基于 BP 神经网络的线性回归预测与非线性规划的商超蔬菜补货定价策略模型

摘要

本文针对商超的补货定价问题，首先结合需求价格理论推导成本加成定价法的理论模型，并基于 BP 神经网络及一元线性回归建立预测模型，随后根据收益最大的目标建立非线性规划模型并运用模拟退火算法进行求解，最后给出最优补货定价策略。

针对问题一探究不同类别及单品蔬菜得销售量的分布规律与其相关关系。首先对附件数据进行整理得到各品类及单品的销售量的时间序列数据，并基于此数据绘制**箱线图**，直观地反映出各品类及各单品蔬菜的**总体分布规律**，例如花叶类蔬菜的平均销售量最多，茄类蔬菜的平均销售量最少；花叶类蔬菜中，木耳菜的平均销售量最多。其次利用 SPSS 绘制**时间序列图**并进行**季节性分解**得出时间维度上的分布规律：水生根茎类、茄类蔬菜有明显的季节性，且后者还有逐渐增长的长期趋势。对于各蔬菜的相关关系，本文先对各品类及单品的销售量数据进行了 **JB 检验**，发现品类销量数据满足正态分布特征而单品销量数据不满足，于是选择运用**皮尔逊相关系数**度量品类间的相关关系，而用**斯皮尔曼相关系数**度量单品间的相关关系，最后得出结论与合理解释，例如品类间花菜类和花叶类的销售量相关性较高，其相关系数达 0.75。

针对问题二要求给出未来一周各品类蔬菜的补货量与定价策略，本文首先从**供给侧与需求侧**出发考虑，推导出**成本加成定价法**的理论模型公式，并结合经济学原理从附件数据中挖掘出与定价有关的数据如成本、损耗量、进价、定价等得出历史各品类蔬菜的定价。随后分别采用 **ARIMA 预测模型**以及 **BP 神经网络预测模型**对各品类未来七天的销量进行预测，并基于预测值计算补货量。通过比较两个模型的**拟合优度**后，选择预测效果更优的 BP 神经网络预测模型。最后建立**线性回归模型**并输入历史各品类蔬菜的销量与定价数据，得到各品类蔬菜的定价回归方程，从而基于先前预测出的销售量得出未来定价决策：花叶类的补货量决策为 23.75、26.3、24.29、22.81、24.45、25.74、27kg;定价为 10.75、10.68、10.74、10.78、10.73、10.70、10.66 元/kg。

针对问题三，要求在问题二的基础上进一步制定未来一天各蔬菜单品的补货量与定价策略。根据目标及相关约束，考虑建立**非线性规划模型**，设置目标为最大化商超利润，约束条件考虑可售单品种数限制、最小陈列量要求、各类蔬菜需求、进货量上限、定价上下限。引入决策变量表示 7 月 1 日是否售卖该种蔬菜单品。使用**模拟退火算法**进行求解，得出 7 月 1 日售卖的单品为西兰花、金针菇、竹叶菜、螺丝椒、菱角、长线茄等 29 种，其补货量及定价见表 13。

针对问题四，要求寻找有助于优化补货及定价策略的其他有关数据，本文建议采集组合销售数据、价格敏感度数据、随陈列时间变长的价值损耗数据以协助制定补货定价策略。**组合销售数据**是被同时购买的菜品信息数据，有助于补货选品决策及销售组合定价决策。**价格敏感度数据**表示价格的变动引起的对产品需求的变化，有助于根据价格敏感度决定对菜品价格的调整力度以及预测价格调整后的需求情况。**价值损耗数据**是随着蔬菜价值及顾客的价格期望随新鲜度变化的数据，有利于根据新鲜度及价格期望采用动态定价法定价，并选择受新鲜度影响小的蔬菜进行售卖。

关键词：供求原理　斯皮尔曼相关系数　BP 神经网络预测　线性回归　非线性规划

一、 问题重述

1.1 问题背景

生鲜产品是人们日常生活的必需品，也是生鲜类超市的主要商品，决定着生鲜超市的利润。常见的生鲜类产品，如蔬菜，保质期不长，且其品质随着储存时间的变长而变差。大多数种类的蔬菜仅限进货当天售卖，储存一天后将失水变蔫，甚至腐烂，以至于无法再进行售买。因此，生鲜超市根据以往客户对不同种类蔬菜的需求情况及供货商对蔬菜的供应情况，制定合理的补货及定价策略尤为重要。

1.2 问题重述

问题设置层层递进，引导分析以往销售量的内在规律及不同种类类蔬菜销售量的相关关系及销售量与定价相关关系，并据此预测日后的需求量，从而提出合理的补货、定价策略。

问题一要求探究各蔬菜类别及每类蔬菜中的具体单品销售量的分布规律，并分析不同类别的蔬菜销售量之间的相互关系，以及不同蔬菜单品销售量之间的关联关系。

问题二首先提出以蔬菜类别作为后续补货的基本单位，要求分析每个类别的销售量之和与其通过成本加成定价法得到的定价的相关关系，并探究在保证超市利润尽可能大的情况下，制定未来七日中每天各类别的补货量以及对蔬菜的定价办法。

问题三提出因空间限制，需进一步细化各蔬菜的进货计划，以各类别中的具体蔬菜为进货单位，且各蔬菜进货量不低于 2.5kg，总数目介于 25 到 31 之间。要求在参考前一周的售卖种类，尽量同时满足市场需求及超市利润最大化，给出未来一周的补货及定价计划。

问题四要求提出有利于超市优化进货及定价方案的其他有关数据，以协助解决上述问题，并阐述理由。

二、 问题分析

2.1 问题一的分析

本问要求探究不同类别蔬菜销售量及不同蔬菜单品销售量的分布规律，并分析各类别之间与各蔬菜单品之间销售量的相关关系。首先，本问主要涉及附件一和附件二中数据，先进行数据预处理，将销售量数据以月为单位，按数据加总整理，便于后续分析且加总抵消了退货对销售量的影响，并对处理后的数据进行平稳性检验。其次，运用箱线图直观展示数据的总体分布情况，绘制时间序列图观察销售量随时间的变化规律，并在观察分析后，描述分布规律。最后，对各组数据进行 JB 检验，判断其是否符合正态分布，根据检验结果，考虑计算其皮尔逊相关系数或斯皮尔曼相关系数，以并用热力图展示计算结果，便于显示和比较数据间相关关系大小。

2.2 问题二的分析

本问要求运用成本加成法推导销售总量与定价的关系，并对 2023 年 7 月 1 日-7 日的补货总量和定价策略。该题思路如下：首先对附件二、三、四数据进行预处理，按照天数分别将各品类的销售量加总，针对每个品类销量的时间序列数据，对其中单

品批发价格进行销量加权，得到各品类的进货价格，并将附件四中某品类所有单品的损耗率取平均值得到该品类的损耗率。其次分别建立神经网络预测模型和 ARIMA 预测模型，对未来七天的销售量进行预测，并比较两个模型的拟合优度，筛选出预测效果更好的模型为 BP 神经网络预测模型。接着通过查找经济学原理知识，结合已知数据推导出成本加成定价的理论模型，并计算出过去四年各品类的定价额。最后建立线性回归模型，输入先前计算出的各品类定价与销量的历史数据进行回归分析，得到定价与销售量的具体回归方程，并代入提前预测出的未来七天的销售量得到未来七天的定价。

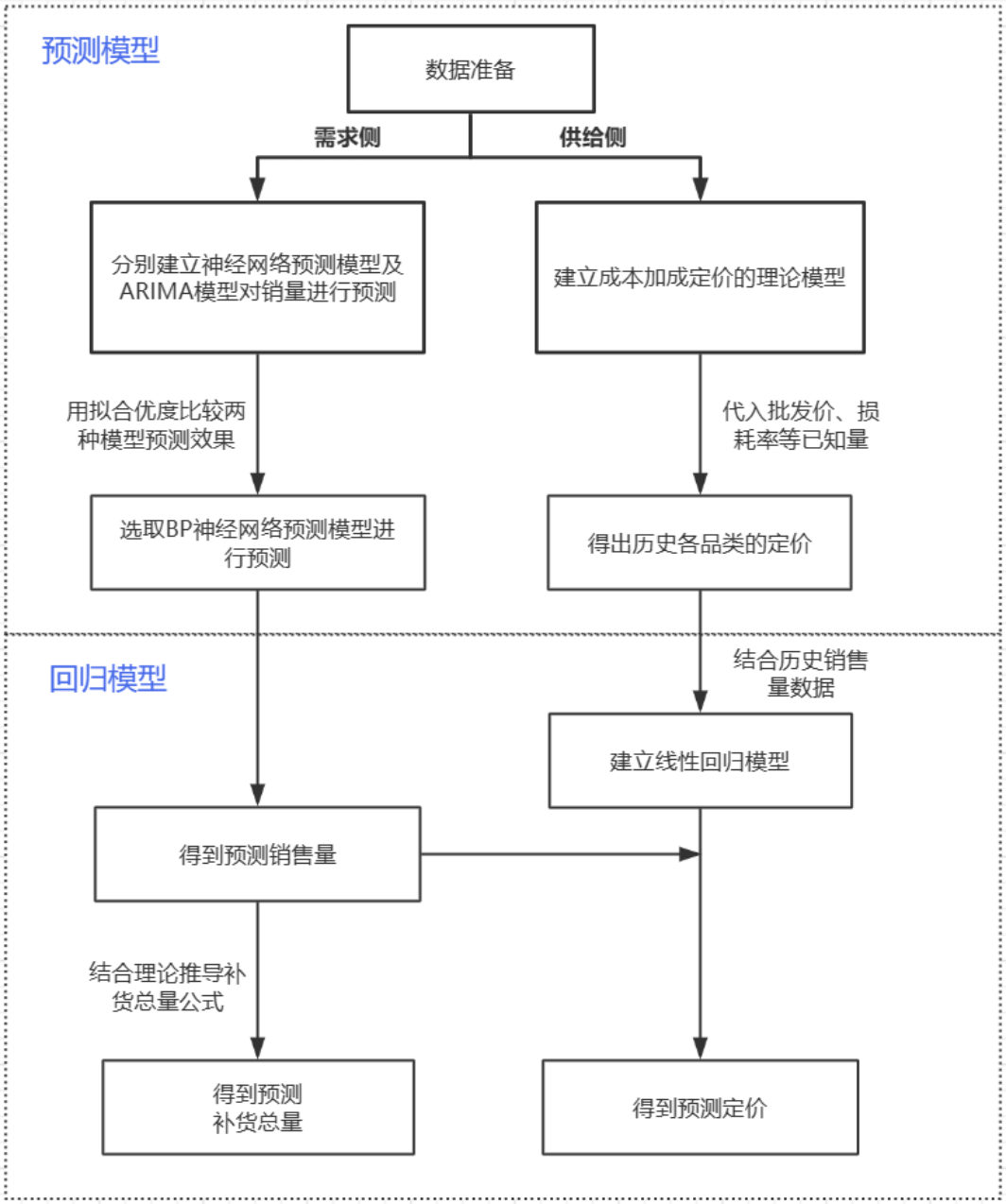


图 1 问题二思路分析

2.3 问题三的分析

本问提出因空间限制，需在问题二的基础上进一步制定各个蔬菜单品进货计划。首先，通过筛选附件三中的数据，整理得出 2023 年 6 月 24-30 日的可售品种。其次，根据附件二的结果，得出 2023 年 7 月 1 日各蔬菜品类销售量的预测值，并综合损耗率计算出各类别蔬菜的需求量。最后，考虑使用规划模型，将可售单品总数要求、最小成列量要求、进货量的上限作为约束条件，并将超市利润最大化作为目标函数，解出模型后得到 7 月 1 日蔬菜单品的补货量及定价策略。

2.4 问题四的分析

本问要求寻找有助于优化补货及定价策略的其他有关数据。首先，考虑到菜品与菜品间的关联关系及搭配，本文建议采集组合销售数据来识别不同菜品组合的关系，以助于补货选品决策及销售组合定价决策。其次，考虑到经济学原理中商品价格与需求的关系，本文建议收集价格敏感度数据，有助于根据价格敏感度决定对菜品价格的调整力度以及预测价格调整后的需求情况。最后，考虑到蔬菜保鲜期短，随着上架时间变长品相越来越差，导致人们购买欲望下降，故建议收集蔬菜价值损耗数据，以识别随着蔬菜价值及顾客的价格期望随新鲜度变化。结合蔬菜销售的特点及经济学原理，采集以上数据，可以有效优化补货定价决策的制定，从而实现商超收益最大化。

三、 模型假设

1.假设退货商品继续售卖，不计入损耗。
2.假设损耗量与销售量之和即为进货量，商超无赠送活动。
3.假设市场价格稳定，商超进价与售价不会剧烈变动。
4.假设价格对销售量存在影响，价格上升，销售量减少；价格下降，销售量增加。

四、 符号说明

表 1 符号说明

符号	含义
i	第 i 个蔬菜单品
J	第 J 个蔬菜品类
t	第 t 天
L	超市总利润
w_i	第 i 种单品在 7 月 1 日的定价

j_i	第 i 种单品的批发价格
y_i	第 i 种单品的进货量
e_i	第 i 种单品的损耗率
x_i	可取 0 或 1，x_i取 0 是表示 7 月 1 日不售卖第 i 种单品，x_i取 1 时，表示 7 月 1 日售卖第 i 种单品。
D_J	第 J 个蔬菜品类的需求量
w_{it}	第 i 种单品的的定价
e_J	第 J 个蔬菜品类中单品的平均损耗率

五、 数据预处理

5.1 缺失值

对于题目给出的数据进行缺失值查找，利用 MATLAB 的 find 函数。若有缺失值，可删除有缺失值的样本或特征，或用平均值、前后的数值对缺失处进行填补。结果显示，数据中不存在缺失值，无需处理。

5.2 数据合并整理

将不同类别的蔬菜及各蔬菜单品 2020 年 8 月至 2023 年 6 月间每月的销售量数据分别合并加总，一是便于后续分析，二是退货通常在一月内完成，加总抵消了退货对销售量的影响。整理方法为使用 Excel 的 VLOOKUP 函数提取各月份的数据后，再分组加总。

5.3 平稳性检验

为了检验时间序列数据是否具有稳定的分布特征，需要对数据进行平稳性检验。平稳性也是使用时间序列模型进行估计的前提，本文采用 ADF 检验进行平稳性检验。

ADF 检验的全称是 Augmented Dickey-Fuller test，是 Dickey-Fuller test 的拓展，DF 检验只可用于一阶情况，当序列存在高阶的滞后相关时，可使用 ADF 检验。进行 ADF 检验时采用的是不指定 trend，也就是检验是否含截距项平稳，显著性水平 p 值>0.05，不拒绝原假设，非平稳；显著性 p<0.05，接受原假设，则为平稳。利用 python 实现了 ADF 检验。以各蔬菜类别结果为例，除花叶类蔬菜外，其余 p 值均小于 0.05，且花叶类 p 值为 0.056，可近似看作通过了平稳性检验。ADF 检验证明数据平稳，可用于后续研究。具体代码及检验结果见附录。

六、 问题一模型的建立与求解

6.1 各品类及单品的分布规律

根据题干要求，首先将附件二数据中单品编号与附件一中的对应品类进行匹配，并将附件二中每个品类的销售量按月份汇总，得到各品类销售量的箱线图以及时间序列图，以此观察得出各品类及单品的总体分布规律，及时间维度上的分布规律。

6.1.1 总体分布规律

从下图可以直观地看出，于品类而言，花叶类蔬菜的平均销售量最多，茄类蔬菜的平均销售量最少，且花叶类蔬菜的不同单品间总销售量差距最大，茄类蔬菜各单品间的总销量差距最小，这可能是因为花叶类蔬菜的单品种类繁多，从而导致价格、销量等存在较大差异。

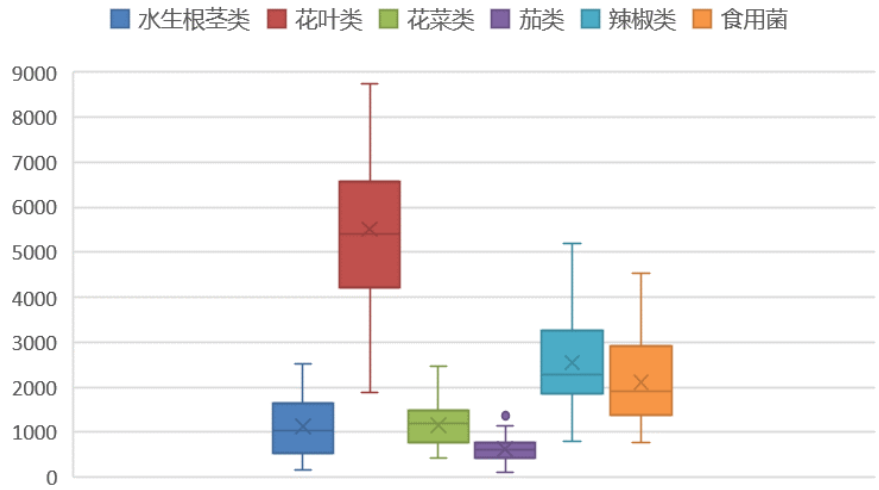
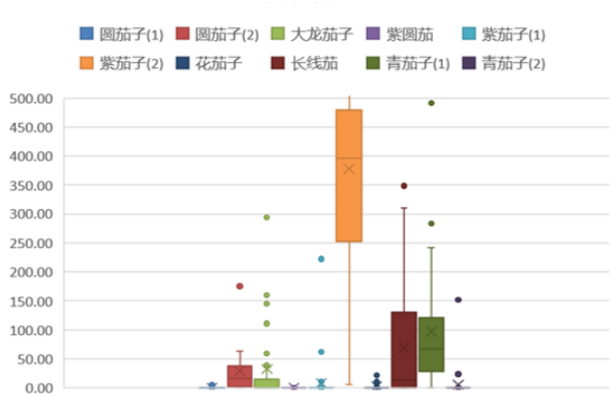


图 2 不同品类总销量箱线图

此外，根据图 3 可知，于单品而言，茄类蔬菜中紫茄子的平均销售量最多，花菜类蔬菜中西兰花的销售量最多，食用菌中双胞菇的平均销售量最多，花叶类蔬菜中，木耳菜的平均销售量最多，辣椒类蔬菜中，小米椒的平均销售量最多，水生根茎类蔬菜中，净藕的平均销售量最多。



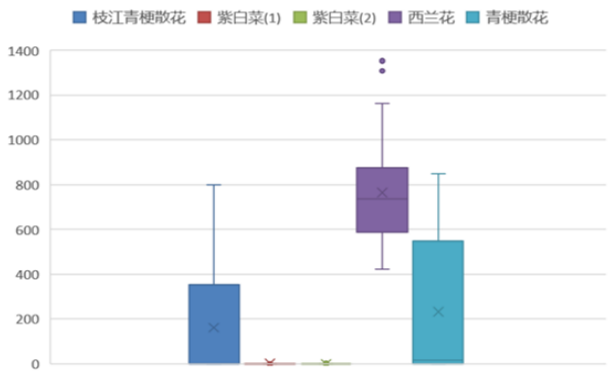


图 3 不同单品总销量箱线图

6.1.2 时间维度上的分布规律

在对上述数据进行取自然对数处理之后，运用 SPSS 时间序列分析器绘制时序图如下，根据图 5，可以看出水生根茎类蔬菜有着明显的季节性：从每年的七月开始，其销售量逐渐走高，并在次年的一月左右销量达到顶峰，从二月起，销量开始大幅回落，直到又一个销售周期开始。茄类蔬菜也有着较为明显的季节性和长期趋势，由图 5 可得，其销售量在每年的十一月从低谷逐渐增加，并在次年五月左右达到最大值后下降，从长期趋势上看，该品类销售量有着减少的趋势。而其他品类的蔬菜没有明显的时间维度上的规律特征。

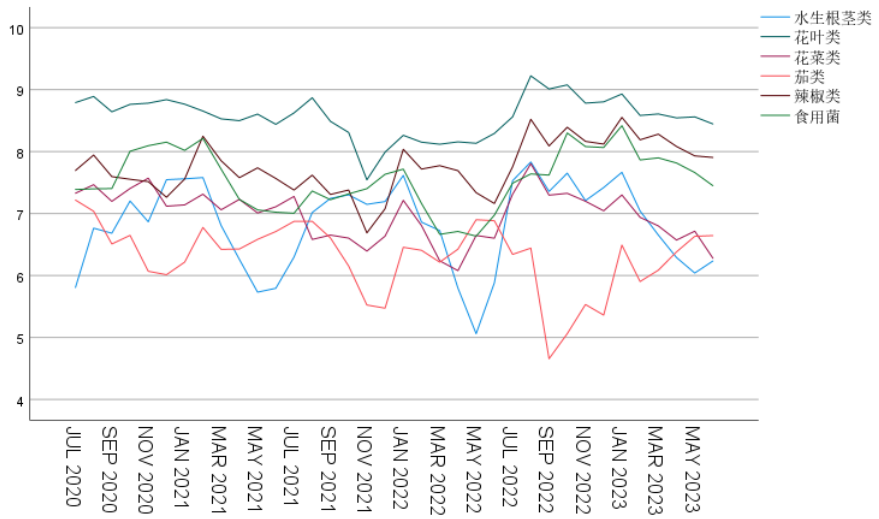


图 4 各品类销量的时间序列图

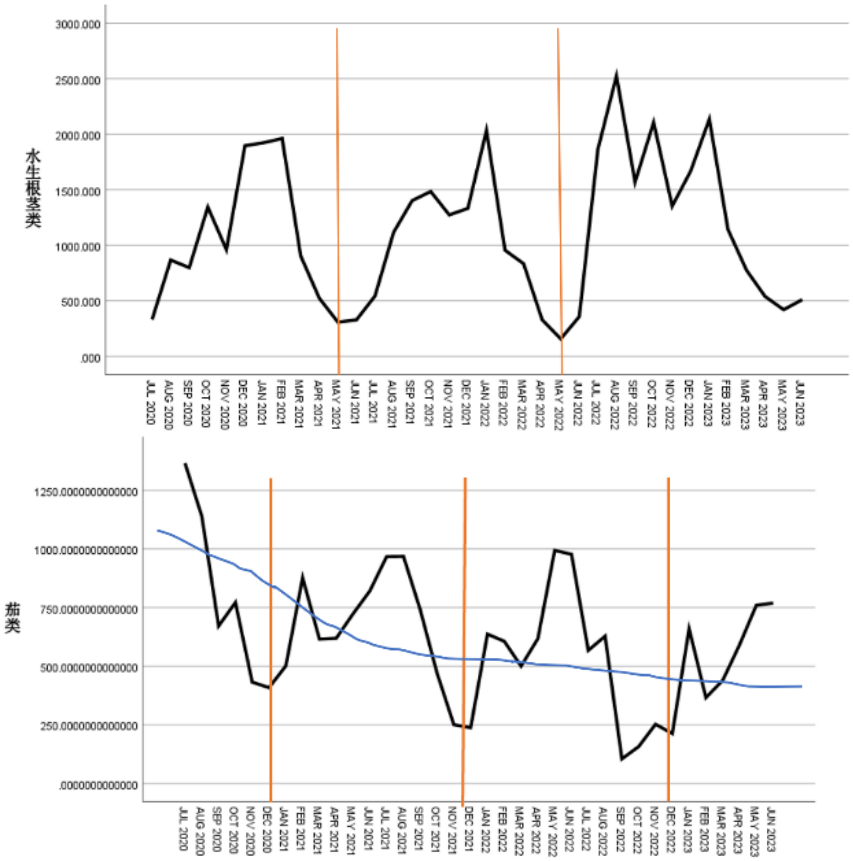


图 5 水生根茎类与茄类销量的时间序列图

6.2 相关性分析

常用相关系数探究变量间相关性的强弱，故考虑在进行正态分布检验后，采用皮尔逊相关系数或斯皮尔曼相关系数度量不同类别的蔬菜销售量之间的相互关系，以及不同蔬菜单品销售量之间的关联关系。

6.2.1 正态分布检验

皮尔逊相关系数适用于呈正态分布的连续变量，故需先对数据进行正态分布检验。本题涉及的数据为大样本，故选用雅克-贝拉检验(Jarque-Bera test)来判断各组数据是否符合正态分布。其检验原理是：对于一个随机变量 $\{X_i\}$，构造 JB 统计量：

$$JB = \frac{n}{6}\left[S^2 + \frac{(K-3)^2}{4}\right]$$

其中，其中 n 为样本容量，S 为样本偏度，K 为峰度。若服从正态分布，那么 S 为 0，K 为 3，JB 统计量逐渐减小。当显著性水平 α =0.05 时，$\chi^2(2)$=5.99147，若 JB 统计量小于临界值 5.99147，则接受原假设，即样本服从正态分布，否则应拒绝原假设。

对六种蔬菜销售量数据的 JB 检验结果如下，均小于 5.99147，故均符合正态分布，可计算其皮尔逊相关系数。

JB 检验结果(h): [2.105402 0.910489 1.798564 0.576506 3.712456 2.787277]
JB 检验结果(p): [0.348994 0.634293 0.406862 0.749572 0.156261 0.248171]

但对不同蔬菜单品做 JB 检验的结果显示，仅有极个别单品销售量数据服从正态分布，故考虑用斯皮尔曼相关系数刻画蔬菜单品间相关关系强弱，蔬菜单品的 JB 检验结果见附录。

6.2.2 运用皮尔逊相关系数度量品类间相关关系

皮尔逊相关系数，又称皮尔逊积矩相关系数，可用于衡量两组数据线性相关关系的强弱。当两个呈正态分布的变量都是连续变量，且两者之间呈线性关系时，适合选用皮尔逊相关系数来度量二者的相关程度。皮尔逊相关系数具体计算公式如下：

$$r_p = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2} \cdot \sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

上式中r_p表示皮尔逊相关系数，n 代表样本个数，X_i与Y_i分别表示两组变量的第 i 个样本。r_p的数值介于 0 到 1 之间，r_p的绝对值越接近于 1，说明两组变量的线性相关程度越大。相关系数的正、负分别代表正相关、负相关。计算得到六种类别的蔬菜的销售量的相关关系相关系数后，将相关系数表进行可视化处理，绘制热力图，更加清楚直观，便于比较。计算得到的相关性热力图如图 6 所示。

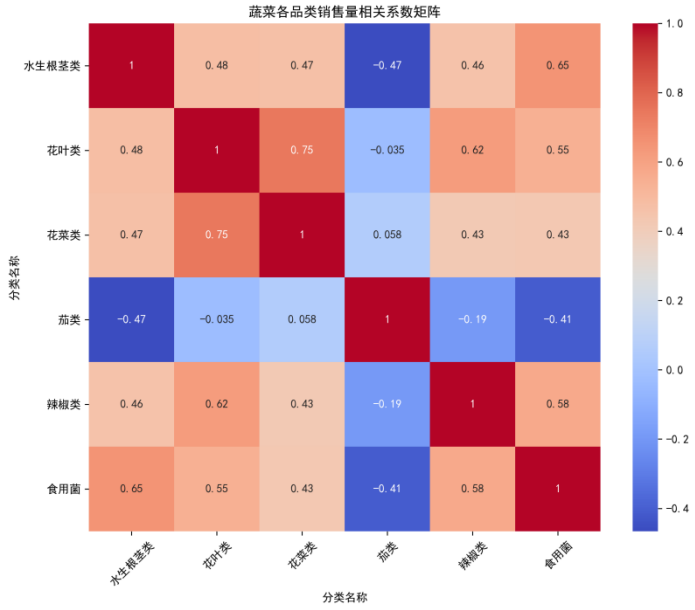


图 6 六种蔬菜的皮尔逊相关系数热力图

观察上图可以得知，花菜类和花叶类的销售量相关性较高，相关系数为 0.75，花叶类与茄类、花菜类与茄类以及辣椒类与茄类销售量的相关性较小，分别为-0.035、0.058 和 0.19，其绝对值均小于 0.3，相关性极弱，认为无相关关系。

6.2.3 运用斯皮尔曼（等级）相关系数度量单品间的相关关系

当变量数据不满足正态分布时，皮尔逊相关系数度量变量间关系的方法将不再适用，此时，可通过计算对数据要求较少的斯皮尔曼相关系数进行变量间相关性的度量。由上文 JB 检验可知，大多数单品的销售量分布并不满足正态分布，因此选择计算斯皮尔曼相关系数的方法衡量各单品间是否单调相关。

斯皮尔曼相关系数是利用两个变量的秩进行线性相关分析，因此两组随机变量$X = (x_1, x_2, \dots, x_n)$和$Y = (y_1, y_2, \dots, y_n)$斯皮尔曼等级相关系数定义如下：

$$r_s = \frac{\sum_{i=1}^{n} (p_i - \bar{p})(q_i - \bar{q})}{\sqrt{\sum_{i=1}^{n} (p_i - \bar{q})^2} \sqrt{\sum_{i=1}^{n} (q_i - \bar{q})^2}}$$

公式中，p_i和q_i分别是x_i和y_i的秩（$i = 1,2,3 \dots, n$），若出现观测值相等的情况，则该观测值对应的秩为这几个值相应的秩的平均值，如果变量间的秩可以计算，则斯皮尔曼（等级）相关系数的公式可以简化为

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

其中，d_i为随机变量 X、Y 中对应观测值x_i和y_i的秩的差值。

若$0 < r_s \leq 1$, 表明 X 和 Y 存在正相关关系；若$-1 \leq r_s < 0$, 表明呈负相关；若$r_s = 0$，表示不相关。

表 2 列举了部分具有代表性单品间的斯皮尔曼（等级）相关系数计算结果，其他全部单品间的斯皮尔曼（等级）相关系数的计算结果见附录。

表 2 部分单品间的斯皮尔曼相关系数

单品 1	单品 2	Correlation
紫白菜(2)	西峡花菇(2)	1
红珊瑚(粗叶)	西峡花菇(2)	1
灯笼椒(份)	黑皮鸡枞菌(盒)	1
海鲜菇(2)	鱼腥草	1
紫尖椒	黑牛肝菌(盒)	1
猪肚菇(盒)	藕尖	0.998367347
鲜粽叶	鲜粽叶(袋)(1)	0.998367347
海鲜菇(2)	马兰头	0.998367347
西峡香菇(份)	龙牙菜	0.997864769
姬菇(1)	海鲜菇(1)	0.991665278
奶白菜(份)	杏鲍菇(2)	0.983546618
云南油麦菜(份)	云南生菜(份)	0.983476332
云南生菜(份)	小米椒(份)	0.979708866
小皱皮(份)	菜心	-0.814290545
云南油麦菜	小米椒(份)	-0.816689741
枝江红菜苔	红薯尖	-0.821425885
云南油麦菜(份)	云南生菜	-0.823417717
云南生菜	小米椒(份)	-0.826068601
云南油麦菜	云南油麦菜(份)	-0.830500647
云南油麦菜	云南生菜(份)	-0.835410423

分析表格可以发现，红珊瑚（粗叶）与绿牛油组合等多种单品组合间的相关系数为一，即二者有强正相关关系，这可能是由于这些单品本身的销售总量极少导致数据不具有代表性，也可能是因为该商品组合构成家常菜搭配组合，顾客选择捆绑购买，从而导致二者间有较强的相关性，例如黑牛肝菌（盒）与紫尖椒组合以及黑皮鸡枞菌（盒）与灯笼辣组合均有强正相关关系，是因为辣椒炒菌类为一家常菜，顾客选择同时购买。

同时，观察表格还可发现同品类单品间存在较强的的负相关关系，例如云南油麦菜与云南生菜、青尖椒与青线椒等单品呈极强的负相关关系，是因为二者类别相似，顾客视作替代品，导致顾客在购买时往往做出二选一或者多选一决策。

七、 问题二模型的建立与求解

7.1 数据准备

首先对附件二、三、四数据进行预处理，将附件二的单品编号与附件一中的品类名称相匹配，分别筛选出六个品类的销售流水数据，并按照天数分别将各品类的销售量加总，得到每一品类销售量随天数变化的时间序列数据。并取每个品类中单品批发价的销量加权平均数作为该品类的批发价，即各品类蔬菜的进价；取每个品类中单品的平均损耗率作为该品类的损耗率

其次，通过将上述数据导入社会科学统计软件包，作出数据的 ACF 和 PACF 情况判断序列的平稳性。

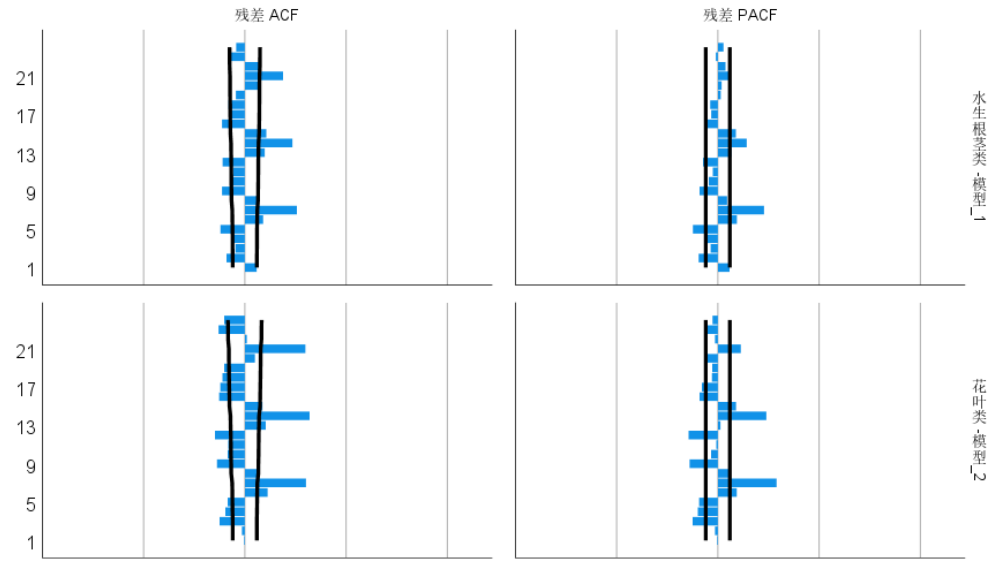


图 7 平稳化前的数据 ACF 与 PACF 图

由上图部分品类的 ACF 和 PACF 可知该数据为非平稳序列，于是考虑采用差分法使其平稳化。

7.2 成本加成定价法理论模型的建立

成本加成定价法是一种常用的定价方法。其基本思路是目标价格=单位成本×（1+加成率），加成率一般由成本利润率来确定，即：

$$定价 = 单位成本 \times (1 + 成本利润率) \tag{1}$$

$$\begin{cases} 单位成本 = （进价 \times 进货量） \div 销售量 \\ 进货量 = 销售量 + 损耗量 \\ 损耗量 = 进货量 \times 损耗率 \\ 进货量 = 销售量 \div （1 - 损耗率） \end{cases}$$

联立上述公式，得式（2）：

$$单位成本 = 进价 \div （1 - 损耗率） \tag{2}$$

$$\begin{cases} 成本利润率 = （前一天售价 - 进价） \times 销售量/(进价 \times 进货量） \times 100\% \\ 进货量 = 销售量 \div （1 - 损耗率） \end{cases}$$

联立上述公式，得式（3）：

$$成本利润率 = （前一天售价 - 进价）\times（1 - 损耗率）\div 进价 \quad （3）$$

将（2）、（3）代入（1）中，得：

$$定价 = 进价 \times 损耗率 \div （1 - 损耗率）+ 前一天售价$$

将各单品的进价、损耗率及前一天售价带入上式即可得到由成本加成定价法确定的各蔬菜单品定价结果见附件。

7.3 初步尝试--ARIMA 预测模型的建立与求解

7.3.1 ARIMA 模型的建立

ARIMA 模型又称差分自回归滑动平均求和模型，其将预测数列在时间维度上的数据序列看作随机序列，随机序列之间的关系折射了预测对象未来发展的可能性，并通过曲线拟合和参数估计建立数学模型，从而进行预测。[1]

根据图 8，通过分析 ACF 和 PACF 可知，所有滞后阶数的自相关系数和偏自相关系数均和 0 没有显著的差异，即我们认为残差就是白噪声序列。

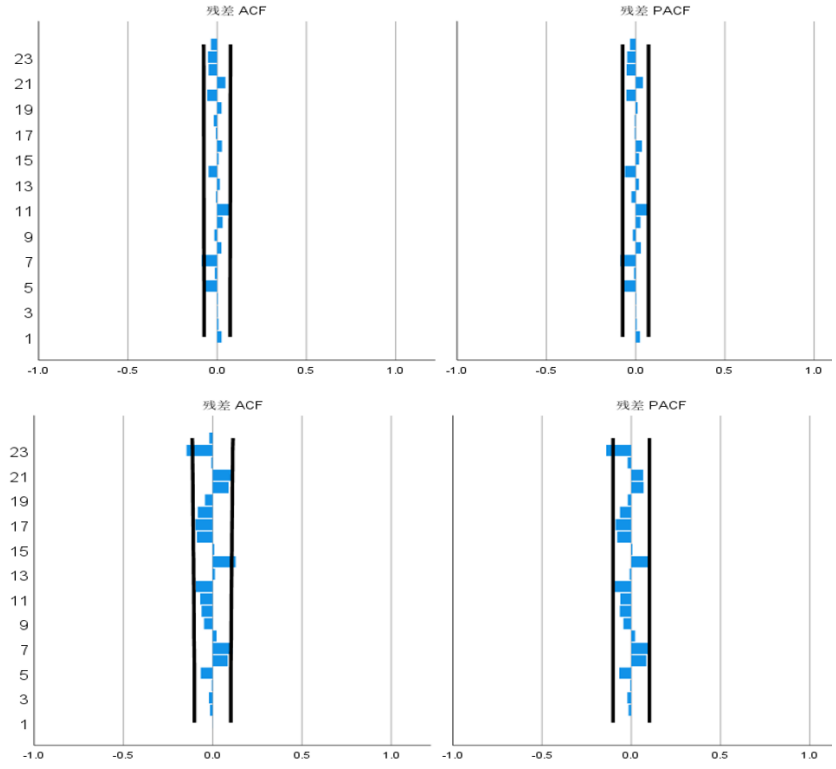


图 8 平稳后数据的 ACF 和 PACF 图

7.3.2 ARIMA 模型的求解

拟合统计结果显示，ARIMA 模型的拟合优度为R^2=0.653（这里只给出水生根茎类蔬菜的预测结果，其他结果见附件）

表 3ARIMA 模型拟合优度

拟合统计	平均值
平稳 R 方	0.512
R 方	0.653
RMSE	24.58
MAPE	30.6
MaxAPE	953.76

7.4 BP 神经网络预测模型的建立与求解

7.4.1 BP 神经网络预测模型的建立

BP 神经网络预测模型是通过最速下降法学习大量的输入-输出对应数据，并不断反向传播以调整网络的参数，不断优化自身预测效果的模型。BP 神经网络拓扑结构见图 9，包括输入层、隐层和输出层。

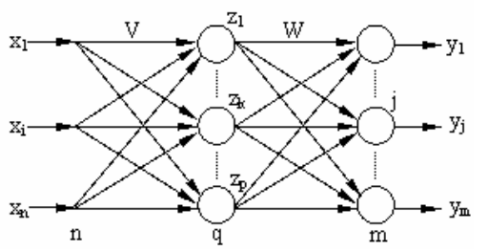


图 9 BP 神经网络拓扑结构

BP 神经网络预测步骤如下：

设 BP 网络的输入层由 m 个节点，隐层有 q 个节点，输出层有 n 个节点，输入层与隐层之间的权值为v_{kj},隐层与输出层之间的权值为w_{jk},，设隐层的传递函数为$f1$,输出层的传递函数为$f2$，则隐层点的输出为：

$$z_k = f_1(\sum_{i=0}^{n} v_{ki}x_i)$$

输出层节点的输出为：

$$y_j = f_2(\sum_{k=0}^{q} w_{jk}z_k)$$

根据以上原理，BP 神经网络的计算步骤图如下[2]：

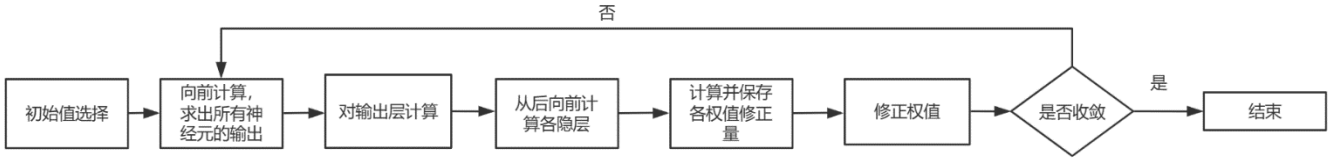


图 10 BP 神经网络计算原理

本问题以过去三年各品类每天的销售总量为学习样本，需要预测出未来七天的销售量，基于此，本文采用非自动回归模型（Nonlinear Autoregressive）进行预测。

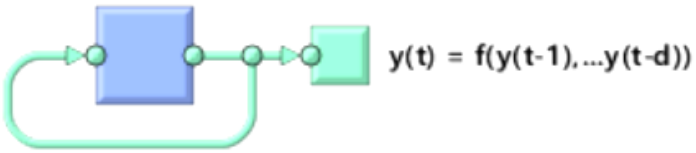


图 11 非自动回归模型图解

7.4.2 BP 神经网络预测模型的求解

由图 12 可知，该模型进行了 22 次训练，在第 15 次训练时，均方误差达到最小值，对应最佳模型。

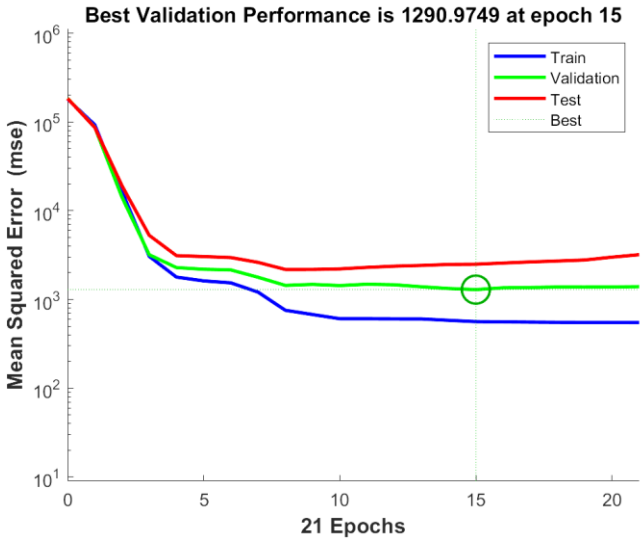


图 12 训练结果 MSE 图

图 13 呈现了该模型的拟合过程。

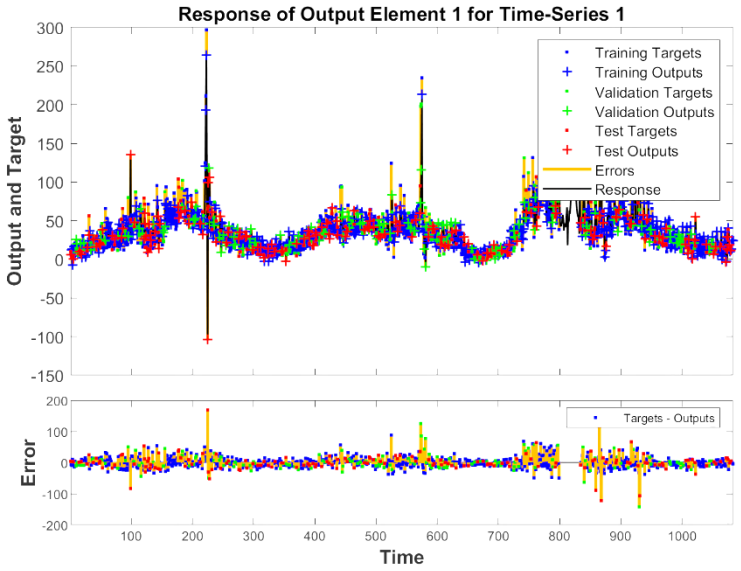


图 13 模型拟合过程

图 14 说明该模型的拟合优度R^2达到了 0.94258，说明预测效果很好，相较于先前

ARIMA 模型的拟合优度$R^2 = 0.653$，BP 神经网络的拟合优度更接近于 1，说明预测效果更好，固选择用 BP 神经网络模型进行销售量的预测，不同品类蔬菜 2023.7.1-2023.7.7 的销售量预测结果见表 4。

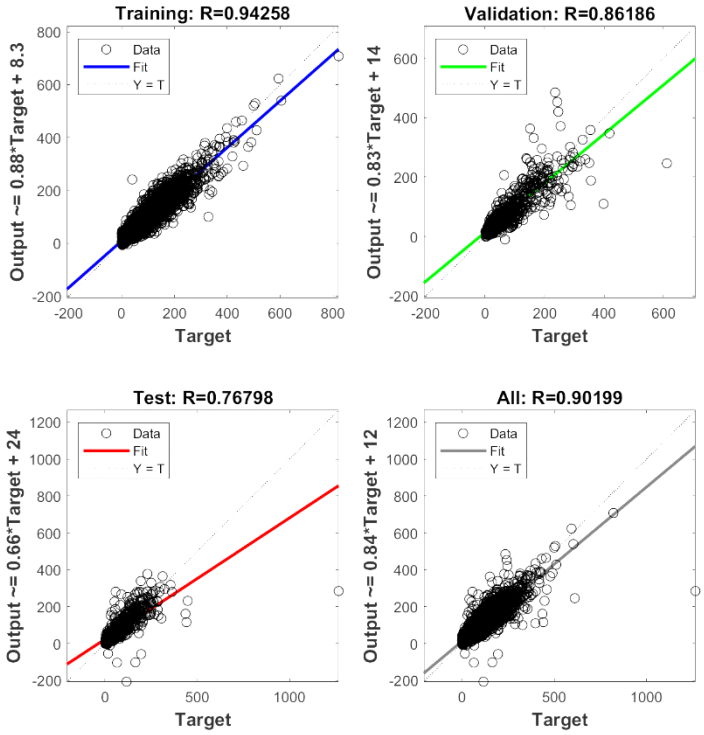


图 14 拟合效果图

表 4 未来一周的各品类销售量预测

销售日期	水生根茎类（KG）	花叶类（KG）	花菜类（KG）	茄类（KG）	辣椒类（KG）	食用菌（KG）
2023-7-1	20.445	143.174	20.392	17.394	92.699	51.593
2023-7-2	20.364	138.617	22.582	21.745	83.412	48.54
2023-7-3	19.408	145.817	20.853	19.753	83.673	49.023
2023-7-4	19.699	145.322	19.586	18.336	81.28	52.043
2023-7-5	19.898	148.593	20.991	17.924	80.91	52.485
2023-7-6	20.041	147.988	22.097	17.069	80.317	53.826
2023-7-7	20.156	149.787	23.18	19.646	80.66	51.859

7.5 线性回归模型的建立和求解

7.5.1 数据准备

将前三年各品类的批发价、损耗率等已知量代入先前建立的成本加成定价理论模型，得出前三年各品类的的定价。

7.5.2 一元线性回归模型的建立

线性回归模型通常用于度量两个变量间的线性关系，并进行预测。本文采用一元线性回归模型度量定价与各品类销售量的关系。

一元线性回归的回归方程是：

$$y = b_0 x + b_1 + \varepsilon$$

其中，b_0、b_1为回归系数、x 为自变量、y 为因变量、ε为误差项。[3]

采用最小二乘法确定回归系数，使得误差平方和即$\sum_{i=1}^{n}\ (y_i - b_0 - b_1 x_i)^2$最小，采用对$x_i$求偏导的方式进行求解，可得回归系数:

$$b_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}$$

$$b_0 = \bar{y} - \beta_1 \bar{x}$$

该模型的检验统计量为 F 统计量：

$$F = \frac{\frac{SSR}{1}}{\frac{SSE}{N-2}} = \frac{\frac{\sum_{i=1}^{n}\ (\hat{y} - \bar{y})^2}{1}}{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{N-2}}$$

$F_\alpha(1, n-2)$ 为临界值，当 F>F_α时，拟合结果显著，回归方程拟合较好。

7.5.3 一元线性回归模型的求解及预测

本文选取过去三年每天各品类的定价与销售量数据，运用 Stata 软件堆区进行一元线性回归分析，得到回归结果如下表 5：

表 5 线性回归方程

品类	回归方程
花菜类	y=-0.0339x+11.44564
花叶类	y=-0.00476x+6.973292
辣椒类	y=-0.01291x+10.21008
茄类	y=-0.03783x+10.00518
食用菌类	y=-0.01429x+10.16748
水生根茎类	y=-0.05156x+12.55402

接着，对其进行显著性检验与异方差检验，结果如下表 6：

表 6 显著性检验与异方差检验结果

品类	P>F_α	Prob>chi2
花菜类	0.000	0.0623
花叶类	0.000	0.9801
辣椒类	0.000	0.0798
茄类	0.000	0.0827
食用菌类	0.000	0.7750
水生根茎类	0.000	0.0946

结果表明，所有品类的回归系数均显著异于 0，且 BP 检验的 P 值均大于 0.05，拒绝原假设，即认为扰动项不存在异方差。

综上所述，一元线性回归回归方程显著，各品类定价与销售量之间存在明显的线性关系，因此该结果可用于后续预测。

7.6 未来七天各品类补货量及定价策略结果

根据理论推导，$补货量=\frac{销售量}{1-损耗率}$，将表 4 销售量预测数据代入可得补货量，同时，将其按品类分别代入各品类定价与销售量的回归方程中，可得各品类未来的定价策略，结果如下表 7：

表 7 花菜类蔬菜的补货量与定价策略结果

日期	补货量（KG）	定价(元/KG)
2023-7-1	23.75	10.7543512
2023-7-2	26.3	10.6801102
2023-7-3	24.29	10.7387233
2023-7-4	22.81	10.7816746
2023-7-5	24.45	10.7340451
2023-7-6	25.74	10.6965517
2023-7-7	27	10.659838

表 8 花叶类蔬菜的补货量与定价策略结果

日期	补货量（KG）	定价(元/KG)
2023-7-1	159.58	6.29178376
2023-7-2	154.5	6.31347508
2023-7-3	162.53	6.27920308
2023-7-4	161.97	6.28155928
2023-7-5	165.62	6.26598932
2023-7-6	164.94	6.26886912
2023-7-7	166.95	6.26030588

表 9 辣椒类蔬菜的补货量与定价策略结果

日期	补货量（KG）	定价(元/KG)
2023-7-1	18.73	9.34716498
2023-7-2	23.41	9.18256665
2023-7-3	21.27	9.25792401
2023-7-4	19.74	9.31152912
2023-7-5	19.3	9.32711508
2023-7-6	18.38	9.35945973
2023-7-7	21.15	9.26197182

表 10 茄类蔬菜的补货量与定价策略结果

日期	补货量（KG）	定价(元/KG)
2023-7-1	56.16	9.43021603
2023-7-2	52.84	9.4738434
2023-7-3	53.36	9.46694133
2023-7-4	56.65	9.42378553

2023-7-5	57.13	9.41746935
2023-7-6	58.59	9.39830646
2023-7-7	56.45	9.42641489

表 11 食用菌类蔬菜的补货量与定价策略结果

日期	补货量（KG）	定价(元/KG)
2023-7-1	23.23	11.4998758
2023-7-2	23.13	11.50405216
2023-7-3	22.05	11.55334352
2023-7-4	22.38	11.53833956
2023-7-5	22.6	11.52807912
2023-7-6	22.77	11.52070604
2023-7-7	22.9	11.51477664

表 12 水生根茎类蔬菜的补货量与定价策略结果

日期	补货量（KG）	定价(元/KG)
2023-7-1	23.23	11.4998758
2023-7-2	23.13	11.50405216
2023-7-3	22.05	11.55334352
2023-7-4	22.38	11.53833956
2023-7-5	22.6	11.52807912
2023-7-6	22.77	11.52070604
2023-7-7	22.9	11.51477664

八、 问题三模型的建立与求解

8.1 非线性目标规划模型

8.1.1 模型的建立

本问对可售单品数及陈列量提出约束，并要求在尽可能满足市场需求的条件下，制定合理的单品补货和定价策略，使得超市利润最大化。根据上述题目条件，考虑建立目标规划模型求解。

（1）目标函数商超总利润 L 的确定

以商超总利润 L 作为目标函数来对商超 7 月 1 日的单品补货量和定价进行决策。商超的利润主要与蔬菜的销量、批发价格、售价等有关。商超的总利润 L 表示如下：

$$L = \sum (w_i - j_i) y_i (1 - e_i)$$

上式中下标 i 均代表第 i 个蔬菜单品。其中，$y_i(1 - e_i)$）表示第 i 个产品的进货量与损耗量之差，即为其销量。

（2）约束条件的确定

根据题目背景、题目要求及假设，主要有以下约束条件：

①约束条件一：可售单品种数应介于 27 至 33 之间

$$27 \le \sum x_i \le 33 \quad x_i \in \{0，1\}$$

x_i可取 0 或 1，x_i取 0 是表示 7 月 1 日不售卖第 i 种单品，反之，x_i取 1 时，表示 7 月 1 日售卖第 i 种单品。x 之和为总售卖单品种数目，不少于 27，不多于 33。

②约束条件二：各单品最小陈列量不能少于 2.5 千克

由于陈列量为销售量与当日未售出量以致损耗量之和，故陈列量大于等于销售量，若销售量大于等于 2.5 千克，陈列量必然大于等于 2.5 千克。又由销售量为进货量减去损耗量，所以，只需使 7 月 1 日售卖的各单品进货量与损耗量之差大于等于 2.5 千克即可。列式如下：

$$y_i(1-e_i) \ge 2.5x_i$$

e_i代表第 i 种单品的损耗率，y_i代表第 i 种单品的进货量，$y_i(1-e_i)$表示第 i 种商品的销售量。因只需使参与售卖的单品满足陈列量要求，故将上限设置为 $2.5x_i$。

③约束条件三：尽可能满足对各类蔬菜的需求

$$\sum x_i > 1 \qquad i \in \mathrm{J}$$

J 表示第 J 类蔬菜品种，7 月 1 日进货的蔬菜单品属于每类品种的单品都要有。

约束条件三：减少滞销数量，规定进货量上限

$$\sum y_i > D_J/(1-e_J) \qquad i \in \mathrm{J}$$

D_J表示第 j 类蔬菜的总需求量，$D_J/(1-e_J)$表示第 J 类蔬菜品种的预计销售量，e_J为第 J 类蔬菜的平均损耗率，由各类蔬菜品类中所含所有蔬菜单品的损耗率求平均得出。将预测得到的各个品类的需求量加上预计损耗量作为进货量的上限，以保证进货量能够基本售出，减少因滞销所导致的损失。

④约束条件四：设置定价上下限

$$\min w_{it} \le w_i \le \max w_{it}$$

w_{it}表示第 i 个单品在第 t 天的售价。每个蔬菜单品的定价介于其三年来的最低售价与最高售价之间，以免为追求利润而制定过高售价，而根据销量与利润的负相关关系，可能会导致利润不增反减，导致决策失误。

⑤约束条件五：定价、进价、进货量、需求量大于 0

$$w_i > 0；\ w_{it} > 0；\ \mathrm{j_i} > 0；\ y_i > 0；\ D_J > 0$$

⑥约束条件六：定价高于进价

$$w_i > \mathrm{j_i}$$

商品售价与进价之间的价差是商品的利润来源。

将目标函数及约束条件整理如下：

$$\max L = \max \sum (\mathrm{w_i} - \mathrm{j_i})\mathrm{y_i}(1 - \mathrm{e_i})$$

$$s.t. \begin{cases} 27 \le \sum x_i \le 33 \\ x_i \in \{0，1\} \\ y_i(1 - e_i) \ge 2.5x_i \\ \sum x_i > 1 \quad i \in \mathrm{J} \\ \sum y_i > D_J/(1 - e_i) \quad i \in J \\ \min w_{it} \le w_i \le \max w_{it} \\ w_i > 0； \ w_{it} > 0； \ \mathrm{j_i} > 0； \ y_i > 0； \ D_J > 0 \\ w_i > \mathrm{j_i} \end{cases}$$

8.1.2 模型的求解

（1）数据准备

首先，需要将附件三中的 2023 年 6 月 24 日至 2023 年 6 月 30 日的数据筛选出来，以确定 7 月 1 日可选择的销售品种。其次，根据问题二的预测结果，整理出 7 月 1 日各类蔬菜的需求量。

（2）运用模拟退火算法求解

模拟退火算法实质上可以大致分为两层循环：循环一，反复迭代产生新解，在降温的过程中的任一温度下通过随机扰动新解，并计算目标函数值的变化，决定是否被接受。循环二，缓慢降温重复迭代过程，在固定温度迭代完成后缓慢的降温算法最终可能收敛到全局最优解[4]，

模拟退火算法求解过程如图：

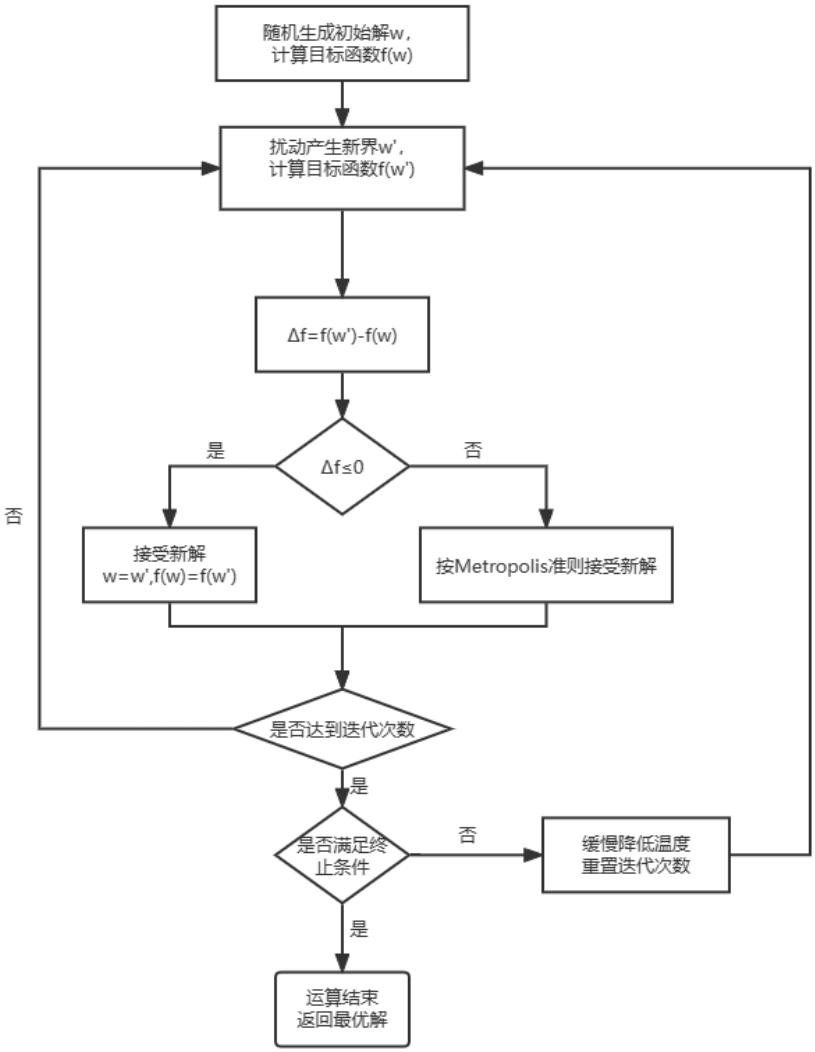


图 15 模拟退火算法图解

（3）结果

计算结果如下表所示：

表 13　7.1 日售卖单品及其补货量与定价策略

单品名称	定价（元/KG）	补货量（KG）	利润
云南生菜(份)	5.22	2.760295904	3.085782541
云南油麦菜(份)	4.57	2.760295904	3.550799721
小米椒(份)	5.95	2.760295904	9.028754728
西兰花	12.00	20.9897041	70.65234522
芜湖青椒(1)	5.42	2.651113468	3.889340003
金针菇(盒)	1.96	2.511300854	2.229605373
竹叶菜	3.81	2.89418847	6.261451259
紫茄子(2)	6.23	2.661556478	6.303886143
螺丝椒(份)	5.76	2.760295904	3.79458078
娃娃菜	6.77	2.563576702	5.374056874

小皱皮(份)	2.81	2.760295904	1.864801519
双孢菇(盒)	5.01	2.50501002	3.985729113
菠菜(份)	5.93	2.760295904	3.223599533
姜蒜小米椒组合装(小份)	4.87	2.760295904	5.656785185
苋菜	4.11	3.068237604	3.269294671
螺丝椒	12.95	2.783344467	7.675230138
海鲜菇(包)	2.62	2.5	1.6751475
奶白菜	4.44	2.964895636	4.208172661
净藕(1)	14.86	13.63899918	48.7269959
西峡花菇(1)	25.89	45.88339322	343.795079
枝江青梗散花	7.07	2.760295904	1.401371452
木耳菜	5.95	2.705920554	6.365130625
上海青	8.69	131.5844392	440.6960697
红薯尖	5.39	2.72985368	5.90276795
长线茄	12.52	13.43658749	62.51033625
小青菜(1)	5.51	2.788000446	6.353264749
洪湖藕带	25.45	3.291639236	4.370743746
青红杭椒组合装(份)	5.92	2.760295904	5.713415174
高瓜(1)	18.33	3.533568905	4.596245477
菱角	13.25	2.765792676	8.046720126
虫草花(份)	3.83	2.760295904	2.305241607
红椒(2)	20.03	82.09406255	443.0672001
青茄子(1)	4.31	2.631855985	2.714458973

九、 问题四的研究

超市经营者一方面想通过低价吸引更多的客户取得更高的销量，另一方面又要保证销售利润，因此，超市商品的合理定价起到了至关重要的作用。同时，蔬菜滞销易导致蔬菜腐烂变质，补货量不宜过多；但面对多余菜品储备量的客户需求时，没有充足的储备又会损失部分利润。综上，合理的蔬菜商品补货和定价决策是超市盈利的关键。我们通过查阅资料和研究发现，采集以下数据有利于补货和定价策略的合理制定。

9.1 菜品间组合销售的数据

收集不同菜品组合销售的数据，可用于确定未来销售的蔬菜单品，在进行补货决策时，对于待选择的几种利润相似的菜品，可以选择与已定补货菜品组合销售情况更好的菜品，使销售量更有保障。而在定价决策方面，组合销售的数据可用于关联销售定价。周上（2013）提出，对于相互关联紧密的商品组合，可以建立考虑关联销售的定价模型，由遗传算法得出商品组合中每件商品的最优价格，超市经营者就可以调整这些商品组合的价格，获取最佳的销售额与利润。[5]不同蔬菜单品的价格敏感度数据

价格敏感度是一个经济学名词，表示由于价格的变动引起的对产品需求的变化。了解消费者对不同蔬菜单品的价格敏感度，能够使在营销活动中掌握更多的主动权，也更具有实际意义。在定价决策方面，对于价格敏感度高的菜品，可以通过调整价格以适应库存量，或实施降价促销；而对于价格敏感度不高的菜品，调整价格对其销量影响不大，故无需降价。王艳（2014）的研究指出[6]，超市在对商品实行降价促销时要根据历史数据分析确定该商品的销售价格弹性，只有弹性大于 1 的商品降价或定价比竞争对手低才有意义。在补货决策方面，价格敏感度数据可以帮助预测价格调整后的需求情况，从而协助制定补货量决策，也可以根据数据，结合营销目的，有意识地选择敏感度高或低的蔬菜单品

9.2 各蔬菜单品随陈列时间变长的销量变化及价值损耗数据

蔬菜容易腐烂，在销售期结束后其残值直接归零。即便是在可供销售的有限时间内，由于蔬菜保鲜期短，品相逐渐变差，所以顾客对于蔬菜的价格期望也是时刻都在变化的，因而愿意为该类商品支付的货币也是动态变化的。随着陈列时间变长各蔬菜单品的销量数据可以反映销量受品相、新鲜度大小的影响程度，从而反映消费者对蔬菜的价格期望的变化。在定价决策时，可以采用动态定价法，通过定时打折等方式在蔬菜新鲜度变差后进行减价处理，以保证蔬菜的销售量。[7]卢亚杰对生鲜蔬菜的新鲜度变化引起的价值损耗进行定量分析，建立基于价值损耗的超市优质生鲜蔬菜的定价模型。在订货决策方面，可以尽量少地选择销量受新鲜度影响大的蔬菜，以减少损耗。陈军和但斌等（2009）将新鲜度函数引入基本变质库存模型中，并以控制损耗为目的，研究了弹性需求下的时鲜产品订货策略[8]

十、 模型的评价

10.1 模型的优点

1.本文从供给侧与需求侧出发，利用现有数据以及成本加成定价法原理，推导出商超的定价公式，充分考虑了现实情况，解决了实际问题。

2.本文将 BP 神经网络预测模型及线性回归模型结合使用，模型新颖，考虑全面。

3.针对相关关系、销量预测等问题，本文尝试了多个模型，并结合题干数据特征、各模型适用条件，对比各模型计算效果，最终选择出最优的数学模型。

4.本文问题三结合供需原理，采用合适的数学语言转化目标函数与约束条件，考虑全面充分，从而更准确地求得最优解。

10.2 模型的缺点

1.在分析各单品间的相关关系时，未搜集到足够充分的资料更加合理地解释其相关系数。

2. 模拟退火算法可能陷入局部最优解， 可能收敛到局部最小值， 在大规模数据上收敛较慢。

十一、 参考文献

[1] 欧阳泽龙,李锦琼,汪官嬰等.基于时间序列模型的全球猴痘周新增病例预测[J/OL].中国动物传染病学报:1-9[2023-09-10].https://doi.org/10.19958/j.cnki.cn31-2031/s.20230906.004.

[2] 曾庆扬,丁楚衡,谷战英等.基于 BP 神经网络的油茶产量预测模型构建[J].经济林研究,2022,40(03):87-95.DOI:10.14067/j.cnki.1003-8981.2022.03.009.

[3] 杨宜平,赵培信,黄霞.面板数据下带固定效应的线性回归模型的稳健变量选择[J/OL].数理统计与管理:1-8[2023-09-10].https://doi.org/10.13860/j.cnki.sltj.20230626-002.

[4] 李朝迁,裴建朝.新型模拟退火遗传算法在路径优化的应用[J].组合机床与自动化加工技术,2022,No.577(03):52-55.DOI:10.13462/j.cnki.mmtamt.2022.03.013.

[5] 周上,黄章树.基于数据挖掘技术的超市商品定价研究[J].科技和产业,2013,13(02):62-64+99.

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

[7] 卢亚杰. 我国超市优质生鲜蔬菜动态定价问题研究[D].北京交通大学,2010.

[8] 但斌,陈军.基于价值损耗的时鲜产品供应链协调[J].中国管 理科学,2008,05:42-49.

附录

附录 1

支撑材料的文件列表

1. 附件.zip

（1）文件夹：问题一

main1.1_预处理及描述性分析.ipynb
main1.2_时间序列前的平稳性检验.ipynb
main1.3_正态分布检验.ipynb
main1.4_用皮尔逊相关系数求分类.ipynb
main1.5_用斯皮尔曼相关系数求单品.ipynb

（2）文件夹：问题二

第二问预处理.xlsx
main2.1_数据的预处理（2）.ipynb
main2.2_时间序列预测.pdf
main2.3_BP 神经网络预测.docx

（3）文件夹：问题三

main3.1_时间序列预测销售量.ipynb
main3.2_单品成本加成定价.ipynb
main3.3_时间序列预测成本加成定价.ipynb
main3.4_时间序列预测进价.ipynb
main3.5_算法求解.ipynb
第三问预处理. xlsx
第三问结果. xlsx

（4）文件夹：results

销售量按分类-月统计. xlsx
销售量按分类-日统计. xlsx
销售量按单品-月统计. xlsx
销售量按单品-日统计. xlsx
斯皮尔曼求得各单品相关系数. xlsx
水生根茎类求加权成本加成定价. xlsx
蔬菜品类正态分布检验结果.docx
蔬菜品类的 ADF 检验结果(按天）.docx
蔬菜各品类销售量相关系数矩阵图.docx
蔬菜单品正态分布检验结果.docx
食用菌求加权成本加成定价. xlsx
茄类求加权成本加成定价. xlsx
品类销量图.png
辣椒类求加权成本加成定价. xlsx
花叶类求加权成本加成定价. xlsx
花菜类求加权成本加成定价. xlsx

各单品求加权成本加成定价. xlsx
各单品按时间排列的批发价格. xlsx
单品销量前 10 图.png
单品 7.1 销售量预测. xlsx
单品 7.1 批发价格预测. xlsx
单品 7.1 成本加成定价预测. xlsx
49 个符合的单品.xlsx

附录 2

main1.1_预处理及描述性分析.ipynb

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
# 读取附件 1 至附件 4 的数据
# 附件 1-4 为原版赛题文件 因文件字节限制故为放至压缩包内
original_1 = pd.read_excel('附件 1.xlsx')
original_2 = pd.read_excel('附件 2.xlsx')
original_3 = pd.read_excel('附件 3.xlsx')
original_4 = pd.read_excel('附件 4.xlsx')
# 合并 data_1 和 data_2
merged_data = pd.merge(original_1, original_2, on='单品编码', how='outer')

merged_data['销售日期'] = pd.to_datetime(merged_data['销售日期'])
merged_data['销售月份'] = merged_data['销售日期'].dt.to_period('M')
# 按照单品名称进行分组，计算每个单品的销售量
sales_by_Single = merged_data.groupby('单品名称')['销量(千克)'].sum()
# 按照分类名称进行分组，计算每个单品的销售量
sales_by_Catagory = merged_data.groupby('分类名称')['销量(千克)'].sum()
# 按照时间顺序排序
merged_data.sort_values('销售日期', inplace=True)

# 按照单品名称和日期进行分组，计算销售总量
sales_by_Single_and_Date = merged_data.groupby(['单品名称', '销售日期'])['销量
(千克)'].sum()

# 重塑数据，按照时间顺序排序
sales_by_Single_and_Date = sales_by_Single_and_Date.unstack('单品名称')
sales_by_Single_and_Date.sort_index(axis=1, inplace=True)
sales_by_Single_and_Date = sales_by_Single_and_Date.fillna(0)
# 导出到 Excel
```

```
sales_by_Single_and_Date.to_excel('./results/销售量按单品-日统计.xlsx')

# 按照时间顺序排序
merged_data.sort_values('销售月份', inplace=True)

# 按照单品名称和月份进行分组，计算销售总量
sales_by_Single_and_Month = merged_data.groupby(['单品名称', '销售月份'])['销量(千克)'].sum()

# 重塑数据，按照时间顺序排序
sales_by_Single_and_Month = sales_by_Single_and_Month.unstack('单品名称')
sales_by_Single_and_Month.sort_index(axis=1, inplace=True)
sales_by_Single_and_Month = sales_by_Single_and_Month.fillna(0)

# 导出到 Excel
sales_by_Single_and_Month.to_excel('./results/销售量按单品-月统计.xlsx')
# 按照时间顺序排序
merged_data.sort_values('销售日期', inplace=True)

# 按照品类名称和日期进行分组，计算销售总量
sales_by_Catagory_and_Date = merged_data.groupby(['分类名称', '销售日期'])['销量(千克)'].sum()

# 重塑数据，按照时间顺序排序
sales_by_Catagory_and_Date = sales_by_Catagory_and_Date.unstack('分类名称')
sales_by_Catagory_and_Date.sort_index(axis=1, inplace=True)
sales_by_Catagory_and_Date = sales_by_Catagory_and_Date.fillna(0)

# 导出到 Excel
sales_by_Catagory_and_Date.to_excel('./results/销售量按分类-日统计.xlsx')
# 按照时间顺序排序
merged_data.sort_values('销售日期', inplace=True)

# 按照品类名称和月份进行分组，计算销售总量
sales_by_Catagory_and_Month = merged_data.groupby(['分类名称', '销售月份'])['销量(千克)'].sum()

# 重塑数据，按照时间顺序排序
sales_by_Catagory_and_Month = sales_by_Catagory_and_Month.unstack('分类名称')
sales_by_Catagory_and_Month.sort_index(axis=1, inplace=True)
sales_by_Catagory_and_Month = sales_by_Catagory_and_Month.fillna(0)
# 导出到 Excel
sales_by_Catagory_and_Month.to_excel('./results/销售量按分类-月统计.xlsx')
```

```
import matplotlib.pyplot as plt
import matplotlib

# 指定中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']

# 设置图形的大小为 10x6 英寸
plt.figure(figsize=(10, 6))

# 只选择销量前 10 的数据
top20_sales = sales_by_Single.nlargest(10)

# 绘制柱状图
plt.bar(top20_sales.index, top20_sales.values)

# 设置 x 轴标签的字体样式，并将标签旋转 45 度
plt.xticks(rotation=45, size=16)

# 设置 x 轴标签的文字内容以及字体样式
plt.xlabel('单品名称')

# 设置 y 轴标签的文字内容以及字体样式
plt.ylabel('销售量（千克）')

# 设置图形的标题以及字体样式
plt.title('蔬菜单品销售量分布（前 10）')

# 保存图形为 PNG 格式文件，设定 dpi 为 300 以获得更高的分辨率，并通过
bbox_inches='tight'参数确保保存整个图形
plt.savefig('./results/单品销量前 10 图.png', dpi=300, bbox_inches='tight')
# 设置图形的大小为 10x6 英寸
plt.figure(figsize=(10, 6))

# 绘制柱状图
plt.bar(sales_by_Catagory.index, sales_by_Catagory.values)

# 设置 x 轴标签的字体样式，并将标签旋转 45 度
plt.xticks(rotation=45, size=16)

# 设置 x 轴标签的文字内容以及字体样式
plt.xlabel('分类名称')

# 设置 y 轴标签的文字内容以及字体样式
plt.ylabel('销售量（千克）')
```

```
# 设置图形的标题以及字体样式
plt.title('蔬菜各品类销售量分布')

# 保存图形为 PNG 格式文件，设定 dpi 为 300 以获得更高的分辨率，并通过
bbox_inches='tight'参数确保保存整个图形
plt.savefig('./results/品类销量图.png', dpi=300, bbox_inches='tight')
```

附录 3

main1.2_时间序列前的平稳性检验.ipynb

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import scipy.stats as stats
import matplotlib.pyplot as plt
import matplotlib
import scipy
from docx import Document
#读取预处理过的数据
original_Catagory_Groupbydate = pd.read_excel('./results/销售量按分类-日统
计.xlsx')
original_Catagory = pd.read_excel('./results/销售量按分类-月统计.xlsx')
#进行品类的 ADF 检验
from arch.unitroot import ADF
column_data  = original_Catagory[['水生根茎类', '花叶类', '花菜类', '茄类', '辣椒
类', '食用菌']]
for column in column_data.columns:
    result = ADF(column_data[column])
    adf = ADF(column_data[column], trend='ct')


    # 打印出检验结果
    print(f"\n\n{column}数据的检验\n\n")
    print("不指定 trend 默认为检验是否含截距项平稳\n")
    print(result.summary().as_text())
#进行单品的 ADF 检验
column_data  = original_Catagory_Groupbydate[['水生根茎类', '花叶类', '花菜类',
```

```
'茄类', '辣椒类', '食用菌']]
for column in column_data.columns:
    result = ADF(column_data[column])
    adf = ADF(column_data[column], trend='ct')

        # 打印出检验结果
    print(f"\n\n{column}数据的检验\n\n")
    print("不指定 trend 默认为检验是否含截距项平稳\n")
    print(result.summary().as_text())

    # print("\n\n 指定 trend 为检验是否含截距项和时间趋势项平稳\n")
    # print(adf.summary().as_text())
```

附录 3

main1.3_正太分布检验.ipynb

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import scipy.stats as stats
import matplotlib.pyplot as plt
import matplotlib
import scipy
from docx import Document
#读取预处理过的数据
original_Single = pd.read_excel('./results/销售量按单品-月统计.xlsx')
original_Catagory = pd.read_excel('./results/销售量按分类-月统计.xlsx')
# 进行分类的正态分布检验

# 提取需要检验的列数据
column_data  = original_Catagory[['水生根茎类', '花叶类', '花菜类', '茄类', '辣椒
类', '食用菌']]
alpha = 0.05
n_c = column_data.shape[1]
H = np.zeros(n_c)
P = np.zeros(n_c)
# 进行蔬菜单品的正态分布检验

# 提取需要检验的列数据
```

```
column_data  = original_Single[['七彩椒(1)', '七彩椒(2)', '七彩椒(份)', '上海青', '
上海青(份)', '东门口小白菜', '丝瓜尖', '云南油麦菜', '云南油麦菜(份)', '云南生菜
', '云南生菜(份)', '余干椒', '保康高山大白菜', '冰草', '冰草(盒)', '净藕(1)', '净藕(2)',
'净藕(3)', '南瓜尖', '双孢菇', '双孢菇(份)', '双孢菇(盒)', '双沟白菜', '和丰阳光海鲜
菇(包)', '四川红香椿', '圆茄子(1)', '圆茄子(2)', '外地茼蒿', '外地茼蒿(份)', '大白菜
', '大白菜秧', '大芥兰', '大龙茄子', '奶白菜', '奶白菜(份)', '奶白菜苗', '姜蒜小米椒
组合装(小份)', '姬菇(1)', '姬菇(2)', '姬菇(份)', '姬菇(包)', '娃娃菜', '小白菜', '小白
菜(份)', '小皱皮', '小皱皮(份)', '小米椒', '小米椒(份)', '小青菜(1)', '小青菜(2)', '小
青菜(份)', '平菇', '快菜', '春菜', '木耳菜', '木耳菜(份)', '本地上海青', '本地小毛白
菜', '本地黄心油菜', '杏鲍菇(1)', '杏鲍菇(2)', '杏鲍菇(250 克)', '杏鲍菇(份)', '杏鲍
菇(袋)', '枝江红菜苔', '枝江红菜苔(份)', '枝江青梗散花', '槐花', '水果辣椒', '水果
辣椒(份)', '水果辣椒(橙色)', '油菜苔', '泡泡椒(精品)', '洪山菜苔', '洪山菜薹珍品
手提袋', '洪山菜薹莲藕拼装礼盒', '洪湖莲藕(粉藕)', '洪湖莲藕(脆藕)', '洪湖藕带
', '活体银耳', '海鲜菇(1)', '海鲜菇(2)', '海鲜菇(份)', '海鲜菇(包)', '海鲜菇(袋)(1)', '
海鲜菇(袋)(2)', '海鲜菇(袋)(3)', '海鲜菇(袋)(4)', '灯笼椒(1)', '灯笼椒(2)', '灯笼椒
(份)', '牛排菇', '牛排菇(盒)', '牛首油菜', '牛首生菜', '猪肚菇(盒)', '猴头菇', '甘蓝叶
', '甜白菜', '田七', '白玉菇(1)', '白玉菇(2)', '白玉菇(盒)', '白玉菇(袋)', '白菜苔', '白
蒿', '秀珍菇', '竹叶菜', '竹叶菜(份)', '紫圆茄', '紫尖椒', '紫白菜(1)', '紫白菜(2)', '紫
苏', '紫苏(份)', '紫茄子(1)', '紫茄子(2)', '紫螺丝椒', '紫贝菜', '红尖椒', '红尖椒(份)',
'红杭椒', '红杭椒(份)', '红椒(1)', '红椒(2)', '红椒(份)', '红橡叶', '红灯笼椒(1)', '红灯
笼椒(2)', '红灯笼椒(份)', '红珊瑚(粗叶)', '红线椒', '红莲藕带', '红薯尖', '红薯尖
(份)', '组合椒系列', '绣球菌', '绣球菌(袋)', '绿牛油', '艾蒿', '芜湖青椒(1)', '芝麻苋
菜', '芥兰', '芥菜', '花茄子', '花菇(一人份)', '苋菜', '苋菜(份)', '茶树菇(袋)', '茼蒿', '
茼蒿(份)', '荠菜', '荸荠', '荸荠(份)', '莲蓬(个)', '菊花油菜', '菌菇火锅套餐(份)', '菌
蔬四宝(份)', '菜心', '菜心(份)', '菠菜', '菠菜(份)', '菱角', '萝卜叶', '蒲公英', '蔡甸藜
蒿', '蔡甸藜蒿(份)', '薄荷叶', '藕尖', '虫草花', '虫草花(份)', '虫草花(盒)(2)', '虫草
花(袋)', '螺丝椒', '螺丝椒(份)', '蟹味菇(1)', '蟹味菇(2)', '蟹味菇(盒)', '蟹味菇(袋)', '
蟹味菇与白玉菇双拼(盒)', '襄甜红菜苔(袋)', '西兰花', '西峡花菇(1)', '西峡花菇
(2)', '西峡香菇(1)', '西峡香菇(2)', '西峡香菇(份) ', '豌豆尖', '赤松茸', '赤松茸(盒)',
'辣妹子', '野生粉藕', '野藕(1)', '野藕(2)', '金针菇(1)', '金针菇(2)', '金针菇(份)', '金
针菇(盒)', '金针菇(袋)(1)', '金针菇(袋)(2)', '金针菇(袋)(3)', '银耳(朵)', '长线茄', '随
州泡泡青', '青尖椒', '青尖椒(份)', '青杭椒(1)', '青杭椒(2)', '青杭椒(份)', '青梗散花
', '青红尖椒组合装(份)', '青红杭椒组合装(份)', '青线椒', '青线椒(份)', '青茄子(1)',
'青茄子(2)', '青菜苔', '面条菜', '马兰头', '马齿苋', '高瓜(1)', '高瓜(2)', '鱼腥草', '鱼
腥草(份)', '鲜木耳(1)', '鲜木耳(2)', '鲜木耳(份)', '鲜粽叶', '鲜粽叶(袋)(1)', '鲜粽叶
(袋)(2)', '鲜粽叶(袋)(3)', '鲜粽子叶', '鲜藕带(袋)', '鸡枞菌', '鹿茸菇(盒)', '黄心菜
(1)', '黄心菜(2)', '黄白菜(1)', '黄白菜(2)', '黄花菜', '黑油菜', '黑牛肝菌', '黑牛肝菌
(盒)', '黑皮鸡枞菌', '黑皮鸡枞菌(盒)', '龙牙菜']]
alpha = 0.05
n_c = column_data.shape[1]
H = np.zeros(n_c)
P = np.zeros(n_c)

# 指定中文字体
```

```
matplotlib.rcParams['font.sans-serif'] = ['SimHei']

# 遍历每一列的检验结果，并添加到文档中
for i in range(column_data.shape[1]):
    h, p = stats.jarque_bera(column_data.iloc[:, i])
    H[i] = h
    P[i] = p

    if p < alpha:
        result = "不满足正态分布"
    else:
        result = "满足正态分布"

    # 添加每一列的结果到文档中
    doc.add_paragraph(f'第{i+1}列数据：{result}')

# 添加 JB 检验结果
doc.add_heading('JB 检验结果', level=2)
doc.add_paragraph(f'h: {H}')
doc.add_paragraph(f'p: {P}')

# 保存文档
doc.save('./results/蔬菜单品正态分布检验结果.docx')

for i in range(column_data.shape[1]):
    h, p = stats.jarque_bera(column_data.iloc[:, i])
    H[i] = h
    P[i] = p

    if p < alpha:
        print("第", i+1, "列数据不满足正态分布")
    else:
        print("第", i+1, "列数据满足正态分布")

print("JB 检验结果(h):", H)
print("JB 检验结果(p):", P)
```

附录 5

main1.4_用皮尔逊相关系数求分类.ipynb

```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import scipy.stats as stats
import matplotlib.pyplot as plt
import matplotlib
import scipy
#读取预处理过的数据
original_Single = pd.read_excel('./results/销售量按单品-月统计.xlsx')
original_Catagory = pd.read_excel('./results/销售量按分类-月统计.xlsx')
# 使用系统默认字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
#处理负号不显示的问题
plt.rcParams['axes.unicode_minus'] = False
original_Catagory['销售月份'] = pd.to_datetime(original_Catagory['销售月份'])
original_Catagory_1 = original_Catagory.drop(original_Catagory.columns[0],
axis=1)

corr_matrix = original_Catagory_1.corr()

# 相关系数矩阵的绘制
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', ax=ax)
ax.set_title('蔬菜各品类销售量相关系数矩阵')

for tick in ax.get_xticklabels():
    tick.set_rotation(45)
for tick in ax.get_yticklabels():
    tick.set_rotation(0)

plt.xlabel('分类名称')
plt.ylabel('分类名称')
plt.savefig('./results/蔬菜各品类销售量相关系数矩阵图.png', dpi=300,
bbox_inches='tight')
plt.show()
corr_matrix = original_Single.corr()

# 可视化相关系数矩阵
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', ax=ax)
ax.set_title('蔬菜各单品销售量相关系数矩阵')
```

```
for tick in ax.get_xticklabels():
    tick.set_rotation(45)
for tick in ax.get_yticklabels():
    tick.set_rotation(0)

plt.xlabel('单品名称')
plt.ylabel('单品名称')
plt.savefig('./results/蔬菜各单品销售量相关系数矩阵图.png', dpi=300, bbox_inches='tight')
plt.show()
```

附录 6

main1.5_用斯皮尔曼相关系数求单品.ipynb

```
#导入所需软件包
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import scipy.stats as stats
import matplotlib.pyplot as plt
import matplotlib
import scipy
#读取预处理过的数据
original_Single = pd.read_excel('./results/销售量按单品-月统计.xlsx')
original_Catagory = pd.read_excel('./results/销售量按分类-月统计.xlsx')
# 选择需要计算相关系数的列
columns = original_Single.columns[1:253]
# 计算相关系数
corr_matrix = original_Single[columns].corr(method='spearman')
# 将相关系数矩阵转换为上三角形矩阵
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
upper_triangle = corr_matrix.mask(mask)
# 将上三角形矩阵展平，并按照相关系数大小进行排序
sorted_corr = upper_triangle.unstack().sort_values(ascending=False)

# 创建一个 DataFrame 对象，包含相关系数和对应的指标
df = pd.DataFrame({'Index 1': sorted_corr.index.get_level_values(0),
                   'Index 2': sorted_corr.index.get_level_values(1),
                   'Correlation': sorted_corr.values})

# 删除包含 NaN 值的行
```

```
df = df.dropna()

# 导出 DataFrame 对象为 Excel 文件
df.to_excel('./results/斯皮尔曼求得各单品相关系数.xlsx', index=False)
# 绘制相关系数矩阵的热力图
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)
plt.title('Correlation Matrix')
plt.show()
```

附录 7

main2.1_数据的预处理(2).ipynb

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import datetime
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
# 读取预处理过的数据
original = pd.read_excel('./第二问预处理.xlsx')
original
original.columns
# 删除指定列
original.drop(columns=['扫码销售时间', '辅助列', '销售单价(元/千克)', '销售类型',
'是否打折销售', '分类编码', '批发价格(元/千克)', '损耗率(%)'], inplace=True)

# 按照时间顺序排序
original.sort_values('销售日期', inplace=True)

# 按照分类名称和日期进行分组，计算销售总量
grouped = original.groupby('分类名称')

# 遍历每个分组并保留其他列的数据
for name, group in grouped:
# 打印每个分组的数据
    print(f"分类: {name}")
    print(group)
```

```
# 保存蔬菜品类分组结果
    group.to_excel(f"./results/蔬菜品类/{name}.xlsx", index=False)
# 读取预处理过的数据
huacai = pd.read_excel('./results/蔬菜品类/花菜类.xlsx')
huaye = pd.read_excel('./results/蔬菜品类/花叶类.xlsx')
qielei = pd.read_excel('./results/蔬菜品类/茄类.xlsx')
lajiaolei = pd.read_excel('./results/蔬菜品类/辣椒类.xlsx')
shiyongjun= pd.read_excel('./results/蔬菜品类/食用菌.xlsx')
shuishenggenjing = pd.read_excel('./results/蔬菜品类/水生根茎类.xlsx')
import pandas as pd

def calculate_daily_sales(dataframe, date):
    # 将日期列转换为日期类型
    dataframe['销售日期'] = pd.to_datetime(dataframe['销售日期'])

    # 根据日期分组并计算总销量
    daily_sales = dataframe[dataframe['销售日期'] == date]['销量(千克)'].sum()

    # 返回当天的总销量
    return daily_sales

# 调用函数计算当天的总销量 检验花菜类2020年7月1日的销量是否为46.64
daily_sales = calculate_daily_sales(huacai, '2020-7-1')
print(daily_sales)
#TRUE
# 水生根茎类
# 将日期列转换为日期类型
shuishenggenjing['销售日期'] = pd.to_datetime(shuishenggenjing['销售日期'])

# 按照时间顺序排序
shuishenggenjing.sort_values('销售日期', inplace=True)

# 分别求每天的成本加成定价-第一步 加总
shuishenggenjing['成本加成定价*单次销量'] = shuishenggenjing['成本加成定价(元/千克)'] * shuishenggenjing['销量(千克)']

# 按照分类名称进行分组，计算每个单品的销售量
shuishenggenjing = shuishenggenjing.groupby('销售日期').agg({'销量(千克)': 'sum', '成本加成定价*单次销量': 'sum'})

# 分别求每天的成本加成定价-第二步 除以当天该品种总销量
shuishenggenjing['成本加成定价'] = shuishenggenjing['成本加成定价*单次销量'] / shuishenggenjing['销量(千克)']
```

```
# 删除多余项
shuishenggenjing.drop(columns=['成本加成定价*单次销量'], inplace=True)

# 导出到 Excel
shuishenggenjing.to_excel('./results/问题二/水生根茎类求加权成本加成定价.xlsx')
```

附录 8

main2.2_时间序列预测.pdf

```
GET DATA
  /TYPE=XLSX
  /FILE='C:\Users\86180\Desktop\ri danpin.xlsx'
  /SHEET=name 'Sheet1'
  /CELLRANGE=FULL
  /READNAMES=ON
  /DATATYPEMIN PERCENTAGE=95.0
  /HIDDEN IGNORE=YES.
EXECUTE.
DATASET NAME 数据集 1 WINDOW=FRONT.
DATE D 1.

The following new variables are being created:

  Name            Label

  DAY_            DAY, not periodic
  DATE_           Date.  Format:  "DDDD"
PREDICT THRU DAY 1092.
* 时间序列建模器.
TSMODEL
   /MODELSUMMARY  PRINT=[MODELFIT]
   /MODELSTATISTICS  DISPLAY=YES MODELFIT=[ SRSQUARE]
   /MODELDETAILS  PRINT=[ PARAMETERS]  PLOT=[ RESIDACF
RESIDPACF]
   /SERIESPLOT OBSERVED FORECAST
   /OUTPUTFILTER DISPLAY=ALLMODELS
   /SAVE  PREDICTED(预测)
   /AUXILIARY  CILEVEL=95 MAXACFLAGS=24
   /MISSING USERMISSING=EXCLUDE
   /MODEL DEPENDENT=水生根茎类 花叶类 花菜类 茄类 辣椒类 食用菌
INDEPENDENT=DAY_
      PREFIX='模型'
```

```
    /EXPERTMODELER TYPE=[ARIMA EXSMOOTH]
    /AUTOOUTLIER DETECT=ON TYPE=[ ADDITIVE LEVELSHIFT
INNOVATIONAL TRANSIENT LOCALTREND
     ADDITIVEPATCH].
```

附录 9

main2.3_BP 神经网络预测.pdf

```
function [Y,Xf,Af] = myNeuralNetworkFunction(X,Xi,~)
%MYNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 09-Sep-2023
23:06:38.
%
% [Y,Xf,Af] = myNeuralNetworkFunction(X,Xi,~) takes these arguments:
%
%   X = 1xTS cell, 1 inputs over TS timesteps
%   Each X{1,ts} = 6xQ matrix, input #1 at timestep ts.
%
%   Xi = 1x2 cell 1, initial 2 input delay states.
%   Each Xi{1,ts} = 6xQ matrix, initial states for input #1.
%
%   Ai = 2x0 cell 2, initial 2 layer delay states.
%   Each Ai{1,ts} = 10xQ matrix, initial states for layer #1.
%   Each Ai{2,ts} = 6xQ matrix, initial states for layer #2.
%
% and returns:
%   Y = 1xTS cell of 1 outputs over TS timesteps.
%   Each Y{1,ts} = 6xQ matrix, output #1 at timestep ts.
%
%   Xf = 1x2 cell 1, final 2 input delay states.
%   Each Xf{1,ts} = 6xQ matrix, final states for input #1.
%
%   Af = 2x0 cell 2, final 0 layer delay states.
%   Each Af{1ts} = 10xQ matrix, final states for layer #1.
%   Each Af{2ts} = 6xQ matrix, final states for layer #2.
%
% where Q is number of samples (or series) and TS is the number of timesteps.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====
```

```
% Input 1
x1_step1.xoffset = [0.926;31.298;0.632;0.252;6.066;3.012];
x1_step1.gain =
[0.00675981694415715;0.0016205157291308;0.01078033451378;0.016852181093
538;0.0033435590514323;0.0039360471066117 7];
x1_step1.ymin = -1;

% Layer 1
b1 = [1.488230975291968905;-1.7010329570372146879;-
1.7938416534616841158;0.37163788820108512034;-
0.30144517727043834698;0.26527669923702945631;0.40425733743891645311;-
3.9730093366299961133;2.3327046783250788842;0.58580362266937435756];
IW1_1 = [0.28542414215341460615 2.4874620116310497409
0.6023606048580868455 -0.0028533351956805952376 -0.76310349654968478994
-0.88716289381410096127 -0.46814314084097996549 -2.3019714467419460924
0.76823693769103140738 0.59526270454856300418 1.2625088431434590142
0.80173218928845635389;0.5660142640391792046 -3.2137563113730465147
0.13259862954877382757 1.9340149305468243845 -1.002384614529113982
2.9316680257116844999 1.2898653236445651871 -0.38541810736968817874 -
1.5136169408722917051 0.43828649654430740235 -1.5388196296996521362 -
1.4319882272248649624;-0.090515971591818261688 -3.1513025017640283743
0.50508741670027457182 0.37005878740451370046 0.23491273961490991007
0.4491889464903843221 0.33339995343901274172 1.2067908160840983278 -
0.59515833150946961805 -0.16559984829347471069 -0.99352331250338465907 -
0.16823531726330659408;-2.528828715752065 4702 -2.0813719313255938381
0.19460292941435552994 -0.70687777072485824537 2.6437031934258734545
1.8364004185143163461 -1.6295257795923046018 0.41655513516243075633 -
0.046789439579059635688 0.17853729486465583221 1.0625153648936884743
1.2058216000513355404;-0.60623928914250257982 0.19200304899728098951
0.38222986204172643143 -0.19238341459914812259 -1.4049613708774273402 -
0.030277995670679333506 -0.49222874388091875097 0.61593542471008155825
0.00077743305439205347834 0.3333400977463704673 -
0.49862453595118350069 0.96855327071800179084;0.49489105486446494098 -
0.12698632858844721705 -0.32988765530102837076 0.018706288604083590177
0.25446600663871093984 0.48743420164818607176 0.27274106033689948791 -
0.61996297358245622267 0.054843794422113172071 -0.10139934737844893964
0.034585878288883271925 -
0.051092568302847002348;0.72501754570180165782 -2.595455692315326246 -
1.0948408878692632573 -0.042402384787049572645 1.3795142430387059562
3.2667659403663380502 -0.98785169548838225317 0.095283868851211150908 -
0.15728128981733027758 1.6849448083091818074 0.79729277207405568895
0.82857840075162603188;2.1384959737608411068 0.20502848101179332874 -
0.21379565885234230738 0.001416941522025802 4801 -2.457656114234886946 -
3.2485172178544505783 2.0333583546521252217 -3.7505489058899894594 -
```

```
0.11159966902808372835 -0.55708588478547949041 -2.727509042608381673
0.48110016556302925839;0.52489207867900367077 -5.2575161235568037554 -
1.6834089579356745769 -3.286200919934943343 -3.1898350493550537976
4.9251461178234503535 0.74258756688095184373 0.96744523550524030853
2.8538662917480150938 -4.0293934533528386055 -4.7938980998908897746 -
4.595163265709421907;0.18524319846074199791 -0.25030964891152629459
0.37194513092943365384 -0.20453849634316284556 -0.54635236144405019409
0.55015588938803139385 -0.26203246956053166672 -0.35247066548545147313
0.22234502186401494672 0.08363378654584656402 -0.30547460782554602954
1.1117736095748507186];

% Layer 2
b2 = [-1.2289348616962887384;-1.1202725992630044871;-
1.1978539141949626767;-0.86014019707310174212;-1.1896588407068933169;-
1.3576961404898955088];
LW2_1 = [-0.46314752930722691282 0.11768441795638540093 -
0.50488025680516634974 -0.18500246839973125135 -1.0988398257958476023 -
1.4395188455498564295 0.70330893850044640114 0.13394789365179041174
0.90755558070419430523 1.2237254442310392388;-0.50423228470656888955
0.21595476323492082682 -0.83792319596064146481 -0.066037102834457808709
-1.0367034642945160972 -1.9301552909185895057 0.37239261485963170983
0.14887922132303693656 0.38352513628975820614 1.1560123423292103517;-
0.22290589760799101038 0.21009927089209087803 -0.47698437554417227879 -
0.10140423166646114761 -1.5348431004820717316 -2.5030550171438208373
0.49001827142835219231 0.22191138579947308762 0.76220483339965128255
1.6724933835803674764;-0.030981310606274767988 0.38412038479796978274 -
0.51123337844904070959 0.039342044606983775368 -0.61974289476526811526 -
1.5479441128527848814 0.75056281812173342161 0.22233658078484591636
0.71372249233986384542 0.40660959915528754882;-0.31298054038680156452
0.13649256852006605434 -0.50691482468452564447 0.056164135170275034104
-1.0861388633601822828 -1.4260845260870971263 0.64377883634825905368
0.18647447442718678712 0.86679062095262071619 0.85579974145559256637;-
0.47770091806733766537 0.22695251073382063933 -0.70807048357169066755
0.03447400319214589453 -0.83822922138450728369 -1.2864816923006745508
0.59746903828310993223 0.17333718001974310341 0.83284576360685413921
1.231318925418159127];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain =
[0.00675981694415715;0.0016205157291308;0.01078033451378;0.016852181093
538;0.0033435590514323;0.0039360471066117 7];
y1_step1.xoffset = [0.926;31.298;0.632;0.252;6.066;3.012];
```

```
% ===== SIMULATION ========

% Format Input Arguments
isCellX = iscell(X);
if ~isCellX
	X = {X};
end
if (nargin < 2), error('Initial input states Xi argument needed.'); end

% Dimensions
TS = size(X,2); % timesteps
if ~isempty(X)
	Q = size(X{1},2); % samples/series
elseif ~isempty(Xi)
	Q = size(Xi{1},2);
else
	Q = 0;
end

% Input 1 Delay States
Xd1 = cell(1,3);
for ts=1:2
	Xd1{ts} = mapminmax_apply(Xi{1,ts},x1_step1);
end

% Allocate Outputs
Y = cell(1,TS);

% Time loop
for ts=1:TS

	% Rotating delay state position
	xdts = mod(ts+1,3)+1;

	% Input 1
	Xd1{xdts} = mapminmax_apply(X{1,ts},x1_step1);

	% Layer 1
	tapdelay1 = cat(1,Xd1{mod(xdts-[1 2]-1,3)+1});
	a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*tapdelay1);

	% Layer 2
	a2 = repmat(b2,1,Q) + LW2_1*a1;
```

```
        % Output 1
        Y{1,ts} = mapminmax_reverse(a2,y1_step1);
    end

    % Final Delay States
    finalxts = TS+(1: 2);
    xits = finalxts(finalxts<=2);
    xts = finalxts(finalxts>2)-2;
    Xf = [Xi(:,xits) X(:,xts)];
    Af = cell(2,0);

    % Format Output Arguments
    if ~isCellX
        Y = cell2mat(Y);
    end
    end

    % ===== MODULE FUNCTIONS ========

    % Map Minimum and Maximum Input Processing Function
    function y = mapminmax_apply(x,settings)
    y = bsxfun(@minus,x,settings.xoffset);
    y = bsxfun(@times,y,settings.gain);
    y = bsxfun(@plus,y,settings.ymin);
    end

    % Sigmoid Symmetric Transfer Function
    function a = tansig_apply(n,~)
    a = 2 ./ (1 + exp(-2*n)) - 1;
    end

    % Map Minimum and Maximum Output Reverse-Processing Function
    function x = mapminmax_reverse(y,settings)
    x = bsxfun(@minus,y,settings.ymin);
    x = bsxfun(@rdivide,x,settings.gain);
    x = bsxfun(@plus,x,settings.xoffset);
    end
```

stata 回归代码

```
import excel "D:\HuaweiMoveData\Users\volcano\Desktop\问题二成本加成预处理数据\
> 问题二\水生根茎类求加权成本加成定价.xlsx", sheet("Sheet1") firstrow
```

```
 regress 成本加成定价 销量千克
  estat hettest,rhs iid

  clear

  import excel "D:\HuaweiMoveData\Users\volcano\Desktop\问题二成本加成预处理数据\
> 问题二\花菜类求加权成本加成定价.xlsx", sheet("Sheet1") firstrow

. regress 成本加成定价 销量千克
  estat hettest,rhs iid

    regress 成本加成定价 销量千克,robust
     clear

. import excel "D:\HuaweiMoveData\Users\volcano\Desktop\问题二成本加成预处理数据\
> 问题二\花叶类求加权成本加成定价.xlsx", sheet("Sheet1") firstrow

. regress 成本加成定价 销量千克
 estat hettest,rhs iid


import excel "D:\HuaweiMoveData\Users\volcano\Desktop\问题二成本加成预处理数据\
> 问题二\辣椒类求加权成本加成定价.xlsx", sheet("Sheet1") firstrow clear

. regress 成本加成定价 销量千克

estat hettest,rhs iid


. import excel "D:\HuaweiMoveData\Users\volcano\Desktop\问题二成本加成预处理数据\
> 问题二\茄类求加权成本加成定价.xlsx", sheet("Sheet1") firstrow clear

. regress 成本加成定价 销量千克

estat hettest,rhs iid

import excel "D:\HuaweiMoveData\Users\volcano\Desktop\问题二成本加成预处理数据\
> 问题二\食用菌求加权成本加成定价.xlsx", sheet("Sheet1") firstrow clear
```

```
. regress  成本加成定价  销量千克

 estat hettest,rhs iid
 twoway (scatter  成本加成定价  销量千克  ) (lfit  成本加成定价  销量千克  )
```

附录 10

main3.1_时间序列预测销售量.pdf

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings("ignore")
from docx import Document
# 指定中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
# 步骤 1：读取数据
original_1 = pd.read_excel('./results/销售量按单品-日统计.xlsx')
# vegetable_categories = original_1.drop(original_1.columns[0], axis=1)
# 步骤 2: 转换日期格式并设置为索引
original_1['销售日期'] = pd.to_datetime(original_1['销售日期'])
original_1.set_index('销售日期', inplace=True)
# 步骤 3: 获取所有蔬菜品类
vegetable_categories = original_1.columns

# 步骤 3: 循环预测
forecast_steps = 1  # 预测未来一天的销量

# 创建一个空的 DataFrame
df = pd.DataFrame()

for i in range(1, 246):  # 从索引值 1 循环到 245
    # 步骤 4: 选择一个蔬菜品类进行分析
```

```
    specific_vegetable_data = original_1[vegetable_categories[i]]

    # 步骤 5: 建立 ARIMA 模型
    model = ARIMA(specific_vegetable_data, order=(5, 1, 0))
    model_fit = model.fit()

    # 步骤 6: 进行预测
    forecast = model_fit.forecast(steps=forecast_steps)
    rounded_forecast = round(forecast.values[0], 6)  # 提取数值部分并保留 6 位小数

    # 步骤 7: 可视化结果

        # plt.plot(specific_vegetable_data.index, specific_vegetable_data.values, label='Historical Daily Sales')
            # plt.plot(pd.date_range(start=specific_vegetable_data.index[-1], periods=forecast_steps+1)[1:], forecast, label='Forecasted Daily Sales', color='red')
    # plt.xlabel('Date')
    # plt.ylabel('Sales (kg)')
    # plt.title(f'{vegetable_categories[i]}销售量预测')  # 根据索引的列名设置标题
    # plt.legend()
    # plt.show()

    print(f'7.1 日{vegetable_categories[i]}销售量预测量:', rounded_forecast)  # 根据索引的列名打印预测值

    # 将每次循环的数据追加到 df 中
    df = pd.concat([df, pd.DataFrame({'单品名称': [vegetable_categories[i]], '7.1 日销售量预测量': [rounded_forecast]})], ignore_index=True)
    # 导出 DataFrame 对象为 Excel 文件
df.to_excel('./results/第三问/单品 7.1 销售量预测.xlsx', index=False)
# 步骤 3: 前 33
vegetable_categories = ['云南生菜(份)','云南油麦菜(份)','小米椒(份)','西兰花','芜湖青椒(1)','金针菇(盒)','竹叶菜','紫茄子(2)','螺丝椒(份)','娃娃菜','小皱皮(份)','双孢菇(盒)','菠菜(份)','姜蒜小米椒组合装(小份)','苋菜','螺丝椒','海鲜菇(包)','奶白菜','净藕(1)','西峡花菇(1)','枝江青梗散花','木耳菜','上海青','红薯尖','长线茄','小青菜(1)','洪湖藕带','青红杭椒组合装(份)','高瓜(1)','菱角','虫草花(份)','红椒(2)','青茄子(1)']

# 步骤 3: 循环预测
forecast_steps = 1  # 预测未来一天的销量

# 创建一个空的 DataFrame
```

```
df = pd.DataFrame()

for i in range(1, 34):  # 从索引值 1 循环到 34
    # 步骤 4: 选择一个蔬菜品类进行分析
    specific_vegetable_data = original_1[vegetable_categories[i]]

    # 步骤 5: 建立 ARIMA 模型
    model = ARIMA(specific_vegetable_data, order=(5, 1, 0))
    model_fit = model.fit()

    # 步骤 6: 进行预测
    forecast = model_fit.forecast(steps=forecast_steps)
    rounded_forecast = round(forecast.values[0], 6)  # 提取数值部分并保留 6 位小数

    # 步骤 7: 可视化结果
    # 设置图形的大小为 10x6 英寸
    plt.figure(figsize=(10, 6))
        plt.plot(specific_vegetable_data.index, specific_vegetable_data.values, label='Historical Daily Sales')
                    plt.plot(pd.date_range(start=specific_vegetable_data.index[-1], periods=forecast_steps+1)[1:], forecast, label='Forecasted Daily Sales', color='red')
    plt.xlabel('Date')
    plt.ylabel('Sales (kg)')
    plt.title(f' {vegetable_categories[i]} 销售量')  # 根据索引的列名设置标题
    plt.legend()
    plt.xticks(rotation=45)  # 设置横轴刻度的旋转角度为 45 度，以避免重叠
    plt.tight_layout()  # 调整图形布局，确保完整显示横轴刻度标签

    # 保存图形为 PNG 格式文件，设定 dpi 为 300 以获得更高的分辨率，并通过 bbox_inches='tight'参数确保保存整个图形
        plt.savefig(f'./results/第三问/单品 7.1 销售量预测图/{vegetable_categories[i]}7.1 销售量预测图.png', dpi=300, bbox_inches='tight')

    print(f' 7.1 日{vegetable_categories[i]} 销售量:', rounded_forecast)  # 根据索引的列名打印预测值

    # # 将每次循环的数据追加到 df 中
    # df = pd.concat([df, pd.DataFrame({'单品名称': [vegetable_categories[i]], '7.1 日销售量预测量': [rounded_forecast]})], ignore_index=True)
```

附录 11

main3.2_单品成本加成定价.pdf

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings("ignore")
from docx import Document
# 指定中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']

# 读取预处理过的数据
original = pd.read_excel('./第二问预处理.xlsx')
# 删除指定列
original.drop(columns=['扫码销售时间', '辅助列', '销售单价(元/千克)', '销售类型',
'是否打折销售', '分类编码', '批发价格(元/千克)', '损耗率(%)'], inplace=True)

# 按照时间顺序排序
original.sort_values('销售日期', inplace=True)
# 分别求每天的成本加成定价-第一步 加总
original['成本加成定价*单次销量'] = original['成本加成定价(元/千克)'] *
original['销量(千克)']

# 按照分类名称进行分组，计算每个单品的销售量
original = original.groupby(['销售日期','单品编码']).agg({'销量(千克)': 'sum', '成本
加成定价*单次销量': 'sum'})

# 分布求每天的成本加成定价-第二步 除以当天该品种总销量
original['成本加成定价'] = original['成本加成定价*单次销量'] / original['销量(千
克)']
# 删除多余项
original=original.drop(columns=['成本加成定价*单次销量'], inplace=True)

# 使用 pivot_table 将时间作为列，单品编码作为行，并将每个单品编码在该时
间下的成本加成定价作为值
```

```
pivot_table = original.pivot_table(index='销售日期', columns='单品编码', values='
成本加成定价')
# 将 pivot_table 中的 NaN 值填充为 0
pivot_table.fillna(0, inplace=True)
# 导出到 Excel
pivot_table.to_excel('./results/第三问/各单品求加权成本加成定价.xlsx')
```

附录 12

main3.3_时间序列预测成本加成定价.pdf

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings("ignore")
from docx import Document
# 指定中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
# 步骤 1：读取数据
original_1 = pd.read_excel('./results/第三问/各单品求加权成本加成定价.xlsx')
# vegetable_categories = original_1.drop(original_1.columns[0], axis=1)
# 步骤 2: 转换日期格式并设置为索引
original_1['销售日期'] = pd.to_datetime(original_1['销售日期'])
original_1.set_index('销售日期', inplace=True)
# 步骤 3: 获取所有蔬菜单品
vegetables = original_1.columns

# 步骤 3: 循环预测
forecast_steps = 1  # 预测未来一天的成本加成定价

# 创建一个空的 DataFrame
df = pd.DataFrame()
```

```
for i in range(1, 246):  # 从索引值 1 循环到 245
    # 步骤 4: 选择一个蔬菜品类进行分析
    specific_vegetable_data = original_1[vegetables[i]]

    # 步骤 5: 建立 ARIMA 模型
    model = ARIMA(specific_vegetable_data, order=(5, 1, 0))
    model_fit = model.fit()

    # 步骤 6: 进行预测
    forecast = model_fit.forecast(steps=forecast_steps)
    rounded_forecast = round(forecast.values[0], 6)  # 提取数值部分并保留 6 位小数

    # 步骤 7: 可视化结果

      # plt.plot(specific_vegetable_data.index, specific_vegetable_data.values, label='Historical Daily Sales')
            # plt.plot(pd.date_range(start=specific_vegetable_data.index[-1], periods=forecast_steps+1)[1:], forecast, label='Forecasted Daily Sales', color='red')
    # plt.xlabel('Date')
    # plt.ylabel('Sales (kg)')
    # plt.title(f' {vegetable_categories[i]} 成本加成定价预测')  # 根据索引的列名设置标题
    # plt.legend()
    # plt.show()

    print(f' 7.1 日{vegetables[i]} 成本加成定价预测量:', rounded_forecast)  # 根据索引的列名打印预测值

    # 将每次循环的数据追加到 df 中
df = pd.concat([df, pd.DataFrame({'单品名称': [vegetables[i]], '7.1 日成本加成定价预测量': [rounded_forecast]})], ignore_index=True)
    # 导出 DataFrame 对象为 Excel 文件
df.to_excel('./results/第三问/单品 7.1 成本加成定价预测.xlsx', index=False)
```

附录 12

main3.4_时间序列预测进价.pdf

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

```
import matplotlib.font_manager as fm
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings("ignore")
from docx import Document
# 指定中文字体
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
# 步骤 1：读取数据
original= pd.read_excel('./附件 3.xlsx')

# 按照时间顺序排序
original.sort_values('日期', inplace=True)
# 使用 pivot_table 将时间作为列，单品编码作为行，并将每个单品编码在该时间下的成本加成定价作为值
pivot_table = original.pivot_table(index='日期', columns='单品编码', values='批发价格(元/千克)')

# 将 pivot_table 中的 NaN 值填充为 0
pivot_table.fillna(0, inplace=True)
# 步骤 3: 获取所有蔬菜单品
vegetables = pivot_table.columns

# 步骤 3: 循环预测
forecast_steps = 1  # 预测未来一天的成本加成定价

# 创建一个空的 DataFrame
df = pd.DataFrame()

for i in range(1, 246):  # 从索引值 1 循环到 245
    # 步骤 4: 选择一个蔬菜单品进行分析
    specific_vegetable_data = pivot_table[vegetables[i]]

    # 步骤 5: 建立 ARIMA 模型
    model = ARIMA(specific_vegetable_data, order=(5, 1, 0))
    model_fit = model.fit()

    # 步骤 6: 进行预测
```

```
    forecast = model_fit.forecast(steps=forecast_steps)
    rounded_forecast = round(forecast.values[0], 6)  # 提取数值部分并保留 6 位小数

    ## 步骤 7: 可视化结果

      # plt.plot(specific_vegetable_data.index, specific_vegetable_data.values, label='Historical Daily Sales')
            # plt.plot(pd.date_range(start=specific_vegetable_data.index[-1], periods=forecast_steps+1)[1:], forecast, label='Forecasted Daily Sales', color='red')
    # plt.xlabel('Date')
    # plt.ylabel('Sales (kg)')
    # plt.title(f'{vegetables[i]}批发价格预测')  # 根据索引的列名设置标题
    # plt.legend()
    # plt.show()

    print(f'7.1 日{vegetables[i]}批发价格预测量:', rounded_forecast)  # 根据索引的列名打印预测值

    # 将每次循环的数据追加到 df 中
    df = pd.concat([df, pd.DataFrame({'单品名称': [vegetables[i]], '7.1 日批发价格预测量': [rounded_forecast]})], ignore_index=True)
```

附录 12

main3.5_时间序列预测进价.pdf

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.font_manager as fm
import matplotlib.pyplot as plt
import matplotlib
import math
import random
import matplotlib.pyplot as plt
# 读取预处理过的数据
original = pd.read_excel('./第二问预处理.xlsx')
# 选择 2023 年 6 月 24-30 日的数据
original['销售日期'] = pd.to_datetime(original['销售日期'])
recent_data = original[(original['销售日期'] >= '2023-06-24') & (original['销售日期
```

```
'] <= '2023-06-30')]
# 去除重复值并获取去重后的单品编码列表
unique_codes = recent_data['单品编码'].drop_duplicates().unique()

# 计算去重后的单品编码个数
num_unique_codes = len(unique_codes)

print(unique_codes)
print(num_unique_codes)

# 创建一个 DataFrame 对象
df = pd.DataFrame(unique_codes, columns=['单品编码'])

# 导出为 Excel 文件
df.to_excel('./results/49 个符合的单品.xlsx', index=False)

# 读取预处理过的数据
original_1 = pd.read_excel('./第三问 top33.xlsx')
# 根据单品求损耗率
def compute_rate(danpin):

    sunhao = original_1[original['单品编码'] == danpin]['损耗率']
    return sunhao.iloc[0]*0.01
# 给单品和进货量-求出陈列量
def compute_chenlie(danpin,jinhuo):
    chenlie=jinhuo*(1-compute_rate(danpin))

    return chenlie
# 给单品-求进价
def compute_jinjia(danpin):
    sunhao = original_1[original_1['单品编码'] == danpin]['平均进价']
    return sunhao.iloc[0]
# 给单品-求成本加成定价
def compute_dingjia(danpin):
    sunhao = original_1[original_1['单品编码'] == danpin]['修正后成本加成定价
预测']
    return sunhao.iloc[0]
# 给出单品编码、进货量-求出该商品的利润
def compute_lirun(danpin,jinhuo):
                    lirun=compute_chenlie(danpin,jinhuo)*compute_dingjia(danpin)-
jinhuo*compute_jinjia(danpin)

    return lirun
x = 0
```

```
a = 0.97  # 退火速度
T = 100  # 初始温度
min_quanju = T  # 全局最优解初始化
min_x = 0  # 全局最优解对应 X 坐标
n = 0  # 退火计次
listn = []
list_min = []
while T > 1:  # 终止条件

    min_jubu = T
    t = T
    min_x1 = random.randint(0, 500)/100

    for i in range(2):
        x1 = min_x1 + random.randint(-100, 100)/100
        y1 = mbhs(x1)
        dt = y1 - min_jubu
        if dt < 0 or dt == 0:
            t = y1
            min_x1 = x1
        else:
            zhishu = -dt/t
            met = math.exp(zhishu)
            pd = random.randint(0, 1000000)/1000000
            if pd <= met:
                t = y1
                min_x1 = x1
        min_jubu = t

    if min_jubu <= min_quanju:
        min_quanju = min_jubu
        min_x = min_x1

    print('局部最优解：', min_jubu)
    T = T * a

    n += 1
    listn.append(T)
    list_min.append(min_jubu)

print('全局最优解：', min_quanju)
print('最优解时 X 取值:', min_x)

# 制图
```

```
X = list_x = []
Y = list_y = []
c = 0
for i in range(500):
    list_x.append(i/100)
    list_y.append(mbhs(i/100))

#图中使用汉字
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

#最优解绘图
plt.subplot(121)
plt.plot(X, Y)
plt.scatter(min_x, min_quanju, c='r', label='全局最优解')
plt.legend()

#退火过程
plt.subplot(122)
plt.plot(listn, list_min)
plt.gca().invert_xaxis()

plt.show()
```