蔬菜类商品的自动定价与补货决策

摘要

当商超需要做出关于蔬菜商品的补货和定价决策时，数据的收集和分析起着至关重要的作用。本文在题目所给相关资料的基础上，分析销售量分布规律以及相互关系，并构建商超收益模型用于决策。

针对问题一，通过考察统计销售量等信息，注意到大量流水低信息少的单品。为降低处理难度，对文件信息依据销售量累积贡献率，降序取前 95% 对数据进行清洗，商品单品总数降至 86 种。同时注意到退货信息占比量少的特点，一并将退货流水剔除以降低分析难度。而后统计每一单品和每一品类的日销售量，分析其随时间的分布规律，并对每个单品以及品类进行 Spearman 相关性分析，探讨相互关系以及周期性变化特征。

针对问题二，借助问题一处理得到的日销售量等数据，再构建多个变量。例如成本利润率和折扣系数等，同时参考加成定价法，构建商超收益模型。目的是通过寻找最大收益确定补货量与定价。部分数据需进行预测，考虑数据的时间序列特点，选用 ARIMA 模型对日销售量和批发价进行预测，依据预测所得求解商超收益模型。最终求得商超最大收益分别为 (单位元)：649.68、608.68、601.56、603.84、613.69、618.57、624.33。并确定各品类补货量与定价。

针对问题三，进一步细化商超收益模型，并根据相关约束条件提前去除不符合的单品，使得单品种数进一步降低至 30，计算被剔除的单品中若引进则会带来的收益期望，取收益期望为正且排名前列的单品添至已选可售单品。最终单品选择数为 32。考虑到市场对不同品类的需求，引入需求差量用于影响所选 32 种单品的补货量，从而改进商超收益模型。借由问题二中已预测的数据，使用线性规划中的单纯形法对问题三求解，求得最大商超收益为：670.05 元。并确定各品类补货量与定价策略。

针对问题四，提出建议收集库存数据，过程性成本数据以及客户反馈数据等相关信息，以做到分析客户的行为模式和群体画像、分析商品的详细利润组成和其他潜在成本、以及获取更多的约束条件和关系式，进而改进商超收益模型。

最后，比较验证了 ARIMA 模型对预测所需数据参数的敏感程度，探讨其预测的滞后性与保守性。并对比了线性回归预测模型与 ARIMA 模型的预测性能，结论是线性回归预测模型对时间序列的数据预测能力低于 ARIMA 模型。

关键字： Spearman 相关性分析 ARIMA 模型 单纯形法 商超收益模型

一、 问题重述

1.1 问题背景

现某生鲜超市中蔬菜种类商品数量繁多且保鲜期较短, 品相也随时间增加变差, 当日商品无法隔日再售, 因此商超会参考各蔬菜商品的历史销售与需求情况每日补货。

1.2 问题要求

商超提供了多个附件，分别是：附件 1（商超经销的 6 个蔬菜品类的商品信息）、附件 2（2020 年 7 月 1 日至 2023 年 6 月 30 日各商品的销售流水明细）、附件 3（2020 年 7 月 1 日至 2023 年 6 月 30 日各商品批发价格）和附件 4（各商品近期的损耗率数据）。

基于上述背景与附件信息，我们需要建立数学模型解决以下问题：

1. **问题 1**：蔬菜类商品不同品类或不同单品之间可能存在一定的关联关系，请分析蔬菜各品类及单品销售量的分布规律及相互关系。
2. **问题 2**：考虑商超以品类为单位做补货计划，请分析各蔬菜品类的销售总量与成本加成定价的关系，并给出各蔬菜品类未来一周 (2023 年 7 月 1-7 日) 的日补货总量和定价策略，使得商超收益最大。
3. **问题 3**：因蔬菜类商品的销售空间有限，商超希望进一步制定单品的补货计划，要求可售单品总数控制在 27-33 个，且各单品订购量满足最小陈列量 2.5 千克的要求。根据 2023 年 6 月 24-30 日的可售品种，给出 7 月 1 日的单品补货量和定价策略，在尽量满足市场对各品类蔬菜商品需求的前提下，使得商超收益最大。
4. **问题 4**：为了更好地制定蔬菜商品的补货和定价决策，商超还需要采集哪些相关数据，这些数据对解决上述问题有何帮助，请给出意见和理由。

二、 问题分析

本文要解决的是蔬菜类商品的定价与补货决策问题，针对每一单品和品类，最终都要决策出其定价和补货数量。问题 1 是在真正进行决策前，先就不同品类或不同单品的蔬菜类商品对销售量分析其中可能蕴涵的分布规律以及相互关系以方便后续作答。问题 2 是在仅针对以品类为单位时，根据所给以及推导信息对补货量与定价做出决策从而使得商超收益最大化。问题 3 是针对每一单品做出补货量与定价策略外，还增添了各种限制条件。问题 4 则是开放性问题，考察对于商品定价与补货决策还需要哪些数据以使得数据可靠性更高且收益更稳定。四个问题层层递进，细化程度不断提升。

2.1 问题一

针对问题一，并考虑到后续问题二和问题三的处理，可以认为从时间关系上考察商超蔬菜类商品品类与单品的销售量之间的分布规律与相互关系是个更优的方向。这样能够一定程度上在完成问题一的同时，为后续问题提供帮助。

时间粒度的选择有“年，月，日”三种粒度。基本上可以认为“年”的粒度过大，对于当前问题的分析并没有太多的帮助。而“月”的粒度也略微粗犷，时间点被降级成 36 个月，在降级的过程中信息也大量的丢失了，从而导致分析的分布规律与相互关系缺乏置信度。所以选择以“日”的时间粒度对数据进行分析。

借由附件 1，附件 2 统计商超 2020 年 7 月 1 日至 2023 年 6 月 30 日各商品的销售流水明细后，注意到在 246 个单品中，大部分单品在销售总量上的占比过低，过多的维度也会导致计算上的难度，权衡之后选择对数据进行清洗。另外，除了清洗所剔除单品的数据以外，也选择将数据占比极小的退货数据剔除。再计算每日各单品与品类的日销售量，在此基础上，使用 Spearman 相关性分析方法试探究单品与单品之间，品类与品类之间日销售量的相互关系。除此之外，通过绘制折线图可察觉部分商品销售量周期性变化特性。整体思路如下图所示：

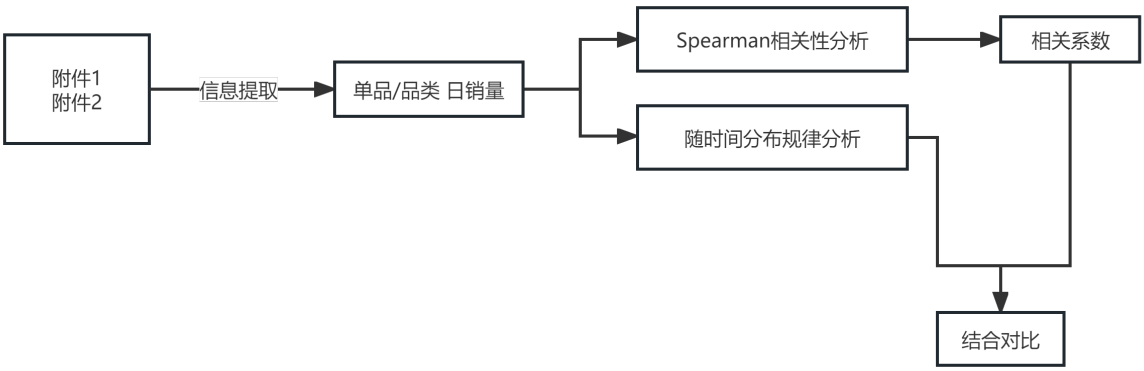

图 **1**　问题一思路流程

2.2 问题二

问题一中统计了各个单品和品类的日销售量，在此基础上，继续对问题二进行分析解决。

针对问题二，其关键在于对以品类为单位进行补货和定价决策，定价方法为加成定价法。为方便问题三的求解，将模型的构建提前细化，以单品为单位构建变量。同时考虑到同一品类商品之间的相似性，也是考虑到降低模型复杂度，对小部分变量以品类为单位构建。除此之外，继续借由附件 3 和附件 4 求得一些必要常量。借由所构建变量和常量以及部分约束关系，建立商超总收益模型。

若要给出各蔬菜品类未来一周的日补货总量和定价策略，需要对构建的部分变量使用预测模型进行预测。考虑到数据的时间特性以及其他各种原因，选用 ARIMA 模型进行预测。将预测的变量作为已知信息从而对商超总收益模型进行求解。流程图如下：

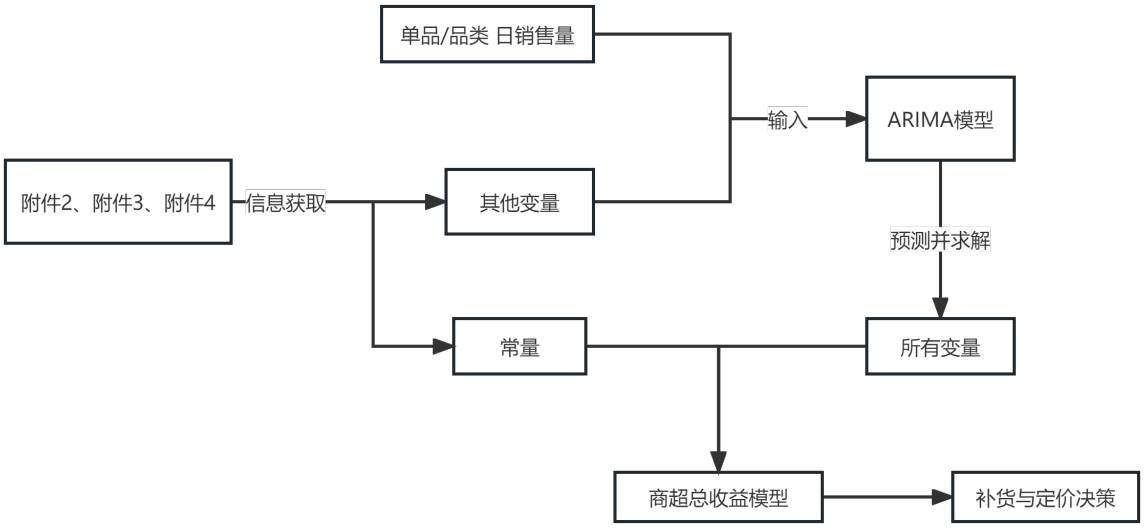

图 **2**　问题二流程图

2.3 问题三

针对问题三，延用问题二的模型并细化为以单品为单位。问题三的关键在于所给的部分约束条件。通过对每条约束条件的分析，从而对原数据进行合理处理，使得模型的约束条件放宽。同时，因为题目要求的变化，对问题二模型进行改进。模型最终呈现为线性规划的模式。

用单纯形法求解线性规划，最终求解。本题流程如下：

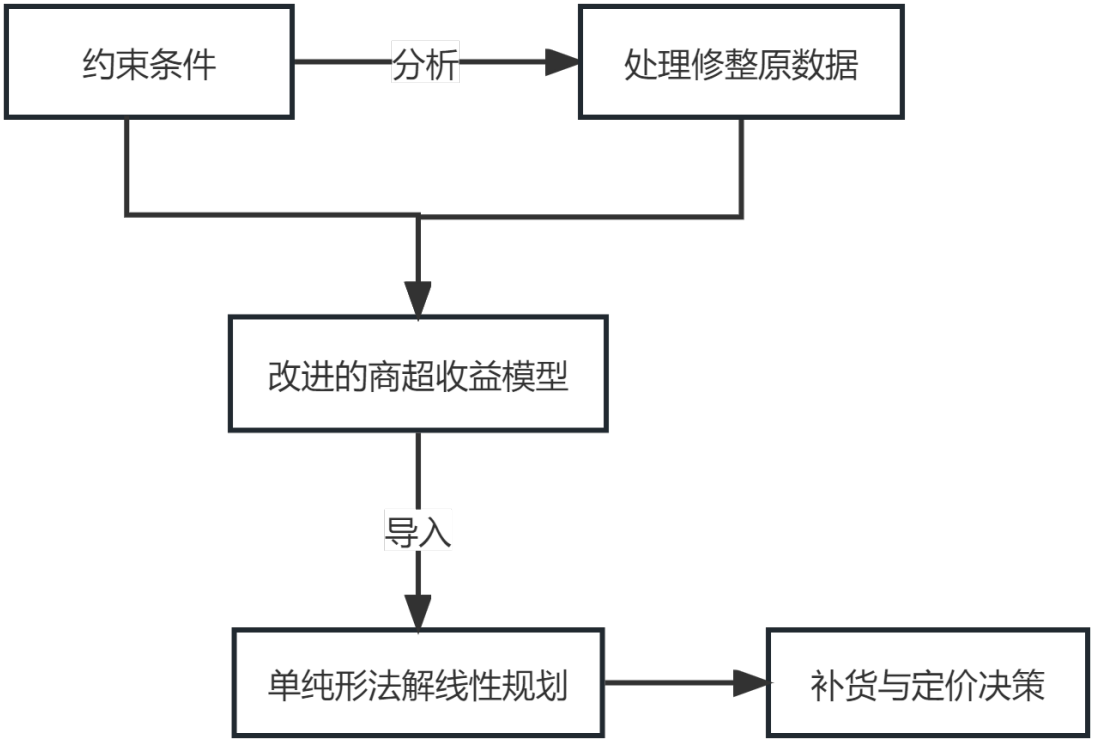

图 **3**　问题三流程图

2.4 问题四

通过查找相关资料，了解现实生活中商家会收集的相关数据。同时，通过前面建模过程中对模型的化简，了解到模型如果有哪些数据能够更好的改进，从而给出建议。

三、 模型的假设

本文提出以下合理假设：

- 所有商品都无法隔日售出，不考虑小部分商品的长时间存储可能性。
- 近期的商品损耗率与往日及以后所有时期的商品损耗率相同
- 不考虑可能存在的社会因素影响（例如疫情封控对客户流量的影响使得商品销量降低）。
- 不考虑除进货成本以外的其他可能存在的成本
- 售价与销量往往存在一定的相关性，但考虑到食品作为生活必需品的性质，食品的售量存在一个此消彼涨的情况，若此种食品价格涨了，人们往往会购买别的类似食品，从而使总收益变化程度不大。因此，不考虑售价与销量的可能存在的相关性。

四、 符号说明

符号	含义
P	原图
SP	嵌入数字水印后的图片

五、 模型建立与求解

5.1 问题一模型建立与求解

5.1.1 数据清洗与处理统计

统计各个单品的总销售量，注意到在 246 个单品中，存在大量单品其销售流水占比远远小于主流单品的销售量，在极大地影响数据处理速度的同时，本身也并没有太多的分析意义。在此种原因的加持下，将单品销售量从大到小排列并选取，计算累计占比率，取至 95%，将之后的单品剔除。

除此之外，统计销售流水中的退货数据数量，在 878503 条数据中，退货数据仅有 461 条，可以认为退货数据对定价与补货决策无太大实际效用，故将退货数据剔除。

从处理后的数据中统计各单品的每日销售量 α_i（千克）和各品类的每日销售量 β_i（千克）。

α_i 的定义如下：

$$\alpha_i = \sum d_j \tag{1}$$

其中，d_j 为销售流水中同一单品的每单销售量，单位为千克。

β_i 的定义如下：

$$\beta_i = \sum \alpha_j \tag{2}$$

其中，α_j 为属于同一品类各单品的每日销售量，单位为千克。

5.1.2 分析分布规律与相关性

借由统计后的 α_i 和 β_i，试探究其随时间变化下的统计规律。

而后，由于现实中商品的销售量受大量其他因素影响，其分布特征并非为简单的正态分布，在此条件下，选用 Spearman 相关性分析方法分析各单品 α_i 和各品类 β_i 的相互关系是较优的选择。具体过程如下[1]：

(1) 秩次转换并计算秩差

(2) 计算 Spearman 秩相关系数：使用秩次差异来计算 Spearman 秩相关系数

$$\rho = 1 - \frac{6\sum_{i=1}^{N} d_i^2}{N\left(N^2 - 1\right)} \tag{3}$$

其中，N 为数据个数，d 为两个变量的秩差，与两个相关变量的具体值无关，仅与其值之间的大小关系有关，适用于时间序列间的相关性分析。

此外，Spearman 相关性分析不仅可以揭示单品之间的关系，还可以帮助排名单品的销售趋势，有助于识别销售排名前列的蔬菜单品。因此，这项分析为了解和利用蔬菜销售数据的价值提供了深刻的见解。

5.1.3 问题一求解

先剔除销售流水中的退货数据，剩余 878042 条数据。而后从大到小计算各单品累计销售量占比，剔除 95% 之后的单品。剩余单品数目从 246 降至 86，销售流水数据也降至 826623 条。部分所选单品以及其销售总额如下表：

对 86 个单品的 α_i 使用 SPSSpro 软件计算其 Spearman 相关系数，从而得到相关系数矩阵，由于矩阵过大不方便展示，已存放在支撑材料。当相关系数大于 0.8 时认为 α_i 与 α_j 之间的关系为强相关，该矩阵中共有 62 个数据符合标准，故 31 对变量符合强相关。受篇幅影响，下面陈列相关系数大于 0.87 的四组数据。

“金针菇（1）”和“青梗散花”的相关系数为 0.872，$\alpha_{\text{金针菇(1)}}$ 与 $\alpha_{\text{青梗散花}}$ 随时间分布规律如下：

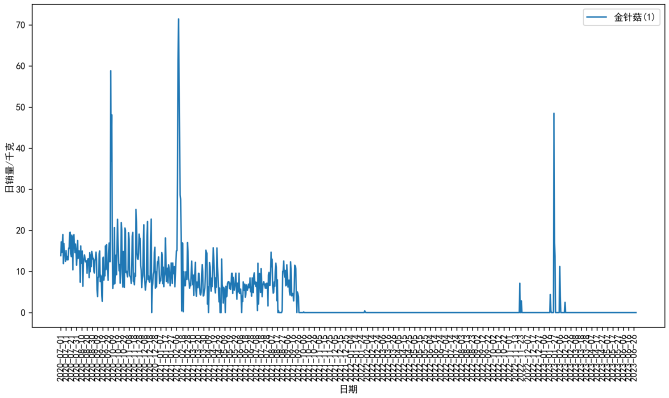

图 **4**　金针菇 **(1)**

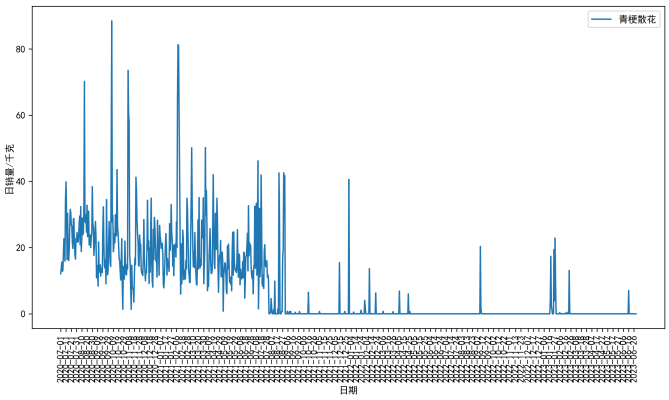

图 **5**　青梗散花

“小米椒（份）”和“小皱皮（份）”的相关系数为 0.872，$\alpha_{\text{小米椒 (份)}}$ 与 $\alpha_{\text{小皱皮 (份)}}$ 随时间分布规律如下：

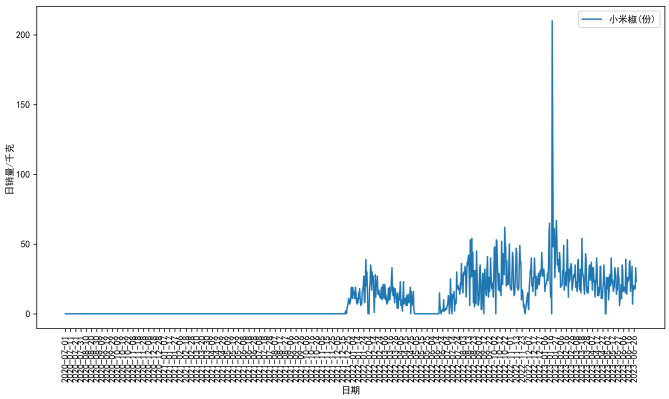

图 **6**　小米椒 **(**份**)**

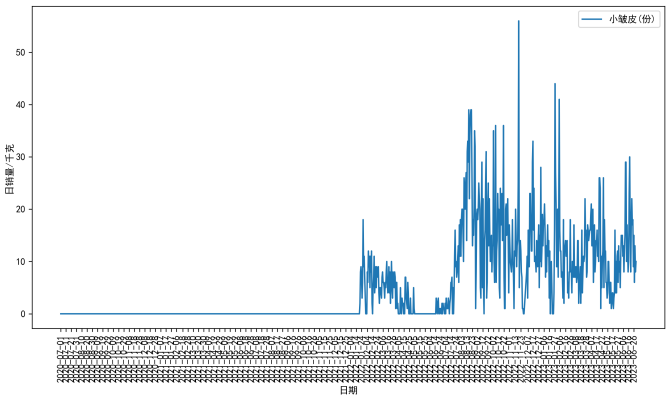

图 **7**　小皱皮 **(**份**)**

“杏鲍菇（1）”和“小皱皮（份）”的相关系数为 0.872，$\alpha_{\text{杏鲍菇 (1)}}$ 与 $\alpha_{\text{金针菇 (1)}}$ 随时间分布规律如下：

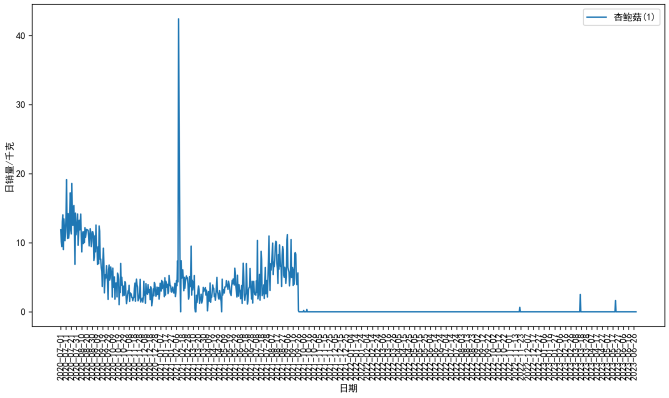

图 **8**　杏鲍菇 **(1)**

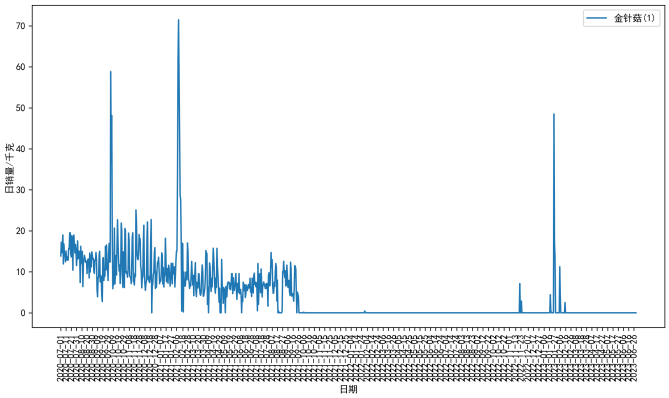

图 **9**　金针菇 **(1)**

“云南生菜（份）”和“云南油麦菜（份）”的相关系数为 0.872，$\alpha_{\text{云南生菜(份)}}$ 与 $\alpha_{\text{云南油麦菜(份)}}$ 随时间分布规律如下：

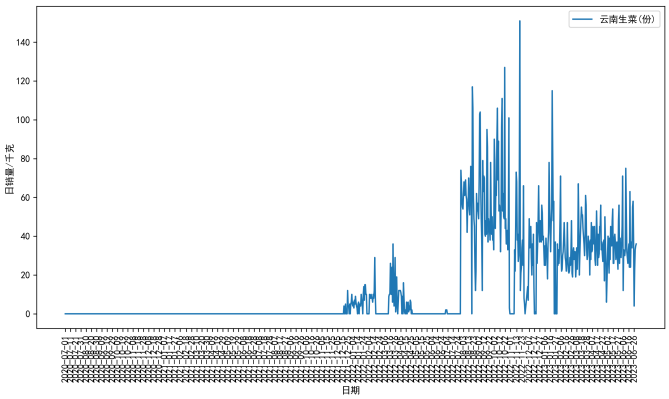

图 **10**　云南生菜 **(份)**

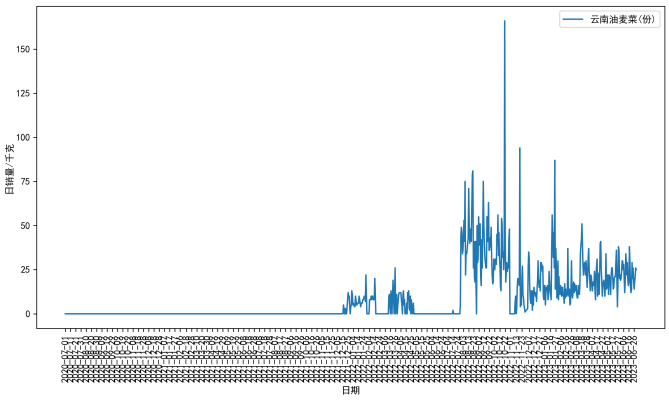

图 **11**　云南油麦菜 **(**份**)**

以上四组仅为所有组合中表现出强相关性的代表，观察散点图可易得这些单品往往售卖时期相近似，这一特征为相关系数较高的原因之一。

针对以上四个组合中出现的七个单品，给出它们的相关系数热力矩阵图以供参考。

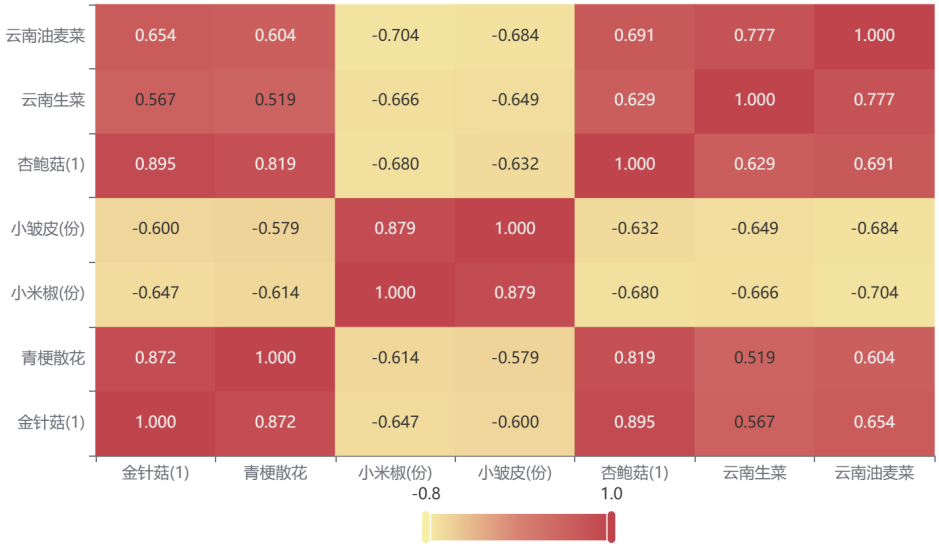

图 **12**　相关系数热力图

此外，对每个品类的 β_i，绘制其随时间变化的散点图，观察易得其中三类商品存在一定的周期性变化，散点图如下：

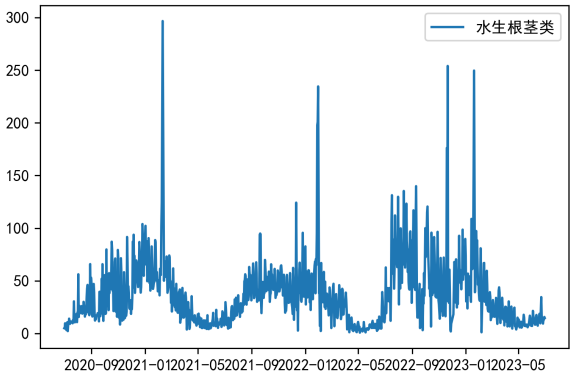

图 **13**　水生根茎类

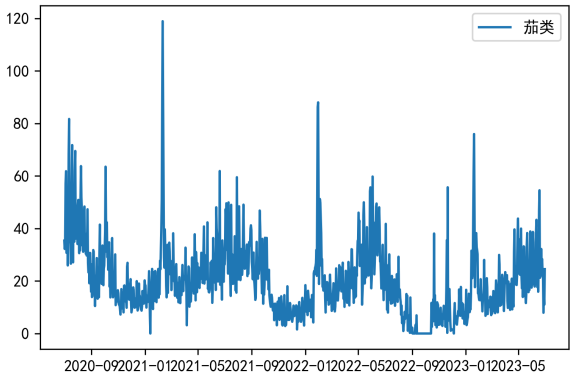

图 **14**　茄类

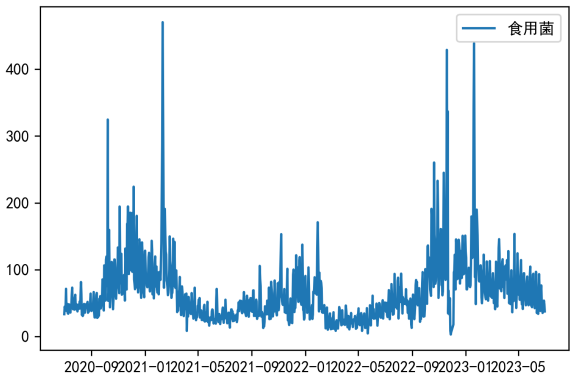

图 **15**　食用菌

5.2 问题二

5.2.1 相关变量的构建

(1) 商品损耗率 μ

其定义为，针对每一单品，每天因各种原因自然损耗或是没收卖出去而剩余的量的占比。其值已在附件 4 中给出。

(2) 补货量 γ

定义为一天内每一单品各自的补货量。

(3) 折扣系数 r

在销售流水明细中，有小部分的商品是以打折销售的形式售出的。这些商品打折的原因往往可能是放置时间过长或是卖相受损，为体现打折对商超收益的影响，定义折扣系数 r。考虑到同一品类的商品往往进货渠道一致，受损所致的打折幅度相同，爱好人群相近，故对每一品类使用同一折扣系数。其定义公式如下：

$$r_i = \frac{\sum(\frac{\text{折扣价}}{\text{当日售价}} - 1)}{N} \tag{4}$$

其中，N 为单一品类总销售单数。

(4) 单品定价 x

其定义为，针对每一单品，其各自的价格。

(5) 成本利润率 σ

利润率是成本加成定价法中必要的，而成本利润率系利润率的一种变体。其广义定义式如下：

$$\sigma = \frac{\text{收益}}{\text{成本}} \times 100\% \tag{5}$$

考虑到同一品类的商品其作为食品口感等因素大致相同，成本也基本近似，故对同一品类使用相同的利润率。

(6) 总收益 w

定义为商超一天内的总收益额度。

(7) 品类加权定价 X

其定义为，针对每一品类的商品所求的对其旗下所有商品具有代表性的定价。

5.2.2 商超总收益模型的构建

对于任意 γ_i, 其应满足售货需求和自然损失，故其与 μ_i 和 α_i 应满足以下不等式：

$$\gamma_i \left(1 - \mu_i\right) \geq \alpha_i \tag{6}$$

利润率的本质是收益与成本之间的占比，在当前条件假设下，成本可由 γ 和 d 简单表示。加成定价法是指，为了确定一种产品的价格而把一个百分比（或绝对的）数量加到所估计的产品成本上，故 σ 与 γ 和 d 满足以下关系式:

$$\sigma_i = \frac{\left(\sum \alpha_j x_j\right)(1+r_i) - \sum \gamma_j d_j}{\sum \gamma_j d_j} \tag{7}$$

类似的，可推导出总收益与各变量的关系式如下：

$$w = \sum_i^n \left[\alpha_i(x_i(1+r_i) - d_i) - (\gamma_i - \alpha_i)d_i\right] \tag{8}$$

由以上，可推导出商超总收益模型：

$$\text{求} \max w$$
$$st. \begin{cases} w = \sum_i^n \left[\alpha_i(x_i(1+r_i) - d_i) - (\gamma_i - \alpha_i)d_i\right] \\ \gamma_i\left(1-\mu_i\right) \geq \alpha_i \\ \sigma_i = \frac{\left(\sum \alpha_j x_j\right)(1+r_i) - \sum \gamma_j d_j}{\sum \gamma_j d_j} \end{cases} \tag{9}$$

5.2.3 商超总收益模型的求解

商超总收益额度主要取决于商超的补货量与定价策略，这包括对各单品销量与各单品批发价的提前预测，以此为基础用式子 [7] 对每一单品 x_i 求解，继而用加权法确定其品类加权定价 X_j, 计算公式如下:

$$X_j = \sum_n^i \frac{\alpha_i}{\beta_j} x_i \tag{10}$$

其中，j 为 i 所属的品类。

除此之外，仍需要对 α 进行预测。考虑销售流水数据的时间序列特点，使用一般的线性回归法等方法对 α 进行预测和拟合容易受到老数据的流水影响。下图是“泡泡椒(精品)”的每日销量曲线：

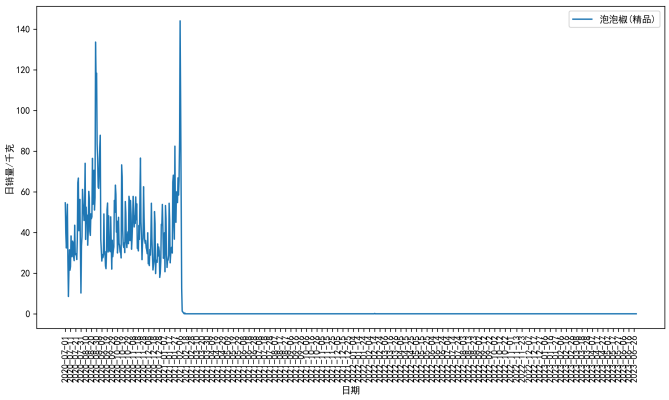

图 **16**　泡泡椒 **(**精品**)**

容易观察到，其曲线显著特点在于：日销售量在某一日期之后完全归于 0，不再售卖。一个导致此种特点的可能原因是该商品的原产家被检测出食品安全问题等使其被勒令停止出售。而具有此种曲线特征的单品不止一家，若使用线性回归等方法预测，会将所有时间点的数据视为同权的数据，得到的回归线仍会在 0 的上方，预测出的数值大于 0，与现实情况完全不符，这就体现了时间久远程度对于预测未来数据的权重不同。也正因为这种特点，使用 ARIMA 模型是本文认为的更优选择。

ARIMA 模型具有以下优点[2]:

1. 不需要借助事物发展的因果关系去分析这个事物过去的和未来的联系。
2. 对趋势性、随机性和相关性的数据都能有很好的预测结果。
3. 该模型的结构比较简单，预测原理也很容易理解，只需要内生变量而不需要借助一些其他类型的外生变量。

ARIMA 模型的建模过程如下：

(1) 对原始时序数据进行平稳性检验。如果序列不满足平稳性条件则进行差分平稳化处理。

(2) 模型识别。对平稳序列分别进行 ACF 和 PACF 图分析, 综合差分阶数，得到最佳参数 p,q 和 d 。

(3) 模型估计。通过可视化观察拟合效果并检验参数显著性。

(4) 模型检验。使用不同配置方式进行建模, 对比拟合参数结果并选定最终模型。

(5) 残差白噪声检验。根据检验结果决定是否继续建模。

(6) 数据预测。

根据上述过程，将 α 与其对应的时间日期输入进模型中，训练完成后预测 7 月 1 日至 7 日七天的 α, 根据式 [2] 计算 β，从而进一步解得 x 和 X。

同样的，继续使用 ARIMA 模型预测 7 月 1 日至 7 日七天的 d。

观察式 [6], 容易知道当取等时，w 取得最大。

由以上三点，可解模型，求得 7 月 1 日至 7 日商超的最大收益如表 2。

表 **2**　**2023** 年 **7** 月 **1** 日至 **7** 日商超总收益

日期	总收益
2023/7/1	649.6865139
2023/7/2	608.6804961
2023/7/3	601.5645523
2023/7/4	603.8445866
2023/7/5	613.6927919
2023/7/6	618.5767892
2023/7/7	624.3340563

各品类加权定价 X 如表 3。

表 **3**　**2023** 年 **7** 月 **1** 日至 **7** 日各品类加权定价

分类名称	7/1	7/2	7/3	7/4	7/5	7/6	7/7
水生根茎类	20.792	20.598	20.983	21.587	22.204	21.772	20.784
花叶类	5.486	5.439	5.503	5.523	5.517	5.513	5.531
花菜类	11.909	11.833	11.814	11.793	11.771	11.750	11.729
茄类	6.896	6.995	7.102	7.188	7.257	7.329	7.395
辣椒类	5.771	5.690	5.652	5.721	5.773	5.747	5.695
食用菌	6.151	6.162	5.815	5.544	5.395	5.307	5.347

各品类进货量通过求各单品 γ 的和可求，如表 4:

表 **4**　**2023** 年 **7** 月 **1** 日至 **7** 日各品类进货量

分类名称	7/1	7/2	7/3	7/4	7/5	7/6	7/7
水生根茎类	17.463	15.81	15.040	16.39	17.564	18.685	18.927
花叶类	157.84	155.64	155.622	154.53	154.12	154.37	156.6
花菜类	22.888	20.094	19.788	19.665	19.616	19.596	19.588
茄类	23.962	22.762	22.218	22.111	22.24	22.243	22.282
辣椒类	97.881	95.382	91.033	87.471	88.508	89.66	93.237
食用菌	43.217	36.655	39.614	41.670	43.292	43.665	42.468

5.3 问题三

5.3.1 问题三模型建立思路

在问题二中所建立的商超总收益模型，是建立在 α 之上，以品类角度求得的总收益。在此基础上对其进行适当修改并添加部分限制条件可进一步对问题三进行求解。

问题三相比于问题二，除了单位粒度细分为单品之外，还有以下几点约束：

1. 可售单品总数控制在 27-33 个。
2. 各单品订购量满足最小陈列量 2.5 千克。
3. 参考 2023 年 6 月 24-30 日的可售品种。
4. 尽量满足市场对各品类蔬菜商品需求。

因计算单位细化至单品，决定将成本利润率 σ 细化成针对每一单品的成本利润率，计算所得部分单品成本利润率如表 5：

表 **5**　部分单品 σ

单品名称	σ
上海青	0.446
苋菜	0.272
西兰花	0.362
红薯尖	0.523
芜湖青椒 (1)	0.345

参考 6 月 24 日至 30 日的可售单品，将这个时段内未出售的商品剔除，可选单品数从问题二的 86 种降至 40 种。考虑最小陈列量为 2.5 千克，且认为最小陈列量理应与 γ 含义相同。利用从问题二中所解 γ, 剔除 $\gamma < 2.5$ 的单品，单品数量降至 30 种。

容易证明，对于剔除所剩的 30 种单品，在商超全部选择售卖时，商超的总收益相较于不选满时更大。将 β 视作市场对各品类商品的需求量，此时，各品类 β 在 40 种时的情况与删减至 30 种时的情况对比如下：

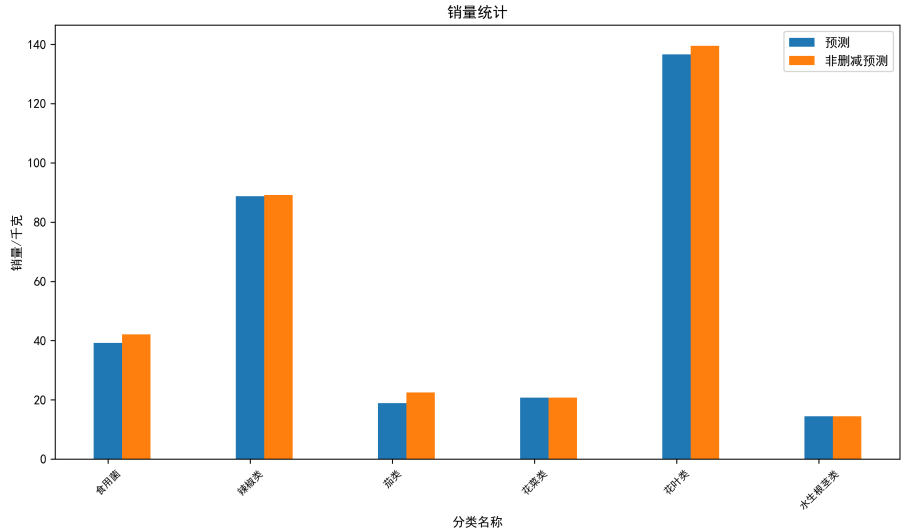

图 **17**　**40** 种与 **30** 种对比

现在，最大可售单品数量仍剩 3 种，考虑到剔除的 10 种单品中可能存在收益可能性，决定对剩余的 10 种商品以 $\gamma = 2.5$ 作为条件，用问题二模型计算收益并从大到小陈列如表 6：

表 **6**　未选择的部分单品在 $\gamma = 2.5$ 时的收益 **(**降序**)**

单品名称	收益 (元)
青茄子 (1)	3.816
虫草花 (份)	2.661
圆茄子 (2)	-0.502
云南生菜	-1.500
···	···

其中，即使 γ 增至 2.5kg 的情况下仍能盈利的单品仅剩 2 种，选择将这 2 种单品添至所选可售单品中。

此时，各品类 β 在 40 种时的情况与删减至 32 种时的情况对比如下：

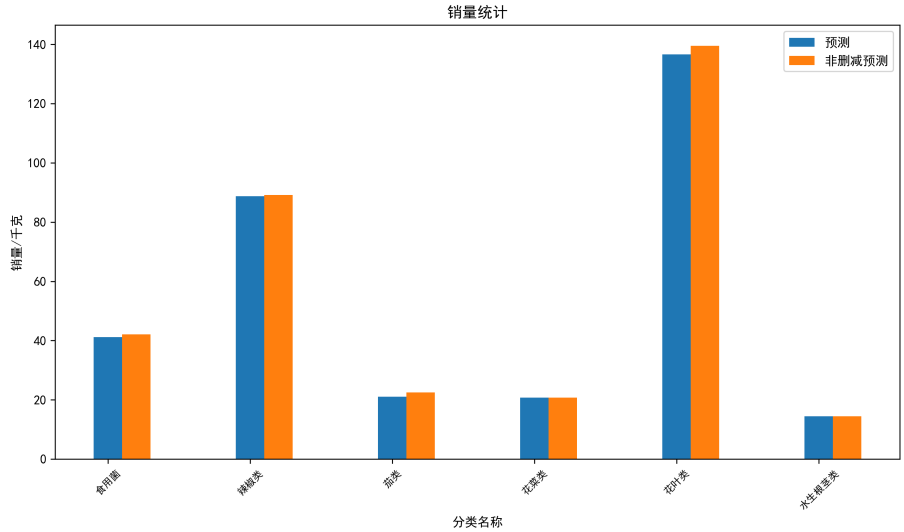

图 **18**　**40** 种与 **32** 种对比

认为，当客户在商超上买不到某一单品时，会倾向于选择同一品类的另一近似商品，因此可以认为，客户对各品类蔬菜的需求期望在删减前后几乎近似。为尽量满足市场对各品类蔬菜商品的需求，选择将该部分的市场空缺用已选择的 32 种商品进行填补。为方便表达，构建需求差量 Δ, 有以下限制条件：

$$\sum \Delta_i \geq \beta_{\text{未删减预期 j}} - \beta_{\text{预期 j}} \tag{11}$$

同时 γ 与 μ, Δ 和 α 存在以下关系式：

$$\gamma_i \left(1-\mu_i\right) \geq \alpha_i + \Delta_i \tag{12}$$

商超总收益模型改进后如下:

$$\text{求 max } w$$

$$st. \begin{cases} w = \sum_i^n \left[\alpha_i(x_i(1+r_i) - d_i) - (\gamma_i - \alpha_i)d_i\right] \\ \gamma_i \left(1-\mu_i\right) \geq \alpha_i + \Delta_i \\ \sigma_i = \frac{(\sum \alpha_j x_j)(1+r_i) - \sum \gamma_j d_j}{\sum \gamma_j d_j} \\ i \in \{\text{选中的单品}\} \\ \gamma_i \geq 2.5 \end{cases} \tag{13}$$

5.3.2 问题三模型求解

将模型以 Δ 为未知量，对 $\gamma_i \left(1-\mu_i\right) \geq \alpha_i + \Delta_i$ 取等号（容易证明，取等号时出现极值），原模型变为了常规的线性规划方程。将方程导入 SPSSpro 软件，使用单纯形法将模型求解，得到各单品 γ 和 max w。

所有选择的单品已在支撑材料中给出。

部分单品 γ 如表 7：

表 **7**　求解所得部分单品 γ

单品名称	γ(千克)
金针菇 (盒)	15.206
海鲜菇 (包)	8.1448
小皱皮 (份)	13.998
小米椒 (份)	29.399
···	···

经求解，商超最大收益为：670.05 元。部分单品 x 如表 8:

表 **8**　求解所得部分单品 x

单品名称	x(元)
金针菇 (盒)	2.268
海鲜菇 (包)	2.798
小皱皮 (份)	3.876
小米椒 (份)	4.272
···	···

所有问题三数据已在支撑材料中给出。

5.4 问题四

为更好地制定蔬菜商品的补货和定价决策，除对过去几年的历史销售数据、主要成本及对应损耗率进行分析外，商超还需采集每个蔬菜品类及单品的库存数据、过程性成本数据、竞争对手相关数据以及客户反馈数据等进行分析，以不断满足客户需求，使商超收益最大化。

首先，需采集每种商品的库存数据。通过了解每个蔬菜品类及单品的库存数量、库存周转率、库存价值等库存数据，可以在有效监控和管理库存水平的同时，更准确地决定是否需要对某类或某种蔬菜进行补货及对应需要补货的数量。补货数量的决策关乎蔬菜类商品的大部分收益能力，因此至关重要。

第二，需要采集每种商品的过程性成本数据。通过了解每个蔬菜品类及单品的运输成本、包装成本、存储成本等过程性成本数据，商超可以明确各种蔬菜的总成本及对应

成本结构，计算商品的毛利润，为制定合理的定价策略、评估商品的盈利能力提供依据。

第三，需要采集客户的反馈数据。通过对客户调研，了解不同细分群体的购买力、购买偏好及购买行为等，可以更准确识别目标客户群体，调整产品组合、定价策略及销售渠道等，提高客户的满意度与忠诚度。例如，通过对客户调研，了解大部分客户的群体画像，有助于知晓客户群体的组成，进而对客户的行为模式做出分析。

这些数据的采集和分析将为商超提供更全面、准确的信息，有助于制定更具针对性和效益的补货和定价策略，以实现商超的收益最大化目标。商超可以借助现代数据分析技术来处理和利用这些数据，从而在激烈的市场竞争中保持竞争力。

六、 模型检验

6.1 检验 ARIMA 模型的对参数设置的灵敏度

试探究通过已有的 α 数据对 ARIMA 模型的预测性能进行探究检验。让 ARIMA 模型预测 2023 年 6 月 1 日至 2023 年 6 月 30 日的 $\alpha_{西兰花}$，参与训练的数据分别设置为预测当天前 10、30、50 天的数据。绘制比较图像如下：

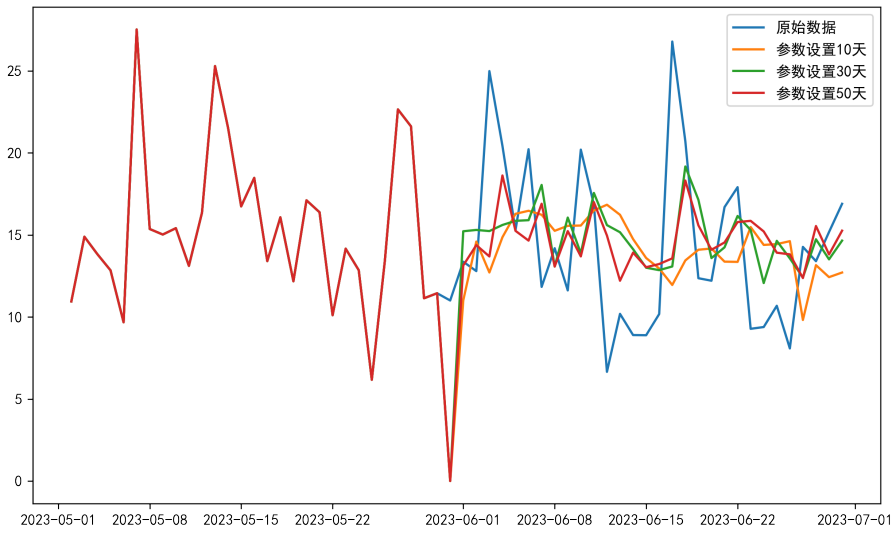

图 **19**　原始数据与参数设置为 **10** 天、**30** 天、**50** 天预测情况

观察发现，参数设置为 30 天与 50 天的数据预测情况较为相近，说明时间更加久远的数据对模型的影响权重正逐渐减弱，而且两者变化趋势相较于真实数据有滞后性，这也是所有以时间序列分析为基的预测模型所带有的特征，但变化幅度与真实而言更为保守，但总体上与真实数据基本上是在同一回归线附近徘徊。而参数设置为 10 天的预测数据，除滞后性外在变化幅度上的保守程度更甚。

变化趋势的保守是符合现实情况，在现实生活中，由于太多的不可控不可察因素，商家无法做到对所有的事情完全知悉，导致了其在销量预测上会更加保守，且变化趋势

会参考前几天的走势，无法即时对走势的改变做出预知，导致了滞后。

6.2 ARIMA 模型与线性回归模型的性能比较

试比较线性回归模型与 ARIMA 模型对 $\alpha_{西兰花}$ 的预测性能，同样预测 6 月 1 日至 30 日的 $\alpha_{西兰花}$，使用 SPSSpro 软件训练线性回归模型并绘制图像如下图：

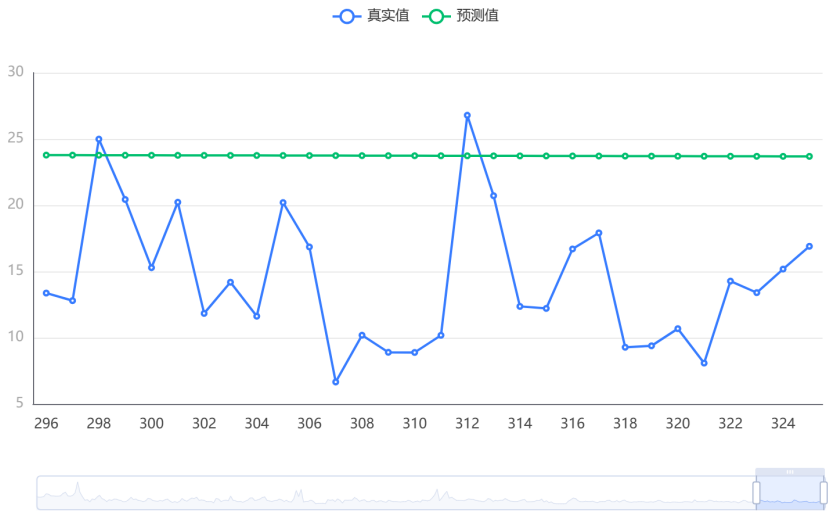

图 **20**　线性回归模型

并绘制参数设置为 30 天时的 ARIMA 模型预测图像：

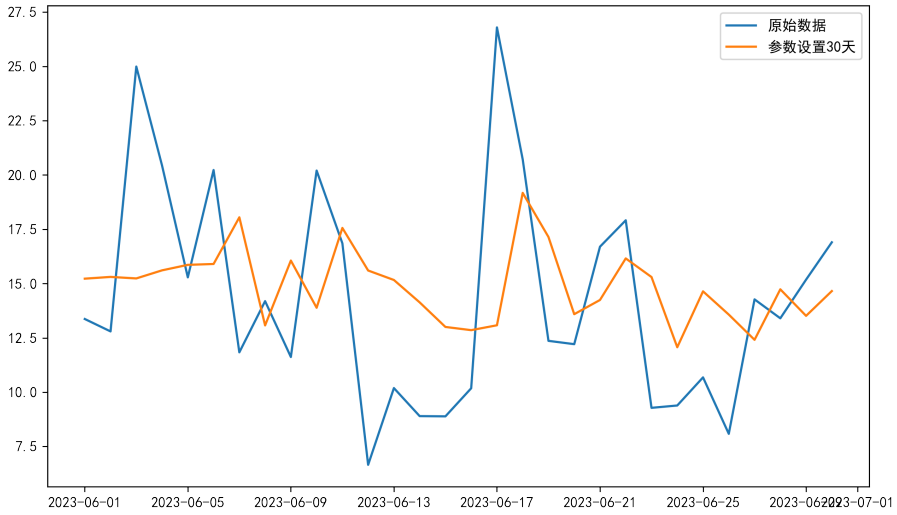

图 **21**　**ARIMA** 模型

肉眼可见的差异，线性回归模型对 $\alpha_{西兰花}$ 计算求得的回归线几乎完全脱离真实数据，其预测作用在本题上几乎不起到作用。而 ARIMA 模型则与真实数据结合更为紧密，使用其预测数据是可行有效的。

七、 模型评价

7.1 优点与创新

1. 在使用 Spearman 相关性分析前，合理探讨该题数据的大致分布是否令 Spearman 相关分析更为合理。
2. 引入 ARIMA 模型这一强大的时间数据预测模型，能够更为合理的利用时期和销量的关联，在离预测时间点较近的数据拥有更高的加权，提高了模型预测的精确性。
3. 模型构建过程引入自行定义的折扣系数以体现打折对总收益的影响，使模型更接近现实。

7.2 不足之处

1. 从所给附件中搜集的数据信息偏少。
2. 简单的模型有弊有利，坏处在于其模型对于现实情况的拟合不一定好。
3. 缺少自我创新，没有闪光点。

参考文献

[1] 司守奎、孙玺菁. 数学建模算法与应用[M]. 国防工业出版社, 2011.

[2] 周雨蕙, 唐旗英. 基于 ARIMA 的多元线性回归模型的全球气候变暖分析[J]. Modeling and Simulation, 2023, 12: 1257.

附录 A

支撑材料

1. 第二第三问求解相关.py
2. 折扣率利润率相关.py
3. 题目一数据清洗与绘图.ipynb
4. 题目一计算日销量.ipynb
5. 题目二计算.ipynb
6. 题目二批发价预测.ipynb
7. 题目二销量预测.ipynb
8. 题目三.ipynb
9. ARIMA 测试.ipynb
10. 日销量.xlsx

11. 销量预测.xlsx
12. 折扣力度.xlsx
13. 批发价预测.xlsx
14. 品类日销量.xlsx
15. 相关系数代表.xlsx
16. 相关系数.xlsx
17. 利润率.xlsx
18. 用于计算.xlsx
19. output1.xlsx
20. 单品利润率.xlsx
21. 筛除单品中盈利单品
22. output2.xlsx
23. 单品定价.xlsx

源代码

Listing 1: 题目一数据清洗与绘图

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn

df1 = pd.read_excel("附件1.xlsx")
df1["单品编码"] = df1["单品编码"].astype(str)
df1["分类编码"] = df1["分类编码"].astype(str)
df2 = pd.read_excel("附件2.xlsx")
df2["单品编码"] = df2["单品编码"].astype(str)
df = pd.merge(df1,df2,on="单品编码")
df = df[df['销售类型'] != '退货']#删除退货的数据
df['扫码销售时间'] = pd.to_timedelta(df['扫码销售时间'])
df["销售时间"]= df["销售日期"]+df["扫码销售时间"]
df = df.sort_values(by="销售时间").reset_index(drop=True)

# %%
all_vegetables=pd.Series(df["单品编码"].unique())
all_kinds = pd.Series(df["分类编码"].unique())
all_vegetables.name = "单品编码"
all_kinds.name = "分类编码"

vegetables_sold = pd.DataFrame(columns=["单品编码","单品销售量"])
vegetables_sold["单品编码"]= all_vegetables
vegetables_sold["单品销售量"] = 0
```

```
# %%
def compute_sold(df,all,sold):
    for i in range(len(all)):
        sold.iloc[i,1]=df.loc[(df["单品编码"] == all[i]),["销量(千克)"]].sum()
    return sold
#计算单品销售量
vegetables_sold = compute_sold(df,all_vegetables,vegetables_sold)
vegetables_sold

# %%
vegetables_sold=vegetables_sold.sort_values(by="单品销售量",ascending=False).reset_index(drop=True)
vegetables_sold

# %%
def clear(percent,sum,vegetables_sold):
    cur=0
    i=0
    while cur < percent:
        cur+=vegetables_sold.loc[i,"单品销售量"]/sum
        i+=1
    return vegetables_sold.iloc[:i,:]

# %%
vegetables_sold = clear(0.95,vegetables_sold.loc[:,"单品销售量"].sum(),vegetables_sold)
vegetables_sold

# %%
condition = df['单品编码'].isin(vegetables_sold["单品编码"].unique())

# 根据条件删除不符合条件的行
df_filtered = df[condition]
df_filtered

# %%
all_vegetables=pd.Series(df_filtered["单品编码"].unique())
all_kinds = pd.Series(df_filtered["分类编码"].unique())

kinds_sold = pd.DataFrame(columns=["分类编码","单类销售量"])
kinds_sold["分类编码"]= all_kinds
kinds_sold["单类销售量"] = 0
kinds_sold

# %%
def compute_sold_2(df,all,sold):
    for i in range(len(all)):
        sold.iloc[i,1]=df.loc[(df["分类编码"] == all[i]) ,["销量(千克)"]].sum()
```

```
    return sold
#计算分类销售量

# %%
kinds_sold = compute_sold_2(df,all_kinds,kinds_sold)
kinds_sold

# %%
df_filtered.to_excel("数据清洗后.xlsx",index=False)

# %%
def figure_Day(df,df_time):
    dateRange = df["销售日期"].unique()
    one_day = np.timedelta64(1, 'D')
    for i in range(len(dateRange)):
        for j in range(df_time.shape[1]-1):
            condition1 = (df["销售时间"] >= dateRange[i]) & (df["销售时间"] <
                dateRange[i]+one_day) & (df["单品编码"] == df_time.columns[j+1])
            df_time.iloc[i,j+1]=df.loc[condition1,"销量(千克)"].sum()
    return df_time

# %%
dateRange = df["销售日期"].unique()
df_day = pd.DataFrame(dateRange,columns=["Date"])
for name in all_vegetables:
    df_day[name] = 0
df_day = figure_Day(df_filtered,df_day)
df_day

# %%
all_vegetables=pd.Series(df_filtered["单品名称"].unique())
df_day_copy = df_day
new_dict = {}
for code,items in zip(df_filtered["单品编码"].unique(),df_filtered["单品名称"].unique()):
    new_dict[code]=items
new_dict
df_day_copy.rename(columns=new_dict, inplace=True)
df_day_copy

# %%
df_day_copy.to_excel("日销量.xlsx",index=False)

# %%

#df_day['Date'] = pd.to_datetime(df_day['Date'])
R=range(df_day.shape[0])
for j in range(1,df_day.shape[1]):
```

```
# 创建图表
    fig, ax = plt.subplots(figsize=(10, 6), dpi=300,facecolor='white') # 设置图表的尺寸和分辨率

# 设置字体以支持中文
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False

# 绘制曲线图，这里使用自定义的R变量

    ax.plot(df_day['Date'].iloc[R], df_day[df_day.columns[j]].iloc[R], label=df_day.columns[j])

# 设置X轴刻度
    custom_xticks = df_day['Date'].iloc[R][::10] # 自定义X轴刻度间隔
    ax.set_xticks(custom_xticks)
    ax.tick_params(axis='x', rotation=90) # 设置X轴刻度标签旋转角度

# 添加图例
    ax.legend(loc='upper right')
    plt.xlabel("日期")
    plt.ylabel("日销量/千克")
# 显示图形
    plt.tight_layout() # 自动调整布局以防止标签被裁剪
    plt.savefig("outPNG/{}.png".format(df_day.columns[j]), dpi=300) # 保存图像为文件（可选）
    plt.close()


# %%
from pandas.plotting import scatter_matrix
scatter_matrix(df_day.iloc[:, 1:49], figsize=(15, 15), marker='o', hist_kwds={'bins': 20},
    s=60, alpha=0.8,cmap=mglearn.cm3)

# %%
grouped = df_filtered.groupby("分类名称")
df_day_copy = df_day.copy()
for name, group_data in grouped:
    # 获取同一分类下的单品名称列表
    single_products = group_data["单品名称"].unique()

    # 在df_day_copy中添加一列，该列的值为同一分类下所有单品的销售量之和
    df_day_copy[name] = df_day_copy[single_products].sum(axis=1)


# %%
df_day_copy.iloc[:, [0] + list(range(-1, -7, -1))].to_excel("品类日销量.xlsx",index=False)

# %%
# 创建一个白色背景的坐标系
```

```
fig, ax = plt.subplots(figsize=(15, 15))
ax.set_facecolor('white') # 设置背景颜色为白色

# 绘制散点矩阵
scatter_matrix = pd.plotting.scatter_matrix(df_day_copy.iloc[:, list(range(-1, -7, -1))],
    ax=ax,
                                        marker='o', hist_kwds={'bins': 20}, s=60, alpha=0.8,
                                            cmap=mglearn.cm3)

# 如果需要，可以进一步自定义散点矩阵的样式
# 例如，可以设置对角线上的图形，添加标签等
for ax in scatter_matrix.ravel():
    ax.set_xlabel(ax.get_xlabel(), fontsize=12)
    ax.set_ylabel(ax.get_ylabel(), fontsize=12)

plt.show()

# %%
vegetables_sold['单品名称'] = vegetables_sold['单品编码'].map(new_dict)
vegetables_sold.iloc[:,[1,2]].to_excel("选择的单品.xlsx",index=False)

# %%
plt.figure(dpi=300)
for i in range(1, 7):
    plt.plot(df_day_copy['Date'].iloc[:], df_day_copy[df_day_copy.columns[-i]].iloc[:],
        label=df_day_copy.columns[-i])
    plt.legend(loc='upper right',fontsize=10)
# 设置图像分辨率为 300 dpi
    plt.savefig("outPNG/{}.png".format(df_day_copy.columns[-i]), facecolor='white')
    plt.clf()
```

Listing 2: 题目一计算日销量

```
# %%
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
#读取附件一
df1 = pd.read_excel("附件1.xlsx")
df1["单品编码"] = df1["单品编码"].astype(str)
df1["分类编码"] = df1["分类编码"].astype(str)
#读取附件二
df2 = pd.read_excel("附件2.xlsx")
df2["单品编码"] = df2["单品编码"].astype(str)
#按单品编码合并
df = pd.merge(df2,df1,on="单品编码")
```

```
df = df[df['销售类型'] != '退货']#删除退货的数据
#计算销售具体时间
df['扫码销售时间'] = pd.to_timedelta(df['扫码销售时间'])
df["销售时间"]= df["销售日期"]+df["扫码销售时间"]
#df=df.drop(columns=["销售日期","扫码销售时间"])
df=df.sort_values(by="销售时间").reset_index(drop=True)


# %%
all_vegetables=pd.Series(df["单品名称"].unique())
df

# %% [markdown]
# #计算各单品日销量

# %%
dateRange=df["销售日期"].unique()


# %%
#def figure_Day(df,df_time,dateRange):
#    one_day = np.timedelta64(1, 'D')
#    for i in range(len(dateRange)):
#        for j in range(df_time.shape[1]-1):
#            condition1 = (df["销售时间"] >= dateRange[i]) & (df["销售时间"] <
    dateRange[i]+one_day) & (df["单品名称"] == df_time.columns[j+1])
#            df_time.iloc[i,j+1]=df.loc[condition1,"销量(千克)"].sum()
#    return df_time

# %%
#df_day = pd.DataFrame(dateRange,columns=["Date"])
#for name in all_vegetables:
#    df_day[name] = 0
#df_day = figure_Day(df,df_day,dateRange)
#df_day

# %%
def figure_Day(df, df_time, dateRange):
    # 将销售时间列转换为 datetime64 数据类型
    df['销售时间'] = pd.to_datetime(df['销售时间'])

    # 将日期范围添加到数据框中
    df['日期范围'] = pd.cut(df['销售时间'], bins=dateRange, right=False, labels=False)

    # 使用 pivot_table 执行汇总操作
    pivot_df = pd.pivot_table(df, index=['日期范围', '单品名称'], values='销量(千克)',
        aggfunc='sum').unstack(fill_value=0)
```

```
    # 将结果填充到 df_time 中
    df_time.iloc[:, 1:] = pivot_df.values

    # 删除日期范围列
    df.drop(columns=['日期范围'], inplace=True)

    return df_time

# %%
```

Listing 3: 题目二计算

```
# %%
import pmdarima as pm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn

df1=pd.read_excel("批发价预测.xlsx")
df1 = df1.add_prefix('批发_')
df2=pd.read_excel("销量预测.xlsx")
df2 = df2.add_prefix('销量_')
df=pd.concat([df1, df2.iloc[:,1:]], axis=1)
df = df.rename(columns={'批发_Date': 'Date'})
df_copy=df.T.copy()

# %%


# %%
df_filtered = pd.read_excel("数据清洗后.xlsx")
grouped = df_filtered.groupby("分类名称")
new_dict={}
for name, group_data in grouped:
    # 获取同一分类下的单品名称列表
    single_products = group_data["单品名称"].unique()
    for j in single_products:
        new_dict["批发_{}".format(j)]= name
        new_dict["销量_{}".format(j)]= name



# %%
df_copy['分类名称'] = df_copy.index.map(new_dict)
df_copy
```

```
# %%
new_dict={}
for name, group_data in grouped:
    # 获取同一分类下的单品名称列表
    single_products = group_data["单品名称"].unique()
    for j in single_products:
        new_dict[j]= name

# %%
df_lirun = pd.read_excel("利润率.xlsx")
df_lirun

# %%
lirun_dict={}
for i in range(6):
    lirun_dict[df_lirun.iloc[i,0]]=df_lirun.iloc[i,3]

# %%
df_copy['利润率'] = df_copy['分类名称'].map(lirun_dict)
df_copy

# %%
df_zhesun = pd.read_excel("附件4s.xlsx")
df_zhesun

# %%
zhesun_dict={}
for i in range(251):
    zhesun_dict["批发_"+df_zhesun.iloc[i,1]]=df_zhesun.iloc[i,2]
    zhesun_dict["销量_"+df_zhesun.iloc[i,1]]=df_zhesun.iloc[i,2]

# %%
df_copy['损耗率(%)'] = df_copy.index.map(zhesun_dict)
df_copy

# %%
df_copy.to_excel("用于计算.xlsx")

# %%
```

Listing 4: 题目二批发价预测

```
# %%
import pmdarima as pm
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
import mglearn

df = pd.read_excel("附件3.xlsx")
df

# %%
df2 = pd.read_excel("数据清洗后.xlsx")
df2

# %%
new_dict = {}
for code,items in zip(df2["单品编码"].unique(),df2["单品名称"].unique()):
    new_dict[code]=items
new_dict
# 使用 .map() 方法将编码映射为名字并添加为新列
df['单品名称'] = df['单品编码'].map(new_dict)
df.dropna(inplace=True)
df.reset_index(drop=True,inplace=True)

# %%
data = {'Date': ['2023-07-01', '2023-07-02', '2023-07-03', '2023-07-04', '2023-07-05',
     '2023-07-06','2023-07-07']}

df_forcast = pd.DataFrame(data,columns=["Date"])

# %%
grouped=df.groupby("单品名称")
fig, ax = plt.subplots(figsize=(10, 6), dpi=300,facecolor='white')
#for name in grouped["单品名称"].unique():
#    group_A = grouped.get_group(name)
#    fig, ax = plt.subplots(figsize=(10, 6), dpi=300,facecolor='white')
#    plt.rcParams['font.sans-serif'] = ['SimHei']
#    plt.rcParams['axes.unicode_minus'] = False
#    ax.plot(grouped['Date'].loc[name], grouped[name].loc["批发价"], label=name)
#    ax.legend(loc='upper right')
#    ax.tick_params(axis='x', rotation=90)
#    plt.tight_layout() # 自动调整布局以防止标签被裁剪
#    plt.savefig("outPNG2/{}.png".format(name), dpi=300) # 保存图像为文件（可选）
#    plt.close()
for name, group_data in grouped:
    fig, ax = plt.subplots(figsize=(10, 6), dpi=300, facecolor='white')
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    ax.plot(group_data['日期'], group_data['批发价格(元/千克)'], label=name)
    ax.legend(loc='upper right')
    ax.tick_params(axis='x', rotation=90)
```

```
    plt.xlabel("日期")
    plt.ylabel("批发价格(元/千克)")
    plt.tight_layout() # 自动调整布局以防止标签被裁剪
    plt.savefig("outPNG2/{}.png".format(name), dpi=300) # 保存图像为文件（可选）
    plt.close()

# %%
data = {'Date': ['2023-07-01', '2023-07-02', '2023-07-03', '2023-07-04', '2023-07-05',
     '2023-07-06','2023-07-07']}

df_forcast = pd.DataFrame(data,columns=["Date"])

# %%

num_periods = 7

for name, group_data in grouped:
    time_series_data = group_data['批发价格(元/千克)']
# 自动选择最佳ARIMA模型
    model = pm.auto_arima(time_series_data, seasonal=False)

# 预测未来值
    forecast = model.predict(n_periods=num_periods)
    df_forcast[name]=forecast.tolist()

# %%
df_forcast.to_excel("批发价预测.xlsx",index=False)

# %%
```

Listing 5: 题目二销量预测

```
# %%
import pmdarima as pm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn

df = pd.read_excel("日销量.xlsx")

# %%
df.set_index('Date', inplace=True)
df

# %%
data = {'Date': ['2023-07-01', '2023-07-02', '2023-07-03', '2023-07-04', '2023-07-05',
```

```
    '2023-07-06','2023-07-07']}

df_forcast = pd.DataFrame(data,columns=["Date"])
# 将"Date"列转换为日期时间类型
#df_forcast['Date'] = pd.to_datetime(df_forcast['Date'])

# 将"Date"列设置为索引
#df_forcast.set_index('Date', inplace=True)



df_forcast

# %%
num_periods = 7

for i in range(df.shape[1]):
    time_series_data = df.iloc[:,i]
# 自动选择最佳ARIMA模型
    model = pm.auto_arima(time_series_data, seasonal=False)

# 预测未来值
    forecast = model.predict(n_periods=num_periods)
    df_forcast[df.columns[i]]=forecast.tolist()


# %%
df_forcast.to_excel("销量预测.xlsx",index=0)

# %%
```

Listing 6: 题目三

```
# %%
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn

df=pd.read_excel("数据清洗后.xlsx")

# %%
df_forcast = pd.read_excel("销量预测.xlsx")
df_forcast

# %%
df_zhesun=pd.read_excel("附件4s.xlsx")
```

```
map_dict={}
for name,num in zip(df_zhesun.iloc[:,1],df_zhesun.iloc[:,2]):
    map_dict[name]=num
map_dict

# %%
df_solds = df_forcast.copy()

# %%
for i in range(df_forcast.shape[0]):
    for j in range(1,df_forcast.shape[1]):
        df_forcast.iloc[i,j]=df_forcast.iloc[i,j]/(1-map_dict[df_forcast.columns[j]]/100)
df_forcast

# %%
df_forcast=df_forcast.T



# %%
df_forcast

# %%
df_forcast["预测销量"]=df_solds.iloc[0,:]
df_forcast

# %%
df_forcast.rename(columns={0:"补货量"}, inplace=True)
df_forcast

# %%
df_forcast.reset_index(drop=False, inplace=True)
df_forcast.rename(columns={"index":"单品名称"}, inplace=True)
df_forcast

# %%
df_forcast=df_forcast.iloc[:,[0,1,-1]]
df_forcast

# %%
df_forcast=df_forcast.iloc[1:,:]

# %%
df_forcast["是否>=2.5kg"]=0
for i in range(df_forcast.shape[0]):
    if df_forcast.iloc[i,1] >= 2.5:
        df_forcast.iloc[i,-1] =1
df_forcast
```

```
# %%
df_forcast.reset_index(drop=True, inplace=True)
df_forcast

# %%
df_day=pd.read_excel("日销量.xlsx")
df_day

# %%
df_day=df_day.iloc[-7:,:]
df_day

# %%
for name in df_day.columns[1:]:
    if df_day.loc[:, name].sum() == 0:
        df_day.drop(columns=[name], inplace=True)
df_day

# %%
left_vege = df_day.columns[1:]
left_vege

# %%
df_forcast_cl = df_forcast[df_forcast['单品名称'].isin(left_vege.to_list())]
df_forcast_cl

# %%
df_forcast_cl.reset_index(drop=True, inplace=True)

# %%
df_forcast_cl

# %%
df_forcast_cl.iloc[:,-1].sum()

# %%
chosen_vege = df_forcast_cl.loc[df_forcast_cl["是否>=2.5kg"] == 1,"单品名称"]

# %%
chosen_vege = chosen_vege.to_list()
chosen_vege

# %%
category = pd.read_excel("附件1.xlsx")
grouped = category.groupby("分类名称")
new_dict = {}
```

```
for name,group_data in grouped:
    single_products = group_data["单品名称"].unique()
    for key in single_products:
        new_dict[key] = name
new_dict

# %%
df_forcast_cl["分类名称"]=df_forcast_cl.loc[:,"单品名称"].map(new_dict)
df_forcast_cl

# %%
grouped = df_forcast_cl.groupby("分类名称")
new_dict = {}
for name,group_data in grouped:
    new_dict[name]=group_data.loc[group_data["是否>=2.5kg"] == 1,"预测销量"].sum()
new_dict

# %%
grouped = category.groupby("分类名称")
cate = {}
for name,group_data in grouped:
    single_products = group_data["单品名称"].unique()
    for key in single_products:
        cate[key] = name
cate

# %%
old_dict =
    {'水生根茎类':0.1479,'花叶类':0.4241,'花菜类':0.0555,'茄类':0.0648,'辣椒类':0.2629,'食用菌':0.0449}#手动计算占

# %%
ssum=0
for key in df_forcast_cl["分类名称"].unique():
    ssum+=new_dict[key]
ssum

# %%
new_percent={}
for key in df_forcast_cl["分类名称"].unique():
    new_percent[key] = new_dict[key]/ssum
new_percent

# %%
dict1 = old_dict
dict2 = new_percent
```

```
from matplotlib.font_manager import FontProperties

# 指定中文字体
font = FontProperties(fname="C:/Windows/Fonts/msyh.ttc", size=12)
plt.rcParams['font.sans-serif'] = ['SimHei'] # 替换为你选择的中文字体
plt.rcParams['axes.unicode_minus'] = False # 用于正常显示负号
plt.figure(figsize=(8, 6), dpi=300, facecolor='white')
# 确定统一的键顺序（这里按照 dict1 的键顺序）
keys = dict1.keys()

# 提取字典中的值
values1 = [dict1[key] for key in keys]
values2 = [dict2[key] for key in keys]

# 设置条形图的位置
x = range(len(keys))

# 绘制条形图
plt.bar(x, values1, width=0.4, label='前7日占比', align='center')
plt.bar(x, values2, width=0.4, label='预测占比', align='edge')

# 设置 x 轴标签，并使用中文字体
plt.xticks(x, keys, fontproperties=font)

# 添加图例
plt.legend()

# 添加标题和标签，并使用中文字体
plt.title('占比统计', fontproperties=font)
plt.xlabel('分类名称', fontproperties=font)
plt.ylabel('占比', fontproperties=font)



# %%
old_df = pd.read_excel("品类日销量.xlsx")
old_df = old_df.iloc[-7:,:]
old_df

# %%
mean_amount={}
for i in range(1,len(old_df.columns)):
    mean_amount[old_df.columns[i]]=old_df.iloc[-1,i]
mean_amount

# %%
grouped = df_forcast_cl.groupby("分类名称")
new_dict_nodelete = {}
```

```
for name,group_data in grouped:
    new_dict_nodelete[name]=group_data.loc[:,"预测销量"].sum()
new_dict_nodelete

# %%
dict1 = mean_amount
dict2 = new_dict
dict3 = new_dict_nodelete
plt.figure(figsize=(8, 6), dpi=300, facecolor='white')
# 确定统一的键顺序（这里按照 dict1 的键顺序）
keys = dict1.keys()

# 提取字典中的值
#values1 = [dict1[key] for key in keys]
values2 = [dict2[key] for key in keys]
values3 = [dict3[key] for key in keys]
# 设置条形图的位置
x = range(len(keys))

# 绘制条形图
plt.figure(figsize=(10, 6), dpi=300, facecolor='white')
#plt.bar(x, values1, width=0.2, label='某日', align='center')
plt.bar([i + 0.2 for i in x], values2, width=0.2, label='预测', align='center')
plt.bar([i + 0.4 for i in x], values3, width=0.2, label='非删减预测', align='center')

# 设置 x 轴标签
plt.xticks([i + 0.2 for i in x], keys, rotation=45, fontsize=8)

# 添加图例
plt.legend()

# 添加标题和标签
plt.title('销量统计')
plt.xlabel('分类名称')
plt.ylabel('销量/千克')

# 显示图形
plt.tight_layout()
plt.show()

# %%
df_forcast_cl.to_excel("限制选择的单品.xlsx",index=False)

# %%
new_dict["茄类"]+=2.1787440176
new_dict["食用菌"]+=1.9316905093
```

```
dict1 = mean_amount
dict2 = new_dict
dict3 = new_dict_nodelete
plt.figure(figsize=(8, 6), dpi=300, facecolor='white')
# 确定统一的键顺序（这里按照 dict1 的键顺序）
keys = dict1.keys()

# 提取字典中的值
#values1 = [dict1[key] for key in keys]
values2 = [dict2[key] for key in keys]
values3 = [dict3[key] for key in keys]
# 设置条形图的位置
x = range(len(keys))

# 绘制条形图
plt.figure(figsize=(10, 6), dpi=300, facecolor='white')
#plt.bar(x, values1, width=0.2, label='某日', align='center')
plt.bar([i + 0.2 for i in x], values2, width=0.2, label='预测', align='center')
plt.bar([i + 0.4 for i in x], values3, width=0.2, label='非删减预测', align='center')

# 设置 x 轴标签
plt.xticks([i + 0.2 for i in x], keys, rotation=45, fontsize=8)

# 添加图例
plt.legend()

# 添加标题和标签
plt.title('销量统计')
plt.xlabel('分类名称')
plt.ylabel('销量/千克')

# 显示图形
plt.tight_layout()
plt.show()

# %%
```

Listing 7: 第二第三问求解相关

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

# In[12]:


import pandas as pd
data = pd.read_excel('D:/qq/聊天文件/销量预测.xlsx')
```

```
print(data.head())


# In[9]:


data1 = pd.read_excel('D:/qq/聊天文件/附件1.xlsx')
print(data1.head())


# In[7]:




# 将表二的蔬菜名称和分类信息作为字典
classification_dict = dict(zip(data1['单品名称'], data1['分类名称']))

# 修改表一的列名
data.rename(columns=classification_dict, inplace=True)

# 打印修改后的DataFrame
print(data)


# In[13]:


data2 = pd.read_excel('D:/qq/聊天文件/批发价预测.xlsx')
print(data2.head())


# In[14]:


combined_df = pd.concat([data, data2], ignore_index=True)

# 打印连接后的数据
print(combined_df)


# In[31]:


import pandas as pd
data = pd.read_excel('D:/qq/聊天文件/用于计算.xlsx')
print(data.head())
```

```
# In[62]:


data['1日进货量']=data['1日销量']/(1-data['损耗率(%)']/100)
data['2日进货量']=data['2日销量']/(1-data['损耗率(%)']/100)
data['3日进货量']=data['3日销量']/(1-data['损耗率(%)']/100)
data['4日进货量']=data['4日销量']/(1-data['损耗率(%)']/100)
data['5日进货量']=data['5日销量']/(1-data['损耗率(%)']/100)
data['6日进货量']=data['6日销量']/(1-data['损耗率(%)']/100)
data['7日进货量']=data['7日销量']/(1-data['损耗率(%)']/100)
data


# In[33]:


data['1日销售额']=data['1日成本']*(1+data['利润率'])
data['2日销售额']=data['2日成本']*(1+data['利润率'])
data['3日销售额']=data['3日成本']*(1+data['利润率'])
data['4日销售额']=data['4日成本']*(1+data['利润率'])
data['5日销售额']=data['5日成本']*(1+data['利润率'])
data['6日销售额']=data['6日成本']*(1+data['利润率'])
data['7日销售额']=data['7日成本']*(1+data['利润率'])
data


# In[34]:


df = pd.DataFrame(data)


# In[63]:


result = df.groupby('分类名称').agg({'1日销量': 'sum', '1日销售额': 'sum', '1日成本':
    'sum','1日进货量': 'sum','2日销量': 'sum', '2日销售额': 'sum','2日成本': 'sum','2日进货量':
    'sum','3日销量': 'sum', '3日销售额': 'sum','3日成本': 'sum','3日进货量': 'sum','4日销量':
    'sum', '4日销售额': 'sum','4日成本': 'sum','4日进货量': 'sum','5日销量': 'sum','5日销售额':
    'sum','5日成本': 'sum','5日进货量': 'sum','6日销量': 'sum', '6日销售额': 'sum','6日成本':
    'sum','6日进货量': 'sum','7日销量': 'sum', '7日销售额': 'sum','7日成本': 'sum','7日进货量':
    'sum',}).reset_index()
result
```

```
# In[64]:


import pandas as pd
data2 = pd.read_excel('D:/qq/折扣力度.xlsx')
data2


# In[65]:


data2 = pd.DataFrame(data2)
result2 = pd.merge(result, data2, on=['分类名称'], how='inner')
result2


# In[66]:


result2['1日定价']=result2['1日销售额']/(result2['1日销量']*(1+result2['折扣系数']))
result2['2日定价']=result2['2日销售额']/(result2['2日销量']*(1+result2['折扣系数']))
result2['3日定价']=result2['3日销售额']/(result2['3日销量']*(1+result2['折扣系数']))
result2['4日定价']=result2['4日销售额']/(result2['4日销量']*(1+result2['折扣系数']))
result2['5日定价']=result2['5日销售额']/(result2['5日销量']*(1+result2['折扣系数']))
result2['6日定价']=result2['6日销售额']/(result2['6日销量']*(1+result2['折扣系数']))
result2['7日定价']=result2['7日销售额']/(result2['7日销量']*(1+result2['折扣系数']))
result2


# In[67]:


result2.to_excel('D:/qq/聊天文件/output2.xlsx', index=False)


# In[1]:


import pandas as pd
data = pd.read_excel('D:/qq/聊天文件/限制选择的单品.xlsx')
data


# In[39]:


data1 = pd.read_excel('D:/qq/聊天文件/批发价1号预测.xlsx')
```

```
data1
data2 = pd.read_excel('D:/qq/附件四修改版.xlsx')
data2


# In[43]:


data = pd.DataFrame(data)
data1 = pd.DataFrame(data1)
data2 = pd.DataFrame(data2)
result = pd.merge(data, data1, on=['单品名称'], how='left')
result1 = pd.merge(result, data2, on=['单品名称'], how='left')
result1


# In[45]:


result1['成本']=result1['2023-07-01']/(1-result1['损耗率(%)']/100)
result1


# In[48]:


result1 = result1.sort_values(by='成本', ascending=True)
result1


# In[49]:


result1.to_excel('D:/qq/聊天文件/问题三.xlsx', index=False)


# In[61]:


result2 = pd.read_excel('D:/qq/聊天文件/问题三 (1).xlsx')
result2


# In[62]:


data = pd.read_excel('D:/qq/聊天文件/单品利润率.xlsx')
```

```
data
data1 = pd.read_excel('D:/qq/折扣力度.xlsx')
data1


# In[64]:


result2 = pd.DataFrame(result2)
data = pd.DataFrame(data)
data1 = pd.DataFrame(data1)
result2 = pd.merge(result2, data, on=['单品名称'], how='left')
result2 = pd.merge(result2, data1, on=['分类名称'], how='left')
result2


# In[66]:


result2['定价']=result2['成本']*(1+result2['利润率_x'])/(1+result2['折扣系数_x'])
result2


# In[67]:


result2.to_excel('D:/qq/聊天文件/单品定价.xlsx', index=False)


# In[ ]:
```

Listing 8: 折扣率利润率相关

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

# In[ ]:


import pandas as pd


# In[5]:


excel_file_path = 'D:/qq/聊天文件/数据清洗后.xlsx'
```

```
# 使用pandas的read_excel方法导入Excel文件
data = pd.read_excel(excel_file_path)


# In[6]:


print(data.head())


# In[7]:


sorted_data = data.sort_values(by='销售单价(元/千克)')


# In[8]:


print(sorted_data.head())


# In[9]:


grouped_data = data.groupby('分类名称')


# In[10]:


for category, group in grouped_data:
    # 使用category和group进行操作，例如打印分类名和数据
    print(f"Category: {category}")
    print(group)


# In[11]:


discount_ratio = grouped_data['是否打折销售'].apply(lambda x: x.map({'是': True, '否':
    False}).mean())

# 打印每个组中的打折比例
print(discount_ratio)
```

```
# In[12]:


import pandas as pd

# 假设您已经按照分类名称分组并存储在grouped_data中

# 创建一个新的DataFrame来存储结果
result_data = pd.DataFrame()

# 计算每个分类内是否打折的数量，并将结果转换为布尔值
result_data['是否打折销售'] = grouped_data['是否打折销售'].transform(lambda x: x.eq('是').any())

# 计算每个分类内的折扣率
def calculate_discount_rate(group):
    if group['是否打折销售'].any():
        discount_mean = group.loc[group['是否打折销售'], '销售单价(元/千克)'].mean()
        group['折扣率'] = 1 - group['销售单价(元/千克)'] / discount_mean
    else:
        group['折扣率'] = 0 # 如果没有打折数据，则折扣率为0
    return group

result_data = grouped_data.apply(calculate_discount_rate)

# 使用groupby()按照'分类名称'列进行分组，并计算每个分类的平均折扣率
average_discount_rate = result_data.groupby('分类名称')['折扣率'].mean().reset_index()


# 打印每个分类的平均折扣率
print(average_discount_rate)


# In[13]:


import pandas as pd
excel_file_path2 = 'D:/qq/附件四修改版.xlsx'
# 使用read_excel导入数据
data2 = pd.read_excel('D:/qq/附件四修改版.xlsx')


# In[14]:


print(data2.head())
```

```
# In[15]:


table1 = pd.DataFrame(data)
table2 = pd.DataFrame(data2)


# In[16]:


merged_data = pd.merge(table1, table2, on=['单品名称', '单品编码'], how='left')
merged_data


# In[17]:


import pandas as pd
excel_file_path3 = 'D:/qq/聊天文件/附件3.xlsx'
# 使用read_excel导入数据
data3 = pd.read_excel('D:/qq/聊天文件/附件3.xlsx')
print(data3.head())


# In[18]:


table3 = pd.DataFrame(data3)
merged_data3 = pd.merge(merged_data, table3, on=['销售日期', '单品编码'], how='left')
merged_data3


# In[19]:


merged_data3['利润']=merged_data3['销量(千克)']*merged_data3['销售单价(元/千克)']-merged_data3['批发价格(元/千克)


# In[20]:


merged_data3


# In[21]:
```

```
merged_data3['成本']=merged_data3['批发价格(元/千克)']*merged_data3['销量(千克)']/(1-merged_data3['损耗率(%)']/10


# In[22]:


merged_data3


# In[27]:


result = merged_data3.groupby('单品名称')['利润'].sum().reset_index()
result


# In[28]:


result1 = merged_data3.groupby('单品名称')['成本'].sum().reset_index()
result1


# In[34]:


result2 = pd.merge(result, result1, on=['单品名称'], how='inner')
result2['利润率']=result2['利润']/result2['成本']
result2
result2.to_excel('D:/qq/聊天文件/单品利润率.xlsx', index=False)


# In[25]:


import pandas as pd
data_sale = pd.read_excel('D:/qq/聊天文件/日销量.xlsx')
print(data_sale.head())


# In[39]:


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

```
# In[40]:


import matplotlib.pyplot as plt

# 示例数据
categories = ['1日收益', '2日收益', '3日收益', '4日收益', '5日收益', '6日收益', '7日收益']
values = [649.6865139, 608.6804961, 601.5645523, 603.8445866, 613.6927919, 618.5767892,
    624.3340563]

# 创建一个条形图
sns.barplot(x=categories, y=values)

# 添加标题和标签
plt.title('未来七日收益')
plt.xlabel('日期')
plt.ylabel('收益')

# 显示图形
plt.show()


# In[26]:


df = pd.DataFrame(data_sale)

# 转换月份列为日期时间类型
df['Date'] = pd.to_datetime(df['Date'])

# 创建虚拟变量（独热编码）以处理分类变量
df = pd.get_dummies(df, columns=['蔬菜分类'], drop_first=True)

# 添加截距项
df['截距'] = 1

# 定义自变量（特征）和因变量
X = df[['截距', '蔬菜分类_蔬菜B', '蔬菜分类_蔬菜C', '月份']]
y = df['销售量']

# 拟合多元回归模型
model = sm.OLS(y, X).fit()

# 打印回归结果
print(model.summary())
```

```
# In[ ]:
```

Listing 9: ARIMA 测试

```
# %%
import pmdarima as pm
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn

df = pd.read_excel("日销量.xlsx")
df.set_index('Date', inplace=True)
df

# %%
df_example = df.loc[:,["西兰花"]]
df_example

# %%
df_test = df_example.copy()

# %%
df_test.iloc[range(df_test.shape[0]-31,df_test.shape[0]),0]=0
df_test

# %%
import matplotlib.pyplot as plt
import pmdarima as pm

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 原始数据
R = range(df_test.shape[0]-30, df_test.shape[0])
original_data = df_example.iloc[R, 0]

# 创建一个画布
plt.figure(figsize=(10, 6), dpi=300)

# 绘制原始数据
plt.plot(df_example.index[R], original_data, label="原始数据")

# 绘制其他三幅图像
model_periods = [30]
for period in model_periods:
    df_test_copy = df_test.copy()
    for i in range(df_test_copy.shape[0]-30, df_test_copy.shape[0]):
```

```
        time_series_data = df_example.iloc[range(i-period, i), 0]
        model = pm.auto_arima(time_series_data, seasonal=False)
        df_test_copy.iloc[i, 0] = model.predict(n_periods=1)
    plt.plot(df_test.index[R], df_test_copy.iloc[R, 0], label=f"参数设置{period}天")

# 添加图例
plt.legend(loc='upper right', fontsize=10)


# %%


# %%
```