销售量的分布规律**：

(1): 在以一天为统计单位时，各品类的销售量都呈现出显著的周期性。

(2): 随时期间推移，周期基本保持不变，品类销售量与时间的自相关系数呈递减趋势。

5.2 品类销量的相互关系

5.2.1 基于 Spearman 秩相关系数分析各品类销量的相关性

Spearman 秩相关系数（Spearman's Rank Correlation Coefficient）是一种非参数统计方法，用于衡量两个变量之间的相关性，特别适用于数据不满足正态分布或具有秩次数据的情况。与 Pearson 相关系数不同，Spearman 秩相关系数基于变量的秩次而不是原始数据值进行计算。**由各品类销售量的散点图（图 4），可得出各品类销售量不具有线性关系**，故皮尔森 pearson 相关性分析失效；由描述性统计的结果可知，品类销售量也不具有正态性。**综上，我们选用 Spearman 秩相关系数进行相关性分析**。

茄类在 2022 年 9 月 19 日至 2022 年 10 月 10 日时间段中没有采购、销售记录。故而，**此时段中各品类的销售量不参与相关性分析**。

使用以下公式计算 Spearman 秩相关系数。其中，r_s 是 Spearman 秩相关系数，d 是秩次差，n 是样本容量的大小：

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

[3]如：若将计量时间单位由“一周”扩展至“三周”左右，则时间序列分析图中的纵坐标值都将靠近水平直线 0

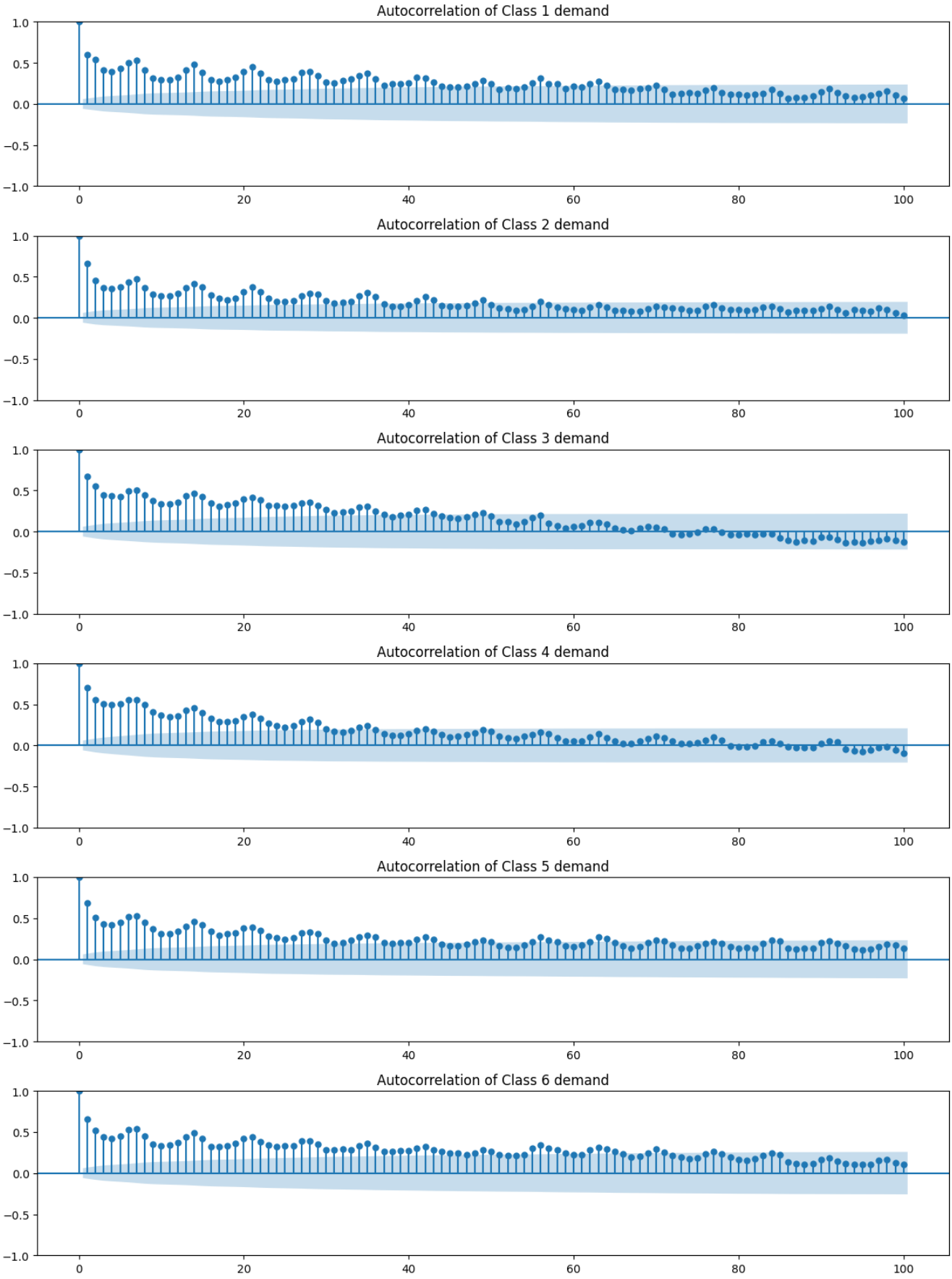

Autocorrelation of Class 1 demand
Autocorrelation of Class 2 demand
Autocorrelation of Class 3 demand
Autocorrelation of Class 4 demand
Autocorrelation of Class 5 demand
Autocorrelation of Class 6 demand
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100

图 3: 自相关系数图

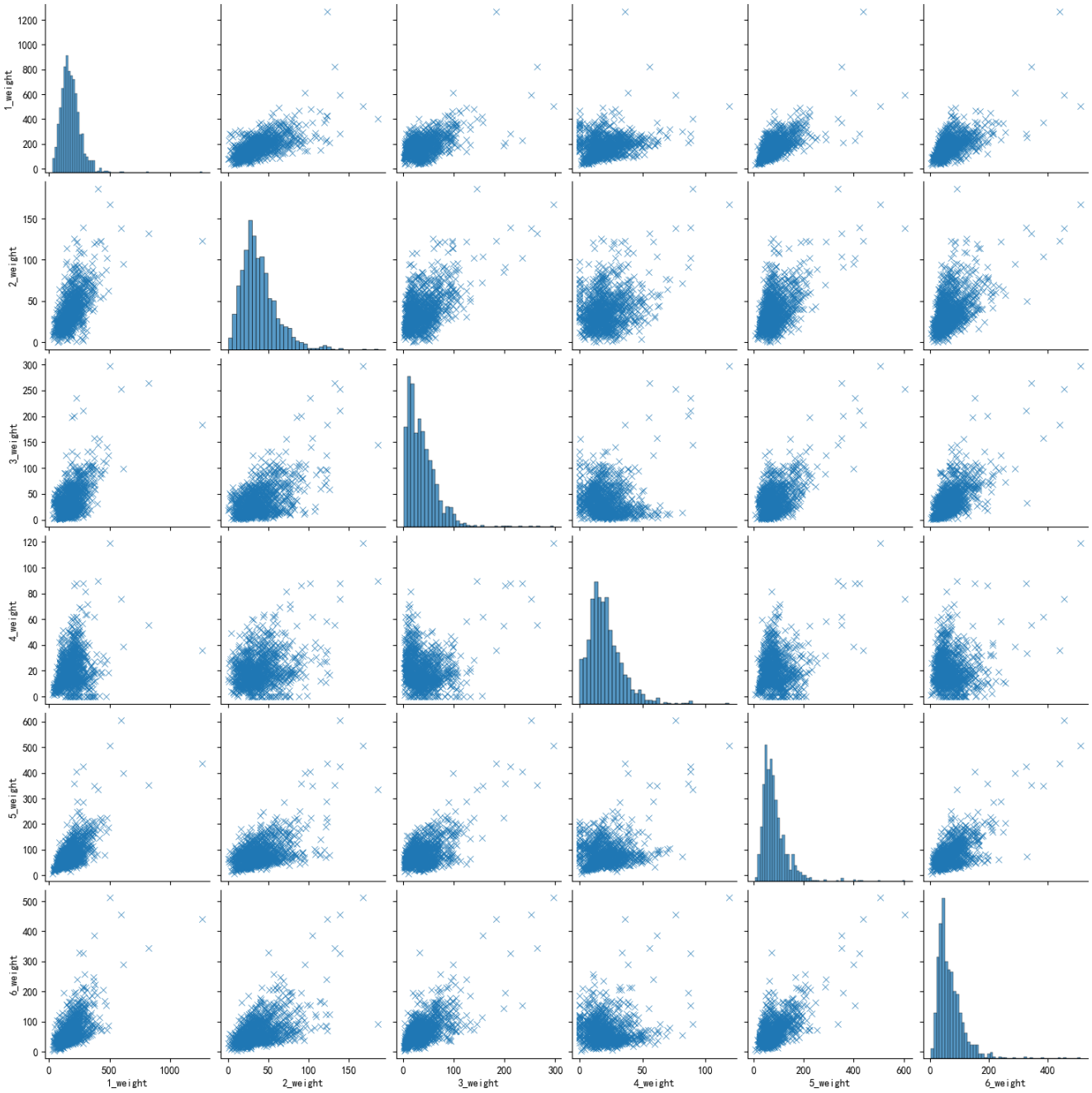


图 4: 六个品类销售量的的散点图

相关性

			花叶类	花菜类	水生根茎类	茄类	辣椒类	食用菌
斯皮尔曼 Rho	花叶类	相关系数	1.000	.661**	.439**	.178*	.534**	.585**
		显著性（双尾）	.	<.001	<.001	.028	<.001	<.001
		N	157	157	157	153	157	157
	花菜类	相关系数	.661**	1.000	.395**	.166*	.358**	.458**
		显著性（双尾）	<.001	.	<.001	.040	<.001	<.001
		N	157	157	157	153	157	157
	水生根茎类	相关系数	.439**	.395**	1.000	-.377**	.290**	.656**
		显著性（双尾）	<.001	<.001	.	<.001	<.001	<.001
		N	157	157	157	153	157	157
	茄类	相关系数	.178*	.166*	-.377**	1.000	.034	-.270**
		显著性（双尾）	.028	.040	<.001	.	.680	<.001
		N	153	153	153	153	153	153
	辣椒类	相关系数	.534**	.358**	.290**	.034	1.000	.490**
		显著性（双尾）	<.001	<.001	<.001	.680	.	<.001
		N	157	157	157	153	157	157
	食用菌	相关系数	.585**	.458**	.656**	-.270**	.490**	1.000
		显著性（双尾）	<.001	<.001	<.001	<.001	<.001	.
		N	157	157	157	153	157	157

**. 在 0.01 级别（双尾），相关性显著。

*. 在 0.05 级别（双尾），相关性显著。

图 5: 各品类之间销售量相关性

r_s 接近于 1 或 -1，则表示变量之间有强烈的秩次相关性，其中 1 表示正相关，-1 表示负相关。

结合上表中相关系数和显著性可知，茄类与花菜类、茄类与辣椒类的销售量存在较为显著的相关性。花叶类与花菜类、水生根茎类与食用菌的销售量都具有较强烈的秩次相关性，但不具有显著性。

5.3 基于抽样分析各单品销量的分布规律

由于数据总量过大，我们先对数据进行抽样处理，再作出样本的销量随时间变化的图表。抽样总量占样本容量的 10% 以上，既减少了图表的复杂程度，也使抽样数据不失一般性和代表性。

仅以**花菜类和辣椒类中的单品为例**作图，并分析花菜类和辣椒类的单品销售量分布规律。下图描述了辣椒类单品和花菜类单品的销售量随时间变化的数值和趋势：

由图 6、图 7 可以看出，花菜类和辣椒类中的大部分单品的销售量随着随着时间变化都呈现出一部分不稳定性，仅有小部分单品的销售量较为稳定，其对应的销售量数值随时间波动幅度较小。同时，花菜类和辣椒类中的大部分单品都具有较为明显的峰值和低谷，具有周期性。

5.4 基于层次聚类法分析单品销量的相互关系

对单品销售量进行聚类分析，寻找单品在销售量上的共同特征。

先提取出 251 个单品每天的销售量总和。在 SPSS 中，利用 “分析-描述性统计-描述” 工具对这些单品的销售量数据做标准化。运用层次聚类 (Hierarchical Clustering) 绘制谱系图，从而反映这些销售量趋势的总体特征。

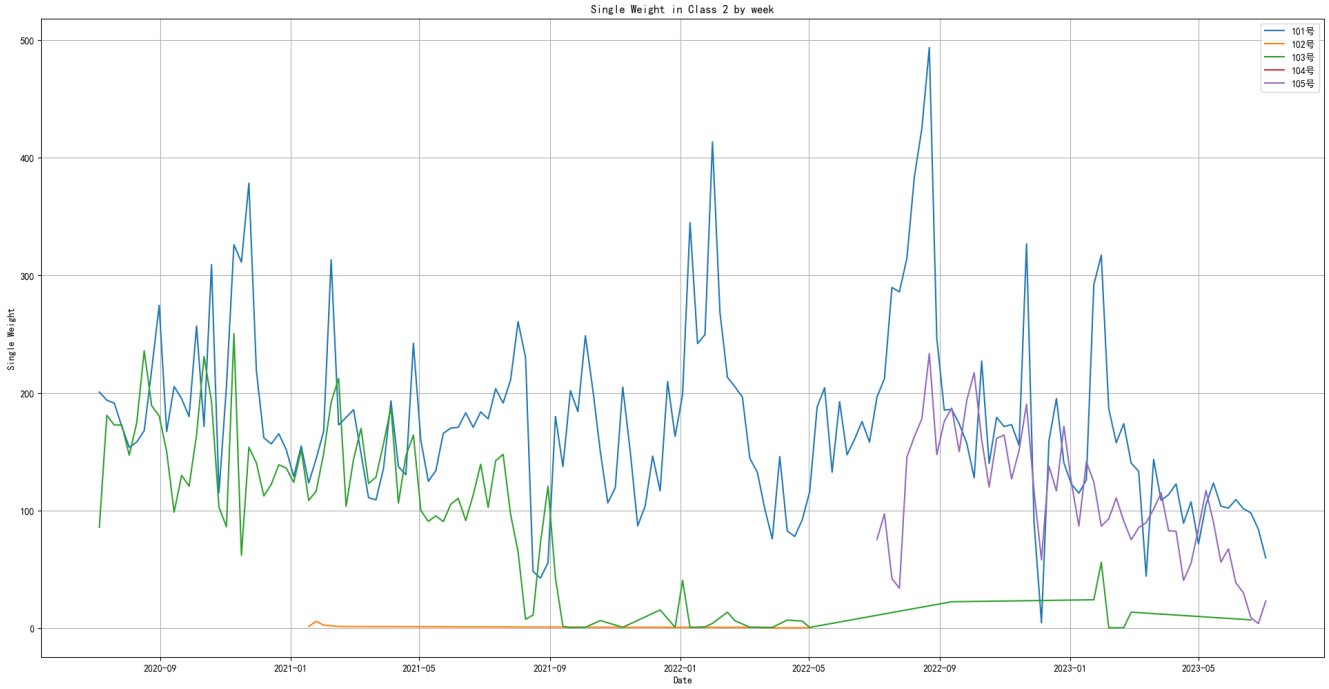


图 6: 花菜类单品销售量随时间变化的图表

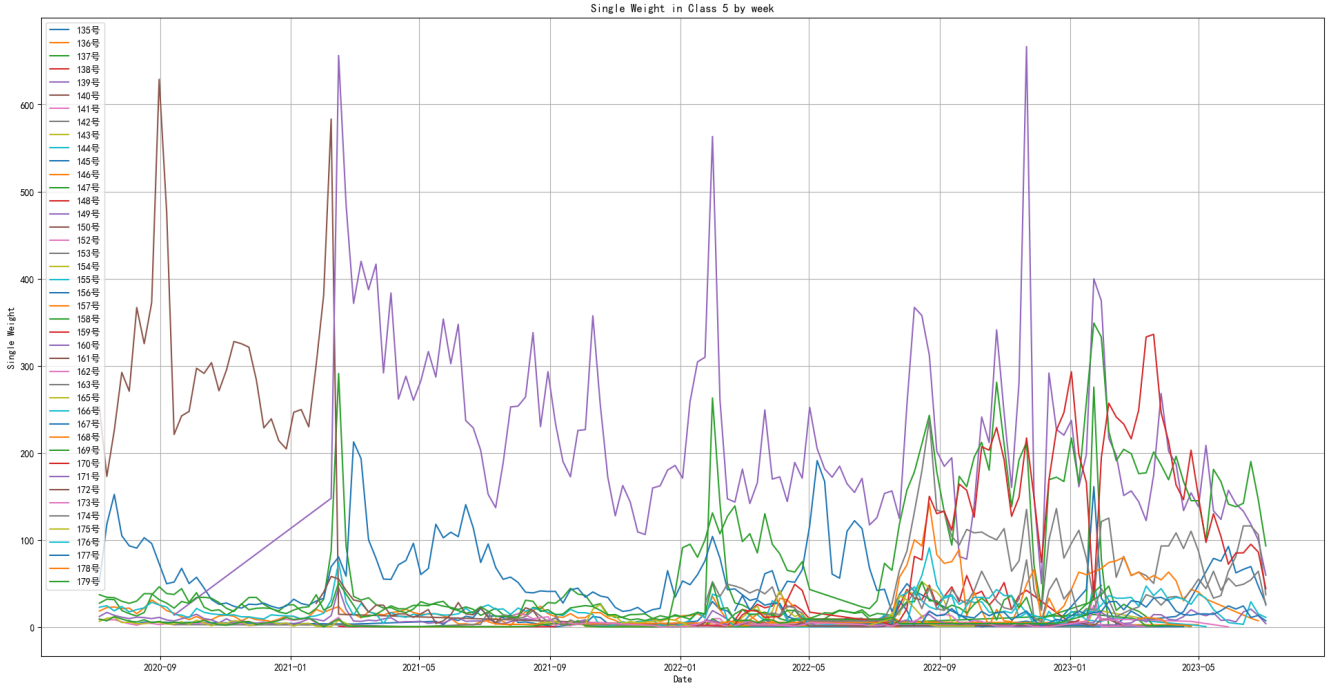


图 7: 辣椒类单品销售量随时间变化的图表

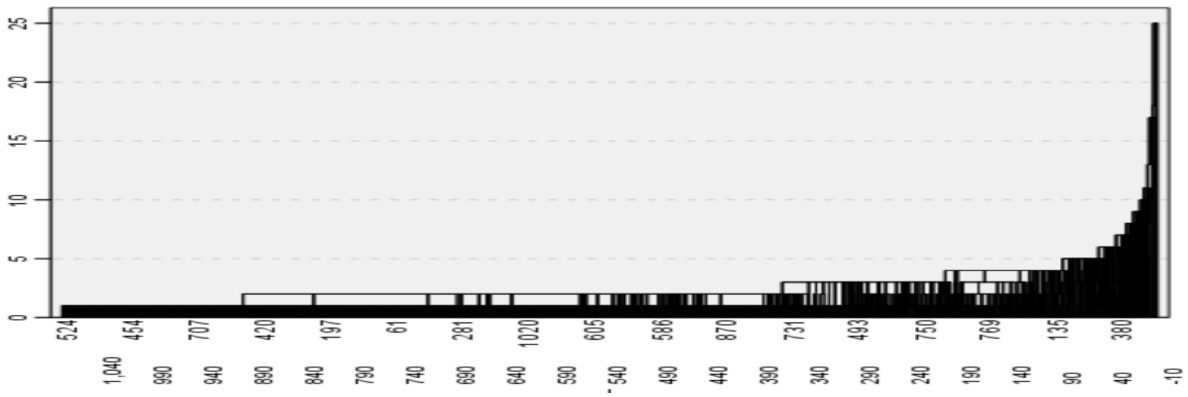


图 8: 单品销售量聚类谱系图

由谱系图（图 8）可知，大部分单品的销售量随时间的变化都不存在明显的的相似之处，只有少部分单品的销售量随时间变化时呈现出相似的规律。单品的销售量分布表现出更多的随机性。

6 问题二的模型建立与求解

6.1 分析销量和成本加成定价的关系

6.1.1 成本加成定价

资料显示，成本加成定价法指产品价格要补偿生产和销售成本并得到合理回报。该理论假设销售者决定价格的主导权，消费者仅能影响加成率。分析成本加成定价的结果（售价）及平均成本和加成率这二价格构成部分，计算三者与销量的 Pearson 相关系数。由于消费者影响加成率是一个双向过程，即商家基于成本和消费者购买情况决定加成率，加成率过高消费者不选择购买，使得商家选择降低加成率，反之亦然。因此在分析销量和加成率时，**因考虑到自相关和滞后性的可能，同时选择 TLCC 进行时间滞后互相关分析**。

6.1.2 热图——Pearson 相关性分析结果

图 9 从上至下，分别为加成率与销量的相关性热图、成本与销量的相关性热图、价格与销量的相关性热图。

一般相关系数在 0.7 以上说明关系非常紧密；[0.4,0.7) 说明关系紧密；[0.2,0.4) 说明关系一般。对于热图分析，可得如下结论：

(1) 三个因素和销量相关性均属于不强相关；

(2) 加成率相关性各异，其中类 4、类 5 加成率与销量正相关;

(3) 成本、价格与销量关系均为负相关，符合市场规律;

(4) 水生根茎类成本、水生根茎类价格与辣椒类销量有相较其他类的较强的关系，对于辣椒类成本、辣椒类价格和水生根茎类销量也有一定的正相关性。猜测二类产品走势相关。

6.2 基于 TLCC 分析销量和加成率的相关性

市场传递价格需要一定时间，影响需求后再影响价格也有一定时间差——时间序列之间存在一定滞后性。时间滞后互相关（TLCC）可以定义两个信号之间的方向性，例如引导-追随关系，在这种关系中，引导信号会

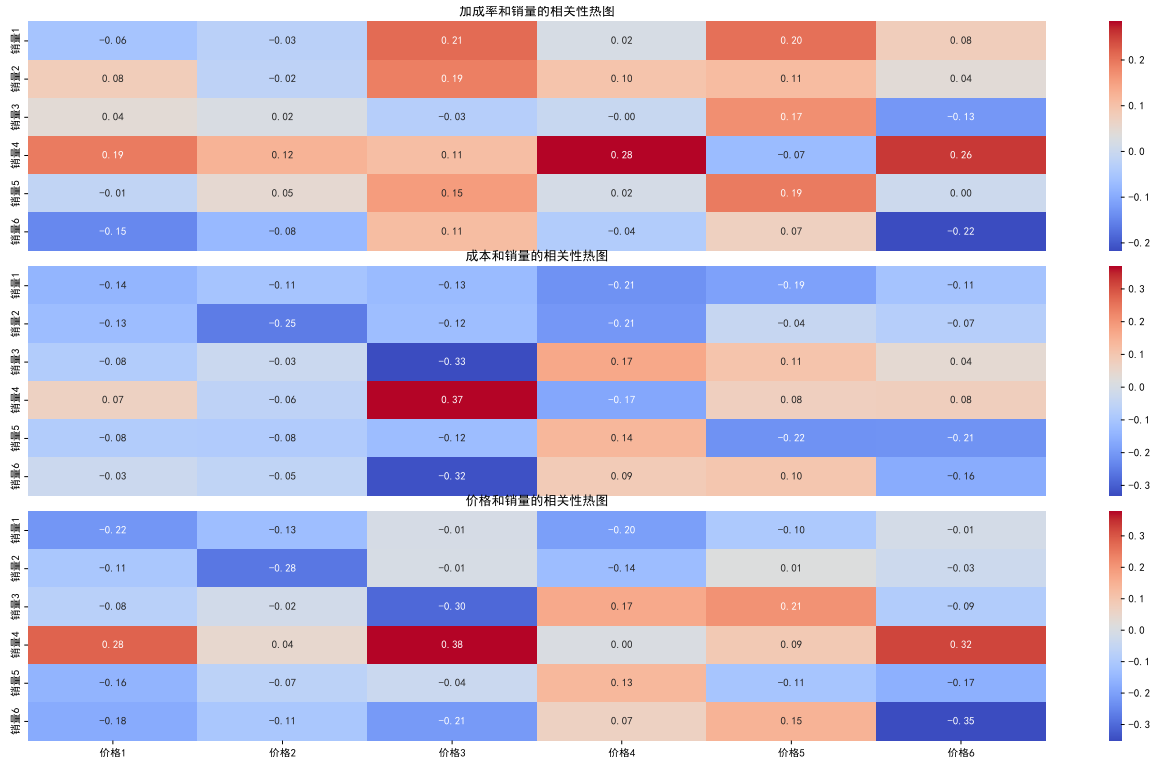


图 9: 不同变量之间的相关性热图

将响应初始化，追随信号则重复它。[4]

	TLCC 滞后	最大 TLCC
1	8033.91452417	1
2	3346.35571439	4
3	2068.58287063	-5
4	2360.38010573	-10
5	9471.61610496	3
6	14345.85039721	4

表 3: 加成率和销量 TLCC 分析

表中最大 TLCC 数均远大于 1，表明 ω 加成率与销量具有较为明显的相关性。也可以发现，水生根茎类和辣椒类的 TCLL 滞后步数均为负值，表示销量变化领先于加成率的变化。二者表现出相同的趋势，也验证热图体现的二者相关趋势。

[4]利用 python 中 Numpy 包的 correalte 函数和 roll 函数选择 2023 年已有数据进行计算，具体程序见附录。

6.3 极坐标变换回归分析

6.3.1 对销售量数据进行反对数化变换

问题一的分析中提到，销售量数据偏离正态分布，且数值均大于 1，体现出左偏分布特征，故在 python 中采用反对数化的处理方式使其减少左偏性。

$$SP_i^{'} = \ln SP_i \tag{2}$$

6.4 极坐标变换下的线性回归

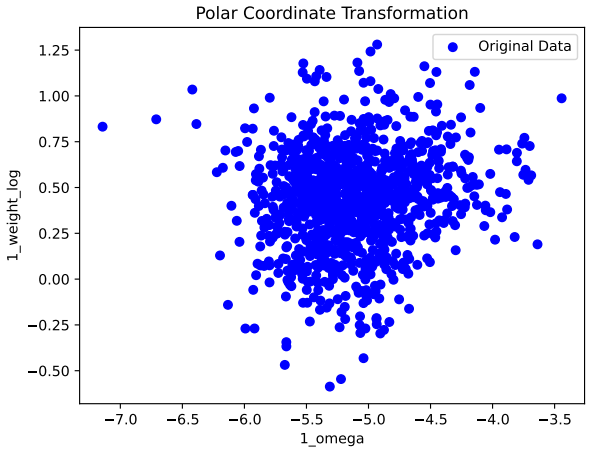
由于数据集的分布为团状，在笛卡尔坐标表示下，难以对数据集进行回归分析，故考虑采用广义坐标——极坐标，对数据进行表示。选择 $\{\theta.r\}$ 作为特征，Ω 作为预测值。进行广义的线性回归。下图为花叶类的预测情况，R^2=0.99187, 均方误差 (MSE): 0.00065, 均方根误差 (RMSE): 0.02552,线性回归系数: [0.10260352, -5.07525507],线性回归截距: 7.443441585515765, 模型回归效果较好。其他品类进行相同操作，也得出较为良好的结果。

极坐标变换公式和线性回归结果

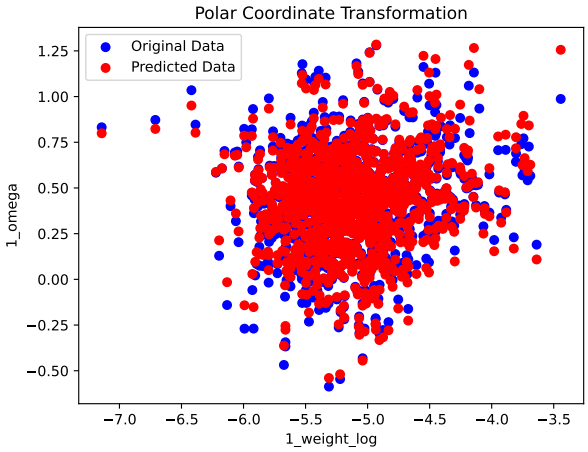
$$\begin{cases} \theta = \arctan \dfrac{SP_i^{'}}{\Omega_i} \\ r = SP_i^{'2} + \Omega_i^2 \end{cases} \tag{3}$$

$$\Omega_i = B \arctan \frac{SP_i^{'}}{\Omega_i} + A\sqrt{SP_i^{'2} + \Omega_i^2} + C \tag{4}$$

Dataset	R^2 Score	MSE	RMSE	Slope 1	Slope 2	Intercept
1	0.9912	0.00064	0.0253	0.1032	-5.0809	7.4491
2	0.6547	0.02836	0.1684	0.2148	-2.1321	2.8321
3	0.6191	0.03696	0.1923	0.0842	-1.6915	2.5225
4	0.5152	0.03281	0.1811	0.1657	-1.2087	1.7382
5	0.9733	0.00299	0.0547	0.1057	-4.1657	6.1020
6	0.9542	0.00719	0.0848	0.2453	-3.7083	4.8742



(a) 销量反对数化数据集



(b) 极坐标为广义坐标线性回归后的品类一的预测集和原始集

6.4.1 总结：各品类销售量和成本生成定价的关系

模型从 Pearson 相关系数、TLCC 滞后相关性分析、以及在极坐标化后的线性回归模型，可发现在同时间比较的 Pearson 相关中，销售量与成本生成定价的关系并不明显，主要关联来自于平均生产成本。而考虑到销售量是反映需求侧的重要指标之一，利用 python 中的 TLCC 方法分析需求侧在成本生成定价中可影响的 Ω 加成率，可发现关系比较明显，能够计算得到 {1,4,-5,-10,3,4} 的滞后天数关系。进一步分析数据集，可发现极坐标变换后，能够进行很好的线性回归，R^2 均大于 0.6，多数达到 0.95 以上。

6.5 收益模型的建立

6.5.1 供应侧 (Supply-side) 的分析

在供应侧中，生鲜商超作为上游供应商的承接者存在，担任提供商超周围生鲜蔬菜的责任。上游供应商提供蔬菜类商品采购价格，在 ADF 检测后，发现序列 1、2、4、6 具有良好平稳性 (p<0.05)，3,5 序列一阶差分后具有良好平稳性。

在此基础上，应用进行**快速傅里叶分析**，分析周期性 (与 TCLL 结果吻合较好, 结果见附件)，后采用 ARIMA、SARIAM 和机器学习算法（随机森林、支撑向量机、长短期学习机和神经网络）进行预测。在实际操作中发现，随机森林等机器学习方法拟合精度普遍优于以 ARIMA 为代表的传统模型 (R^2 与 MAE 等评价)，并且能够更好的捕捉极端变化 (见图 11)，故据此构建供货价矩阵 B，储存预测的 7 月 1 日到 7 月 7 日间 6 类商品采购价格信息。

$$\mathbf{B} = \begin{pmatrix} 3.92796401 & 8.7206494 & 12.408897 & 5.13409722 & 6.72844761 & 3.80936104 \\ 4.04834393 & 8.52775217 & 12.488233 & 4.94504438 & 6.52890667 & 3.73802459 \\ 4.18285916 & 8.47153752 & 12.40216642 & 4.82616765 & 6.87243398 & 3.77243089 \\ 4.20512679 & 8.29263467 & 13.15446222 & 4.84834069 & 6.51101256 & 3.70364068 \\ 4.18331341 & 8.3877922 & 12.45982425 & 4.79643532 & 6.69001952 & 3.82366216 \\ 3.93499153 & 8.28288627 & 12.22803583 & 4.61871145 & 6.87231376 & 3.79368168 \\ 3.73855612 & 8.19721161 & 12.43793402 & 4.3343684 & 6.18831415 & 3.9209787 \end{pmatrix}$$

6.5.2 打折力度分析

通过识别同一单品日内低价获取打折率，计算不同时间段不同品类的平均打折程度 D, 计算公式如下：

$$(Dis)_j = \frac{1}{n}\sum_j \frac{p_o - p_d}{p_o}$$

计算各类单品的平均打折率，计算公式如下：

$$(Dis)_j = \sum w_i dis$$

其中，i 表示日期，j 表示品类内单品序号，n 为当日售卖的某品类 p_0 表示第 i 日 j 单品原价，p_d 表示第 i 日 j 单品折扣价。w_i 表示单品过往平均占总售出的比例。

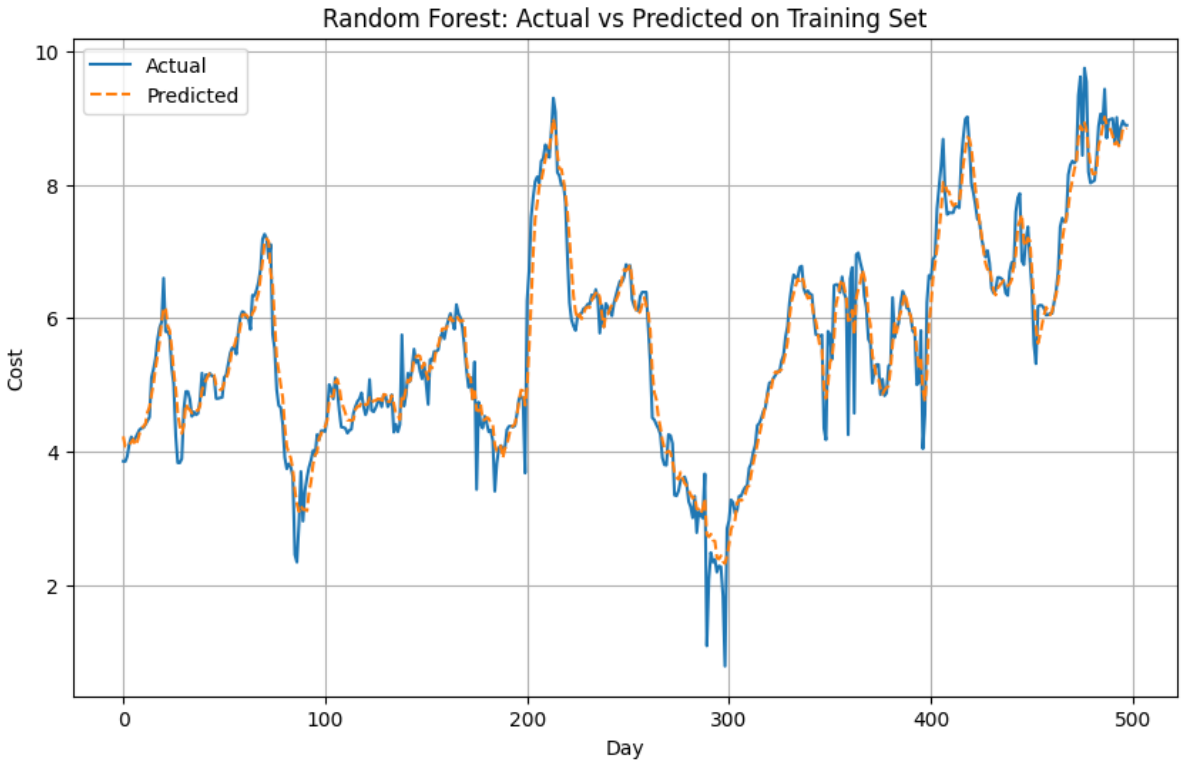


图 11: 第 2 类商品随机森林拟合结果

6.5.3 损耗率分析

打折比例是指同一品类在当日参与打折的销售量在当日该品类总销售量的比例 d，计算公式如下：

$$d_j = \sum w_i s_i$$

其中，j 表示品类序号。选择使用对销量的贡献率加权计算平均损耗率。w_i 表示单品过往平均占总售出的比例，s_i 表示单品的损耗率

6.5.4 平均生产成本分析

平均生产成本是成本加成定价中重要的一个元素。模型考虑针对单品计算平均采购价格，再根据其占其品类的销量比例加权而得。计算公式如下：

$$\bar{C}_j = \sum w_i b_i$$

其中,j 表示品类序号，w_i 表示单品过往平均占总售出的比例，b_i 表示单品过往平均采购价格。

6.5.5 7.6.2-7.6.3 计算结果展示

6.5.6 收益分析

$$Profit = \sum_{i=1}^{7}\sum_{j=1}^{6}(W_{ij}(1-d_j)P_{ij} + d_j W_{i-1,j} D_j P_{ij}) - \sum_{i=1}^{7}\sum_{j=1}^{6} W_{ij} B_{ij}$$

其中，W_{ij} 表示第 i 天，第 j 个品类的补货量;P_{ij} 表示第 i 天，第 j 个品类的定价;B_{ij} 表示第 i 天，第

	C	S 率 (%)	DIS
花叶类	5.590833222	11.19356766	0.229276861
花菜类	5.321991971	5.901571	0.436871983
水生根茎类	5.691733071	9.95251023	0.285928393
茄类	2.442000175	9.382468737	0.434877101
辣椒类	2.951530766	9.423415943	0.258097541
食用菌类	3.240369808	7.368864635	0.209469595

表 4: 7.6.2-7.6.3 计算结果

j 个品类的采购价格。在这个 Profit 函数中，唯一的变量是 W_{ij}, 因为根据问题二第一小问模型提供的回归方程，可以求得对应的类加成率。因此只需对 W 矩阵做优化。在这个函数中，假定 6 月 30 日没有库存残留。经过**遗传算法**的优化选取（具体内容见附录），最终得出预期最大收益为 8922.48 元。

	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
菜叶类	144.69	259.00	267.84	145.23	146.21	144.17	144.92
菜花类	45.53	62.45	66.25	49.67	54.76	50.63	62.57
水生根茎类	15.87	25.46	56.64	29.50	34.88	26.82	37.36
茄类	24.86	42.53	44.50	40.06	39.02	32.91	30.70
辣椒类	49.15	57.09	63.43	53.32	63.49	63.36	42.54
食用菌类	34.61	32.98	69.79	36.26	69.08	33.05	32.01
	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
菜叶类	1.0351	0.4949	0.5034	1.0232	1.0012	1.0460	1.0298
菜花类	0.7081	0.7893	0.8041	0.7309	0.7560	0.7359	0.7898
水生根茎类	0.2544	0.2904	0.3549	0.3021	0.3155	0.2945	0.3211
茄类	0.6073	0.6921	0.6994	0.6825	0.6783	0.6513	0.6403
辣椒类	0.4181	0.4395	0.5594	0.4287	0.5530	0.5661	0.4013
食用菌类	1.3575	1.4420	0.7963	1.2662	0.7885	1.4385	1.4889
	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
菜叶类	7.9938	6.0519	6.2885	8.5078	8.3716	8.0510	7.5885
菜花类	14.8957	15.2587	15.2835	14.3537	14.7290	14.3783	14.6714
水生根茎类	15.5657	16.1148	16.8037	17.1284	16.3909	15.8292	16.4318
茄类	8.2520	8.3675	8.2016	8.1573	8.0499	7.6269	7.1097
辣椒类	9.5416	9.3984	10.7169	9.3023	10.3896	10.7627	8.6717
食用菌类	8.9806	9.1283	6.7764	8.3932	6.8386	9.2509	9.7589

表 6: 补货矩阵、加成率矩阵、定价矩阵

7 问题三的模型建立与求解

查阅经济学资料得到：在给定的市场需求情况下，当价格为 $P_i = \mathrm{C}\ (1 + \Omega_i)$ 时，消费者购买意愿最强，商超盈利可以达到最大化。其中 C 为进货价，Ω 是与市场需求相关的复杂变量，两者存在相互关系。

在问题二的定价策略中，我们考虑了 Ω 与市场需求的相互作用；在问题三中对单品进行定价，由于单品种类较多已经大大提高了问题复杂度；故而对问题二中的定价策略进行化简：即**不再考虑 Ω 对市场需求的反作用**，从以往销售量中判断市场需求，并用预测 7 月 1 日的市场需求情况，从而给出相应 Ω 的值和价格 P_i。于假设中已知滞销率极低，对单品补货量和定价策略的影响较小，故而我们在时间序列模型中忽略滞销率的影响。由于单品种类较多，遍历各个单品较为复杂，故而需要先建立评价模型，在 251 个可售单品中筛选出使商超收益更大的 27-30 个单品进行补货。

在建立评价模型之前，需要对数据做筛选：题目要求**对 6 月 24 日-30 日的可售品种做分析，故而在已有数据中先剔除在这段时间内无销售记录的单品、和销售记录少（非零记录低于两天）且销售量少的单品**。这一定程度上减少了单品销售量数据中的噪声部分，使余下的数据具有可预测的规律性。

7.0.1 基于熵权法的 TOPSIS 综合评价

设事件 X 可能发生的情况分别为:$x_1, x_2, ...x_n$,$p(x_i)$ 为该事件发生的概率。定义事件 X 的信息熵为:

$$H(X) = \sum_{i=1}^{n}[p(x_i)I(x_i)] = -\sum_{i=1}^{n}[p(x_i)\ln p(x_i)]$$

可以看出，信息熵的本质就是对信息量的期望值。

为了方便, 我们可以将这个离散函数当作连续函数 $-xlnx$ 讨论.[5]易知

$$-\lim_{x\to 0} xlnx = 0,$$

将其求导可得到

$$-\ln x - 1$$

再次求导可得到

$$-\frac{1}{x}$$

由此可知，这个函数有一个极大值点 $x = \frac{1}{e}$ 且二阶导不为 0, 故极值点就是最值点, 且唯一。那么类似的,

$$H(X)\text{有最大值} \quad iff \quad p(x_1) = p(x_2) = ... = p(x_n) = -\frac{1}{n}$$

$H(x)$ 取到最大值, 此时 $H(X) = lnx$。

下图给出计算熵权的公式：

令信息熵为

$$e_j = -\frac{1}{\ln n}\sum_{i=1}^{n} p_{ij}\ln(p_{ij})\ (j = 1, 2, \cdots, m)$$

把结果规范到 $[0, 1]$ 之间. 信息熵越大 $\rightarrow$ 指标可靠性越差 $\rightarrow$ 信息有效性越小. 信息效用值: $d_j = 1 - e_j$ 再将信息效用值归一化就是熵权:

$$W_j = d_j / \sum_{j=1}^{m} d_j (j = 1, 2, \cdots, m)$$

在 2023 年 6 月 24 日到 6 月 30 日中，选择单品的日销量、因运损和品相变差的折损率、售价、采购价格及单品打折后的折扣率 5 项指标作为权值，代入上述公式确定各权值的权重。加权结果如下图：

利用图 12 中的熵权值进行 TOPSIS 评价，得到 33 个最优单品如下：

[5]严格证明见《数学建模算法与程序》一书

索引	正理想解距离 (D+)	负理想距离 (D-)	综合得分指数	排序
西兰花	0.35386	0.87008	0.71089	1
竹叶菜	0.49449	0.54456	0.52409	2
菠菜	0.53315	0.50672	0.48729	3
红薯尖	0.54817	0.47091	0.46209	4
净藕 (1)	0.86911	0.4516	0.34194	5
杏鲍菇 (1)	0.75573	0.37213	0.32994	6
青线椒	0.84617	0.38405	0.31218	7
高瓜 (1)	0.72332	0.31472	0.30319	8
圆茄子 (2)	0.75721	0.32693	0.30155	9
云南生菜 (份)	0.86703	0.36255	0.29485	10
菜心	0.84326	0.34063	0.28772	11
奶白菜	0.85921	0.34289	0.28524	12
娃娃菜	0.80241	0.31575	0.28238	13
红尖椒	0.86325	0.32825	0.27549	14
双孢菇 (盒)	0.81052	0.30727	0.27489	15
木耳菜	0.78006	0.2938	0.27359	16
青茄子 (1)	0.87406	0.32621	0.27178	17
长线茄	0.91569	0.33952	0.27049	18
外地茼蒿	0.82566	0.2973	0.26475	19
小青菜 (1)	0.77875	0.27965	0.26422	20
洪湖藕带	0.882	0.30708	0.25825	21
红椒 (1)	0.9216	0.3193	0.25731	22
云南生菜	0.78836	0.27164	0.25626	23
菠菜 (份)	0.87127	0.30004	0.25616	24
七彩椒 (2)	0.86929	0.29719	0.25477	25
小米椒 (份)	0.90029	0.30752	0.25461	26
紫茄子 (2)	0.88105	0.30058	0.25438	27
平菇	0.88801	0.29872	0.25172	28
云南油麦菜	0.87237	0.2914	0.2504	29
上海青	0.8919	0.29677	0.24966	30
大白菜	0.85626	0.2842	0.24919	31
苋菜	0.89473	0.29437	0.24756	32
螺丝椒	0.89896	0.29379	0.24632	33

表 7: 基于熵权法—TOPSIS 评价方法选出的 33 个最优单品

熵权法			
项	信息熵值e	信息效用值d	权重(%)
Q3-折扣	0.981	0.019	4.578
Q8-平均售价	0.928	0.072	17.097
Q6-6月24日至30日平均销量	0.712	0.288	68.604
Q4-损耗率(%)	0.982	0.018	4.4
Q7-平均采购价	0.978	0.022	5.321

图 12: 基于熵权法求得的各权值比重

7.1 单品补货量策略

在表 5 给出的 33 个最优单品中选取补货单品，假设运损或品相变差后仍能打折出售。图 2 是对这些单品近半年的销售量做一阶差分后所作的时间序列图，从中可以得出各品类销售量随时间变化的大致分布规律。首先剔除数据中的异常值。对于 2023 年 6 月 24 日到 2023 年 6 月 30 日间 7 天连续销售的商品，使用 LSTM (Long Short-Term Memory)，预测出 2023 年 7 月 1 日与 6 月 30 日的一阶差分，从而预测出 7 月 1 日的单品销售量；对于近 7 天内销售数据有缺失但少于 3 天的，使用线性插值后对一阶差分进行回归；而对于 7 天内有效数据不足 4 天的，取历史同时期 (剔除异常值) 的平均值作为预测销售数据。

这样做的合理性: 首先，单一商品相较于其所在大类，更容易受到市场需求波动因素影响，滞后性不强，且滞销后直接影响自身第二天的销售量，故绝大多数单品有 1 天为周期的自相关性 (符合检验结果与直观认识)。其次，7 天周期较短，线性模型已经可以较好的反应变化趋势。最后，同一商品有着较强的年际周期性。

7.2 单品定价策略

对于 7 月 1 日单天，简化利润公式

$$Profit = \sum_{j=1}^{33}(W_j(1-d_j)P_j) - \sum_{j=1}^{33} W_j B_j$$

当 W_i 固定后，仅有单品价格 P_i 为变量。其中，我们在问题二中分析出了品类价格加成率 Ω_i 与品类销售量的关系:

$$\Omega_i = A\arctan\frac{SP_i^{'}}{\Omega_i} + B\sqrt{SP_i^{'2} + \Omega_i^2} + C$$

同样的，对于单品 ω_i，也符合品类加成率与销量的关系。故仅需满足约束关系

$$\begin{cases} 27 \le n \le 33 \\ 0 \le SN_i \le W_i \\ W_i \ge 2.5 \end{cases}$$

使用遗传算法优化 W_i 后得出最大收益 931.65 元。

8 问题四的探索

在进一步优化蔬菜类商品的补货策略时，可考虑更多维度的数据来源。首先，从**产地信息**出发，通过深入了解各产地的气候、土壤和农业技术来确保补货的蔬菜质量更加上乘。

除此之外，还可通过收集和分析蔬菜的**主要有效营养价值数据**，分析购买情况和蔬菜主要有效营养价值和销售量变换的关系，以便补充更受消费者喜爱的蔬菜。

物流是补货策略的重要参考因素，运输成本和运输损失直接影响蔬菜采购成本，运输时间影响蔬菜交易的流转率，从而影响**自动定价**的过程。因此，需要收集与物流有关的各种数据，包括**运输价格**、**运输时间**、**运输条件**和**运输损坏率**。这可以帮助商超找到更经济、更环保的物流方案，并**实时调整补货策略**，以确保商品的新鲜和减少损耗。

再者，可研究和对比**同行的业务模式和补货策略**，从中找出模型不足，并在此基础上进行创新和改进补货策略与定价模式。

综合来看，通过整合和利用多元化的数据，能从蔬菜商品成本、蔬菜商品属性市场动向、流转性、商业模式商超可以更好地理解市场动态和消费者需求，从而更加精准和高效地进行补货决策，确保商品的新鲜和高品质。

9 模型评价

9.1 优点与创新

· 使用多种方法包括遗传算法、时间序列分析（ARIMA）等对补货和自动定价策略进行预测
· 分析能够充分使用三年市场需求、供应的数据信息，能够看出特殊事件的作用
· 在数据处理时，对坐标进行极坐标变换，在极坐标空间中较好地完成了回归任务
· 利用人工智能对模型的建立提供协助

9.2 不足之处

· 求解最大收益时，未对初始滞销情况进行考虑，较为粗糙认定滞销量极小
· 使用遗传算法优化收益函数时，参数的调整可能并非最优
· 未考虑突发事件的影响
· 未对打折情况进行分类，较为粗糙地直接采用平均值作为预测值
· 构建问题三模型时，为简化模型，仅考虑卖方市场作用，未考虑折损率对市场需求的影响

9.3 未来预期与改进

· 获取问题四中提及的更多参量，丰富模型维度
· 收集更多有关滞销和市场边际效益的资料，使模型预测情况更加真实

A 模板所用的宏包

模板中已经加载的宏包				
graphicx	ctex	floatrow	amsmath	fontspec
amssymb	float	appendix	array	atbegshi
booktabs	muttirow	booktabs	subcaption	caption

表 8: 已加载的宏包

B python

TCLL 分析所用的代码（问题二中用到）

```
import numpy as np
import pandas as pd

df = pd.read_csv('4-merged_data.csv')
timestamps = df['date'].to_numpy()
data1 = df['1_weight'].iloc[907:1086]
data2 = df['1_cost'].iloc[907:1086]

lag_range = range(-10, 11)  # 滞后范围从 -10到 10
tlcc_values = []
for lag in lag_range:
    shifted_data1 = np.roll(data1, lag)
    tlcc = np.correlate(shifted_data1, data2, 'valid')
    tlcc_values.append(tlcc)

max_tlcc = max(tlcc_values)
max_lag = lag_range[tlcc_values.index(max_tlcc)]
print("最大TLCC:", max_tlcc)
print("最大TLCC对应的滞后:", max_lag)
```

图 9，即热图的生成代码

```
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus']=False
```

```
# 第一组数据
data1 = [
    [-0.055440874, -0.02707306, 0.21097525, 0.019871725, 0.203619687, 0.076904366],
    [0.078332178, -0.020615462, 0.188835296, 0.096194954, 0.110737346, 0.041830121],
    [0.043076222, 0.023684187, -0.03289575, -0.004599736, 0.170648119, -0.126436231],
    [0.191825682, 0.124472128, 0.106814736, 0.283441416, -0.072349543, 0.255079121],
    [-0.01472695, 0.049106618, 0.154806388, 0.017174707, 0.191602637, 0.00185992],
    [-0.146731898, -0.076102276, 0.108724695, -0.039319719, 0.07164913, -0.217606253]
]

# 第二组数据
data2 = [
    [-0.144307617, -0.106109761, -0.131830928, -0.214655552, -0.185252093, -0.108931434]
    [-0.125327119, -0.252499645, -0.122932103, -0.205243265, -0.036998978, -0.073887201]
    [-0.081327021, -0.028555584, -0.331591013, 0.166312454, 0.106695566, 0.038331731],
    [0.06937923, -0.056280612, 0.368139172, -0.174753204, 0.075195962, 0.078085538],
    [-0.080363879, -0.082992104, -0.124548519, 0.137844146, -0.215655306, -0.21329288],
    [-0.025716487, -0.051095625, -0.320859375, 0.088319235, 0.101672598, -0.163565418]
]

# 第三组数据
data3 = [
    [-0.222758948, -0.133134185, -0.008797761, -0.203045064, -0.101819673, -0.014863676]
    [-0.106767348, -0.279217159, -0.005632197, -0.136806181, 0.012934678, -0.031904181],
    [-0.075312205, -0.019643971, -0.304762664, 0.165504773, 0.208511712, -0.091848746],
    [0.281985465, 0.043992543, 0.377827318, 0.004699597, 0.091465313, 0.320923922],
    [-0.158849581, -0.066863402, -0.037738631, 0.133541369, -0.114064099, -0.167953884],
    [-0.179790382, -0.108540035, -0.2124729, 0.066868203, 0.146422245, -0.350904005]
]

# 横坐标标签

y_labels = ["销量1", "销量2", "销量3", "销量4", "销量5", "销量6"]

# 创建3x1的子图
plt.figure(figsize=(12, 18))

# 子图1
x_labels = ["加成率1", "加成率2", "加成率3", "加成率4", "加成率5", "加成率6"]
plt.subplot(3,1,1)
```

```
sns.heatmap(data1, annot=True, fmt=".2f", yticklabels=y_labels, xticklabels=[], cmap="coo
plt.title("加成率和销量的相关性热图")

# 子图2
x_labels = ["成本1", "成本2", "成本3", "成本4", "成本5", "成本6"]
plt.subplot(3,1,2)
sns.heatmap(data2, annot=True, fmt=".2f", yticklabels=y_labels, xticklabels=[], cmap="coo
plt.title("成本和销量的相关性热图")

# 子图3
x_labels = ["价格1", "价格2", "价格3", "价格4", "价格5", "价格6"]
plt.subplot(3,1,3)
sns.heatmap(data3, annot=True, fmt=".2f", yticklabels=y_labels, xticklabels=x_labels, cm
plt.title("价格和销量的相关性热图")

# 显示子图
plt.tight_layout()
plt.show()
```

创建一个从 1 到 251 的列表, 分别代表 251 种不同单品, 来存放提供的数据

```
import numpy as np
# 创建一个从1到251的列表
sequence = list(range(1, 252))

# 创建一个列表来存放你提供的数据
data = [
    (0, np.nan)
    (1, 0.4410609766215462)
    (2, np.nan)
    (3, np.nan)
    (4, 0.25224965550891254)
    (5, 0.4171410897177343)
    (6, 0.17456532599624813)
    (7, 0.20536436224448892)
    (8, np.nan)
    (9, 0.12687135218385379)
    (10, np.nan)
    (11, 0.21382268452785624)
    (12, 0.5631919896593734)
    (13, 0.45922215683301676)
    (14, 0.22039195685075616)
    (15, 0.2899905116939662)
```

(16, 0.35150879579621386)
(17, np.nan)
(18, 0.3606449559822136)
(19, 0.7090809485909563)
(20, 0.8418967797501757)
(21, np.nan)
(22, 0.4701890450799806)
(23, 0.14932504470512603)
(24, 0.21493143710565554)
(25, 0.29617268677206593)
(26, 0.34810668958172075)
(27, 0)
(28, 0.1657798643714136)
(29, 0.094384143463898)
(30, np.nan)
(31, 0.21527740572359716)
(32, 0.14560073044412872)
(33, np.nan)
(34, 0.6506011196276683)
(35, 0.6916693849073183)
(36, 0.4754273504273504)
(37, np.nan)
(38, 0.39031776013484915)
(39, 0.4664259311449861)
(40, 0.8740650899535072)
(41, 0.788537256653298)
(42, 0.2993417156177593)
(43, 0.7260253534282707)
(44, np.nan)
(45, np.nan)
(46, 0)
(47, np.nan)
(48, np.nan)
(49, 0.45750211250526873)
(50, np.nan)
(51, np.nan)
(52, np.nan)
(53, np.nan)
(54, np.nan)
(55, 0.3861651169343477)
(56, 0.21065505309716404)

```
(57, 0.31947139147972003)
(58, 0.3131176631176631)
(59, 0.30787241942036697)
(60, 0.43779400457180884)
(61, 0.42116951420033344)
(62, 0.3083333333333333)
(63, 0)
(64, 0)
(65, 0.5826164874551971)
(66, 0.9285714285714286)
(67, 0.7666666666666667)
(68, 0)
(69, 0)
(70, 0.44684355502269707)
(71, 0.3054766226077754)
(72, np.nan)
(73, np.nan)
(74, 0.38094583965538703)
(75, 0.6095238095238096)
(76, np.nan)
(77, np.nan)
(78, np.nan)
(79, 0.09593203229563715)
(80, 0.2408354266388548)
(81, 0.4863763060125289)
(82, 0.3911574957029502)
(83, 0.23600502764430709)
(84, 0.23558612317117825)
(85, np.nan)
(86, np.nan)
(87, np.nan)
(88, 0.49291125541125536)
(89, 0.48801912069749526)
(90, 0.48610969730199444)
(91, np.nan)
(92, 0.9090909090909091)
(93, np.nan)
(94, np.nan)
(95, 0.3656612995738053)
(96, 0.2955974842767295)
(97, np.nan)
```

(98, 0.328125)
(99, 0.44084934992217745)
(100, np.nan)
(101, 0.395038160980001)
(102, np.nan)
(103, 0.21980354605809704)
(104, 0.4308310087964168)
(105, 0.3476357190548168)
(106, 0.475)
(107, 0.6422282489994832)
(108, np.nan)
(109, 0.7257501246415683)
(110, 0.4710490642410308)
(111, np.nan)
(112, 0.3222957668485273)
(113, 0.1004197271773347)
(114, np.nan)
(115, np.nan)
(116, 0.4715159249779485)
(117, 0.6750000000000002)
(118, 0.2608506709904194)
(119, np.nan)
(120, 0.300497290039814)
(121, np.nan)
(122, 0.510119839257211)
(123, 0.13532017965732826)
(124, np.nan)
(125, 0.09950944327216586)
(126, np.nan)
(127, 0.043120240835842)
(128, 0.6294103943262969)
(129, 0.14277306714608662)
(130, np.nan)
(131, 0.4634289057928613)
(132, 0.1184268669907202)
(133, 0.1612154104463532)
(134, 0.1969313945237)
(135, np.nan)
(136, 0.2848490225427522)
(137, 0.173723363257067)
(138, 0.49896078680773376)

(139, np.nan)
(140, 0.47529032135205374)
(141, 0.06919470353900746)
(142, 0.31553896767442113)
(143, np.nan)
(144, 0.23404420881552004)
(145, np.nan)
(146, 0.47347918627279206)
(147, 0.8)
(148, np.nan)
(149, np.nan)
(150, np.nan)
(151, np.nan)
(152, 0)
(153, 0.3589896823722681)
(154, 0.18798940990333365)
(155, 0.4827021996203031)
(156, 0.5)
(157, 0.3333333333333333)
(158, np.nan)
(159, 0.30016498171944833)
(160, 0.477089843916767)
(161, 0.5213050997533756)
(162, 0.51650958994709)
(163, 0.3837208342469836)
(164, 0.5771726700971983)
(165, 0.2928594687814616)
(166, 0.49312084594846556)
(167, 0.6339445422350015)
(168, 0.46133587921881597)
(169, 0.4075937816467572)
(170, 0.7)
(171, 0.3703001274324803)
(172, 0.2788228959500984)
(173, 0.5416666666666666)
(174, 0.08086892450911183)
(175, 0.2780269507080115)
(176, 0.6350596028000975)
(177, 0.08585634642938342)
(178, 0.7711602943104523)
(179, 0.14612227144312642)

```
(180, np.nan)
(181, 0.3586386850309315)
(182, 0.5833333333333334)
(183, 0.17968864995184408)
(184, 0.15051769505549564)
(185, 0.5104938271604939)
(186, 0.3390151515151515)
(187, 0.5816666666666667)
(188, np.nan)
(189, np.nan)
(190, np.nan)
(191, 0)
(192, 0.6527777777777778)
(193, np.nan)
(194, 0.7916666666666666)
(195, 0.2904564315352697)
(196, 0.6)
(197, 0.44801208858504)
(198, np.nan)
(199, np.nan)
(200, np.nan)
(201, np.nan)
(202, np.nan)
(203, np.nan)
(204, 0.9242424242424242)
(205, 0)
(206, np.nan)
(207, 0)
(208, 0.45760990204509044)
(209, 0.5924866818591517)
(210, 0.5113095238095238)
(211, 0)
(212, np.nan)
(213, 0.49376305346700083)
(214, 0.5717144805992844)
(215, 0.2848278934001992)
(216, 0.6612622663640053)
(217, 0.544201042064129)
(218, 0.7895344455091705)
(219, 0.37553561801044016)
(220, 0.3742327128697888)
```

```
    (221, 0.4937388193202146)
    (222, 0.292502585159909)
    (223, np.nan)
    (224, 0)
    (225, 0.6646103896103897)
    (226, 0.24781297134238311)
    (227, 0.6888888888888888)
    (228, np.nan)
    (229, 0.4751678332767608)
    (230, np.nan)
    (231, 0.2257025369138372)
    (232, np.nan)
    (233, np.nan)
    (234, 0.11041666666666666)
    (235, 0.22636336476712599)
    (236, 0.18927489270803413)
    (237, 0.4375)
    (238, np.nan)
    (239, 0.25)
    (240, 0.29166666666666663)
    (241, 0.3246134817563388)
    (242, 0.3615188775536033)
    (243, 0.3307541807510598)
    (244, np.nan)
    (245, 0.4)
]

# 根据你提供的数据更新序列
for index, value in data:
    sequence[index] = value

# 打印序列来验证结果
for index, value in enumerate(sequence):
    print(f"({index}, {value})")

#求解销量和加成率值
from scipy.optimize import root
import numpy as np
# 定义数据集参数
params = [
    {"slope": [0.10315267, -5.08087583], "intercept": 7.449133700824736},
    {"slope": [0.21480943, -2.13213308], "intercept": 2.832090782075916},
```

```
    {"slope": [0.08416178, -1.69146344], "intercept": 2.5224794110391766},
    {"slope": [0.1657369, -1.20873817], "intercept": 1.7381574732280578},
    {"slope": [0.10571773, -4.16573431], "intercept": 6.102025043977348},
    {"slope": [0.24527483, -3.7082868], "intercept": 4.874226986003713},
]
# 定义函数来计算Omega_i


def calc_Omega_i(Omega_i, slope, intercept, SP_i_prime):
    A, B = slope
    C = intercept
    return Omega_i - (B * np.arctan(SP_i_prime / Omega_i) + A * np.sqrt(SP_i_prime**2 +


# 为每个数据集找到Omega_i的解
solutions = []
for param in params:
    slope, intercept = param['slope'], param['intercept']

    # 使用root函数找到解,
    sol = root(calc_Omega_i, [1.0], args=(slope, intercept, 30))
    solutions.append(sol.x[0])

# 输出解决方案
print(solutions)

#对极坐标变换后的数据进行线性拟合
#对不同数据集运行,只需要改变运行列名

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# 读取数据
df = pd.read_csv(
    r'4-merged_data.csv')
df = df[(df['4_omega'] != 0) & (df['4_weight'] != 0)]
# 对 '1_weight' 列进行对数变换
df['1_weight_log'] = np.log(df['1_weight'])
```

```
# 计算极坐标的径向坐标 r 和角度坐标 theta
df['r'] = np.sqrt(df['1_omega']**2 + df['1_weight_log']**2)
df['theta'] = np.arctan2(df['1_weight_log'], df['1_omega'])

# 选择特征和目标变量
X = df[['r', 'theta']]  # 特征
y = df['1_omega']  # 目标变量

# 创建和训练线性回归模型
model = LinearRegression(fit_intercept=True)
model.fit(X, y)

# 预测
y_pred = model.predict(X)

# 计算 R² 分数

##变换dis数据

import pandas as pd
import numpy as np
df = pd.read_csv(
    r'␣2-2023-discount-rate.csv')
average_non_zero = []
for i in range(1, 252):
    if i == 21 or i == 106 or i == 151 or i == 164 or i == 199:
        average_non_zero.append(np.nan)
        continue
    non_zero_values = df[df[f'{i}_dis_rate'] != 0][f'{i}_dis_rate']
    average_non_zero.append(non_zero_values.mean())


indices_of_ones = [index for index, value in enumerate(
    average_non_zero) if (value == 1 or value == np.nan)]

for index in indices_of_ones:
    average_non_zero[index] = 0

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

```
import matplotlib
# Read the file
sales_data = pd.read_csv('.\\results\\2-daily_class_sales.csv')
sales_data['datetime'] = pd.to_datetime(sales_data['datetime'])
sales_data.set_index('datetime', inplace=True)
# Display the first few rows of the data
sales_data.head()

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

# %%

# Calculate the pairwise correlation matrix for the *_price variables
_weight = sales_data.filter(like='_weight')
_price  = sales_data.filter(like='_price')
new_df = pd.concat([_weight,_price],axis=1)
price_corr = new_df.corr()

price_corr.to_csv('.\\results\\2-price-weight_corr.csv',encoding='utf-8')

# %%
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose

# Plot the autocorrelation function (ACF) to help identify a probable period for the sea
plt.figure(figsize=(20,8))
plot_acf(sales_data['1_price'], lags=100)
plt.show()



# %% [markdown]
#  In this case, we see significant peaks at lags around 7, 14, 21, etc., suggesting a u

# %%
sales_data['1_price_avg'] = sales_data['1_price'].rolling(window=7).mean()
sales_data['1_price_avg'].fillna(method='bfill', inplace=True)
# Perform ETS decomposition on the "1_price" time series using a period of 7 (weekly sea
sales_data['1_price_avg'] = sales_data['1_price_avg'].fillna(method='bfill')
result = seasonal_decompose(sales_data['1_price_avg'], model='multiplicative', period=7)
```

```
# Plot the original time series along with the decomposed components
plt.figure(figsize=(12,8))

plt.subplot(4,1,1)
plt.plot(result.observed)
plt.title('Original Series')
plt.ylabel('1_price')

plt.subplot(4,1,2)
plt.plot(result.trend)
plt.title('Trend Component')

plt.subplot(4,1,3)
plt.plot(result.seasonal)
plt.title('Seasonal Component')

plt.subplot(4,1,4)
plt.plot(result.resid)
plt.title('Residual Component')
plt.ylabel('Residual')

plt.tight_layout()
plt.show()



# %%
from statsmodels.tsa.stattools import adfuller

# Performing Augmented Dickey-Fuller test to check for stationarity
adf_result = adfuller(sales_data['1_price_avg'])

# Create a series to hold the results
adf_series = pd.Series(adf_result[0:4],
                       index=['Test Statistic', 'p-value', '# of Lags Used', 'Number of

# Add the critical values to the series
for key, value in adf_result[4].items():
    adf_series[f'Critical Value ({key})'] = value

# Display the ADF test results
adf_series
```

```
# %%
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Fit an ETS model to the "1_price" series
ets_model = ExponentialSmoothing(sales_data['1_price'], trend='multiplicative', seasonal=
ets_fit = ets_model.fit()

# Generate forecasts for the period from July 1, 2023, to July 10, 2023
forecast_period = pd.date_range(start='2023-07-01', end='2023-07-10', freq='D')
forecast_values = ets_fit.forecast(steps=len(forecast_period))

# Plot the historical data along with the forecast
plt.figure(figsize=(12, 6))
plt.plot(sales_data.index, sales_data['1_price'], label='Historical Data')
plt.plot(forecast_period, forecast_values, label='Forecast', color='red')
plt.xlabel('Date')
plt.ylabel('1_price')
plt.title('1_price Forecast (July 1, 2023 - July 10, 2023)')
plt.legend()
plt.grid(True)
plt.show()

# Display forecast values
forecast_values
# %%
```

第三问熵权法与 TOPSIS 评价使用，SPSSpro 代码

```
import numpy
import pandas
from spsspro.algorithm import quantify_analysis
#生成案例数据
data = pandas.DataFrame({
    "A": [1, 2, 3, 4, 5]
})
forward = pandas.DataFrame({
    "B": [1, 2, 3, 4, 5]
})
reverse = pandas.DataFrame({
    "C": [1, 2, 3, 4, 5]
})
```

```
index = pandas.Series([1, 2, 3, 4, 5], name="D")
weight = pandas.Series([1, 2, 3], name="E")
#TOPSIS分析，输入参数详细可以光标放置函数括号内按shift+tab查看，输出结果参考spsspro模板分
print(quantify_analysis.topsis_analysis(data, forward, reverse, index, weight))

    # %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib

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

# %%
df = pd.read_csv('.\\C\\2-2023.csv')
df['datetime'] = pd.to_datetime(df['datetime'],format='mixed')
#df.set_index('datetime',inplace=True)
df.head()

# %%
# Cover by item

# %%
df['weight_dis'] = df['weight']*df['discount']
daily_discount = df.groupby(['no', pd.Grouper(key='datetime', freq='D')]).agg({
    'weight': 'sum',
    'weight_dis': 'sum',
})

daily_discount.head()
daily_discount = daily_discount.unstack(level='no')
daily_discount.fillna(0, inplace=True)
daily_discount.sort_index(inplace=True)
daily_discount.head()

# %%
invaild_cols = [21,106,151,164,199]
cols = []
for i in range(1,252):
    if i not in invaild_cols:
```

```
        cols.append(f'{i}_weight')
for i in range(1,252):
    if i not in invaild_cols:
        cols.append(f'{i}_weight_dis')

daily_discount.columns = cols
daily_discount.to_csv('.\\results\\2-2023-discount.csv')

# %%
daily_discount = pd.read_csv('.\\results\\2-2023-discount.csv')
daily_discount['datetime'] = pd.to_datetime(daily_discount['datetime'])
daily_discount.head()
for i in range(1,252):
    if i not in invaild_cols:
        daily_discount[f'{i}_dis_rate'] = daily_discount[f'{i}_weight_dis']/daily_discou
daily_discount.head()
daily_discount.fillna(0, inplace=True)
daily_discount.to_csv('.\\results\\2-2023-discount-rate.csv')

# %% [markdown]
# # Cover by class

# %%
df['weight_dis'] = df['weight']*df['discount']
daily_discount = df.groupby(['class', pd.Grouper(key='datetime', freq='D')]).agg({
    'weight': 'sum',
    'weight_dis': 'sum',
})

daily_discount.head()
daily_discount = daily_discount.unstack(level='class')
daily_discount.fillna(0, inplace=True)
daily_discount.sort_index(inplace=True)
daily_discount.head()

# %%
cols = []
for i in range(1,7):
    if i not in invaild_cols:
        cols.append(f'{i}_weight')
for i in range(1,7):
```

```
    if i not in invaild_cols:
        cols.append(f'{i}_weight_dis')

daily_discount.columns = cols
daily_discount.to_csv('.\\results\\2-2023-discount-class.csv')

# %%
daily_discount = pd.read_csv('.\\results\\2-2023-discount-class.csv')
daily_discount['datetime'] = pd.to_datetime(daily_discount['datetime'])
daily_discount.head()
for i in range(1,7):
    daily_discount[f'{i}_dis_rate'] = daily_discount[f'{i}_weight_dis']/daily_discount[f
daily_discount.head()
daily_discount.fillna(0, inplace=True)
daily_discount.to_csv('.\\results\\2-2023-discount-rate-class.csv')

# %%
cost = pd.read_csv(".\\C\\3cost.csv")
cost.head()

# %%
# 按周、class进行分组，然后对sales进行求和
cost['date']=pd.to_datetime(cost['date'])
daily_cost = cost.groupby(['class', pd.Grouper(key='date', freq='D')])['cost'].mean().re
display(daily_cost.head())
plt.figure(figsize=(24, 12))

for class_id in daily_cost['class'].unique():
    class_sales = daily_cost[daily_cost['class'] == class_id]
    plt.plot(class_sales['date'], class_sales['cost'], label=class_dict[class_id])

plt.xlabel('Date')
plt.ylabel('Daily Cost')
plt.title('Daily Cost by Class')
plt.legend()
plt.grid(True)
plt.show()
daily_cost.to_csv('.\\results\\3-cost-class.csv',index=False)

# %%
import pandas as pd
```

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

data = pd.read_csv('.\\results\\3-cost-class.csv')


# Extract the data for class 1
class1_data = data[data['class'] == 1]['cost'].values

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf


# Plot the ACF and PACF
plt.figure(figsize=(12,6))

# Plot ACF
plt.subplot(2,1,1)
plot_acf(class1_data, lags=40, ax=plt.gca())
plt.title('ACF of Differenced Series (Class 1)')

# Plot PACF
plt.subplot(2,1,2)
plot_pacf(class1_data, lags=40, ax=plt.gca())
plt.title('PACF of Differenced Series (Class 1)')

plt.tight_layout()
plt.show()



# %%

# Apply FFT
fft_class1 = np.fft.fft(class1_data)

# Define a frequency cutoff value; we will set a fraction of the highest frequencies to
freq_cutoff = 0.05

# Set a fraction of the highest frequencies to zero
num_zeroed = int(freq_cutoff * len(fft_class1))
fft_class1[-num_zeroed:] = 0
```

```
fft_class1[:num_zeroed] = 0

# Apply ifft to get the smoothed data
smoothed_class1 = np.fft.ifft(fft_class1)

# Plot the original and smoothed data
plt.figure(figsize=(10,6))
plt.plot(class1_data, label='Original Data')
plt.plot(smoothed_class1, label='Smoothed Data (FFT)', linestyle='dashed')
plt.xlabel('Day')
plt.ylabel('Cost')
plt.title('Original Data for Class 1')
plt.legend()
plt.grid(True)
plt.show()

# %%
from statsmodels.tsa.stattools import adfuller

# Convert the smoothed data to a real number to avoid issues in modeling
smoothed_class1_real = smoothed_class1.real

# Conduct the Augmented Dickey-Fuller test to check for stationarity
result = adfuller(class1_data)

# Extract and display the test results
adf_statistic = result[0]
p_value = result[1]
critical_values = result[4]

adf_statistic, p_value, critical_values


# %% [markdown]
# *alpha value* << critical value
# the null hypothesis is rejected
# time series methods can be applied

# %%
from scipy.signal import find_peaks, periodogram
```

```
# Calculate the periodogram
frequencies, power = periodogram(smoothed_class1_real)

# Find the peak frequency
peak_idx = find_peaks(power)[0]
peak_freq = frequencies[peak_idx[np.argmax(power[peak_idx])]]

# The period is the reciprocal of the frequency
if peak_freq != 0:
    period = 1 / peak_freq
else:
    period = None

# Plot the periodogram with the peak frequency highlighted
plt.figure(figsize=(10,6))
plt.semilogy(frequencies, power)
plt.semilogy(peak_freq, power[peak_idx[np.argmax(power[peak_idx])]], 'ro')
plt.title('Periodogram')
plt.xlabel('Frequency')
plt.ylabel('Power')
plt.grid(True)
plt.show()

period



# %%
from sklearn.metrics import mean_absolute_error, mean_squared_error
def calculate_performance(test, predictions):
    mae = mean_absolute_error(test, predictions)
    rmse = mean_squared_error(test, predictions, squared=False)
    return mae, rmse

# %%
from statsmodels.tsa.arima.model import ARIMA

train_days = 520
class1_data_test = class1_data[int(len(class1_data))-7:]
class1_data_train = class1_data[int(len(class1_data))-train_days:int(len(class1_data)-7)

# Fit the ARIMA(1, 1, 1) model using the new ARIMA class
```

```
arima_model = ARIMA(class1_data_train, order=(6, 0, 1))
arima_result = arima_model.fit()

# Display the summary of the ARIMA model
display(arima_result.summary())

model_performance = pd.DataFrame(columns=[])
arima_predictions = arima_result.forecast(steps=len(class1_data_test))
model_performance['ARIMA(6,0,1)'] = calculate_performance(class1_data_test, arima_predic
model_performance['ARIMA(6,0,1)']

# %%
from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from pmdarima import auto_arima

# Use auto_arima to find the best SARIMA model
sarima_result = auto_arima(class1_data_train, seasonal=True, m=7, d=0, D=1,
                           start_p=0, start_q=0, max_p=6, max_q=6, start_P=0,trace=True,

# Display the summary of the best SARIMA model
display(sarima_result.summary())


# %%

sarima_predictions = sarima_result.predict()[:7]
print(sarima_predictions)
model_performance['SARIMA'] = calculate_performance(class1_data_test, sarima_predictions
model_performance['SARIMA']

# %%
# Exponential Smoothing (ETS) model
ets_model = ExponentialSmoothing(class1_data_train, trend=None, seasonal='add', seasonal_
ets_fit = ets_model.fit()
ets_predictions = ets_fit.forecast(steps=len(class1_data_test))
model_performance['ETS'] = calculate_performance(class1_data_test, ets_predictions)
model_performance['ETS']
```

```
# %% [markdown]
# # Machine Learning

# %%
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler

# Function to create lag features
def create_lag_features(data, lags):
    lag_features = pd.DataFrame(index=data.index)
    for i in range(1, lags+1):
        lag_features[f"Lag_{i}"] = data.shift(i)
    return lag_features

# Define the number of lags to use as features
lags = 15

# Create lag features for the training and test sets
train_lag_features = create_lag_features(pd.Series(class1_data_train), lags).dropna()

test_lag_features = create_lag_features(pd.Series(class1_data[-lags-7:]), lags).dropna()
display(test_lag_features)
# Get the corresponding target values for the training and test sets
train_targets = class1_data_train[lags:]
test_targets = class1_data[-7:]

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(train_lag_features, train_targets)

# Make predictions on the test set
rf_predictions = rf_model.predict(test_lag_features)
display(rf_predictions)
# Calculate performance metrics
model_performance['Random Forest'] = calculate_performance(class1_data_test, rf_predictio



model_performance['Random Forest']
```

```
# %%
#hyperparameter tuning
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt'],
    'bootstrap': [True, False]
}
X_train, X_test = train_lag_features, test_lag_features
y_train, y_test = train_targets, test_targets

mpl = RandomForestRegressor()
grid_search = GridSearchCV(mpl, param_grid, cv=5, scoring='neg_mean_squared_error', verb
grid_search.fit(X_train, y_train)

print('Best parameters found: ', grid_search.best_params_)

# 在测试集上评估最优模型的性能
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
model_performance['ELM'] = calculate_performance(y_pred, y_test)
model_performance['ELM']

# %%
best_model = grid_search.best_estimator_
for i in range(7):
    test_lag_features = create_lag_features(pd.Series(class1_data[-lags-1:]), lags).drop
    rf_predictions = best_model.predict(test_lag_features)
    class1_data = np.append(class1_data, rf_predictions)
print(class1_data[-7:])

# %%
# Standardize the features (important for SVM)
scaler = StandardScaler()
train_lag_features_scaled = scaler.fit_transform(train_lag_features)
test_lag_features_scaled = scaler.transform(test_lag_features)
```

```
# Train a Support Vector Machine (SVM) Regressor
svm_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svm_model.fit(train_lag_features_scaled, train_targets)

# Make predictions on the test set
svm_predictions = svm_model.predict(test_lag_features_scaled)

# Calculate performance metrics
model_performance['SVM'] = calculate_performance(test_targets, svm_predictions)

# Performance metrics
model_performance['SVM']



# %%
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from keras.models import Sequential
from keras.layers import Dense, LSTM
import numpy as np



# Scale the data to the range [0, 1] using MinMaxScaler
scaler = MinMaxScaler()
train_lag_features_scaled = scaler.fit_transform(train_lag_features)
test_lag_features_scaled = scaler.transform(test_lag_features)
# Step 2: Feature Engineering

X_train, X_test = train_lag_features_scaled, test_lag_features_scaled
y_train, y_test = train_targets, test_targets

# Reshape the data to 3D array as required by the LSTM model
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

# Step 3: Model Building
# Define the LSTM model architecture
model = Sequential([
    LSTM(50, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(1)
```

```
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model on the training data
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test),

# Evaluate the model on the testing data
y_pred = model.predict(X_test)

# Calculate the R-squared and MSE for the model

model_performance['LSTM'] = calculate_performance(class1_data_test, y_pred)

model_performance['LSTM']

# %%
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score, accuracy_score, mean_squared_error, mean_absolute_
from scipy.optimize import minimize
#extreme learning machine
class OP_ELMRegressor():
    def __init__(self, n_hidden):
        self.n_hidden = n_hidden
    def ReLU(self, x):
        return np.maximum(x, 0, x)
    def fit(self, X, y):
        self.X = X
        self.y = y
        n_samples, n_features = X.shape
        self.W = np.random.randn(n_features + 1, self.n_hidden)

        def loss_func(W_vec):
            W = W_vec.reshape((n_features + 1, self.n_hidden))
            H = np.dot(np.hstack((X, np.ones((n_samples, 1)))), W)
            #print(H)
            H = self.ReLU(H)  # 加上激活函数
            beta = np.dot(np.linalg.pinv(H), y)
            y_pred = np.dot(H, beta)
            mse = np.mean((y - y_pred) ** 2)
```

```
            return mse

        # 用拟牛顿法优化权重矩阵
        res = minimize(loss_func, self.W.ravel(), method='BFGS')
        self.W = res.x.reshape((n_features + 1, self.n_hidden))



        H = np.hstack((X, np.ones((n_samples, 1))))  # 添加偏置项
        H = np.dot(H, self.W)
        H = self.ReLU(H)  # 加上激活函数
        self.beta = np.dot(np.linalg.pinv(H), y)

    def predict(self, X):
        n_samples = X.shape[0]
        H = np.hstack((X, np.ones((n_samples, 1))))
        H = np.dot(H, self.W)
        H = self.ReLU(H)  # 加上激活函数
        y_pred = np.dot(H, self.beta)
        return y_pred

def evaluation(y_test, y_predict):
    mae = mean_absolute_error(y_test, y_predict)
    mse = mean_squared_error(y_test, y_predict)
    rmse = np.sqrt(mean_squared_error(y_test, y_predict))
    mape=(abs(y_predict -y_test)/ y_test).mean()
    r_2=r2_score(y_test, y_predict)
    return mae, rmse, mape,r_2  #mse

# %%
# Train a OP_ELMRegressor
elm_model = OP_ELMRegressor(n_hidden=16)
elm_model.fit(train_lag_features, train_targets)

# Make predictions on the test set
rf_predictions = elm_model.predict(test_lag_features)
display(rf_predictions)
# Calculate performance metrics
model_performance['ELM'] = calculate_performance(class1_data_test, rf_predictions)



model_performance['ELM']
```

```
# %%
# Train a simple feedforward neural network (MultiLayer Perceptron)
nn_model = MLPRegressor(hidden_layer_sizes=(100,), activation='relu', solver='adam', ran
nn_model.fit(train_lag_features_scaled, train_targets)

# Make predictions on the test set
nn_predictions = nn_model.predict(test_lag_features_scaled)

# Calculate performance metrics
model_performance['Neural Network'] = calculate_performance(test_targets, nn_predictions

# Performance metrics
model_performance['Neural Network']



# %%
#hyperparameter tuning
from sklearn.model_selection import GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(10,), (50,), (100,), (50, 50)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001],
}
X_train, X_test = train_lag_features_scaled, test_lag_features_scaled
y_train, y_test = train_targets, test_targets

mpl = MLPRegressor(max_iter=10000)
grid_search = GridSearchCV(mpl, param_grid, cv=5, scoring='neg_mean_squared_error', verb
grid_search.fit(X_train, y_train)

print('Best parameters found: ', grid_search.best_params_)

# 在测试集上评估最优模型的性能
test_score = grid_search.score(X_test, y_test)
print('Test set score with best parameters: ', test_score)

# %%
display(model_performance)

# %%
```

```
# Function to plot actual and predicted values
def plot_actual_vs_predicted(actual, predicted, title):
    plt.figure(figsize=(10,6))
    plt.plot(actual, label='Actual')
    plt.plot(predicted, label='Predicted', linestyle='dashed')
    plt.title(title)
    plt.xlabel('Day')
    plt.ylabel('Cost')
    plt.legend()
    plt.grid(True)
    plt.show()

# Get predictions on the training set
rf_train_predictions = best_model.predict(train_lag_features)
svm_train_predictions = svm_model.predict(scaler.transform(train_lag_features))
nn_train_predictions = nn_model.predict(scaler.transform(train_lag_features))
# Plot actual vs predicted values for the training set
plot_actual_vs_predicted(train_targets, rf_train_predictions, 'Random Forest: Actual vs
plot_actual_vs_predicted(train_targets, svm_train_predictions, 'SVM: Actual vs Predicted
plot_actual_vs_predicted(train_targets, nn_train_predictions, 'Neural Network: Actual vs


# %%
fitted_values = weight_series[1:] - arima_result.resid
```