# 基于机器学习构建蔬菜类商品的自动定价与补货决策模型

## 摘要

在生鲜商超中，由于蔬菜类商品的保质期很短。商超通常会根据各商品的历史销售和需求情况每天进行补货。因此，本文基于组委会给出的相关数据，对蔬菜类商品的自动定价与补货决策进行相关研究。

针对问题一，对于不同品类之间的关联关系分析，首先对 2020 年到 2023 年的不同品类的销量做折线图分析，得出不同品类的销量趋势和变化幅度大致相同。其次分别以不同时间单位建立时间-销量直方图，得出花叶类的销量且波动范围区间最大，茄类的销量波动范围最小。最后，建立以月为单位的不同销量品类河流图，得出在所有单品特定时间段销量的变化有周期性规律; 对于不同单品之间的关联关系分析，我们首先对各个品类绘制散点图和直方图, 发现大多数点分布不均匀很难找出他们之间的相关性, 于是我们基于 Apriori 算法, 算出各个单品之间的频繁集, 其优势在于适合稀疏数据集的关联规则挖掘, 算法采用逐层搜索的迭代方法，我们首先按照蔬菜的季节性将蔬菜类商品分为四个季度, 再将数据按扫码销售时间每 10 分钟分为一个组, 分析每 10 分钟内的单品相关性并可视化分析结果, 支持度越高, 表示项集内单品相关关系越大, 尤其是第四季度中西兰花, 西峡香菇相关性最强烈。

针对问题二，以品类为单位做补货计划, 分析各蔬菜品类的销售总量与成本加成定价的关系，建立各蔬菜品类的销售量（X）和利润（Y）线性回归模型，然后收集每个蔬菜品类的历史销售数据, 包括销售总量和成本加成情况, 利用销售价格和采购成本数据, 计算每个蔬菜品类的成本加成百分比。采用 LSTM 时间序列分析, 预测未来一周每个蔬菜品类的需求, 利用销售量预测结果, 通过建立利润和销售量的线性回归预测模型，制定定价方案。通过预测后七天每个品类的数据, 来制定补货策略和定价策略。在这七天预测中, 花叶类的预测销量最高, 可达到 825.997 千克。

针对问题三, 为了有效应对组合爆炸的情况, 我们选择了遗传算法, 通过 6 月 24 号到 6 月 30 号的数据可以得到平均每天的利润为 653.894 元, 通过遗传算法可以预测出符合问题三要求的最大收益为 360.386 元, 而在该题目要求中, 可选择的单品只有 6 月 24 号到 6 月 30 号这七天的单品的 55%, 所以该预测合理。

针对问题四，采用动态定价法，主要的考虑因素包括了目标函数、约束时间、容量、地点、成本和资源等。决策变量是控制变量，通过控制变量的数值和范围，观察目标函数的值的变化情况，从而能够达到更好地制定蔬菜商品的补货和定价决策。例如通过动态定价法，在过年等节假日期间进行更大规模的进货量可以获得更大收益。

**关键字**：　Apriori 算法　LSTM 时间序列模型　遗传算法　动态定价模型

# 一、 问题重述

## 1.1 问题背景

在生鲜超市中，蔬菜类商品通常拥有相对较短的保鲜期限，且随着时间的流逝，它们的品质会逐渐下降。绝大多数蔬菜品种，如果当日没有售出，第二天就难以再次销售。因此，生鲜超市通常会根据商品的历史销售情况和需求，每天进行必要的补货，以确保顾客能够购买到新鲜、高品质的蔬菜产品。一方面有助于维持商品的新鲜度和吸引力，另一方面能让商家利益最大化。

## 1.2 问题重述

1. 问题一需要分别分析不同品类和不同单品之间存在的相关关系，从而可以进一步分析蔬菜各品类及单品销售量的分布规律及相互关系。

2. 问题二要求以品类为单位做补货计划，首先分析各蔬菜品类的销售总量与成本加成定价的关系，其次需要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。

3. 在售单品总数控制在 27-33 个，且各单品订购量满足最小陈列量 2.5 千克的前提下，进一步详细的制定单品的补货计划，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，使得商超收益最大。

4. 对于在问题三制定的优化蔬菜商品的补货和定价决策，需要采集哪些相关数据，这些数据对解决制定蔬菜商品的补货和定价决策问题有何帮助。写出意见和理由分析。

# 二、 问题分析

蔬菜在短期内容易受时间的影响而产生变质腐烂，这会导致它们失去营养价值和经济价值。商家通常需要在不确切知道具体单品和进货价格的情况下，做出当日各蔬菜品类的补货决策。蔬菜的定价方法一般采用” 成本加成定价”，而对于运损或品相变差的商品，商家通常会选择打折销售。现题目给出附件 1 某商超经销的 6 个蔬菜品类的商品信息、附件 2 和附件 3 该商超 2020 年 7 月 1 日至 2023 年 6 月 30 日各商品的销售流水明细与批发价格的相关数据和附件 4 各商品近期的损耗率数据。

本文基以上信息建立数学模型解决以下问题:

### 2.1 问题一

对于不同蔬菜品类之间的关联关系，绘制折线图和直方图比较分析不同品类之间的关系，可得出不同蔬菜品类的分布规律和相互关系。对于不同单品之间的关联关系，首先绘制散点图和直方图观察发现散点分布不均匀，所以采取基于 Apriori 算法，将数据按扫码销售时间每 10 分钟分为一个组，通过频繁集分析每 10 分钟内的单品相关性。

### 2.2 问题二

以品类为单位做补货计划，分析各蔬菜品类的销售总量与成本加成定价的关系，针对这一小问首先建立各蔬菜品类的销售量（X）和利润（Y）线性回归模型，然后收集每个蔬菜品类的历史销售数据，包括销售总量和成本加成情况，利用销售价格和采购成本数据，计算每个蔬菜品类的成本加成百分比。采用 LSTM 时间序列分析，预测未来一周每个蔬菜品类的需求，利用销售量预测结果，通过建立利润和销售量的线性回归预测模型，制定定价方案。制定补货策略和定价策略。

### 2.3 问题三

该问题要求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。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。此题使用的是遗传算法，基于达尔文的进化理论，通过模拟“优胜劣汰”和“适者生存”的原理，逐步改进候选解以找到最优解或接近最优解。

### 2.4 问题四

对于其他数据的采集，可以提供有关产品、消费者和市场的更多信息。竞争对手数据、消费者行为数据、促销效果数据、天气数据等。这些数据将为商超提供更全面的信息，帮助其更准确地预测需求、制定补货计划、调整定价策略，并升级产品和提高市场竞争力。

## 三、 模型假设

为了更好的建立模型我们做出如下假设:

1 、销售数据稳定性假设：假设销售数据在短期内是相对稳定的，不受突发事件 (如自然灾害) 的影响。

2 、销售时间假设：假设扫码销售时间在每十分钟为一次消费，十分钟内销售的蔬菜单品为一个项集。

3 、市场销量假设：假设市场销量反映市场的需求。

## 四、 符号说明

| 符号 | 意义 |
| --- | --- |
| $Loss$ | 损失值 |
| $X$ | 销售量 |
| $Y$ | 利润 |
| $Sell$ | 销量 |
| $Day$ | 日期 |
| $H$ | 不同品类 |

## 五、 数据预处理

首先对于给出的数据进行缺失值查找, 通过 Excel 的筛选与查找功能寻找空缺值, 得出对于附件的数据不存在缺失值。根据问题, 对 4 个表进行灵活的连接, 通过关键字提取出对问题有用的信息以及去掉异常值。

## 六、 模型的建立与求解

### 6.1 问题一

#### 6.1.1 不同品类之间的关联关系分析

针对题目一要求先分析不同品类之间是否存在一定的关联关系。流程图如图1：

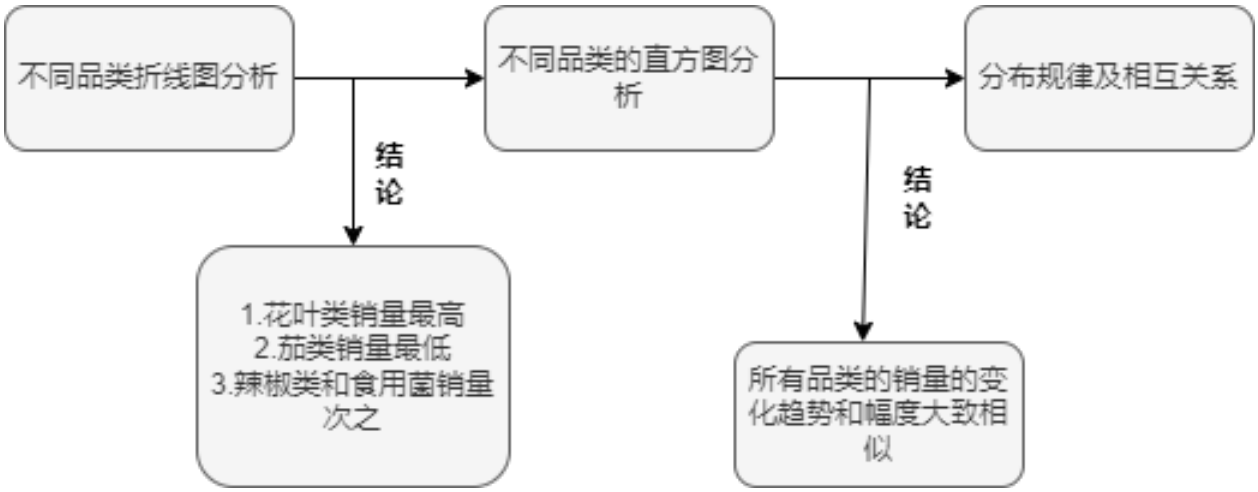



**图 1　第一小问流程图**

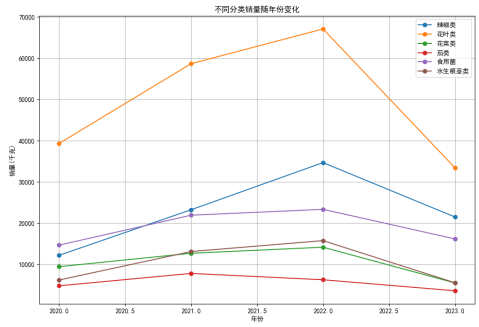
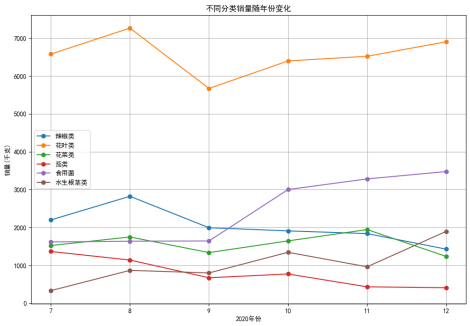
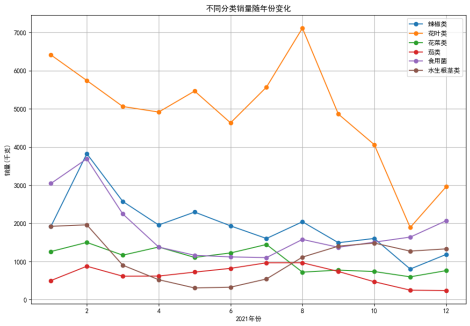



**图 2　2020 年至 2023 年不同品类的销售量**

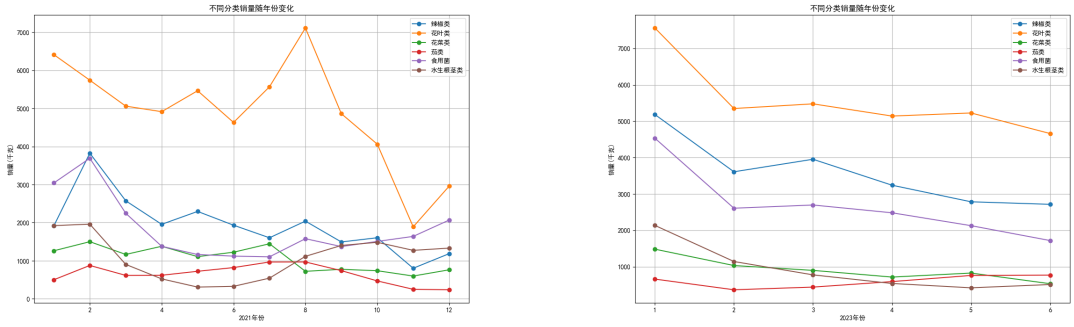



**(a) 2020 年不同品类的销售量**




**(b) 2021 年不同品类的销售量**

**图 3**




**(a) 2022 年不同品类的销售量**　　**(b) 2023 年不同品类的销售量**

**图 4**

由从 2020 年到 2023 的六种菜品的销量折线图曲线图2, 3, 4中观察得出从 2020 年到 2023 年的不同菜品的销量趋势和变化幅度大致相同，其中虽然六种菜品的变化趋势相近但是在这三年中花叶类的销量远远高于其他五类，且销量受季节波动性大，茄类全年销售量都位于六种菜品的最低，其次辣椒类和食用菌全年的商品销量也较高。

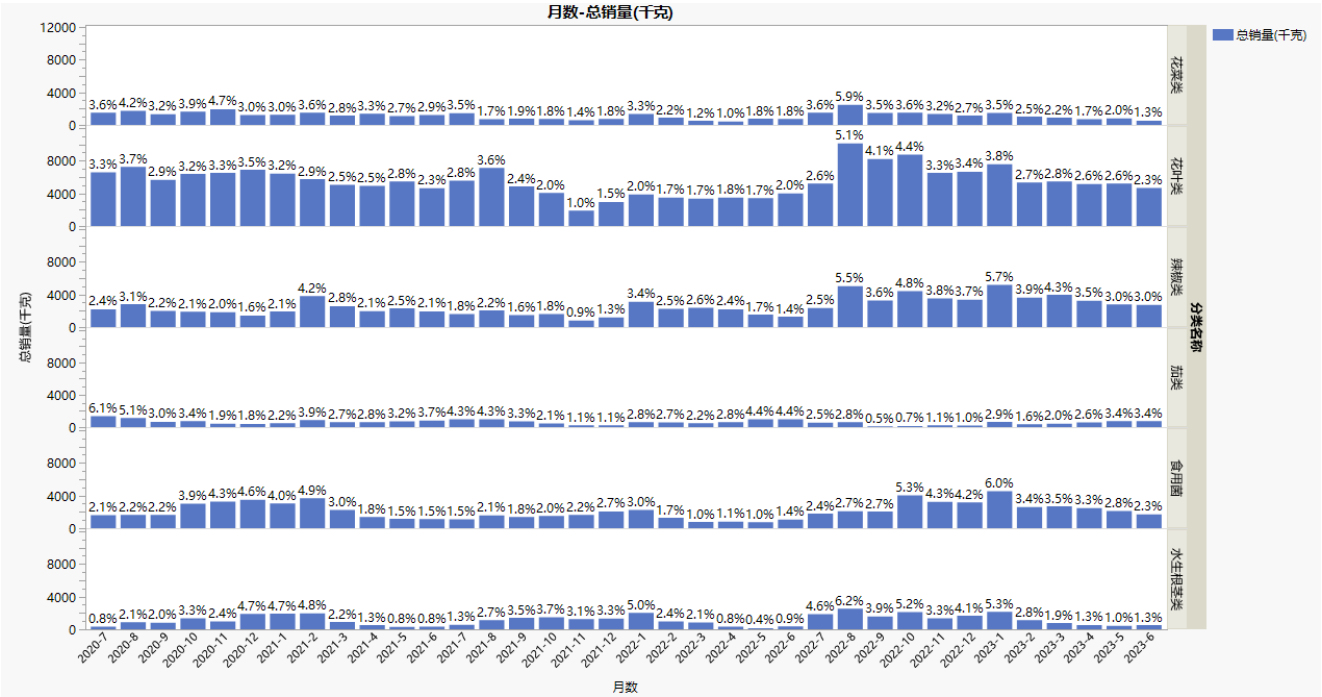



**图 5　以月位单位计算不同蔬菜品类的销量的直方图**

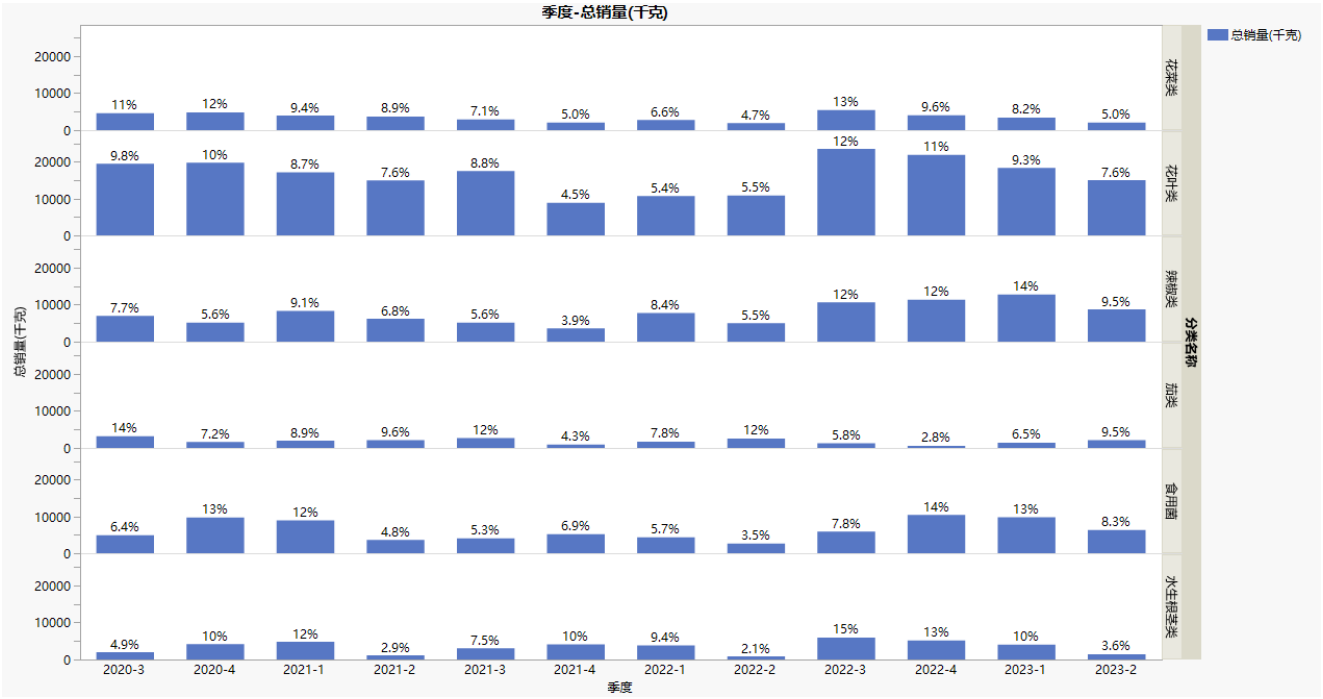



**图 6　以季度位单位计算不同蔬菜品类的销量的直方图**

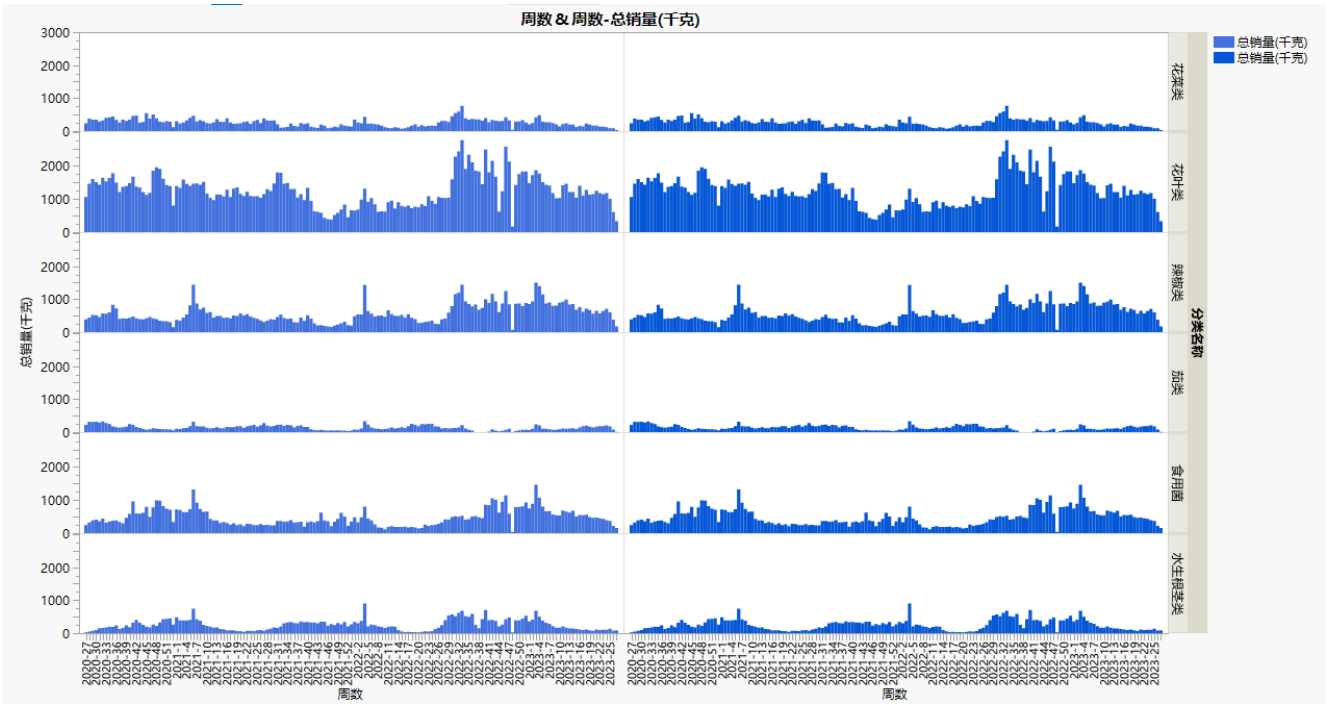



**图 7　以周位单位计算不同蔬菜品类的销量的直方图**

以周数、月份、季度为单位的销售量绘制直方图，由直方图5, 6, 7观察图可得花叶类的销量且波动性随时间变化区间最大，辣椒类和食用菌销量次之，其他的品类与花叶类的销量有近似的波动范围。但是茄类的销量波动虽然有近似花叶类的变化趋势但是变化范围不大。

### 6.1.2 不同单品之间的关联关系分析

对于不同单品之间的关联关系分析，我们首先对各个品类绘制散点图和直方图，如图8，发现大多数点较分散很难找出他们之间的相关性，于是我们基于 Apriori 算法，算出各个单品之间的频繁集，研究单品之间的相关性，Apriori 算法[1]是第一个关联规则挖掘算法，它利用逐层搜索的迭代方法找出数据库中项集的关系，以形成规则，为了发现事物之间的联系，其过程由连接（类矩阵运算）与剪枝（去掉那些没必要的中间结果）组成。该算法中项集的概念即为项的集合。包含 K 个项的集合为 k 项集。项集出现的频率是包含项集的事务数，称为项集的频率。如果某项集满足最小支持度，则称它为频繁项集。

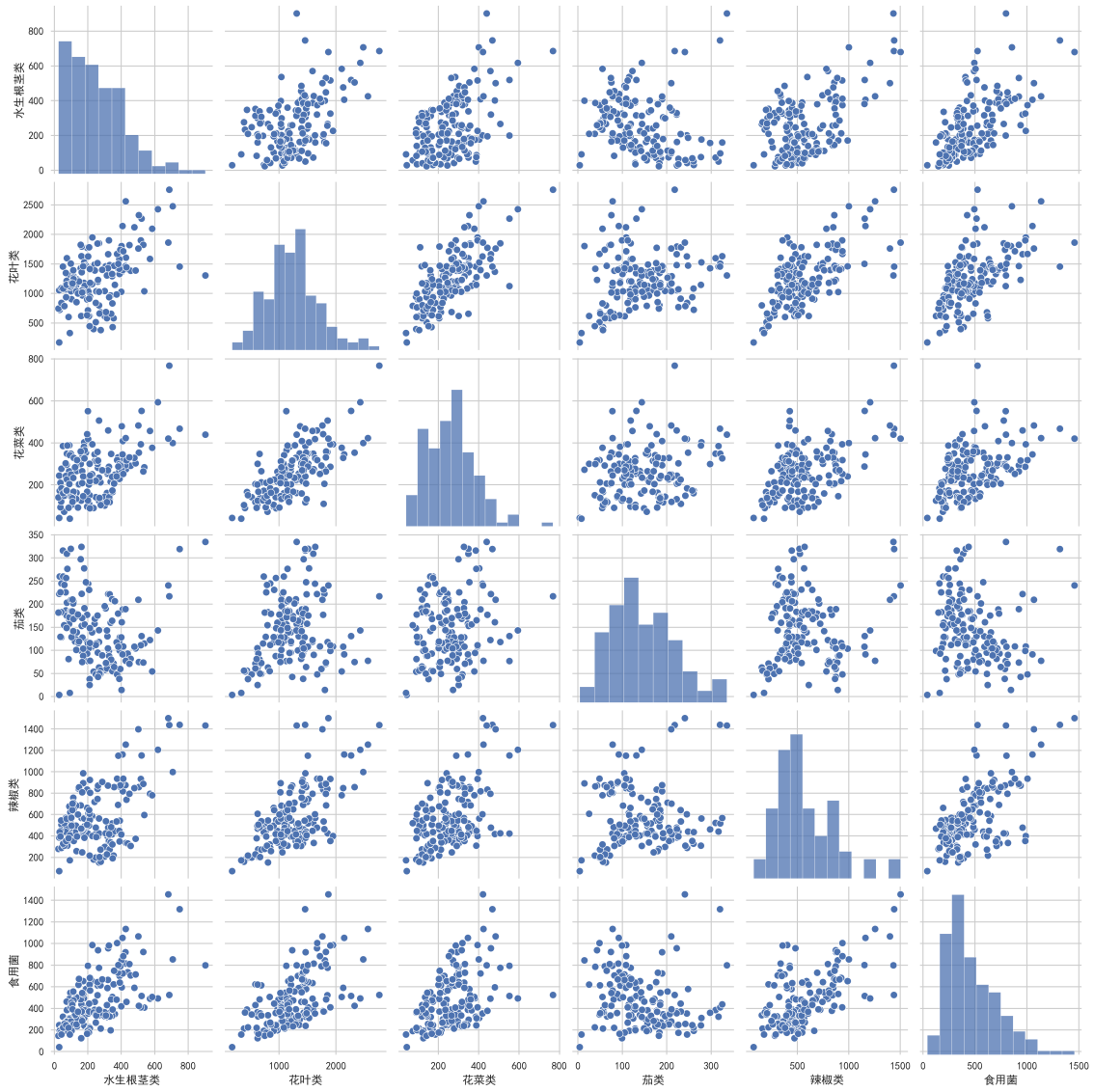



**图 8　各个品类之间散点图和直方图**

具体流程如图9：

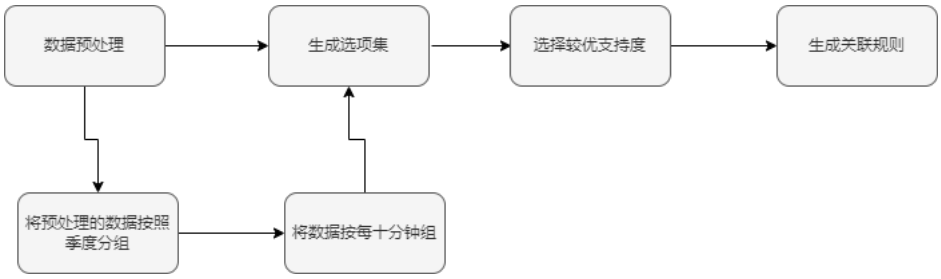



**图 9　Apriori 算法流程图**

在操作之前我们使用预处理的数据, 由于蔬菜的生产和供应确实受到季节性影响, 先将三年数据按照季度分为 4 个季度, 假设扫码销售时间在每十分钟为一次消费, 十分钟内销售的蔬菜单品为一个项集, 去掉小于支持度的非频繁项, 得到表1各个季度的总频繁项集数。频繁项集[2]帮助我们了解到各个单品之间的隐含规律、趋势和关联关系。接着制定最小支持度, 支持度是判定项集内关系的一个标准, 可以更全面地捕捉数据中的模式和规律，但也可能包含一些噪音和不太有用的项集。我们通过暴力搜索, 发现最佳的最小支持度为 0.01。最后选择和筛选结果, 筛选最优的频繁集进行观测, 在大规模数据集中，由于数据的稀疏性，某些项集可能在数据中很少出现, 导致支持度较小的情况如表2, 3, 4, 5分别列出了四季的四季的频繁项集, 支持度越高, 表示项集内关联越大, 例如表2中西兰花, 芜湖青椒相关性较强。

**表 1　四个季度项集汇总**

| 季度 | 项集总和 |
|---|---|
| 第一季度 | 114 |
| 第二季度 | 136 |
| 第三季度 | 377 |
| 第四季度 | 169 |
| 总计 | 796 |

我们可以将上述数据可视化, 更加直观的观察不同单品之间的关联关系, 在 Apriori 算法中还涉及到两个参数: 置信度和提升度, 如果提升度大且置信度高，表示规则非常强。提升度 lift 小但置信度 confidence 高可能表示规则相对较强，但不够强烈。反之，如果提升度小且置信度也低，可能表示规则相对较弱。接下来我们通过画提升度和置信度的散点图分析关联关系, 如图10第四季度的规则非常强, 其他季度较强烈。

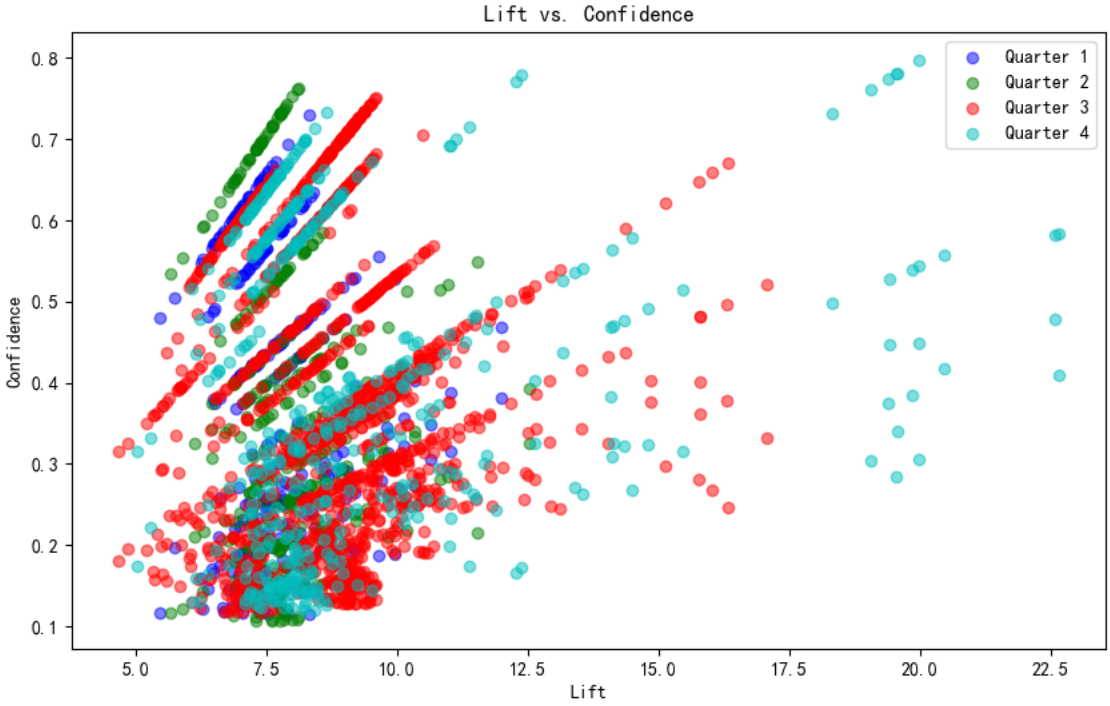



**图 10　提升度和置信度关系**

**表 2　第一季度频繁项集**

| 支持度 | 项集 | 长度 |
|---|---|---|
| 0.032895 | (芜湖青椒 (1), 净藕 (1)) | 2 |
| 0.031980 | (净藕 (1), 西兰花) | 2 |
| 0.032810 | (芜湖青椒 (1), 紫茄子 (2)) | 2 |
| 0.043105 | (芜湖青椒 (1), 西兰花) | 2 |
| 0.033284 | (芜湖青椒 (1), 西峡香菇 (1)) | 2 |
| 0.031852 | (西峡香菇 (1), 西兰花) | 2 |

**表 3　第二季度频繁项集**

| 支持度 | 项集 | 长度 |
|---|---|---|
| 0.031348 | (芜湖青椒 (1), 云南生菜) | 2 |
| 0.032279 | (芜湖青椒 (1), 竹叶菜) | 2 |
| 0.025365 | (西兰花, 竹叶菜) | 2 |
| 0.037518 | (芜湖青椒 (1), 紫茄子 (2)) | 2 |
| 0.027345 | (西兰花, 紫茄子 (2)) | 2 |
| 0.032271 | (芜湖青椒 (1), 螺丝椒) | 2 |
| 0.042554 | (芜湖青椒 (1), 西兰花) | 2 |
| 0.025737 | (西兰花, 螺丝椒) | 2 |

**表 4　第三季度频繁项集**

| 支持度 | 项集 | 长度 |
|---|---|---|
| 0.034175 | (云南生菜, 云南油麦菜) | 2 |
| 0.030709 | (云南生菜, 净藕 (1)) | 2 |
| 0.030608 | (芜湖青椒 (1), 云南生菜) | 2 |
| 0.041681 | (云南生菜, 西兰花) | 2 |
| 0.041596 | (云南生菜, 西峡香菇 (1)) | 2 |
| 0.034614 | (西兰花, 净藕 (1)) | 2 |
| 0.036246 | (芜湖青椒 (1), 西兰花) | 2 |
| 0.039804 | (西兰花, 西峡香菇 (1)) | 2 |

**表 5　第四季度频繁项集**

| 支持度 | 项集 | 长度 |
|---|---|---|
| 0.039476 | (西兰花, 净藕 (1)) | 2 |
| 0.043280 | (西峡香菇 (1), 净藕 (1)) | 2 |
| 0.031928 | (西兰花, 芜湖青椒 (1)) | 2 |
| 0.036137 | (芜湖青椒 (1), 西峡香菇 (1)) | 2 |
| 0.044878 | (西兰花, 西峡香菇 (1)) | 2 |

## 6.2 问题二

### 6.2.1 各蔬菜品类的销售总量与成本加成定价的关系

以品类为单位做补货计划，各蔬菜品类的销售总量与成本加成定价的关系，具体流程如图11:

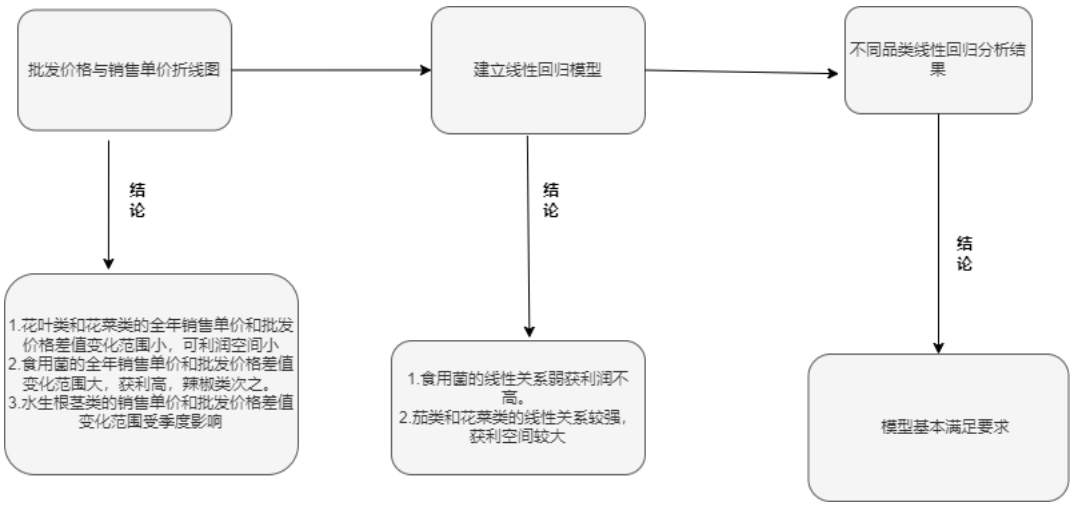



**图 11　第二小问流程图**

绘制以周为时间单位的销售单价与批发价格的时间折线图12和销量与时间关系折线图12所示

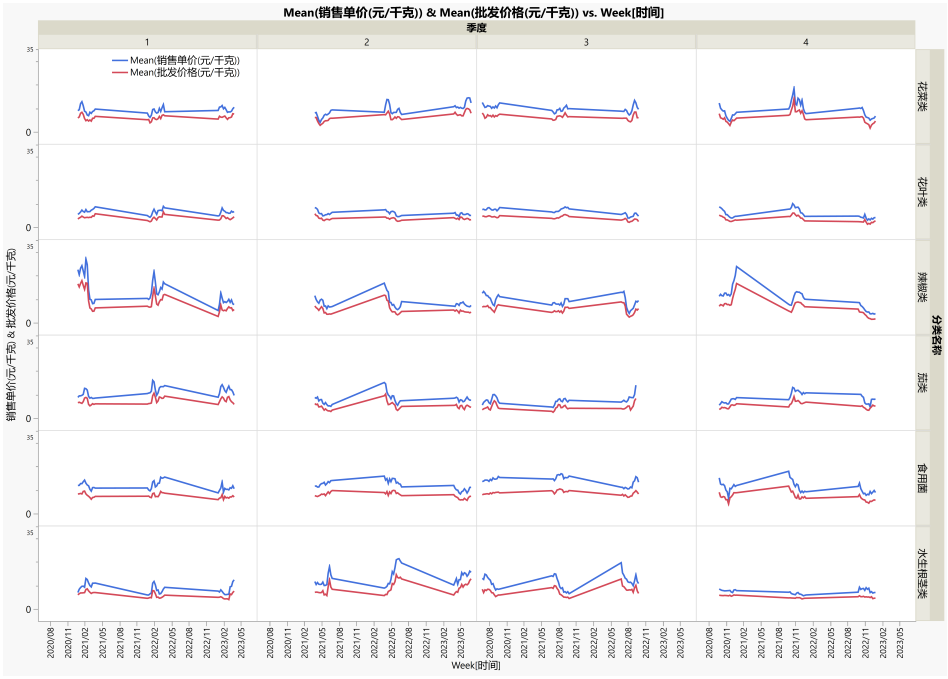



**图 12　销售单价与批发价格与时间折线图**

从图**??**中观察得知有明显的销量和时间线性关系。观察销售单价与批发价格的差值得出，花叶类和花菜类的全年销售单价和批发价格差值变化范围较小，利润空间较小，而食用菌的全年销售单价和批发价格差值变化范围较大，可得利润较高，辣椒类的全年销售单价和批发价格差值变化范围其次，而水生根茎类的销售单价和批发价格差值变化范围受季度影响的而变化范围差异较大，在第一季度，第二季度，第三季度销售单价和批发价格差值变化范围远远大于第四季度的变化范围。

由于销量和时间存在明显的线性关系，假设销量和利润同时也同样存在线性关系，针对该问题我们做同时，计算得出不同蔬菜品类的回归模型如表6所示：所以选择以各蔬菜品类的销售量（X）和利润（Y）建立线性回归中的最小二乘法模型

**表 6　不同蔬菜品类利润的回归分析**

| 品类 | 预测回归模型 |
|---|---|
| 花叶类 | $y = 0.352 + 1.578x$ |
| 花菜类 | $y = 0.237 + 2.642x$ |
| 辣椒类 | $y = 0.485 + 1.950x$ |
| 食用菌 | $y = 1.083 + 0.855x$ |
| 茄类 | $y = 0.157 + 2.780x$ |
| 水生根茎类 | $y = 0.927 + 1.319x$ |

其中食用菌的线性关系较弱，可获利润不高，而茄类和花菜类的线性关系在六种蔬菜品类中较强，可获利空间较大。

**表 7　不同品类线性回归分析结果**

| 品类 | $x^2$ | P |
|---|---|---|
| 花菜类 | 0.37 | 0.001 |
| 茄类 | 0.384 | 0.001 |
| 水生根茎类 | 0.216 | 0.001 |
| 辣椒类 | 0.211 | 0.001 |
| 花叶类 | 0.326 | 0.001 |
| 食用菌 | 0.079 | 0.001 |

$R^2$ 代表曲线回归的拟合程度，越接近 1 效果越好。显著性 P 值为 0.001，水平上呈现显著性，拒绝回归系数为 0 的原假设，因此模型基本满足要求。

### 6.2.2 补货总量和定价策略

针对问题二, 在以品类为单位做补货计划的前提下, 建立基于 PYTORCH[3] 的 LSTM 模型的训练流程, 对每一个品类的销量进行分析预测。LSTM 网络[4] 属于监督学习领域, 因此在建立 LSTM 模型之前，需要将数据序列整理成监督数据集形式, 通常采用滑动窗

口法实现。滑动窗口法是处理时间序列数据的常用方法, 其含义为采用过去的时间点的值去预测下一个时间点的值。在这个过程中涉及到滑动窗口宽度的确定，滑动窗口的宽度指的是一个窗口包含的时间点的个数。本文的单品销售量序列包含一定的周期性, 因此滑动窗口的确定需要考虑周期, 在本文的模型中, 窗口宽度是 7, 表示每个时间步包含过去 7 个时间步的数据。模型的架构: 输入层的大小是 1, 因为每个时间步只有一个特征, 即" 总销量 (千克)(t-n)"。LSTM 层有 1 个堆叠层, 每个堆叠层中包含 4 个隐藏单元)。输出层是一个全连接层, 输出一个值, 用于预测销量。模型训练如下图13所示:

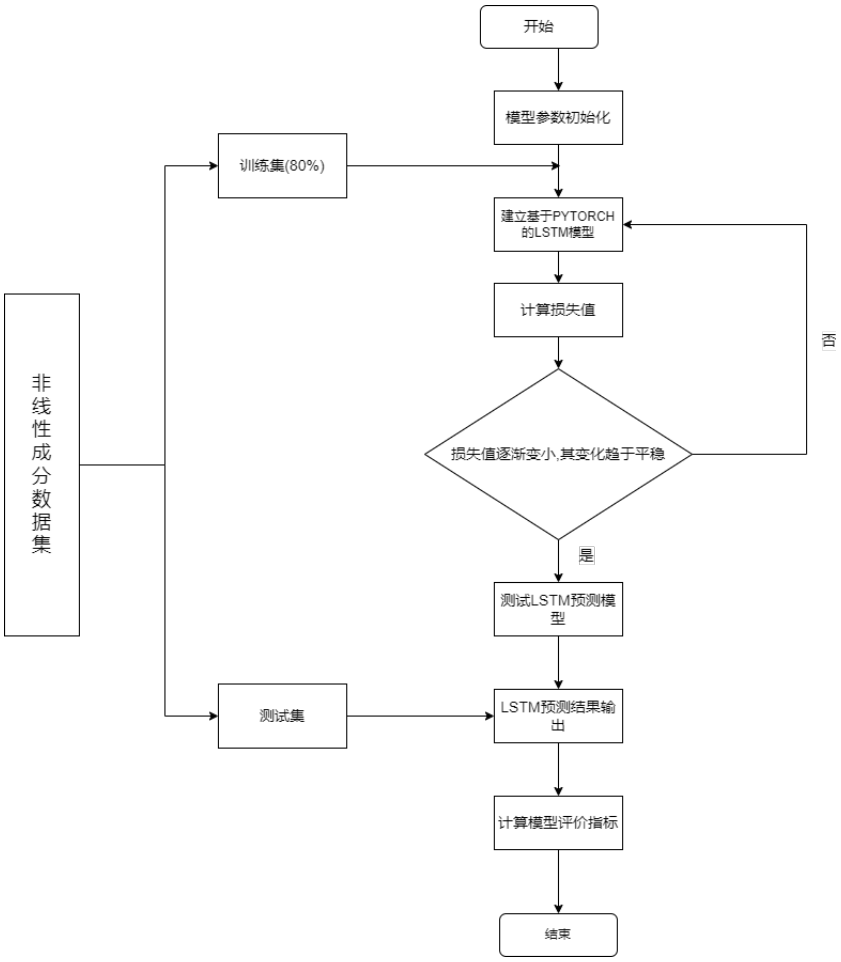
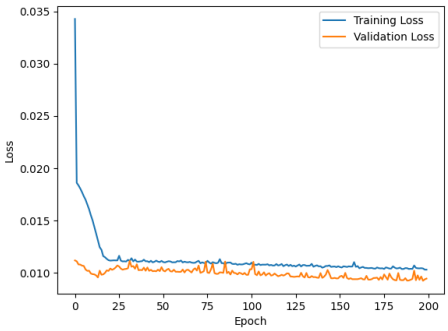
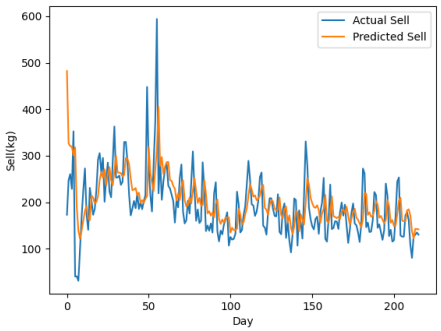



**图 13　LSTM 模型训练流程图**

下面为每个品类的损失函数图与预测效果图, 从图上可以看出 Loss 函数在减小并最后趋于平稳, 由此可见右侧的各单类销量预测值较为可靠。

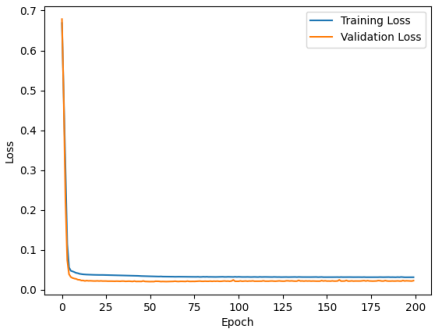



(a) Loss（损失）曲线

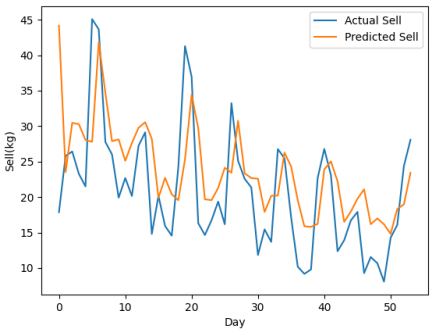



(b) 花叶类

**图 14　花叶类销量预测**

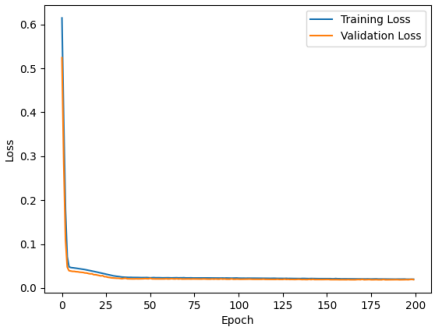



(a) Loss（损失）曲线

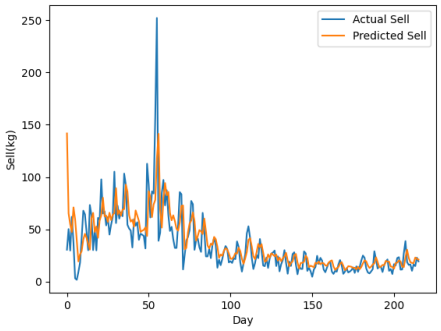



(b) 花菜类

**图 15　花菜类销量预测**

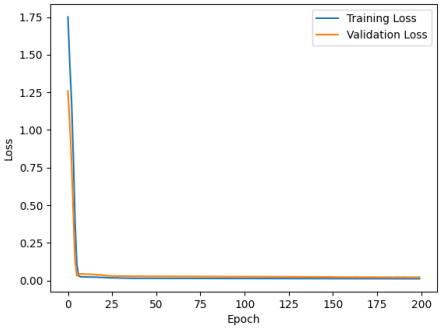



(a) Loss（损失）曲线

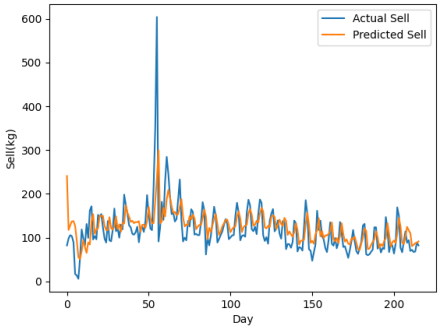



(b) 水生根茎类

**图 16　水生根茎类销量预测**

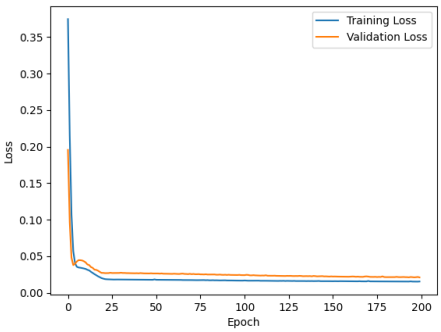



(a) Loss（损失）曲线

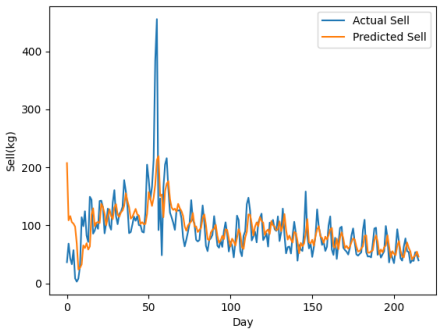



(b) 辣椒类

**图 17　辣椒类销量预测**

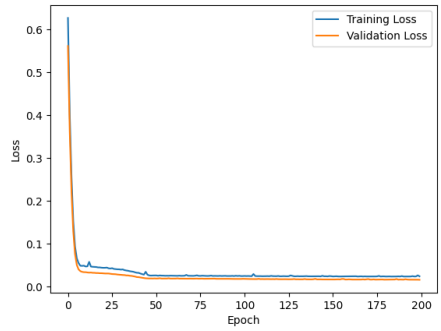



(a) Loss（损失）曲线

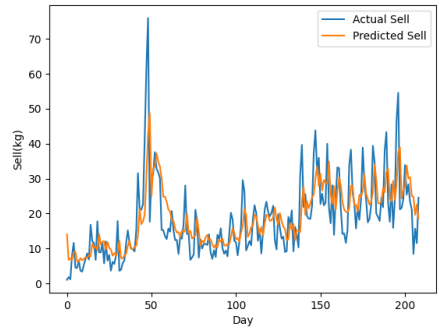



(b) 食用菌

**图 18　食用菌销量预测**




(a) Loss（损失）曲线




(b) 茄类

**图 19　茄类销量预测**

在上面的销量预测图中, 预测曲线的始端都出现了下降趋势, 因此使用曲线后半段进行未来一周 (2023 年 7 月 1-7 日) 的销量预测。根据在第二题的第一个小问建立的销量和利润的线性回归预测模型，确定定价方策略各品类预测表格见下列图8, 9, 10, 11, 12, 13所示。

**表 8　花菜类七日预测表**

| 日期 | 品类 | 销量 (千克) | 定价 | 利润 (元) |
|---|---|---|---|---|
| 7 月 1 日 | 花菜类 | 17.263 | $y = 0.237 + 2.642x$ | 45.846 |
| 7 月 2 日 | 花菜类 | 17.088 | $y = 0.237 + 2.642x$ | 45.383 |
| 7 月 3 日 | 花菜类 | 16.163 | $y = 0.237 + 2.642x$ | 42.938 |
| 7 月 4 日 | 花菜类 | 15.387 | $y = 0.237 + 2.642x$ | 40.889 |
| 7 月 5 日 | 花菜类 | 17.213 | $y = 0.237 + 2.642x$ | 45.714 |
| 7 月 6 日 | 花菜类 | 17.670 | $y = 0.237 + 2.642x$ | 46.922 |
| 7 月 7 日 | 花菜类 | 20.127 | $y = 0.237 + 2.642x$ | 53.412 |
| 总计 | | 120.911 | | 321.104 |

**表 9　花叶类七日预测表**

| 日期 | 品类 | 销量 (千克) | 定价 | 利润 (元) |
|---|---|---|---|---|
| 7 月 1 日 | 花叶类 | 183.760 | $y = 0.352 + 1.578x$ | 290.326 |
| 7 月 2 日 | 花叶类 | 166.418 | $y = 0.352 + 1.578x$ | 262.960 |
| 7 月 3 日 | 花叶类 | 126.461 | $y = 0.352 + 1.578x$ | 199.908 |
| 7 月 4 日 | 花叶类 | 112.833 | $y = 0.352 + 1.578x$ | 178.402 |
| 7 月 5 日 | 花叶类 | 131.640 | $y = 0.352 + 1.578x$ | 208.080 |
| 7 月 6 日 | 花叶类 | 136.969 | $y = 0.352 + 1.578x$ | 216.489 |
| 7 月 7 日 | 花叶类 | 134.334 | $y = 0.352 + 1.578x$ | 212.330 |
| 总计 | | 825.997 | | 1568.495 |

**表 10　辣椒类七日预测表**

| 日期 | 品类 | 销量 (千克) | 定价 | 利润 (元) |
|---|---|---|---|---|
| 7 月 1 日 | 辣椒类 | 114.461 | $y = 0.485 + 1.950x$ | 223.683 |
| 7 月 2 日 | 辣椒类 | 105.012 | $y = 0.485 + 1.950x$ | 205.258 |
| 7 月 3 日 | 辣椒类 | 75.406 | $y = 0.485 + 1.950x$ | 147.526 |
| 7 月 4 日 | 辣椒类 | 80.930 | $y = 0.485 + 1.950x$ | 158.299 |
| 7 月 5 日 | 辣椒类 | 80.457 | $y = 0.485 + 1.950x$ | 157.377 |
| 7 月 6 日 | 辣椒类 | 79.419 | $y = 0.485 + 1.950x$ | 155.353 |
| 7 月 7 日 | 辣椒类 | 86.935 | $y = 0.485 + 1.950x$ | 170.008 |
| 总计 | | 120.911 | | 1117.504 |

**表 11　茄类七日预测表**

| 日期 | 品类 | 销量 (千克) | 定价 | 利润（元） |
|---|---|---|---|---|
| 7 月 1 日 | 茄类 | 28.346 | $y = 0.157 + 2.780x$ | 78.959 |
| 7 月 2 日 | 茄类 | 30.550 | $y = 0.157 + 2.780x$ | 85.087 |
| 7 月 3 日 | 茄类 | 24.683 | $y = 0.157 + 2.780x$ | 68.775 |
| 7 月 4 日 | 茄类 | 24.307 | $y = 0.157 + 2.780x$ | 67.729 |
| 7 月 5 日 | 茄类 | 16.649 | $y = 0.157 + 2.780x$ | 46.441 |
| 7 月 6 日 | 茄类 | 21.863 | $y = 0.157 + 2.780x$ | 60.937 |
| 7 月 7 日 | 茄类 | 17.070 | $y = 0.157 + 2.780x$ | 47.612 |
| 总计 | | 163.468 | | 455.54 |

**表 12　水生根茎类七日预测表**

| 日期 | 品类 | 销量 (千克) | 定价 | 利润（元） |
|---|---|---|---|---|
| 7 月 1 日 | 水生根茎类 | 21.140 | $y = 0.927 + 1.319x$ | 28.811 |
| 7 月 2 日 | 水生根茎类 | 18.014 | $y = 0.927 + 1.319x$ | 24.687 |
| 7 月 3 日 | 水生根茎类 | 18.080 | $y = 0.927 + 1.319x$ | 24.774 |
| 7 月 4 日 | 水生根茎类 | 17.006 | $y = 0.927 + 1.319x$ | 23.357 |
| 7 月 5 日 | 水生根茎类 | 20.739 | $y = 0.927 + 1.319x$ | 28.282 |
| 7 月 6 日 | 水生根茎类 | 19.277 | $y = 0.927 + 1.319x$ | 26.354 |
| 7 月 7 日 | 水生根茎类 | 20.355 | $y = 0.927 + 1.319x$ | 27.775 |
| 总计 | | 134.611 | | 184.04 |

**表 13　食用菌七日预测表**

| 日期 | 品类 | 销量 (千克) | 定价 | 利润 |
|---|---|---|---|---|
| 7 月 1 日 | 食用菌 | 57.737 | $y = 1.083 + 0.855x$ | 50.448 |
| 7 月 2 日 | 食用菌 | 55.273 | $y = 1.083 + 0.855x$ | 48.341 |
| 7 月 3 日 | 食用菌 | 39.401 | $y = 1.083 + 0.855x$ | 34.771 |
| 7 月 4 日 | 食用菌 | 47.179 | $y = 1.083 + 0.855x$ | 41.421 |
| 7 月 5 日 | 食用菌 | 47.877 | $y = 1.083 + 0.855x$ | 42.018 |
| 7 月 6 日 | 食用菌 | 56.550 | $y = 1.083 + 0.855x$ | 49.433 |
| 7 月 7 日 | 食用菌 | 45.847 | $y = 1.083 + 0.855x$ | 40.282 |
| 总计 | | 349.864 | | 306.714 |

### 6.3 问题三

#### 6.3.1 模型的建立

在问题三中,因为存在组合爆炸的情况,所以我们使用遗传算法。遗传算法是一种基于自然选择和遗传机制的优化算法，通常适用于搜索解空间中的候选解。

该问题要求制定 7 月 1 日的单品补货量和定价策略，实际上就是根据 2023 年 6 月 24-30 日的可售品种和问题二求解的补货总量和定价策略再次优化模型。我们选择使用遗传算法进行迭代优化选择，遗传算法是根据基因变异的思想进行随机优化的搜索方法，结合自适应的概率优化算法,能够在全局范围内寻找最优解，满足题目要求。

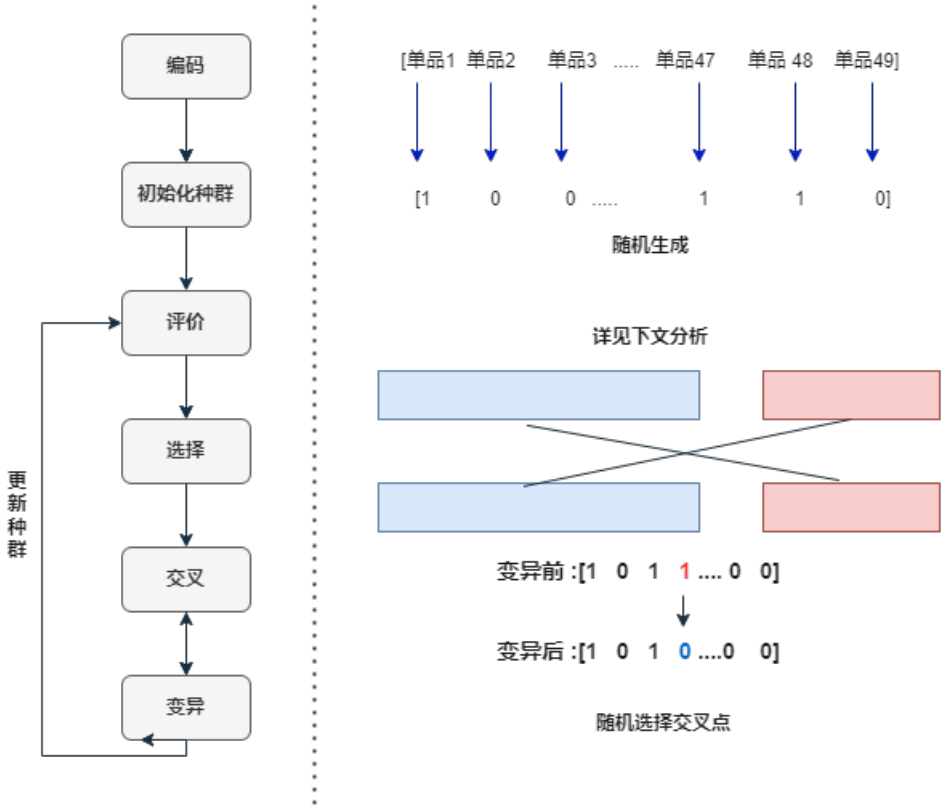



**图 20　遗传算法流程图**

### 6.3.2 七天的利润统计

**表 14　七天的利润**

| 日期 | 利润 |
|---|---|
| 6 月 24 日 | 794.635 |
| 6 月 25 日 | 544.635 |
| 6 月 26 日 | 569.011 |
| 6 月 27 日 | 581.946 |
| 6 月 28 日 | 581.950 |
| 6 月 29 日 | 736.949 |
| 6 月 30 日 | 768.029 |
| 利润平均值 | 653.894 |

### 6.3.3 模型的求解

使用遗传算法[5]，首先确定种群数量，给染色体编码，随机生产第一代种群；然后确定个体评价方案，判断是否满足停止条件，若是则停止，输出最优解；否则继续进行操作，通常第一代种群不会满足停止条件；接着对种群进行选择、交叉、变异操作，得到下一代种群，再进行评价，反复循环，直到满足生产需求。流程图20如下

1. 初始化种群：对 500 个每日购买单品种类编码，随机生成 0 或 1，1 表示选购买该单品，0 表示未购买该单品，即生成初始化种群；为了得到精准数据，至少要有 100 个种群。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, 为此在总群初始化时要保证生成 27-33 个 1，且销量大于 2.5 千克。

2 进行交叉和变异操作：这一轮留下的个体，通过交叉和变异繁衍后代，留下来的个体和它们新产生的子代就形成了新的种群。

3. 再次筛选序列, 重新打乱种群，然后对其中的每个序列计算 1 的数量，如果单品数量在 27 到 33 之间，则保留。

4. 计算现有 100 个种群中数值为 1 的单品总利润, 进行排序, 直到当前群体中选择出一些优良的个体, 并填充种群数量到 100。

5. 重复进行上述 1-4 步，直到求出满足利润最大补货策略的订货方案。此外，还采用总订购成本为个体评价指标, 进行选择。

### 6.3.4

根据第三问参考 2 数据 023 年 6 月 24-30 日的可售品，算出各单品销售和该品类的总销量的单品比例。如下表所示

**表 15　6 月 24 日至 6 月 30 日花菜类单品的销量分析**

| 品类 | 单品名称 | 单品比例 | 单品销量（千克） |
|---|---|---|---|
| 花菜类 | 西兰花 | 0.777 | 87.9.4 |
| | 枝江青梗散花 | 0.223 | 25.203 |

**表 16　6 月 24 日至 6 月 30 日水生根茎类单品的销量分析**

| 品类 | 单品名称 | 单品比例 | 单品销量（千克） |
|---|---|---|---|
| 水生根茎类 | 净藕 (1) | 0.361 | 42.184 |
| | 洪湖藕带 | 0.242 | 28.23 |
| | 高瓜 (1) | 0.180 | 21.031 |
| | 菱角 | 0.105 | 12.224 |
| | 高瓜 (2) | 0.051 | 5.996 |
| | 红莲藕带 | 0.039 | 4.596 |
| | 野生粉藕 | 0.021 | 2.439 |

**表 17　6 月 24 日至 6 月 30 日茄类单品的销量分析**

| 品类 | 单品名称 | 单品比例 | 单品销量（千克） |
|---|---|---|---|
| 茄类 | 紫茄子 (2) | 0.596 | 72.32 |
| | 长线茄 | 0.222 | 29.501 |
| | 青茄子 (1) | 0.140 | 18.727 |
| | 圆茄子 (2) | 0.056 | 7.483 |
| | 紫茄子 (1) | 0.007 | 1,024 |

**表 18　6 月 24 日至 6 月 30 日花叶类单品的销量分析**

| 品类 | 单品名称 | 单品比例 | 单品销量（千克） |
|---|---|---|---|
| 花叶类 | 云南生菜 (份) | 0.259 | 226 |
| | 竹叶菜 | 0.106 | 93.077 |
| | 云南油麦菜 (份) | 0.170 | 149 |
| | 苋菜 | 0.071 | 62.461 |
| | 木耳菜 | 0.048 | 41.534 |
| | 奶白菜 | 0.051 | 44.959 |
| | 小青菜 (1) | 0.039 | 34.306 |
| | 红薯尖 | 0.036 | 31.546 |
| | 娃娃菜 | 0.084 | 73 |
| | 上海青 | 0.031 | 27.029 |
| | 菠菜 (份) | 0.056 | 49 |
| | 云南生菜 | 0.013 | 11.683 |
| | 菜心 | 0.010 | 8.877 |
| | 菠菜 | 0.008 | 6.82 |
| | 外地茼蒿 | 0.007 | 6.538 |
| | 云南油麦菜 | 0.005 | 5.098 |
| | 木耳菜 (份) | 0.048 | 3 |

**表 19　6 月 24 日至 6 月 30 日辣椒类单品的销量分析**

| 品类 | 单品名称 | 单品比例 | 单品销量（千克） |
|---|---|---|---|
| 辣椒类 | 芜湖青椒 (1) | 0.183 | 99.634 |
| | 小米椒 (份) | 0.276 | 150 |
| | 螺丝椒 | 0.088 | 47.897 |
| | 小皱皮 (份) | 0.145 | 79 |
| | 螺丝椒 (份) | 0.145 | 79 |
| | 红椒 (2) | 0.026 | 14.025 |
| | 姜蒜小米椒组合装 (小份) | 0.056 | 49 |
| | 七彩椒 (2) | 0.090 | 7.302 |
| | 青红杭椒组合装 (份) | 0.027 | 15 |
| | 青线椒 (份) | 0.003 | 2 |

**表 20　6 月 24 日至 30 日食用菌单品的销量分析**

| 品类 | 单品名称 | 单品比例 | 单品销量（千克） |
|---|---|---|---|
| 食用菌 | 西峡花菇 (1) | 0.104 | 32.369 |
| | 金针菇 (盒) | 0.365 | 113 |
| | 双孢菇 (盒) | 0.226 | 70 |
| | 海鲜菇 (包) | 0.200 | 62 |
| | 虫草花 (份) | 0.051 | 16 |
| | 蟹味菇与白玉菇双拼 (盒) | 0.026 | 8 |
| | 白玉菇 (袋) | 0.013 | 4 |
| | 鲜木耳 (份) | 0.013 | 4 |

根据问题二的模型,我们可以预测出每种品类一天的销量与销量,根据表15,16,17,18,19,20中每个单品在各自品类中的占比,通过遗传算法,我们可以预测出满足题目需求的单品补

货量和定价策略。最终按题目要求我们可以得到商超收益合理且最大的值为 360.386。

如下图所示, 我们得到了一个最优补货策略, 且经过多次遗传最终保留下来优良序列, 既保证了满足市场对各品类满足市场对各品类蔬菜商品需求，又使得商超收益最大。其中 1 代表补货,0 代表不补货。其列名从左到右依次为七彩椒 (2) 上海青, 云南油麦菜, 云南油麦菜 (份), 云南生菜, 云南生菜 (份), 净藕 (1), 双孢菇 (盒), 圆茄子 (2), 外地茼蒿, 奶白菜, 姜蒜小米椒组合装 (小份), 娃娃菜, 小皱皮 (份), 小米椒 (份), 小青菜 (1), 木耳菜, 木耳菜 (份), 枝江青梗散花, 洪湖藕带, 海鲜菇 (包), 白玉菇 (袋), 竹叶菜, 紫茄子 (1), 紫茄子 (2), 红椒 (2), 红莲藕带, 红薯尖, 芜湖青椒 (1), 苋菜, 菜心, 菠菜, 菠菜 (份), 菱角, 虫草花 (份), 螺丝椒, 螺丝椒 (份), 蟹味菇与白玉菇双拼 (盒), 西兰花, 西峡花菇 (1), 野生粉藕, 金针菇 (盒), 长线茄, 青红杭椒组合装 (份), 青线椒 (份), 青茄子 (1), 高瓜 (1), 高瓜 (2), 鲜木耳 (份), 且每个单品取货量不低于 2.5 千克, 且不同单品数量最多为 30。每个单品的补货量参考图15,16,17,18,19,20

```
0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0
1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0
1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1
1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1
1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1
1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1
0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0
```

**图 21　补货策略对照图**

### 6.4 问题四

#### 6.4.1 模型的建立

考虑蔬菜类商品具有很强的时效性，随时间波动变化很大, 建议从商超的生鲜产品新鲜度和消费者的估价系数等视角来研究蔬菜类商品动态定价的策略[6]该研究结合蔬菜类商品的易腐性特点与消费者的消费心理，提出蔬菜类商品的新鲜度[7]变化和消费者对于蔬菜类商品的时间偏好下的动态定价策略，是对现有蔬菜类商品定价研究的深入与拓展，对具有新鲜度变化的产品动态定价具有较好的理论参考价值

为了更全面地制定蔬菜商品的补货和定价策略，超市可以考虑积极收集以下各种相关数据。这些数据不仅有助于解决市场竞争、消费者需求和库存管理等问题，还可以帮助超市提高经营效率和利润率。动态定价问题四流程图22如下

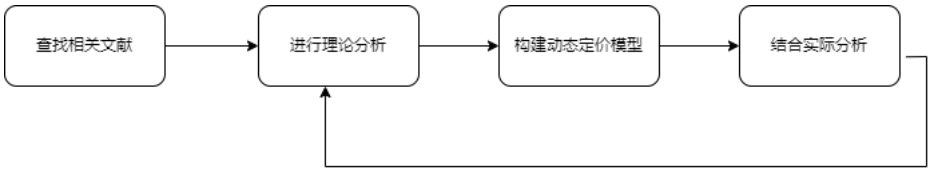



**图 22　第四小问流程图**

1. 收集国内与国外有关蔬菜类商品，新鲜度，蔬菜类商品定价，动态定价相关的学术期刊文献，进行整理与分析，得到相关的文献综述，为本文的研究奠定基础。

2. 充分研究蔬菜类商品的特点，尤其是新鲜度随时间变化这一方面的理论研究。与企业背景问题和数学模型结合起来，将需求函数理论与动态定价方面的方法与理论引入到实际问题中去。

3. 确定市场需求函数，接着根据新鲜度，引出消费者随时间变化的新鲜度偏好函数，将偏好函数带入需求函数，确定最终的需求函数形式。之后确定新鲜期和生鲜衰退期的分解点，分别求出其需求函数，使用积分统计出每个阶段的市场总需求函数。通过市场总需求函数，引入概率密度函数和累计分布函数，确定期望销售量和每个阶段的期望剩余量。最后构造利润函数，对价格 p 求导，得到蔬菜类商品不同阶段的最优定价。

5. 从实际的企业中寻找的补货和定价决策，将这些数据运用到上述的模型中，进行模型求解，并展开数据分析和敏感性分析。

6. 重复 2-5 步直到寻找到最优补货和定价决策中。

### 6.4.2 模型的求解

首先考虑约束条件对于动态定价的模型的建立，一般约束条件有约束时间[8]、目标函数、容量、地点、成本和资源[9]。以约束条件为例

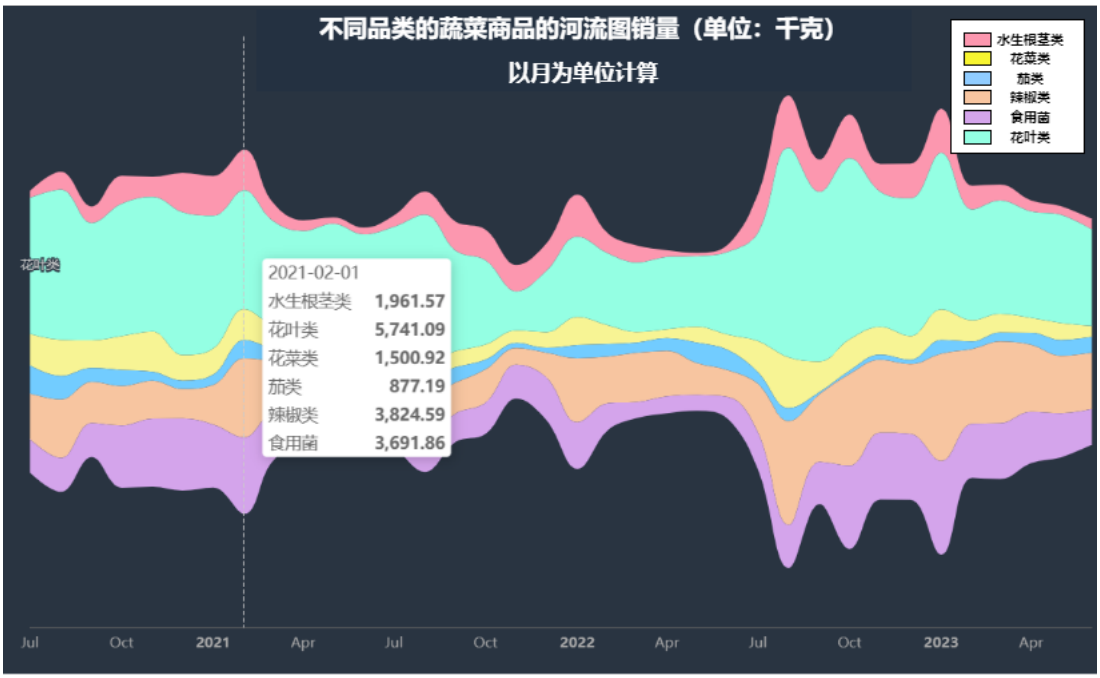



**图 23　2020 年至 2023 年不同品类的河流图**

如图23，观察可得所有品类的销量在 1 月 1 日、8 月 1 日、10 月 1 日出现销量小高峰，且在 2020 年到 2023 年都有改规律，所以在过年等节假日期间进行更大规模的进货量可以获得更大收益

以地点[10]为例，城市比农村的销售量好，靠近学校、医院、政府机构商业街等交通便利的地方的销量高于城郊。因为这些地方的人流量高，人口密度大，人口聚集度稠密。

以蔬菜资源分布为例，特殊品类的蔬菜由于运输距离和成本的不同导致定价比较高，可寻找同类商品的替代单品。减少成本投入。

采取不同的约束条件的组合制定动态定价可优化商品的补货和定价决策。

## 参考文献

[1]　吴海玲. 关联规则 Apriori 算法的改进与应用[Z]. 哈尔滨理工大学, 2023.

[2]　张鸿鸣倪巍伟. 基于差分隐私的数据流频繁项集发布[J]. 计算机工程与设计, 2022, 43(3051-3056).

[3]　郑天宇. 基于神威超算系统的深度学习框架 PyTorch 优化方法研究[Z]. 山东大学, 2023.

[4]　南润. 基于 SARIMA-CNN-LSTM 的车流量预测分析[Z]. 长春工业大学, 2023.

[5] 周跃. 基于遗传算法的机器人路径规划优化研究[J]. 电子元器件与信息技术, 2023.

[6] 程明宝李子怡等. 考虑零售商实行歧视行为的两阶段动态定价策略研究[J]. 管理工程学报.

[7] 翟晓东. 基于比色型智能指示标签的典型冷鲜肉新鲜度实时监测研究[Z]. 江苏大学, 2021.

[8] 曾敏敏. 基于时间情境 A 生鲜社区超市的动态定价策略研究[Z]. 西南财经大学, 2022.

[9] 蒋丹. 农产品消费行为分析及市场动态定价模型研究[Z]. 东北农业大学, 2015.

[10] 李敏. 基于价格弹性和行为选择的共享停车动态定价方法研究[Z]. 宁波大学, 2022.

# 附录 A　apriori 算法

```
# apriori 算法
import warnings
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules

warnings.filterwarnings('ignore')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 创建一个示例 DataFrame
df1 = pd.read_csv(
    "df07.csv",
    parse_dates=[" 时间", " 销售日期", " 扫码销售时间"],
    dtype={
        " 单品编码": "string",
        " 销量（千克）": "float",
        " 销售单价（元/千克）": "float",
        " 销售类型": "category",
        " 是否打折销售": "category",
        " 单品名称": "category",
        " 分类编码": "category",
        " 分类名称": "category",
        " 年": "int",
        " 月": "int",
        " 周数": "int",
        " 季度": "int",
    },
)

# 将时间列转换为时间戳
df1['时间'] = pd.to_datetime(df1['时间'])
```

```
# 创建一个图表 Lift vs. Confidence
plt.figure(figsize=(10, 6))

# 定义颜色列表，每个季度一个颜色
colors = ['b', 'g', 'r', 'c']

for i in [1, 2, 3, 4]:
    # 根据不同的季度创建不同的 df_month
    df_month = df1[df1[" 季度"] == i]
    # 使用 pd.Grouper 按每十分钟分组
    dataset = []
    df_grouped = df_month.groupby(pd.Grouper(key='时间', freq='10T'))

    for name, group in df_grouped:
        tran = group[" 单品名称"].tolist()
        dataset.append(tran)

    te = TransactionEncoder()
    te_ary = te.fit(dataset).transform(dataset)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    # 计算关联规则，保留列名
    frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)
    # 手动计算置信度
    rules = association_rules(frequent_itemsets, metric="support", min_threshold=0.01
    rules["confidence"] = rules["support"] / rules["antecedent support"]

    # 绘制散点图，使用不同的颜色表示不同季度
    plt.scatter(rules['lift'], rules['confidence'], alpha=0.5, label=f'Quarter {i}',

plt.xlabel('Lift')
plt.ylabel('Confidence')
plt.title('Lift vs. Confidence')
plt.legend()
```

```
# 创建一个图表 Support vs. Confidence
plt.figure(figsize=(10, 6))
for i in [1, 2, 3, 4]:
    # 根据不同的季度创建不同的 df_month
    df_month = df1[df1[" 季度"] == i]
    # 使用 pd.Grouper 按每十分钟分组
    dataset = []
    df_grouped = df_month.groupby(pd.Grouper(key='时间', freq='10T'))

    for name, group in df_grouped:
        tran = group[" 单品名称"].tolist()
        dataset.append(tran)

    te = TransactionEncoder()
    te_ary = te.fit(dataset).transform(dataset)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    # 计算关联规则，保留列名
    frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)
    # 手动计算置信度
    rules = association_rules(frequent_itemsets, metric="support", min_threshold=0.01
    rules["confidence"] = rules["support"] / rules["antecedent support"]

    # 绘制散点图，使用不同的颜色表示不同季度
    plt.scatter(rules['support'], rules['confidence'], alpha=0.5, label=f'Quarter {i}
plt.xlabel("Support")
plt.ylabel("Confidence")
plt.title("Support vs. Confidence")
plt.legend()
plt.show()
print(rules.count())
```

# 附录 B　LSTM 时序网络

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import torch
import torch.nn as nn

data = pd.read_csv("shiyongjun.csv")
data = data[['年季度', '总销量（千克）']]
# print(data.iloc[-7:])
# print(data)
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
# print(device)
data[" 年季度"] = pd.to_datetime(data[" 年季度"])
plt.plot(data['年季度'], data['总销量（千克）'])
plt.show()

from copy import deepcopy as dc

def prepare_dataframe_for_lstm(df, n_steps):
    df = dc(df)

    df.set_index('年季度', inplace=True)

    for i in range(1, n_steps + 1):
        df[f'总销量（千克）(t-{i})'] = df['总销量（千克）'].shift(i)

    df.dropna(inplace=True)

    return df

lookback = 7
shifted_df = prepare_dataframe_for_lstm(data, lookback)
```

```
# print(shifted_df)
# print(len(shifted_df))
shifted_df_as_np = shifted_df.to_numpy()
# print(shifted_df_as_np)
# print(shifted_df_as_np.shape)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(-1, 1))
shifted_df_as_np = scaler.fit_transform(shifted_df_as_np)
# print(shifted_df_as_np)
X = shifted_df_as_np[:, 1:]
y = shifted_df_as_np[:, 0]
# print(X.shape)
# print(y.shape)
X = dc(np.flip(X, axis=1))
# print(X)
split_index = int(len(X) * 0.8)
# print(split_index)
X_train = X[:split_index]
X_test = X[split_index:]

y_train = y[:split_index]
y_test = y[split_index:]
# print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
X_train = X_train.reshape((-1, lookback, 1))
X_test = X_test.reshape((-1, lookback, 1))

y_train = y_train.reshape((-1, 1))
y_test = y_test.reshape((-1, 1))
# print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
X_train = torch.tensor(X_train).float()
y_train = torch.tensor(y_train).float()
X_test = torch.tensor(X_test).float()
y_test = torch.tensor(y_test).float()
```

```python
# print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
from torch.utils.data import Dataset

class TimeSeriesDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, i):
        return self.X[i], self.y[i]


train_dataset = TimeSeriesDataset(X_train, y_train)
test_dataset = TimeSeriesDataset(X_test, y_test)
# print(train_dataset)
from torch.utils.data import DataLoader

batch_size = 16

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

for _, batch in enumerate(train_loader):
    x_batch, y_batch = batch[0].to(device), batch[1].to(device)
    print(x_batch.shape, y_batch.shape)
    break


class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_stacked_layers, dropout_prob):
        super().__init__()
        self.hidden_size = hidden_size
```

```
        self.num_stacked_layers = num_stacked_layers

        self.lstm = nn.LSTM(input_size, hidden_size, num_stacked_layers,
                            batch_first=True, dropout=dropout_prob)

        self.fc = nn.Linear(hidden_size, 1)

    def forward(self, x):
        batch_size = x.size(0)
        h0 = torch.zeros(self.num_stacked_layers, batch_size, self.hidden_size).t
        c0 = torch.zeros(self.num_stacked_layers, batch_size, self.hidden_size).t

        out, _ = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

model = LSTM(1, 4, 1, 0.2)
model.to(device)
# print(model)
def train_one_epoch():
    model.train(True)
    print(f'Epoch: {epoch + 1}')
    running_loss = 0.0

    for batch_index, batch in enumerate(train_loader):
        x_batch, y_batch = batch[0].to(device), batch[1].to(device)

        output = model(x_batch)
        loss = loss_function(output, y_batch)
        running_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
            if batch_index % 100 == 99:  # print every 100 batches
                avg_loss_across_batches = running_loss / 100
                print('Batch {0}, Loss: {1:.3f}'.format(batch_index + 1,
                                                        avg_loss_across_batches))
                running_loss = 0.0
        train_losses.append(running_loss / len(train_loader))
        print()


    def validate_one_epoch():
        model.train(False)
        running_loss = 0.0

        for batch_index, batch in enumerate(test_loader):
            x_batch, y_batch = batch[0].to(device), batch[1].to(device)

            with torch.no_grad():
                output = model(x_batch)
                loss = loss_function(output, y_batch)
                running_loss += loss.item()

        avg_loss_across_batches = running_loss / len(test_loader)
        val_losses.append(avg_loss_across_batches)

        print('Val Loss: {0:.3f}'.format(avg_loss_across_batches))
        print('***************************************************')
        print()

    learning_rate = 0.001
    num_epochs = 200
    loss_function = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    # 记录损失函数
    train_losses = []
```

```
val_losses = []

for epoch in range(num_epochs):
    train_one_epoch()
    validate_one_epoch()

# 绘制损失函数图像
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

with torch.no_grad():
    predicted = model(X_train.to(device)).to('cpu').numpy()

plt.plot(y_train, label='Actual Sell')
plt.plot(predicted, label='Predicted Sell')
plt.xlabel('Day')
plt.ylabel('Sell(kg)')
plt.legend()
plt.show()

train_predictions = predicted.flatten()

dummies = np.zeros((X_train.shape[0], lookback+1))
dummies[:, 0] = train_predictions
dummies = scaler.inverse_transform(dummies)

train_predictions = dc(dummies[:, 0])
# print(train_predictions)

dummies = np.zeros((X_train.shape[0], lookback+1))
dummies[:, 0] = y_train.flatten()
```

```
dummies = scaler.inverse_transform(dummies)

new_y_train = dc(dummies[:, 0])
# print(new_y_train)
plt.plot(new_y_train, label='Actual Sell')
plt.plot(train_predictions, label='Predicted Sell')
plt.xlabel('Day')
plt.ylabel('Sell(kg)')
plt.legend()
plt.show()

test_predictions = model(X_test.to(device)).detach().cpu().numpy().flatten()

dummies = np.zeros((X_test.shape[0], lookback+1))
dummies[:, 0] = test_predictions
dummies = scaler.inverse_transform(dummies)

test_predictions = dc(dummies[:, 0])

# print("--------------------")
# print(test_predictions)

dummies = np.zeros((X_test.shape[0], lookback+1))
dummies[:, 0] = y_test.flatten()
dummies = scaler.inverse_transform(dummies)

new_y_test = dc(dummies[:, 0])

plt.plot(new_y_test, label='Actual Sell')
plt.plot(test_predictions, label='Predicted Sell')
plt.xlabel('Day')
plt.ylabel('Sell(kg)')
plt.legend()
plt.show()
```

```
# 获取最后七天的历史数据
last_seven_days_data = X_test[-7:]

# 将数据转换成 PyTorch 张量并移动到相应的设备
last_seven_days_data = torch.tensor(last_seven_days_data).float().to(device)

# 使用模型进行预测
with torch.no_grad():
    predicted_sales = model(last_seven_days_data).detach().cpu().numpy()

# 将预测的销量反归一化
predicted_sales = scaler.inverse_transform(np.column_stack((np.zeros((7, lookback
predicted_sales = predicted_sales[:, -1]  # 获取最后一天的预测值

print("Predicted Sales for the Next 7 Days:")
print(predicted_sales)
```

## 附录 C　遗传算法

```
import copy
import random
import pandas as pd

df2 = pd.read_csv('24-30. 平均销量与平均利润.csv', encoding='utf-8')

singleProfit = df2['平均利润'].tolist()
signalSales = df2['一天的预测销量'].tolist()


group = []
# 初始化种群
# 从 500 条序列中筛选满足条件的序列
for x in range(10000):
```

```
        line = [random.choice([0, 1]) for x in range(49)]
        # for X in range(0,len(line)):
        #     if signalSales[X] < 2.5:
        #         line[X] = 0
        s = sum(line)
        if 27 <= s <= 33:
            group.append(line)

if len(group) < 100:
    exit()

file = open("result.txt", "w+", encoding="utf-8")

# 大于两百条则只取前 100 条数据
group = group[:100]

for N in range(10000):
    file.write("\n")
    file.write(str(N)+"\n")
    # 重新洗牌
    random.shuffle(group)

    # 随机生成 group 一段序列的下标，进行 DNA 序列互换
    i = random.randint(0, 48)
    j = random.randint(i, 48)

    cnt = len(group)
    cnt20 = round(cnt * 0.2)
    if cnt20 % 2 != 0:
        cnt20 -= 1
    rnd20 = random.sample([i for i in range(cnt)], cnt20)
    for v in range(0, cnt20, 2):  # 互换
        x = rnd20[v]
        y = rnd20[v + 1]
        line1 = copy.copy(group[x])
```

```
        line2 = copy.copy(group[y])
        # print(line2)
        # print(line1)
        temp1 = line1[i:j]
        temp2 = line2[i:j]
        line1[i:j] = temp2
        line2[i:j] = temp1
        group.append(line1)
        group.append(line2)
        # print(line2)
        # print(line1)

    # 两两序列互换，若序列为奇数个，舍弃最后一个奇数项
    # if cnt % 2 != 0:
    #     cnt -= 1
    # for x in range(0, cnt, 2):  # 互换
    #     y = x + 1
    #     line1 = group[x]
    #     line2 = group[y]
    #     # print(line2)
    #     # print(line1)
    #     temp1 = line1[i:j]
    #     temp2 = line2[i:j]
    #     line1[i:j] = temp2
    #     line2[i:j] = temp1
    #     # print(line2)
    #     # print(line1)

    # 变异
    # 随机生成 group 下标
    k = random.randint(0, 48)
    # 选择 group 中的百分之 10 进行变异即可
    cnt10 = round(len(group) * 0.1)
    rnd10 = random.sample([i for i in range(len(group))], cnt10)
    for i in rnd10:
```

```
        line = group[i]
        if line[k] == 1:
            line[k] = 0
        else:
            line[k] = 1

# 打乱之后再次筛选 group 内序列
group1 = []
print()
for x1 in range(cnt):
    line1 = group[x1]
    s = sum(line1)
    if 27 <= s <= 33:
        group1.append(line1)



max_total = 0
best_line = []
profit = []
for line in group1:
    # 计算总利润
    total = 0
    # selected = [e for e in line if e == 1]
    q = 0
    for single in line:
        # 计算单品的利润
        total += single * singleProfit[q]
        q += 1
    print(total)
    profit.append((total, line))


sorted(profit, key=lambda w: w[0], reverse=True)
    # profit 排序,
```

```
    # 对应种群
    # top80
# for M in profit:
#     file.write(str(M[1])+"\n")
for K in profit:
    file.write(str(K[0])+"\n")
top80 = round(len(profit) * 0.8)
row = []
sum80 = 0
for H in range(len(profit)):
    sum80 += profit[H][0]
    row.append(profit[H][1])
avg80 = sum80 / top80
file.write(f" 平均利润：{avg80}")
group = row[:top80]

# 补充到 100
cnt3 = 100-top80
rnd3 = random.sample([i for i in range(len(group))], cnt3)
for t in rnd3:
    line3 = copy.copy(group[t])
    group.append(line3)

# file.write(f"{len(profit)}\n")
# file.write(str(top80))
# for H in range(top80):
#     file.write(str(profit[H][0])+"\n")
    # file.write(profit[H][1])


    # 检查是否是新的最大值
    # if total > max_total:
    #     max_total = total
    #     best_line = line
```

```
# print()
# print(f'最大利润是: {max_total}')
# print(f'对应的项集是: {best_line}')
```