基于遗传算法最优蔬菜销售策略

摘　要

随着蔬菜类商超的规模日益庞大，蔬菜采购成本和销售问题成为了商超提高盈利的关键。本文针对商超经营中面临的补货和定价问题，采用线性规划及遗传算法，对销售量和定价的关系进行分析求解，从而制定蔬菜的补货和定价策略。

针对问题一，我们先对品类和单品销售数据进行了描述性统计，得到品类与单品的季节性分布规律。针对品类的相关性，我们采用了 spearman 相关系数进行分析，。对于单品而言，我们利用肘部法则确定了 K-means 聚类的聚类个数，将单品划分成三类，通过分析类别中的单品特点，得出了同一种类间菜品受众度相似的结论。

针对问题二，采用线性回归分别模拟历史情况下非折扣商品各品类销售总量与品类售价、品类加成率之间的关系，得到 6 种品类各自的解析式。首先，结合蔬菜商品成本受季节影响强的特点，以不同节气为特征采用 TCN-Attention 特征融合模型预测未来 7 日不同品类商品的批发价格，与折损率计算得到真实成本。用 bootstrap 样本拓展法 TCN-Attention 时间序列模型预测出未来 7 日期间不同品类商品销售总量范围区间，以此作为寻优中的限制条件。在得到真实成本、销量与售价关系，并确定限制销量范围的条件下，通过线性规划寻找最大利润及该情况下每日的定价、补货量。最后通过历史数据计算各品类中折扣商品的占比及折扣程度，以计算折扣商品利润。用非折扣商品利润与折扣商品利润加和得到 7 天总利润依次为 1801.15、1466.45、1260.20、1518.47、1521.29、1487.33、1433.36。

针对问题三，从批发成本数据中选出 61 件可售单品，采用 topsis 模型，以需求量、利润、折损率为指标对 61 件单品进行评分。为了尽可能多地满足市场对各品类商品的需求，从中选取获得评分最高的 33 种单品作为补货范围，用回归模型模拟这 33 种商品需求量与定价的解析式。再次采用 TCN-Attention 特征融合模型模型预测 7 月 1 日这 33 种单品的批发价格，采用 bootstrap 样本拓展法基于历史售价预测 7 月 1 日当天 33 种农产品价格范围，作为价格范围限制条件，结合题目要求的单品最低 2.5kg 补货量，采用遗传算法通过对捕获策略演利润最大化寻优，求得当日最大利润及该情况下 33 种单品的定价与补货量。得到 7 月 1 日最大利润为 2118.65。

针对问题四，我们提出了将天气现象纳入统计数据的建议，并分析了天气、气温以及湿度对于蔬菜销售和折损的影响，验证了天气现象对补货量和定价策略的影响。

关键词:spearman 相关系数 k-means 聚类　TCN-Attention　BFGS 寻优　遗传算法

一、 问题重述

蔬菜是人们餐桌上必不可少的食品，也是超市经营的重要产品。与其他商品相比，蔬菜新鲜易腐、不耐贮运，若因运输或存放时间过长造成折损，超市往往需要对蔬菜进行降价处理，部分菜品甚至无法再售，大大影响了蔬菜商品的销售与盈利。除此之外，蔬菜类商品种类多样且具有明显的季节性，蔬菜的产量与价格在不同的时间内呈波动趋势。蔬菜的销售量与时间、供求息息相关，为扩大蔬菜行业的盈利，根据历史销售数据合理制定、调整商品单价与补货数量具有重要意义。

为此，本题要求我们建立相关数学模型，完成下列问题：

1. 根据附件数据，分析蔬菜各品类及单品销售量的分布规律与相互关系。

2. 分析各蔬菜品类的销售总量与成本加成定价的关系，并以品类为单位、以收益最大化为目标，给出 2023 年 7 月 1-7 日的日补货总量与定价策略。

3. 由于超市销售空间有限，针对单品的补货计划需将可售单品总数控制在 27-33 个，并满足 2.5 千克的最小陈列量。该计划需依据 2023 年 6 月 24-30 的可售品种，给出 7 月 1 日的单品补货量和定价策略，在尽可能满足市场需求的前提下，实现商超收益最大化。

4. 提供需要采集的其他相关数据，为更好地制定蔬菜商品的补货和定价决策提供帮助与建议，并说明理由。

二、 问题分析

2.1 问题一的分析

问题一旨在分析蔬菜各品类以及单品的销售量分布规律与相互关系。为探究数据分布规律，我们需将附件数据进行合并和整理，利用直方图、折线图等工具进行可视化处理，并对品类和单品进行统计性描述。为研究品类间的相互关系，我们需将日销量按照品类进行统计，并利用 spearmen 相关系数进行求解。针对单品，需要我们通过 K-means 聚类模型分析单品之间的内在联系。

2.2 问题二的分析

问题二要求我们解决两个问题，第一个问题是探究蔬菜品类的销售总量与成本加成定价的关系，首先我们需明确成本加成定价的定价规则，即售价=单位成本*（1+F），由附件中我们可以得到商品的批发价格、销量和损耗率，由此我们可以得到每笔销售的产品真实成本与销售总量，理论满足经济供需，可以对售价与销售总量进行线性回归，从而的到售价与销售总量的关系，又可以通过售价与真实成本确定加成率。

第二个问题要求我们制定 7.1-7.7 日的补货总量和定价策略以得到最大收益，这需要我们预测销售总量区间，并预测相应的商品批发价格，通过销量和成本加成定价的关系进行最优化，从而得到利润最大化的解。

2.3 问题三的分析

问题三要求我们解决两个问题，第一个问题是在 6 月 24 日到 6 月 30 日中可售的蔬菜单品类型中筛选出 27-33 种用于补货。这要求我们构建评价模型，选取合适的特征对各种单品蔬菜进行评分，选取评分最高的 33 种品类用于补货。

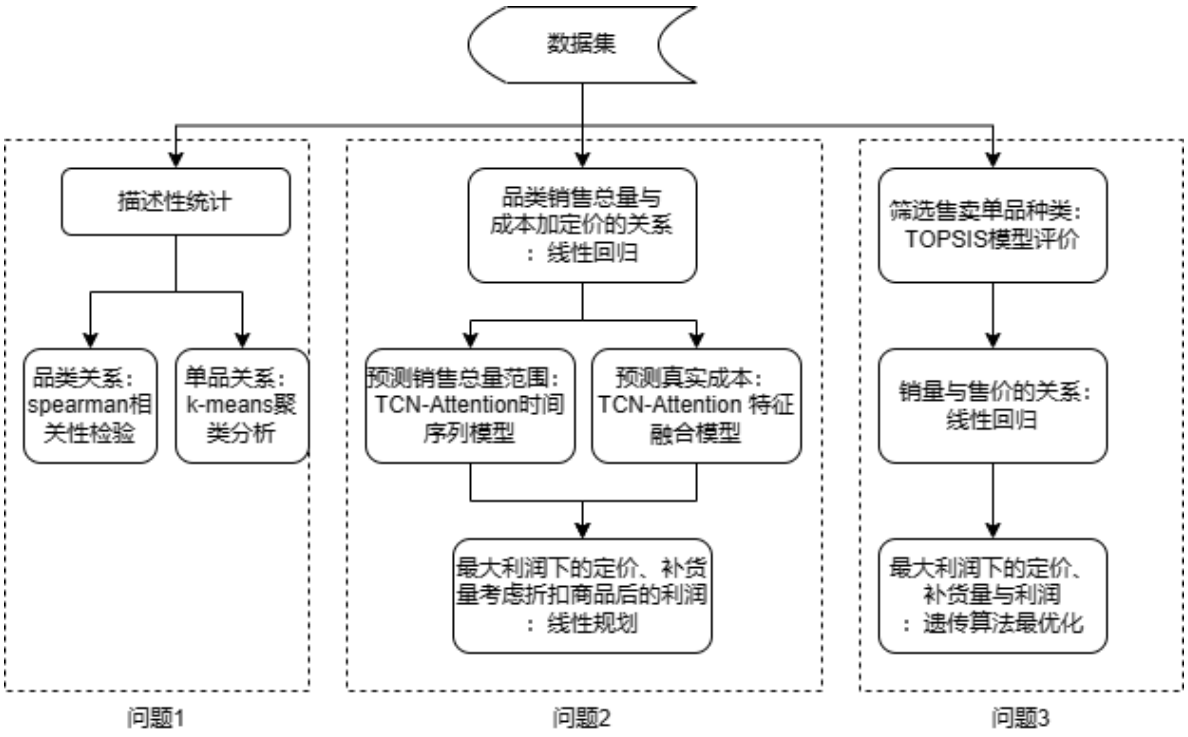
第二个问题要求我们根据第一个问题中筛选出的蔬菜单品种类进行进货，找出最大收益情况下的补货量与定价。这要求我们探究所选单品销量与售价的关系，根据题目要求的限制及历史数据得到的限制进行约束，通过遗传算法最优化使 7 月 1 日收益最大，从而得到对应的补货量与定

价。

2.4 问题四的分析

问题四要求我们添加其他影响因素，以改进蔬菜商品的补货和定价决策，提高商超利润。这要求我们分析并确定未考虑的影响补货和定价的关键因素，从而优化经营策略。

下图为本次建模的工作流程图：



图表 1

三、 模型假设

假设所有商品均在销售当天退货，所有商品退货仅影响当日的销售量，且退货商品不得二次销售。

假设所有折扣商品均在当日处理完毕，未销售完毕的商品隔日不可再销。

假设经营期间供给和需求均满足金融市场规律，无自然灾害或重大事故。

假设批发商供货正常，且批发价格符合市场规律，不存在极端的变动。

假设预测模型中，菜品的销量就是补货所需的数量，所有购进菜品均能及时销售完毕。

四、 符号说明

符号	说明	单位
S	品类销售总量	
C	品类实际成本	
P_C	加权品类售价	
w	单件商品批发价格	
s	单件商品销量	
L	单品损耗率	
r	品类加成率	
c	单件商品实际成本	
R_c	非折扣商品品类利润	
R_d	折扣商品品类利润	
d_1	品类折扣商品利润比例	
d_2	品类折扣商品销量比例	
R_2	第二问总利润	
C_p	品类预测实际成本	
S_p	品类预测销量	
P_p	品类预测价格	
G_i	Topsis 评价得分	

五、 模型的建立与求解

缺失值：通过 find 函数，本数据不存在缺失值。

异常值：1.观测数据我们得到销售数据中存在一次销售记录销量为 160kg，与其他销售量相差极大，我们认定为特殊情况，作为异常值剔除。

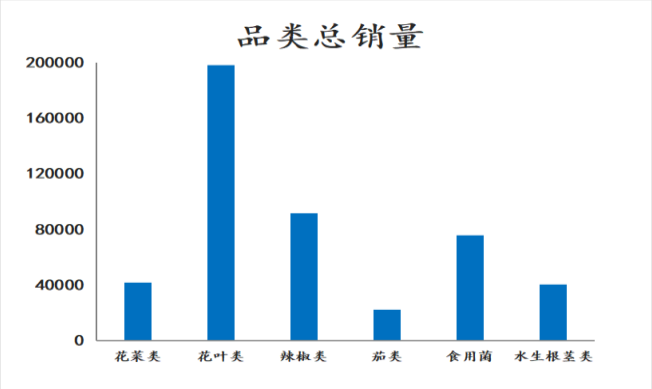
2.根据观测，许多蔬菜的批发价格存在异常。如 0.01 元/kg，明显与实际情况不符，需要剔除异常离群值。对于由附件中批发价格的数据，我们将每单销售的单品的批发价格视为正态分布，在正态分布中，对每个单品的数值大于或小于 μ 加减 3σ 的概率小于 0.27%，下图为其中部分单品批发价分布情况。整合数据后，我们匹配每一单销售记录于其对应成本，共剔除 7897 单（占比 0.89892%）成本异常的销售记录。

3.由于打折商品的打折价格与根据成本加成所得的定价相差较大，打折商品销售数据的存在对寻找销售总量与成本加成定价的关系起不利影响，且打折蔬菜占比重较小，因此在探寻销量与定价时，我们将打折蔬菜的销售数据剔除。但在最终计算销售收益时考虑折扣商品对利润的影响，单独计算折扣商品利润，结合得到总利润。

5.1 问题一

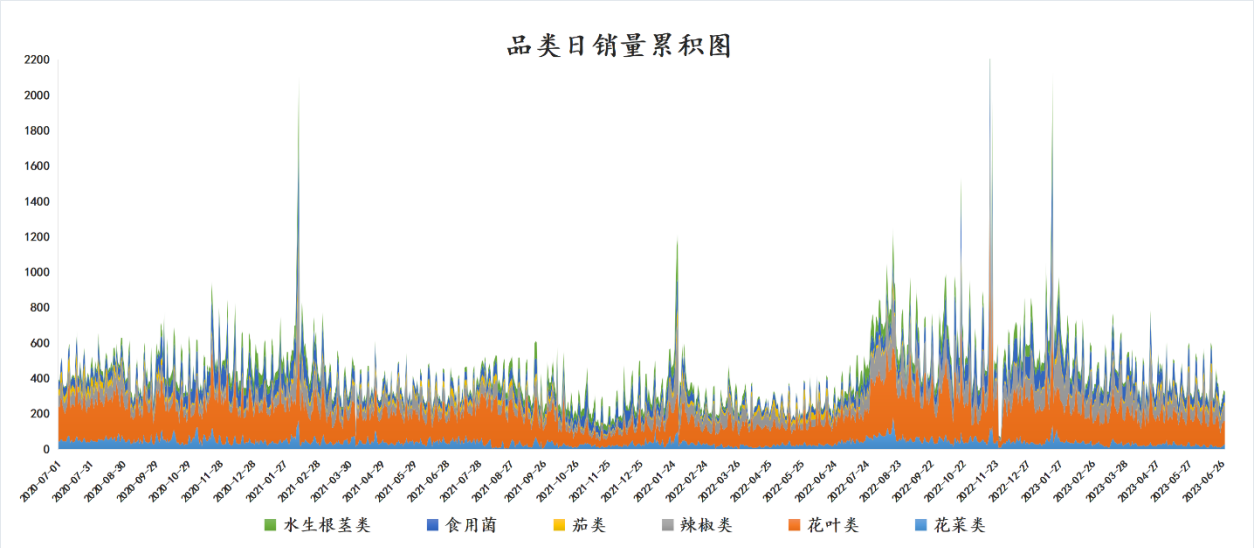
5.1.1 蔬菜各品类和单品销售量的分布规律

为探究蔬菜品类销售量的分布规律，我们首先对六个品类的总销量进行了统计，并绘制直方图将数据可视化，由图表 2 所示，由数据可知，六个品类的销售总量存在较大的差异，其中花叶类销量最高，茄类购买量相对最少。



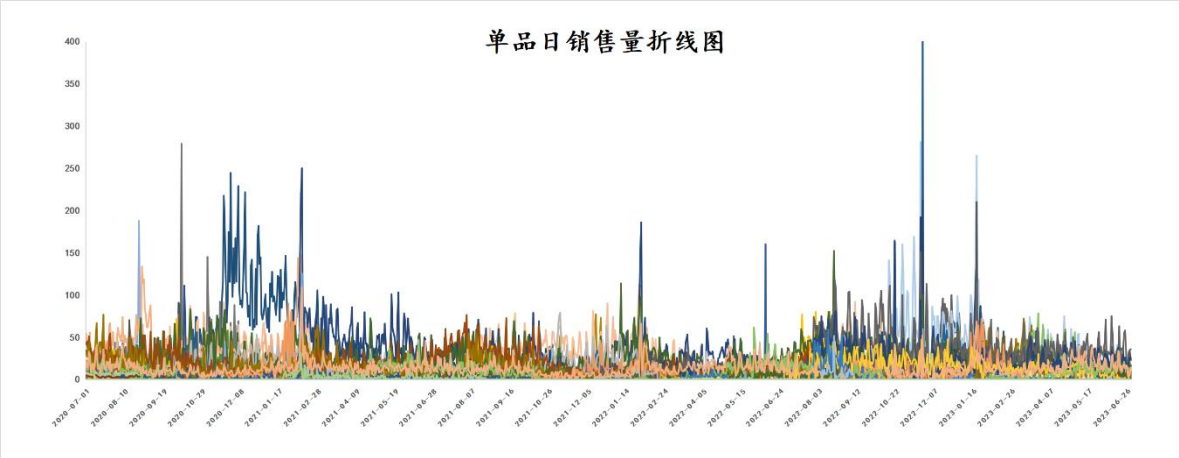
图表 2

为进一步研究各品类分布，我们对品类的日销售量绘制了面积堆积图图表 3，通过观察各品类销量面积占比，我们可以得出在变化过程中，各品类的销量比例几乎恒定，且与图表 2 中的占比基本一致。观察日销量的走势，我们可以得出蔬菜的销售量具有较明显的季节性，每年的销售走势基本相同，并在一月末形成峰值。



图表 3

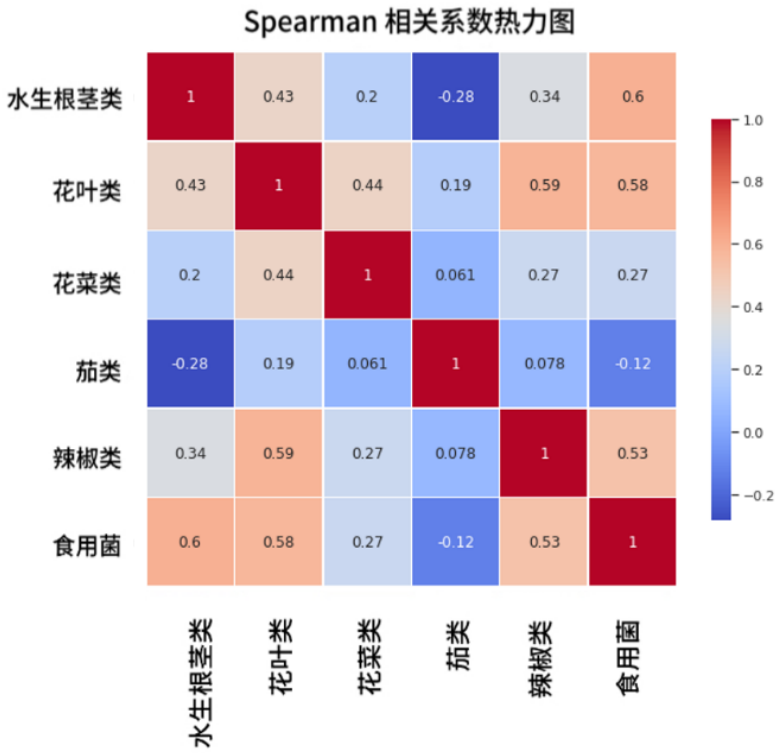
单品品类较为繁多，我们对其绘制了单品日销量的折线图图表 4，各单品也呈现出了明显的季节性，各单品的走势也较为近似，在单品的盛产期销量也将达到短期的峰值，其余时间段内的各单品销量均在对应数值附近平稳波动。



图表 4

5.1.2 蔬菜品类间的相互关系

为探究品类间的相互关系，我们使用 Python 对数据进行斯皮尔曼相关性系数分析，将数据可视化得到下列热力图：



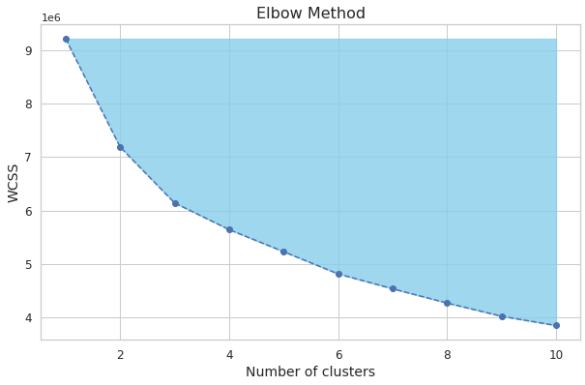
图表 5

由图可得，食用菌与水生根茎类、辣椒与花叶类、辣椒与食用菌、花叶类与食用菌的销售量具有较强的相关性，花叶类与食用菌、花叶类与花菜类的销售量具有一定的相关性，其余品类间的相关性较为薄弱。

5.1.3 蔬菜单品间的相互关系

鉴于单品数量较大，不宜直接进行单品之间的比较分析，我们决定采用 K-means 聚类分析，得到单品的关系。

为确定合适的分类数，我们使用 Python 绘制出了不同 K 值形成的图表 6 中的碎石图，利用肘部法则可知，在 K 值为 3 时，曲线呈现出明显放缓的趋势，故以 3 作为聚类个数。



图表 6

根据以上分析，我们取聚类数 K=3，根据 Python 代码得出以下聚类结果（具体输出结果见附录）：

种类 1（223 种）：艾蒿、白菜苔、白蒿、白玉菇、薄荷叶、本地黄心油菜等

种类 2（10 种）：大白菜、净藕(1)、泡泡椒(精品)、青梗散花、芜湖青椒(1)、西兰花、西峡香菇(1)、云南生菜、云南油麦菜、紫茄子(2)

种类 3（13 种）：保康高山大白菜、菠菜(份)、洪湖莲藕(粉藕)、金针菇(盒)、螺丝椒(份)、奶白菜(份)、娃娃菜、小米椒(份)、小青菜(份)、小皱皮(份)、云南生菜(份)、云南油麦菜(份)、枝江青梗散花

种类 1 包含的单品种数较多，种类 2 和种类 3 包含的种类较少且构成较为相似。种类 3 中主要包含基础菜品或者常见的配料，这些菜品容易获取且需求量大；种类 2 相较种类 3 而言品种更加精确或者包装方式有所更改，其需求量相对缩小，但也属于热销产品；种类 1 中包含了剩余的单品，这些品种适用范围较小或者菜品搭配较为固定，销量相对最少。

5.2 问题二模型的建立与求解

问题要求我们研究销量与成本加成定价的关系，通过变换成本加成定价的概念，可以得到以下公式。

蔬菜销售中不同种类的蔬菜会产生不同程度的折损，折损的部分将会增加蔬菜销售的成本。为了提高准确性，我们需要考虑折损率对成本的影响，求出单件商品实际成本 c。单件商品的真实成本 c 可根据单件商品批发价 w，单件商品销量 s，单品损耗率 L 求得，其公式为：

$$c = w \cdot \left(\frac{s}{1-L} \right)$$

根据销量对品类中的单件商品真实成本进行加权，即可得到品类实际成本。根据成本加成定价原理，售价为单位成本加上特定加成率。在这个问题中，可以得到品类加成

率r，与对一个品类中单品售价的加权得品类售价P_C，单件商品真实成本c，单件商品销量s，品类销量S之间的的关系，公式为：

$$P_c = \left(\frac{\sum_{i}^{n} c_i \times s_i}{S} \right) \cdot (1 + r)$$

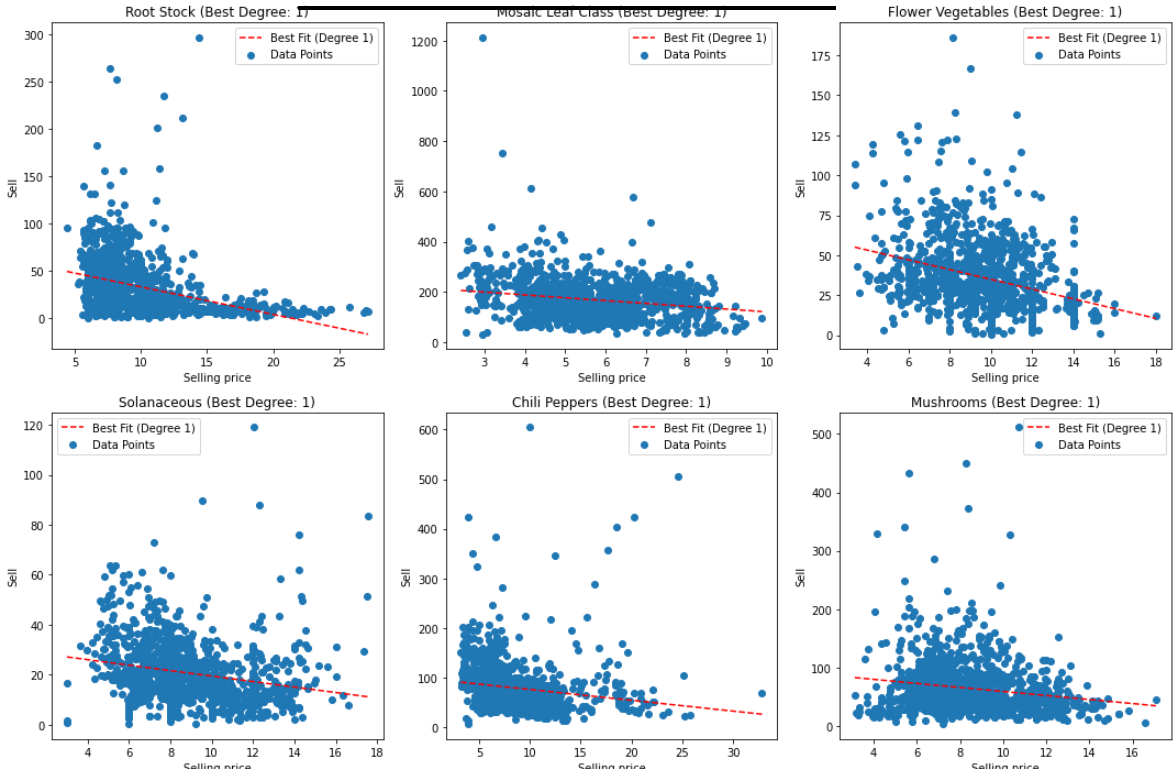
3. 由于打折商品的打折价格与根据成本加成所得的定价相差较大，打折商品销售数据的存在对寻找销售总量与成本加成定价的关系起不利影响，且打折蔬菜占比重较小，因此在探寻销量与定价关系时我们仅研究未打折蔬菜的销售数据。但在最终计算销售收益时考虑折扣商品的存在，对其计算以保留影响。

5.2.1 线性回归分析解销量与售价关系

通过 python 求解销售总量与售价的关系，得到以下关系式，而后根据关系式找出销售总量 y 与售价 x 的关系。

类别	关系式
水生根茎类	y = -2.914 x + 62.26
花叶类	y = -11.24 x + 232.6
花菜类	y = -3.054 x + 65.47
茄类	y = -1.096 x + 30.43
辣椒类	y = -2.18 x + 97.75
食用菌	y = -3.453 x + 93.52

表格 1

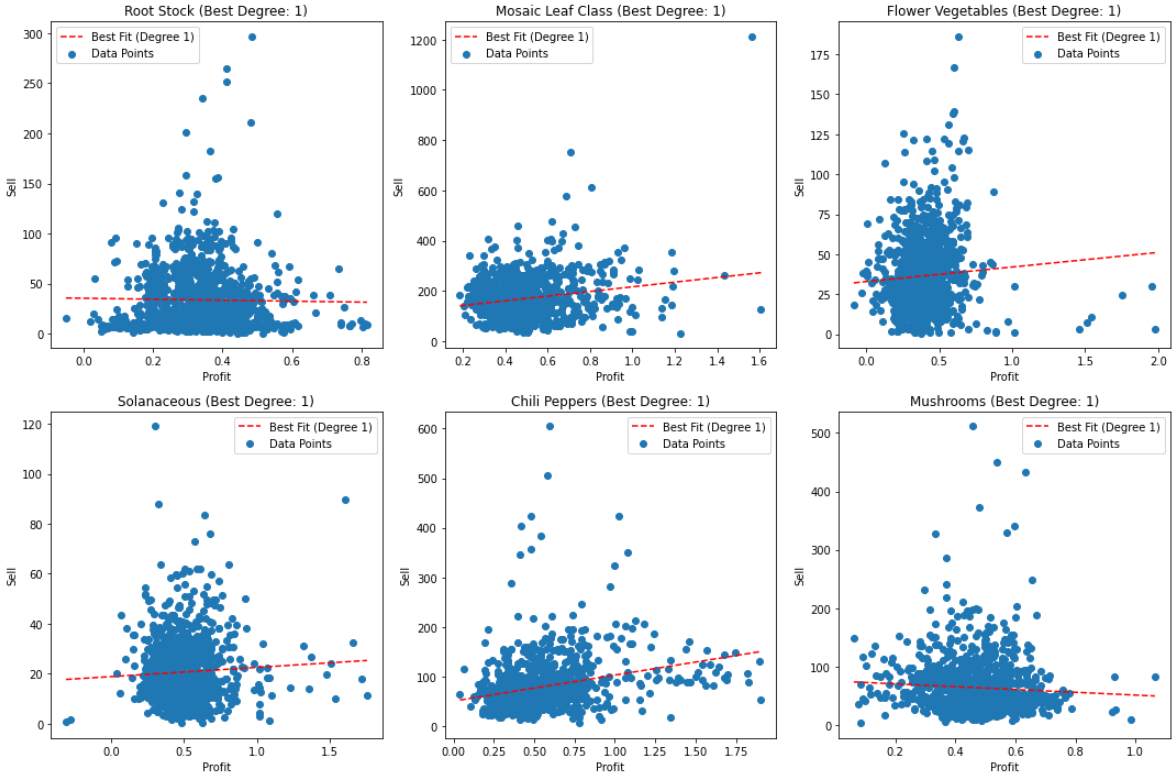


图表 7

进而比较销售总量与利润率的关系，得到：

品类	
水生根茎类	y = -4.869 x + 35.37
花叶类	y = 92.79 x + 123.2
花菜类	y = 9.122 x + 32.89
茄类	y = 3.688 x + 18.86
辣椒类	y = 52.49 x + 50.52
食用菌	y = -23.98 x + 75.07

表格 2



图表 8

可以看出，不同品类的销量与利润率并不是按照经济学正常规律呈现负相关，并不能明显得出二者之间的关系。

问题二要得到商超收益最大情况下未来一周的日补货量和定价策略，要求最大收益情况，我们首先要预测出未来一周的品类真实成本。

5.2.2 TCN-Attention 特征融合成本预测模型

由附件 3 的历史成本数据，不同季节时同一种蔬菜批发价格不同，考虑到蔬菜批发价格受季节变化影响较强（如下图所示），我们将在对 7 月 1 日到 7 月 6 日的各品类蔬菜批发价格预测中考虑季节的影响。我们将二十四节气和对应产品品类的时间序列，预测未来七天品类加权平均成本的变化。成本预测模式使用 TCN-Attention 特征融合神经网络模型，实现多输入单输出的训练和预测。

一．数据预处理

二十四节气层次 one-hot 编码

本文选取中国传统文化中的二十四节气作为季节，时间对某一特定蔬菜品类加权平均成本的影响因子之一，本文将某一个特定日期编码为其相距时间最短的节气为进行神经网络模型训练，需对其进行编码。中国农历中有二十四个节气，如果使用简单的数字编码会导致编码结果缺乏周期信息，同时由于二十四节气在时间序列上是周期性的，如果采用数字编码会导致一年最后一个节气(大寒)和第一个节气(立春)编码结果相差巨大，与实际情况不符。

如果使用简单 one-hot 编码会导致维度稀疏问题，会影响模型学习数据集特征的能力。因此我们使用层次 one-hot 编码方法对二十四节气进行编码。

层次 one-hot 编码的特点在于利用了季节和节气之间的从属关系，一个季节存在六个节气，我们首先对季节进行 one-hot 编码，随后按照一个季节中的六个节气的先后顺序，对六个节气进行 one-hot 编码，将两个结果横向拼接得到编码结果，如下表：

节气	二十四节气层次 one-hot 编码结果
立春	[1,0,0,0,1,0,0,0,0,0]
清明	[1,0,0,0,0,0,0,0,1,0]
大寒	[0,0,0,1,0,0,0,0,0,1]

表格 3

采用层次 one-hot 编码使得节气信息训练数据相较 one-hot 编码维度收缩 53.8%，显著提升编码结果信息表达能力。

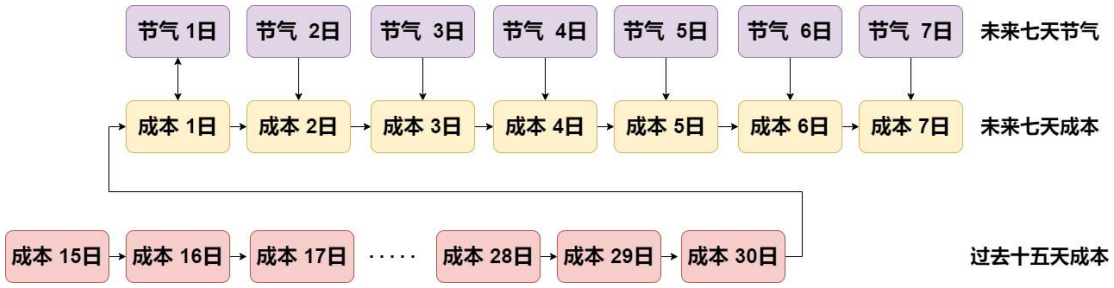
时间序列滑动窗口

本模型为实现对未来七天某一品类成本的预测，需要使用滑动窗口对数据进行预处理，即在模型训练前，将数据集分割为多个特定长度的窗口用于从历史数据中提取有用信息和模式，如下表 12 所示。

窗口设置：滑动窗口被设置为 15 天，意味着模型将考虑过去 15 天的数据来预测未来 7 天的成本。

节气信息：除了过去 15 天的成本数据外，该模型还将使用未来 7 天的节气信息。节气信息作为一个附加的特征输入。

窗口滑动：窗口以一天为步长滑动，生成新的数据子集，用于训练。



图表 9

通过滑动窗口，你可以在一个小的时间段内观察变量，从而更好地捕捉其动态性或季节性。这使得模型能够学习这些短期依赖关系，并据此进行更准确的预测。

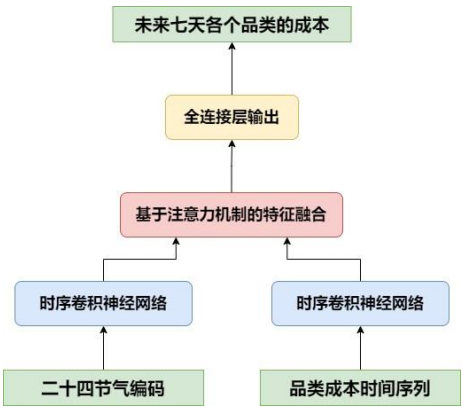
数据归一化

当二十四节气编码和滑动窗口完成之后需要对所有数据进行归一化，本文使用的归一化算法为 MinMaxScaler 算法。

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

二．模型结构

本文构建的模型为时序卷积神经网络-注意力机制特征融合模型(TCN-Attention-feature-fusion Model)，由两个输入层和一个特征融合层以及一个全连接输出层构成，如下图所示：



图表 10

时序卷积层

时序卷积（TCN）层在本模型中是模型的输入层，本模型的两个时序卷积神经网络分别作为二十四节气编码和品类成本时间序列的输入层。时序卷积神经网络的特征在本模型中可以优化神经网络模型的拟合精度，并且增强了模型在时间序列上的推理能力。

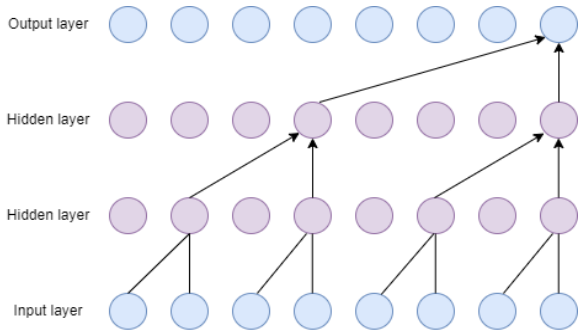
因果卷积：TCN 中的因果卷积是一种特殊的卷积操作，其主要特点是在计算当前时间步（或点）的输出时，仅使用该时间点及其之前的信息，而不使用任何未来信息。这种方式确保了输出序列与输入序列之间存在明确的因果关系，即输出仅由输入及其历史信息决定。数学上，给定一个输入序列$y[t]$，因果卷积可以表述为：

$$y[t] = \sum_{r=0}^{K} w[r] \cdot x[t-r]$$

其中$y[t]$是给定的时间序列，$w[r]$是卷积核在时间r的值，K是卷积核的长度。因果卷积同时限制模型从未来数据获取信息的能力，提高了模型推理能力。

空洞卷积(Dilated Convolution)：相较于传统序列神经网络如 LSTM 等，本模型使用 TCN 作为输入层具有空洞卷积的特征，空洞卷积的重要特征是其卷积核并不是一个连续卷积核，而是在时间序列上以一个较大的间隔进行卷积，在数学上可以表示为：

$$y[t] = \sum_{r=0}^{K} w[r] \cdot x[t-r \cdot d]$$



图表 11

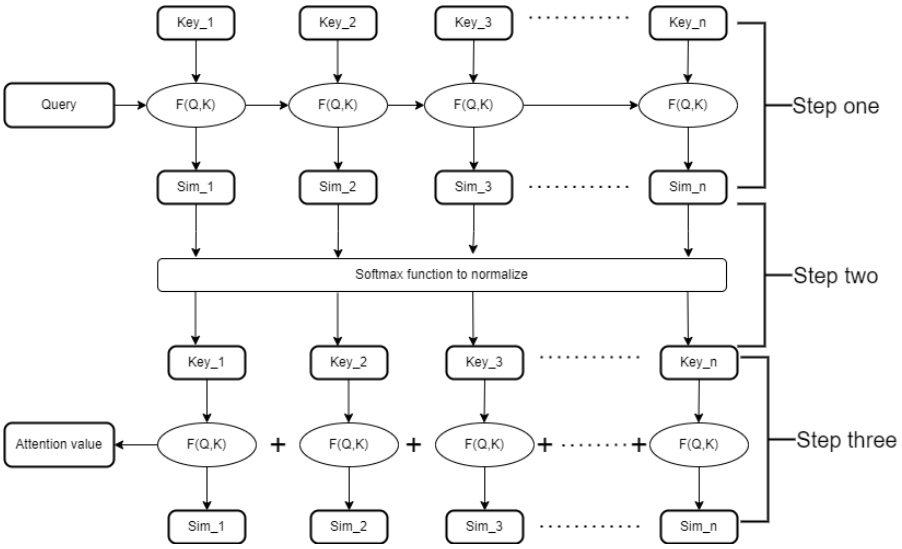
如上图所示，空洞卷积可以增强感受视野，由于空洞卷积具有较大视野有助于模型捕捉不同季节（节气）或长期趋势对蔬菜价格的影响，从而提高预测准确性。

基于注意力机制的特征融合

本模型的特点是多输入单输出，在训练和预测的过程中需要进行特征融合，即将两个输入模型输入的特征矩阵进行融合。

注意力机制自被发表依赖已经广泛地被用于各种神经网络模型中，尤其是处理如本模型一样的多信息源模型或者多模态模型。在外面的模型中，主要的输入有两个，一个是经过时序卷积网络（TCN）处理的蔬菜品类成本时间序列，另一个是经过时序卷积网络（TCN）二十四节气时间序列的层次 one-hot 编码。为了有效结合这两种信息进行准确预测，本文映入了基于注意力机制的特征融合层。

注意力机制如下图所示：



图表 12

在神经网络模型中，注意力机制是一种允许网络模型在处理不同输入同时自动关注重要信息的机制，可以帮助网络实现注入选择性注意力，动态调整，减少干扰和提高可解释性性能。在本文中，我们使用二十四节气特征作为键值，历史成本时间序列特征作为查询值，输出未来成本时间序列作为查询值，本模型注意力机制的运算流程如下：

第一步，横向拼接两个特征矩阵x_1, x_2，计算每个 query 和每个 Key，得到权重系数:

$$K = Concat(x_1, x_2)$$

$$Sim\left(Q, K_i^T\right) = Q \cdot K_i^T$$

第二步，使用 Softmax 对权重进行归一化处理，得到每个特征的注意力权重：

$$\alpha_i = Softmax\left(e^{Sim_i}\right) = \frac{e^{Sim_i}}{\sum_{j=1}^{L_x} e^{Sim_i}}$$

第三步，有了注意力权重后，我们可以计算加权平均的融合特征：

$$Fused\ Features = A(Q, K, V) = \sum_{i=1}^{L_x} \alpha_i \cdot V_i$$

融合特征后，我们通过一个全连接输出层进行最终预测。这一层的目的是将融合的特征映射到成本预测的具体值。

$$Predicted\ Cost = Fused\ Features \cdot W_o + b_o$$

通过对六个品类的农产品分别进行模型训练得到六个预测模型，得到了未来七天六种农

产品的具体单位成本，如下表：

日期	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌类
2023/7/1	10.284833	5.2804837	5.6377425	7.881129	6.693183	5.181829
2023/7/2	13.190946	5.1449084	7.4119754	7.436173	11.00206	6.877834
2023/7/3	11.713792	5.0762324	7.570472	7.916279	17.00632	6.027922
2023/7/4	14.567513	2.4699419	7.765049	7.492321	14.76444	7.714188
2023/7/5	11.58972	3.700917	7.3364487	5.795903	14.83997	5.960561
2023/7/6	15.696482	1.9827238	7.78509	7.303841	16.8858	8.38395
2023/7/7	15.906796	5.041679	7.910936	7.422056	9.941831	8.501409

表格 4

5.2.3 TCN-Attention Bootstrap 抽样时间序列预测模型预测销量范围

为了得到更准确的日补货总量和定价策略，我们希望未来一周各品类蔬菜的补货量与实际情况更接近，所以我们决定对未来七日的销量范围进行预测，并用此销量范围区间作为寻找最优策略的限制条件。为了对未来一周的销售量区间进行预测，我们采用了基于 TCN-Attention 的 Bootstrap 抽样时间序列预测模型，通过对同一数据集不同切分方式得到的数据集分别训练神经网络模型，通过多个模型对未来七天销量的的预测，得到一个置信区间作为未来七天销售量的范围。

数据预处理

1.时间窗口-归一化

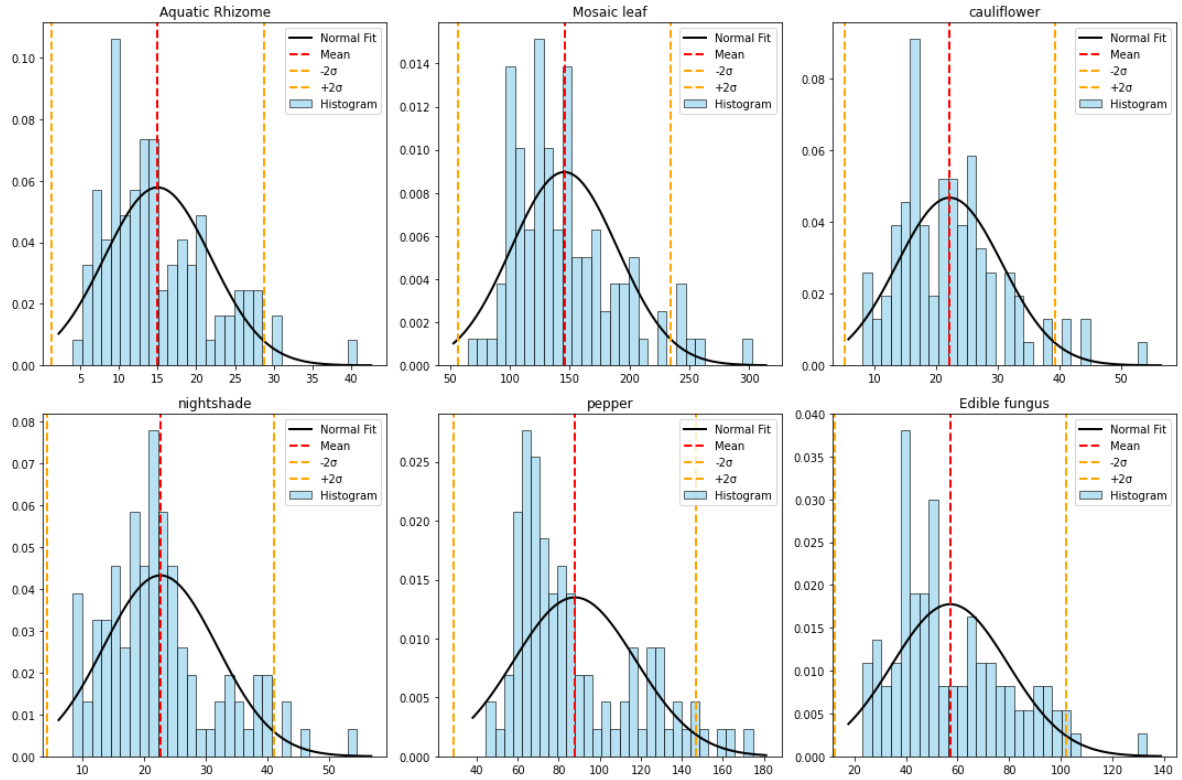
对于本模型的训练，我们使用过去七天的销售量预测未来一天的销售量，所以我们设置时间窗口大小为 7，获得完整的待训练数据。完成滑动窗口处理之后使用 MinMaxScaler 算法对其实现归一化。

2.Bootstrap 样本拓展法

为了预测未来一段时间的六种品类的农产品的销售情况区间，我们需要对各个品类分别训练 100 个模型。为了更好地学习数据集中的依赖模式，同时增加模型训练过程中的噪音，我们对同一品类下不同模型分别训练在不同训练数据集，对于每一个品类的模型，我们在完整归一化之后的完整待训练数据集中切分出 100 个不同的训练集，为了保证时间序列的完整性，我们需要切分的是连续的数据集而非离散的。

我们在每个品类不同数据集上分别训练模型，得到每个品类的 100 个神经网络模型，预测出未来七天，六种特定品类的销售量区间，这个区间符合正态分布。

最终得到 6 个品类在 7 月 1 日到 7 月 6 日的预测真实成本，如下图。



图表 13

限制销售范围如下表：

品类	限制销量范围
水生根茎类	(1.22 ， 28.79)
花叶类	(56.5 ， 234.41)
花菜类	(5.17 ， 39.24)
茄类	(4.20 ， 41.08)
辣椒类	(28.76 ， 147.06)
食用菌	(12.23 ， 102.06)

表格 5

对于不折扣品类利润对于不折扣品类 利润R_c，可根据品类售价P_p，品类预测真实成本C_p，品类销量S_p列出公式=（品类售价-品类预测真实成本）*品类销量

$$R_c = （P_p - C_p） \times S_p$$

5.2.4 BFGS 模型求解最终利润和销量

将以上的各品类销量预测范围作为限制条件，在已知各品类真实成本、需求量于售价关系的情况下，在 python 中使用 BFGS 算法寻优。

TCN-Attention Bootstrap 抽样时间序列预测模型的预测得到了六种品类在未来七天内的销售量区间，我们通过线性回归得到了六种品类，价格和销售量之间的关系，本模型使用 BFGS 算法在限制条件内进行最优化。

1.目标函数

$$Maximize\ Z = \sum_{i=1}^{n}(p_i - c_i) \times Function\ relation(p_i)$$

其中，Z是总收益、n是品类数量、p_i是第i个品类的价格、c_i是第i个品类的 c 成本、$Function\ relation(p_i)$是在p_i价格下的预测销量。

2. **约束条件**

价格-成本约束：$p_i > c_i$，保证产品的售价不能低于成本。

价格上限-下限约束：$p_i > Min(price)$，$p_i < Max(price)$，其中$Min(price)$和$Max(price)$分别是 TCN-Attention Bootstrap 抽样时间序列预测模型预测出六种品类的销售量的上限和下限。

销量约束：$Function\ relation(p_i) > 0$，我们需要预期销量不能为 0。

3. 基本步骤：

1. 初始化：选择一个起始点x_0，在本模型中起始点是随机生成的六个品类在未来七天.的定价和一个初始近似 Hessian 的矩阵H_0。
2. 迭代：对于$k = 0,1,2$ …执行以下操作：
   a. 搜索方向：计算搜索方向p_k，作为解$H_k p_k = -\nabla f(x_k)$的p_k。
   b. 线搜索：找到一个步长SL_k，使得$f(x_{k+1}) < f(x_k)$，其中$x_{k+1} = x_k + SL_{k\times} p_k$。
   c. 更新近似 Hessian：计算x和$f(x)$的差值$s_k = x_{k-1} - x_k$，和$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$，并且用这些更新为$H_{k+1}$。
   d. 收敛检查：如果满足某个收敛标准则停止迭代。

通过上述算法得到收益最大情况下非折扣商品最大利润、品类定价、品类补货量的数据：

品类定价（每公斤）

日期	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌类
2023/7/1	15.82515258	8.658041	10.89556	10.91998	17.17747	16.1328
2023/7/2	17.27840326	8.612902	11.19126	10.69746	18.6138	16.98073
2023/7/3	16.5398043	8.590061	11.21764	10.9375	20.61525	16.55581
2023/7/4	21.36582018	7.721295	11.25008	10.72552	19.86796	17.39893
2023/7/5	16.47777027	8.131623	11.17864	9.877308	19.89314	16.52213
2023/7/6	18.53113733	7.558898	11.25342	10.63127	20.57508	17.73382
2023/7/7	18.63630888	8.578546	11.27439	10.69039	18.26043	17.79256

表格 6

品类补货量（公斤）

日期	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌类
2023/7/1	16.14550537	135.2836	32.19497	6.624434	60.30311	37.81344
2023/7/2	11.91073289	135.791	31.29189	7.109532	57.17191	34.88555

2023/7/3	14.06301026	136.0477	31.21131	6.586252	52.80875	36.3528
2023/7/4	0	145.8126	31.11225	7.048357	54.43786	33.4415
2023/7/5	14.24377745	141.2006	31.33043	8.897468	54.38295	36.46909
2023/7/6	8.260265831	147.638	31.10205	7.253835	52.89634	32.28511
2023/7/7	7.953795924	136.1771	31.03802	7.124951	57.94227	32.0823

表格 7

非折扣品类利润：

日期	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌类
2023/7/1	89.45126	456.9282	169.2752	20.1307	632.2352	414.0939
2023/7/2	48.68461	470.9223	118.261	23.18624	435.1779	352.445
2023/7/3	67.86826	478.0483	113.833	19.89852	190.5834	382.7181
2023/7/4	0	765.7137	108.4272	22.78877	277.8247	323.8723
2023/7/5	69.6243	625.6182	120.3775	36.31417	274.8063	385.1707
2023/7/6	23.41501	823.2551	107.8722	24.13661	195.1492	301.8617
2023/7/7	21.70999	481.6405	104.3949	23.28672	481.9984	298.0814

表格 8

根据 2020 年至 2023 年历史数据，折扣商品利润比例为d_1，品类折扣商品销量比例为d_2。

则折扣品类利润公式为：

$$R_d = \left(S \times \frac{d_1}{1-d_1}\right) \times (P_c \times d_2 - C_p)$$

计算出非折扣商品最大利润后，我们需要考虑打折销售商品对利润的影响。

根据不折扣商品利润和折扣商品利润可列出总利润计算公式：

$$R_2 = \sum_i^n R_{c_i} + \sum_i^n R_{d_i}$$

最终，计算得到利润表：

日期	2023/7/1	2023/7/2	2023/7/3	2023/7/4	2023/7/5	2023/7/6	2023/7/7
非折扣商品利润	1782.114	1448.677	1252.95	1498.626589	1511.911	1475.69	1411.112
折扣商品利润	19.03675	17.77718	7.246233	19.84523816	9.378062	11.64281	22.25089
总利润	1801.151	1466.454	1260.196	1518.471828	1521.289	1487.333	1433.363

表格 9

5.3 问题三模型的建立与求解

根据问题三的要求，我们首先筛选出 2023 年 6 月 24-30 日的可售品种，共计 61 个。为将售卖单品种数控制在 27-33 个之间，我们需对可售卖的蔬菜单品进行评分和排序，选出排名靠前的

销售品种。

题目要求我们尽可能满足市场需求，依据市场供需理论，我们将需求量转换为销售量，作为评分的一个因素。除此之外，收益最大化同样涉及到商品的利润与折扣比率，因此我们认为评判一个蔬菜是否适合售卖的主要因素包含销售量、销售利润以及折扣比率。为客观的评价上述影响因素，我们需使用熵权法对数据进行赋值，在得到权重后，使用 Topsis 法评价每个单品的得分，以此得出最佳组合。

5.3.1 熵权法计算权重

熵权法是一种多标准决策方法，旨在确定不同指标的权重，以便更好地反映各指标在评价过程中的重要性。熵权法较为客观，能够避免主观赋权造成的误差。

在求值之前，我们需对评价指标的数据进行正向化和归一化，保证数据的非负性：

$$z_{ij}=\frac{x_{ij}-x_{\min}}{x_{\max}-x_{\min}}$$

其中，x_{ij}为对应的评价指标数据，min和max分别代表数据中的最小值和最大值，z_{ij}表示处理完成的指标数据。

对于处理过后的数据z_{ij}而言，其对应的概率值为：

$$p_{ij}=\frac{z_{ij}}{\sum_{i=1}^{n}z_{ij}}$$

根据信息熵的定义，对于某一指标，可以用熵值来判断该指标的离散程度，信息熵值越小，指标的离散程度越大，该指标在评价中所占具的权重也就越大。如果求解第 j 个蔬菜的熵值e_j时，我们还需计算信息效用值$\hat{d}_j$，用以正向衡量信息量，计算公式如下：

$$e_j=-\frac{1}{\ln n}\sum_{i=1}^{n}p_{ij}\ln(p_{ij})$$

$$\hat{d}_j=1-e_j$$

最终对$\hat{d}_j$进行归一化处理，得到每个指标类型对应的权重值ω_j。

$$w_j=\frac{d_j}{\sum_{j=1}^{m}d_j}$$

相应的权重结果如下表所示：

指标	信息熵值	信息效用值	权重(%)
总利润	0.957	0.043	18.454
总销售量	0.827	0.173	74.334
折损商品占比	0.983	0.017	7.212

表格 10

5.3.2 Topsis 计算单品评分

TOPSIS 是一种多标准决策方法，用于根据指标的权重和商品的特征来评估商品的综合得分。该方法基于理想解和负理想解的概念，它们分别代表了最佳和最差的性能情况。商品的综合得分取决于其与理想解和负理想解之间的距离，以及权重。

找出每个指标中数据最大和最小的z_i值，构成向量Z^+和Z^-，分别代表了最优和最差的蔬菜指标数据。

$$Z^+ = \{z_1{}^+, z_2{}^+, ..., z_m{}^+\}$$

$$Z^- = \{z_1{}^-, z_2{}^-, ..., z_m{}^-\}$$

对于第 i 类蔬菜到最优数据值、最差数据值的距离分别记作$D_i{}^+$和$D_i{}^-$，计算公式如下：

$$D_i{}^+ = \sqrt{\sum_{j=1}^{m}(z_j{}^+ - z_{ij})}^2$$

$$D_i{}^- = \sqrt{\sum_{j=1}^{m}(z_j{}^- - z_{ij})}^2$$

对于第 i 类蔬菜的得分G_i由以下公式求得，由于$D_i{}^+$和$D_i{}^-$均为距离值，得分G_i的取值范围在[0，1]内，且G_i越大，$D_i{}^+$越小，离理想值越接近，蔬菜种类评分越高。

$$G_i = \frac{D_i{}^-}{D_i{}^+ - D_i{}^-}$$

最终评分和排名结果如下表，由于篇幅有限，我们仅保留了排名最前和最后的 10 个品种，所有数据见附录。

单品名称	排名	单品名称	排名
云南生菜(份)	1	虫草花(份)	41
小米椒(份)	2	红椒(2)	42
云南油麦菜(份)	3	云南生菜	43
西兰花	4	七彩椒(2)	44
芜湖青椒(1)	5	云南油麦菜	45
金针菇(盒)	6	洪湖藕带	46
竹叶菜	7	高瓜(1)	47
紫茄子(2)	8	白玉菇(袋)	48
螺丝椒(份)	9	高瓜(2)	49
娃娃菜	10	木耳菜(份)	50

表格 11

由此我们可以得到评分排名前 33 位的蔬菜单品用于制定 7 月 1 日单品的补货计划。

5.3.3 改进补货策略

探寻 33 个单品自身需求量与定价的关系，根据历史数据，我们确定了各单品的线性回归方差

题目要求我们求解利润最大化情况下的单品销量与定价，首先需要得到 7 月 1 日的各单品真实成本数据，与第二问中预测成本方式类似，我们采用 TCN 根据 6 月 24-30 日的成本数据对 7 月 1 日的各单品成本进行预测。

1. TCN-Attention 农产品单品成本预测模型模型

本模型和前文预测品类成本的模型一致，修改了时间窗口大小为 7-1，即使用过去 7 天的成本历史数据，预测未来一天一个特定农产品的成本。

使用 MinMaxScaler 进行数据归一化，通过训练和预测得到了 33 种农产品的成本价。

2. TCN-Attention Bootstrap 抽样时间序列售价范围预测模型

遗传算法具有贪心属性，其会在一切可能的价格区间中寻求利润最大化，为了使得模型最优化结果更合理,我们需要对价格进行 TCN-Attention Bootstrap 抽样时间序列模型预测。

该模型与问题二的 Bootstrap 模型一样，使用滑动窗口大小为 7-1，使用过去七天的历史售价预测未来一天的售价，33 种农产品每一种切分 100 个子数据集训练 100 个模型，预测获得其价格区间。

3. 遗传算法最优化模型

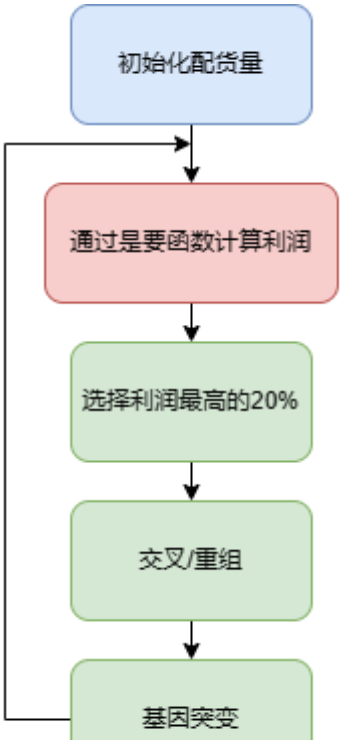
对于 33 种农产品，我们需要在限制条件下对齐实现收益最优化。我们使用遗传算法对其进行最优化处理。遗传算法是一种优化和选择算法，其与自然界中种群进化的过程类似。模型通过在不断迭代中启发式搜索，交叉，变异得到全局最优解。

限制条件

1. 定价在 Bootstrap sampling 预测出的价格区间内。
2. 定价和销量必须大于 0。
3. 如果补货量不为 0，至少等于 2.5。

运算过程

1. 模型首先初始化 100 组配货量矩阵，矩阵中的所有元素均为随机生成。
2. 根据前面线性回归模型得到的 33 个产品价格和销量的关系以及 TCN-Attention 预测出每一个产品当天的进货成本，适应函数通过计算 100 组配货量的总利润作为适应度。
3. 本模型设置了选择数量为 3 个，模型根据第二步计算的适应度选择适应度最优的三个配货计划作为下一迭代的亲本。
4. 对于选择出来的三个配货计划（染色体），本模型混合因子设置为 0.5 即意味着新的后代将“均匀”地从其两个亲本那里继承特质。这种交叉方式旨在找到一个介于两个亲本之间的解，这通常有助于维持种群的多样性，同时还能在一定程度上加速收敛。
5. 本模型自定义了高斯变异（Gaussian Mutation）作为变异函数。高斯变异（Gaussian Mutation）是一种用于连续或实数编码的基因（例如，浮点数）的变异方法。这种变异方法的核心思想是对选定的基因值添加一个从高斯分布



图表 14

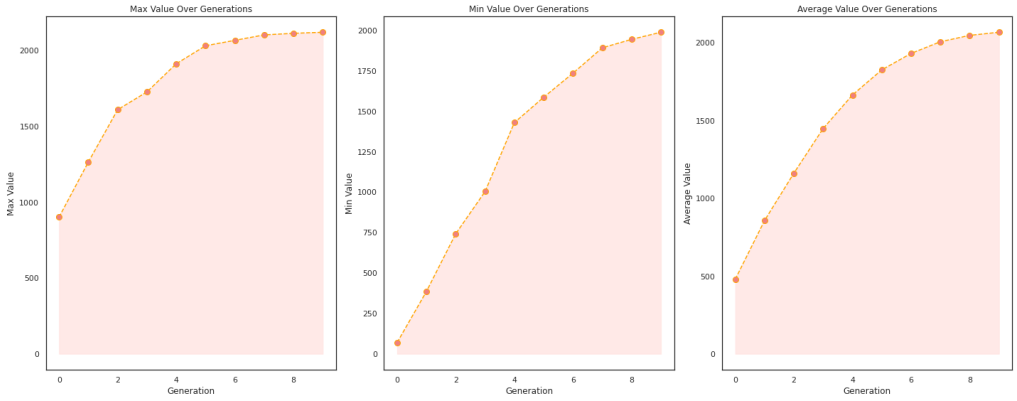
（也称为正态分布）抽取的随机数。

数学上高斯分布的概率密度定义如下：

$$f(x|\mu,\sigma^2)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

在高斯变异中，我们通常得到一个原始值x，然后从一个均值为μ和标准差为σ的高斯分布中取随机数δ，将其添加到原始值x上，以获得新的基因型。

6. 通过不断迭代得到最优化的结果，迭代过程如下图所示：



图表 15

33 种单品配货量及定价如下图所示：

单品	配货量	定价	单品	配货量	定价
云南生菜(份)	31.7901	18.20501	姜蒜小米椒组合装(小份)	9.673302	8.344492
小米椒(份)	6.959814	18.4944	海鲜菇(包)	6.708662	6.433055
云南油麦菜(份)	13.75808	6.805358	红薯尖	11.22093	2.3485
西兰花	21.89312	0.997583	上海青	10.81369	0.508674
芜湖青椒(1)	24.68926	19.53516	枝江青梗散花	9.057766	14.95397
金针菇(盒)	21.08613	2.746575	净藕(1)	15.84707	16.60993
竹叶菜	15.34623	3.852756	青茄子(1)	2.5	7.31
紫茄子(2)	12.71406	12.74369	长线茄	10.16517	0.412276
螺丝椒(份)	9.628669	4.219123	菱角	0	2.197
娃娃菜	8.633408	0.713981	木耳菜	2.967584	3.957403
苋菜	3.757023	9.851776	小青菜(1)	5.234773	6.809505
双孢菇(盒)	5.10158	7.101964	圆茄子(2)	0	4.759
螺丝椒	13.434	2.503405	菜心	3.809025	7.614189
小皱皮(份)	4.906435	8.190681	蟹味菇与白玉菇双拼(盒)	5.044017	6.906949
西峡花菇(1)	5.030443	23.93665	菠菜	6.863266	7.329177
奶白菜	13.03954	0.805665	外地茼蒿	6.943692	8.777288
			红莲藕带	7.794059	8.105957

表格 12

总利润：2118.6540782495063

5.4 问题四的分析

为更好地制定蔬菜商品和的补货和定价策略，商超应当引入更多影响销量和折损的因素，其中，天气是较为常见但又重要的变量，天气、气温和湿度的变化对销量和折损情况均有不同程度的影响。

就天气而言，如遇大雨、烈日等短期的天气情况时，人们出门买菜的意愿将大幅下降，因此，特殊天气下的销售量较往常将有所减少。如果商超仍按照日常销售量进行进货和补货，将会造成货物的堆积，易形成折损和滞销。如遇台风或持续高温、降雨等长期极端天气时，人们更需要一次性囤积足够量的蔬菜产品从而减少外出行为，因此，长期极端天气的前夕的蔬菜需求量将一定程度的上升，如若依照日常销量进行补货，易形成供不应求的局面，存在收益提升的空间。

气温和湿度共同影响蔬菜的折损情况，温度过高容易滋生微生物，造成腐败，严重缩短储存时间；如若温度过低，容易形成冷害，使表面出现凹点、水浸状斑点。湿度过低容易造成蔬菜细胞水分流失、脱水枯萎，影响品质与口感；湿度过高则会为病菌繁殖提供温床，造成蔬菜的腐烂。为减少折损率、提高营收，商超需通过分析不同温度、气温条件下的菜品保存情况，通过风扇、加湿器等工具将储存环境调整至最佳存放状态。预测天气现象带来的销售量变化、气候条件造成的折损情况有助于商超及时调整供求和价格，从而增加额外收益、减少非必要损失。

六、 模型的评价、改进与推广

6.1 模型的优点

1. 在对进货成本的预测中考虑了时间序列因素，引入了不同季节的因素，将日期根据24 节气对蔬菜作物的影响考虑在内。
2. 熵权法能客观地反映各指标在评价过程中的重要性，避免主观赋权造成的误差。
3. Topsis 法能够同时研究多个因素，很好的刻画多个影响指标的综合影响力。

6.2 模型的缺点

1. 最终问中着重最大利润，弱化了满足需求的要求/或算法本身的问题
2. Topsis 的指标选取无法确定合适的个数，需凭借个人分析和评判选取关键指标。

七、 参考文献

[1] 毛莉莎. 供应链视角下蔬菜批发市场定价策略及产销模式研究[D]. 中南林业科技大学, 2023.
[2] 席念楚. 储存蔬菜没那么简单[N]. 中国家庭报, 2021-11-18(013)
[3] 罗海梅. 微观经济学“供求”概念的辨析[J]. 农村经济与科技, 2020, 31(24):108-109.
[4] 李娜. 我国蔬菜价格波动的影响因素研究[D]. 山东农业大学, 2014.
[5] 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.
[6] 李洋, 董红斌. 基于 CNN 和 BiLSTM 网络特征融合的文本情感分析[J]. 计算机应用, 2018, 38(11): 3075-3080.
[7] Hewage P, Behera A, Trovati M, et al. 基于时间卷积神经网络的天气预报方法研究[J]。软计算，2020,24:16453-16482。
[8] 基于时间卷积神经网络的天气预报方法研究[J]。软计算，2020,24:16453-16482。
[9] 葛继科, et al. "遗传算法研究综述." 计算机应用研究 25.10 (2008): 2911-2916.
[10] 段玉倩, 贺家李. 遗传算法及其改进[J]. 电力系统及其自动化学报, 1998, 10(1): 39-52.

附录

数据筛选预处理与其正态分布可视化

```
def Normal_distribution_visulable(ax, DataFrame, Product_name):
    translator = Translator()
    translated_name = translator.translate(Product_name, src='zh-CN', dest='en').text

    filtered_1 = DataFrame[DataFrame['单品名称'] == Product_name]
    mean, std = filtered_1['批发单价'].mean(), filtered_1['批发单价'].std()
    xmin, xmax = filtered_1['批发单价'].min(), filtered_1['批发单价'].max()
    x = np.linspace(xmin, xmax, 100)

    sns.histplot(filtered_1['批发单价'], kde=False, bins=30, ax=ax, label='Data')
    ax.plot(x, norm.pdf(x, mean, std) * 1000, 'k', linewidth=2, label='Fit')
    ax.axvline(mean, color='r', linestyle='--', linewidth=2, label='Mean')
    ax.axvline(mean + 3*std, color='orange', linestyle='--', linewidth=2, label='+3 Std Dev')
    ax.axvline(mean - 3*std, color='orange', linestyle='--', linewidth=2, label='-3 Std Dev')
    ax.set_title(translated_name)
    ax.set_xlabel('Value')
    ax.set_ylabel('Frequency')
    ax.legend()

def Show_all_Normal_distribution(Dataframe, Length, Width):
    Number = Length * Width
    unique_values = Dataframe['单品名称'].unique()
    random_plt_name = random.sample(list(unique_values), Number)

    fig, axes = plt.subplots(Length, Width, figsize=(4*Length, 3*Width))

```

```
    for ax, name in zip(axes.flatten(), random_plt_name):
        Normal_distribution_visulable(ax, Dataframe, name)

    plt.tight_layout()
    plt.show()
```

肘部法确定最优聚类中心算法

```
# 使用 seaborn 设置美观的图表样式
sns.set(style="whitegrid")

# 读入数据
Cluster_prodect_raw_data = pd.read_excel("副本t1 聚类(2).xlsx", engine='openpyxl')

# WCSS 列表
wcss = []

# 尝试不同数量的聚类中心
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(Cluster_prodect_raw_data)
    wcss.append(kmeans.inertia_)

# 绘制肘部法图
plt.figure(figsize=(10, 6))  # 设置图表大小
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')  # 添加点和线
max_wcss = max(wcss)
plt.fill_between(range(1, 11), max_wcss, wcss, color='skyblue', alpha=0.8)  # 填充天蓝色透明区域在曲线上方

plt.title('Elbow Method', fontsize=16)  # 设置标题和字体大小
plt.xlabel('Number of clusters', fontsize=14)  # 设置 x 轴标签和字体大小
plt.ylabel('WCSS', fontsize=14)  # 设置 y 轴标签和字体大小
plt.xticks(fontsize=12)  # 设置 x 轴刻度字体大小
```

```
    plt.yticks(fontsize=12)  # 设置 y 轴刻度字体大小

    plt.show()
```

蔬菜品类之间斯皮尔曼相关性分析及数据可视化

```
    df = Saled_KG_according_category_data

    # 使用Google Translate API翻译列名
    translator = Translator()
    translated_columns = [translator.translate(column, src='zh-cn', dest='en').text for column in df.columns]
    df.columns = translated_columns

    # 计算斯皮尔曼相关性
    spearman_corr = df.corr(method='spearman')

    # 创建热力图
    plt.figure(figsize=(10, 8))  # 设置图形大小
    sns.set(style="white")  # 设置风格为白色背景
    sns.heatmap(spearman_corr, annot=True, cmap="coolwarm", square=True, linewidths=.5, cbar_kws={"shrink": .75})
    plt.title('Spearman Correlation Heatmap')
    plt.show()
```

利润率-销量线性回归及可视化算法

```
    # 计算利润率和销售量之间的关系
    # 创建一个2x3的subplot，高为2，宽为3
    fig, axs = plt.subplots(2, 3, figsize=(15, 10))

    # 使用一个字典来存储所有的DataFrame，这样我们可以使用一个循环来绘制所有的图
    data_dict = {
        'Root Stock': Profit_KG_stat_root_stock,
        'Mosaic Leaf Class': Profit_KG_stat_Mosaic_leaf_class,
        'Flower Vegetables': Profit_KG_stat_flower_vegetables,
        'Solanaceous': Profit_KG_stat_solanaceous,
```

```
    'Chili Peppers': Profit_KG_stat_Chili_peppers,
    'Mushrooms': Profit_KG_stat_mushroomss
}

best_fit_equations = {}  # 用于存储最佳拟合公式

# 循环通过字典绘制每个图
for i, (key, df) in enumerate(data_dict.items()):
    ax = axs[i//3, i%3]

    # 提取数据
    x = df.iloc[:, 3].values
    y = df.iloc[:, 1].values

    # 创建散点图
    ax.scatter(x, y, label='Data Points')

    min_mse = float('inf')
    best_degree = 0
    best_p = None

    # 尝试从 1 到 6 阶多项式
    for degree in range(0,2):
        z = np.polyfit(x, y, degree)
        p = np.poly1d(z)
        mse = mean_squared_error(y, p(x))

        if mse < min_mse:
            min_mse = mse
            best_degree = degree
            best_p = p

    # 绘制最佳拟合线
    xp = np.linspace(min(x), max(x), 100)
    ax.plot(xp, best_p(xp), '--',color="r", label=f'Best Fit (Degree {best_degree})')

    best_fit_equations[key] = best_p

    # 设置图的其他元素
    ax.set_title(f'{key} (Best Degree: {best_degree})')
```

```
        ax.set_xlabel('Profit')
        ax.set_ylabel('Sell')
        ax.legend()

    plt.tight_layout()
    plt.show()

    # 单独输出最佳拟合公式
    print("Best fit equations:")
    for key, equation in best_fit_equations.items():
        print(f"{key}: {equation}")
```

售价-销量线性回归算法及其可视化

```
    # 创建一个2x3的subplot，高为2，宽为3
    fig, axs = plt.subplots(2, 3, figsize=(15, 10))

    # 使用一个字典来存储所有的DataFrame，这样我们可以使用一个循环来绘制所有的图
    data_dict = {
        'Root Stock': Profit_KG_stat_root_stock,
        'Mosaic Leaf Class': Profit_KG_stat_Mosaic_leaf_class,
        'Flower Vegetables': Profit_KG_stat_flower_vegetables,
        'Solanaceous': Profit_KG_stat_solanaceous,
        'Chili Peppers': Profit_KG_stat_Chili_peppers,
        'Mushrooms': Profit_KG_stat_mushroomss
    }

    best_fit_equations = {}  # 用于存储最佳拟合公式

    # 循环通过字典绘制每个图
    for i, (key, df) in enumerate(data_dict.items()):
        ax = axs[i//3, i%3]

        # 提取数据
        x = df.iloc[:, 0].values  # 假设第一列是利润率
```

```
    22.       y = df.iloc[:, 1].values  # 假设第二列是销
售量
    23.
    24.       # 创建散点图
    25.       ax.scatter(x, y, label='Data Points')
    26.
    27.       min_mse = float('inf')
    28.       best_degree = 0
    29.       best_p = None
    30.
    31.       # 尝试从 1 到 6 阶多项式
    32.       for degree in range(0, 2):
    33.           z = np.polyfit(x, y, degree)
    34.           p = np.poly1d(z)
    35.           mse = mean_squared_error(y, p(x))
    36.
    37.           if mse < min_mse:
    38.               min_mse = mse
    39.               best_degree = degree
    40.               best_p = p
    41.
    42.       # 绘制最佳拟合线
    43.       xp = np.linspace(min(x), max(x), 100)
    44.       ax.plot(xp, best_p(xp), '--
',color="r", label=f'Best Fit (Degree {best_degree})')
    45.
    46.       best_fit_equations[key] = best_p
    47.
    48.       # 设置图的其他元素
    49.       ax.set_title(f'{key} (Best Degree: {best_d
egree})')
    50.       ax.set_xlabel('Selling price')
    51.       ax.set_ylabel('Sell')
    52.       ax.legend()
    53.
    54.   plt.tight_layout()
    55.   plt.show()
    56.
    57.   # 单独输出最佳拟合公式
    58.   print("Best fit equations:")
    59.   for key, equation in best_fit_equations.items(
):
    60.       print(f"{key}: {equation}")
```

日期-24 节气转换算法

```
# 二十四节气及其对应的月和日
solar_terms = [
    ("立春", (2, 4)), ("雨水", (2, 19)),
    ("惊蛰", (3, 5)), ("春分", (3, 20)),
    ("清明", (4, 4)), ("谷雨", (4, 20)),
    ("立夏", (5, 5)), ("小满", (5, 21)),
    ("芒种", (6, 5)), ("夏至", (6, 21)),
    ("小暑", (7, 6)), ("大暑", (7, 22)),
    ("立秋", (8, 7)), ("处暑", (8, 23)),
    ("白露", (9, 7)), ("秋分", (9, 22)),
    ("寒露", (10, 8)), ("霜降", (10, 23)),
    ("立冬", (11, 7)), ("小雪", (11, 22)),
    ("大雪", (12, 6)), ("冬至", (12, 21)),
    ("小寒", (1, 5)), ("大寒", (1, 20))
]

# 季节和节气的映射
season_to_terms = {
    '春': ["立春", "雨水", "惊蛰", "春分", "清明", "谷雨"],
    '夏': ["立夏", "小满", "芒种", "夏至", "小暑", "大暑"],
    '秋': ["立秋", "处暑", "白露", "秋分", "寒露", "霜降"],
    '冬': ["立冬", "小雪", "大雪", "冬至", "小寒", "大寒"]
}

def find_previous_solar_term(month, day):
    for i in reversed(range(len(solar_terms))):
        term, (term_month, term_day) = solar_terms[i]
        if (month, day) >= (term_month, term_day):
            return term
    return None

def encode_solar_term(month, day):
    term = find_previous_solar_term(month, day)
    if term is None:
```

```
            return [0]*10

        for season, terms in season_to_terms.items():
            if term in terms:
                season_encoding = [1 if s == season else 0 for s in ['春', '夏', '秋', '冬']]
                term_encoding = [1 if t == term else 0 for t in terms]
                return season_encoding + term_encoding
        return [0]*10
```

滑动窗口算法

```
def slide_windows(Dataframe,window_size):
    processed_df = pd.DataFrame()
    Total_size = window_size+7
    # 滑动窗口
    for i in range(len(Dataframe) - 22):
        past_15_days_cost = Dataframe.iloc[:, 2].iloc[i:i+window_size].values
        future_7_days_solar_term = np.array(Dataframe.iloc[:, 4].iloc[i+window_size:i+Total_size].tolist())

        row_data = {
            '过去时间窗口成本': list(past_15_days_cost),
            '未来七天节气': future_7_days_solar_term.reshape(-1).tolist(),  # 将 7x10 的二维数组转换为 1D 数组
            '未来七天成本': list(Dataframe.iloc[:, 2].iloc[i+window_size:i+Total_size].values)
        }
        processed_df = processed_df.append(row_data, ignore_index=True)
    return processed_df
```

数据预处理-神经网络训练

```
def deal_X2(Dataframe):
    list_test = []
```

```
    for temp in np.array(Dataframe['未来七天节气']):
        for i in temp:
            list_test.append(i)
    X2 = np.array(list_test).reshape(-1, 70, 1)
    return X2

def df_to_nparray(df_col):
    return np.array([np.array(x) for x in df_col])

def Data_process_defore_train(Dataframe):
    # 获取数据集总长度
    n = len(Dataframe)
    # 计算测试集、验证集和训练集的大小
    test_size = int(n * 0.2)
    val_size = int(n * 0.8 * 0.25)
    train_size = n - test_size - val_size

    # 分离自变量和因变量
    X = Dataframe[["未来七天节气", "过去时间窗口成本"]]
    y = Dataframe["未来七天成本"]

    # 按照顺序切分数据集
    X_train = X.iloc[:train_size]
    y_train = y.iloc[:train_size]

    X_val = X.iloc[train_size:train_size+val_size]
    y_val = y.iloc[train_size:train_size+val_size]

    X_test = X.iloc[train_size+val_size:]
    y_test = y.iloc[train_size+val_size:]

    # 打印各个数据集的大小
    print("Training set:", X_train.shape, y_train.shape)
    print("Validation set:", X_val.shape, y_val.shape)
    print("Test set:", X_test.shape, y_test.shape)
```

```
    # 确保 X_train, y_train, X_val, y_val, X_test, y_test 都已正确加载并预处理
    X1_train = df_to_nparray(X_train['过去时间窗口成本']).reshape(-1, 15, 1)
    X2_train = deal_X2(X_train)
    y_train_array = df_to_nparray(y_train).reshape(-1, 7)

    X1_val = df_to_nparray(X_val['过去时间窗口成本']).reshape(-1, 15, 1)
    X2_val = deal_X2(X_val)

    y_val_array = df_to_nparray(y_val).reshape(-1, 7)

    X1_test = df_to_nparray(X_test['过去时间窗口成本']).reshape(-1, 15, 1)
    X2_test = deal_X2(X_test)

    y_test_array = df_to_nparray(y_test).reshape(-1, 7)
    return {"X1 训练集":X1_train,
            "X2 训练集":X2_train,
            "X1 验证集":X1_val,
            "X2 验证集":X2_val,
            "X1 测试集":X1_test,
            "X2 测试集":X2_test,
            "Y 训练集":y_train_array,
            "Y 验证集":y_val_array,
            "Y 测试集":y_test_array}
```

特征融合神经网络

```
# 注意力机制层
class AttentionLayer(layers.Layer):
    def __init__(self):
        super(AttentionLayer, self).__init__()

    def build(self, input_shape):
        self.W = self.add_weight(shape=(input_shape[-1], input_shape[-1]), initializer='random_normal', trainable=True)
```

```
            self.b = self.add_weight(shape=(input_shape[-1],), initializer='zeros', trainable=True)

    def call(self, inputs):

            q = tf.nn.tanh(tf.linalg.matmul(inputs, self.W) + self.b)
            a = tf.nn.softmax(q, axis=1)
            output = tf.reduce_sum(a * inputs, axis=1)
            return output

class Feature_fusion_TCN_model(Model):
    def __init__(self):
            super(Feature_fusion_TCN_model, self).__init__()

            self.tcn1 = TCN()
            self.tcn2 = TCN()

            self.concat = layers.Concatenate(axis=1)
            self.attention_layer = AttentionLayer()

            self.reshape = layers.Reshape((1, -1))
            self.output_layer = Dense(7, activation='linear')

    def call(self, inputs):
            input1, input2 = inputs

            tcn1_output = self.tcn1(input1)
            tcn2_output = self.tcn2(input2)

            merged = self.concat([tcn1_output, tcn2_output])

            attention_output = self.attention_layer(merged)
            attention_output_expanded = self.reshape(attention_output)

```

```
        output = self.output_layer(attention_output_expanded)

        return output
```

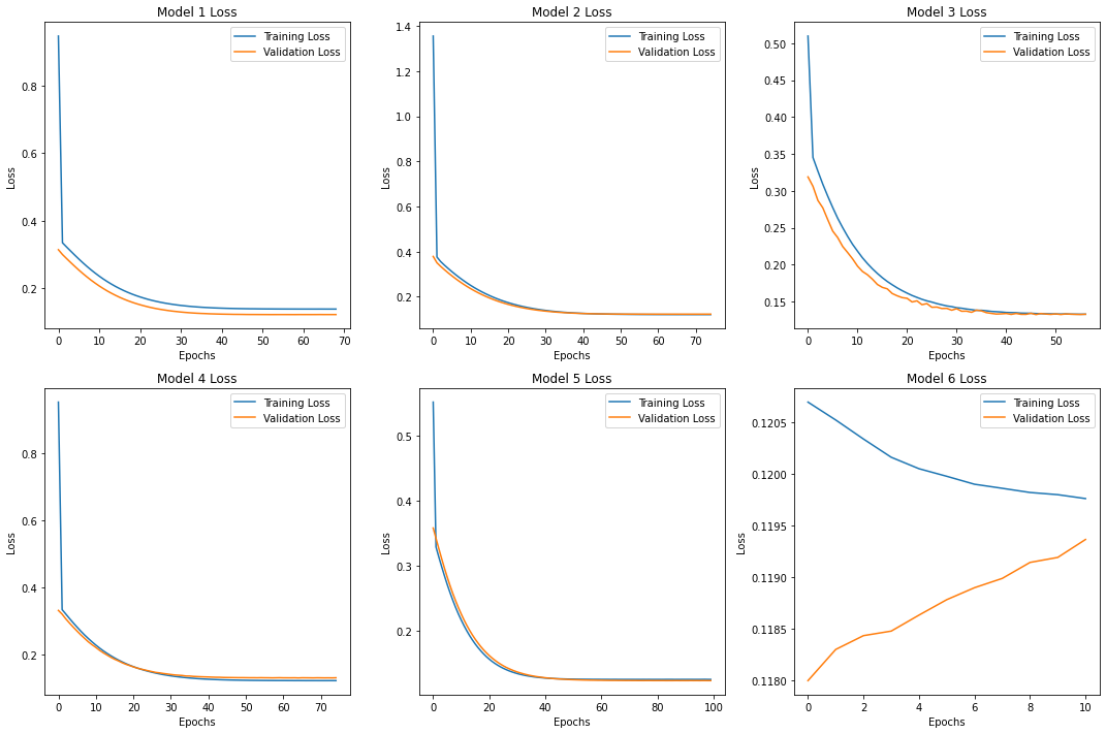
BFGS 最优化

```
prediction_functions = [Root_stock_prediction, Mosaic_Leaf_prediction, Flower_Vegetables_prediction, Solanaceous_prediction, Chili_Peppers_prediction, Mushrooms_prediction]

def objective(prices, costs, prediction_functions):
    revenue = 0
    for i in range(len(prices)):
        quantity = prediction_functions[i](prices[i])
        if quantity < 0:
            quantity = 0
        revenue += (prices[i] - costs[i]) * quantity
    return -revenue

mean_plus_2sigma_values = {
    '水生根茎类': 28.79,
    '花叶类': 234.41,
    '花菜类': 39.24,
    '茄类': 41.08,
    '辣椒类': 147.06,
    '食用菌': 102.06
}
def optimization_price(Dataframe, mean_plus_2sigma_values):
    rows_price = []
    rows_quantity = []
    rows_profit = []
    daily_optimal_revenues = []  # 用于存储每天的最优利润

    for index in range(len(Dataframe)):
        costs = Dataframe.iloc[index].values
        initial_prices = costs * 1.5
```

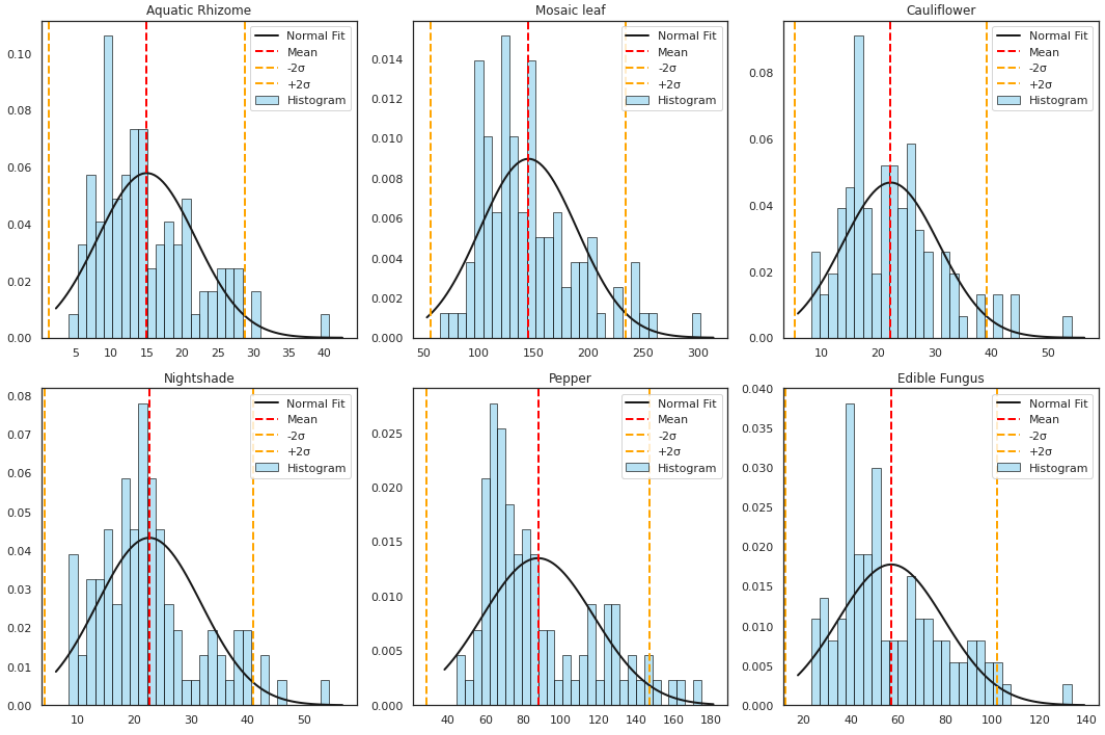
```
            max_prices = [mean_plus_2sigma_values[product] for product in Dataframe.columns]

            constraints = [
                {'type': 'ineq', 'fun': lambda prices, costs=costs: prices - costs},
                {'type': 'ineq', 'fun': lambda prices, max_prices=max_prices: max_prices - prices},
                {'type': 'ineq', 'fun': lambda prices: [prediction_functions[i](prices[i]) for i in range(len(prices))]}
            ]

            result = minimize(objective, initial_prices, args=(costs, prediction_functions), constraints=constraints)

            if result.success:
                optimized_prices = result.x
                rows_price.append(optimized_prices)

                # 计算当天每个品类的销量和利润
                quantities = [prediction_functions[i](optimized_prices[i]) for i in range(len(optimized_prices))]
                profits = [(optimized_prices[i] - costs[i]) * quantities[i] for i in range(len(optimized_prices))]

                rows_quantity.append(quantities)
                rows_profit.append(profits)

                # 计算当天的最优利润（注意：我们最小化了目标函数，所以取负数得到最优利润）
                daily_optimal_revenues.append(-result.fun)

        Result_price_dataframe = pd.DataFrame(rows_price, columns=Dataframe.columns)
        Result_quantity_dataframe = pd.DataFrame(rows_quantity, columns=Dataframe.columns).applymap(lambda x: 0 if np.abs(x) < 1e-10 else x)
```

```
    Result_profit_dataframe = pd.DataFrame(rows_profit, columns=Dataframe.columns).applymap(lambda x: 0 if np.abs(x) < 1e-10 else x)

    # 使用 Seaborn 设置更漂亮的风格
    sns.set(style="whitegrid")

    # 可视化每天的最优利润
    plt.figure(figsize=(8, 7))
    plt.bar(range(1, len(daily_optimal_revenues) + 1), daily_optimal_revenues, color='skyblue', edgecolor='black', alpha=0.6)
    plt.title("Daily Optimal Revenue")
    plt.xlabel("Day")
    plt.ylabel("Optimal Revenue")
    plt.xticks(range(1, len(daily_optimal_revenues) + 1))
    plt.show()

    return Result_price_dataframe, Result_quantity_dataframe, Result_profit_dataframe, daily_optimal_revenues

 # 使用你已经有的 Data_predicted_result
 OPT_price = optimization_price(Data_predicted_result, mean_plus_2sigma_values)
 OPT_price
```

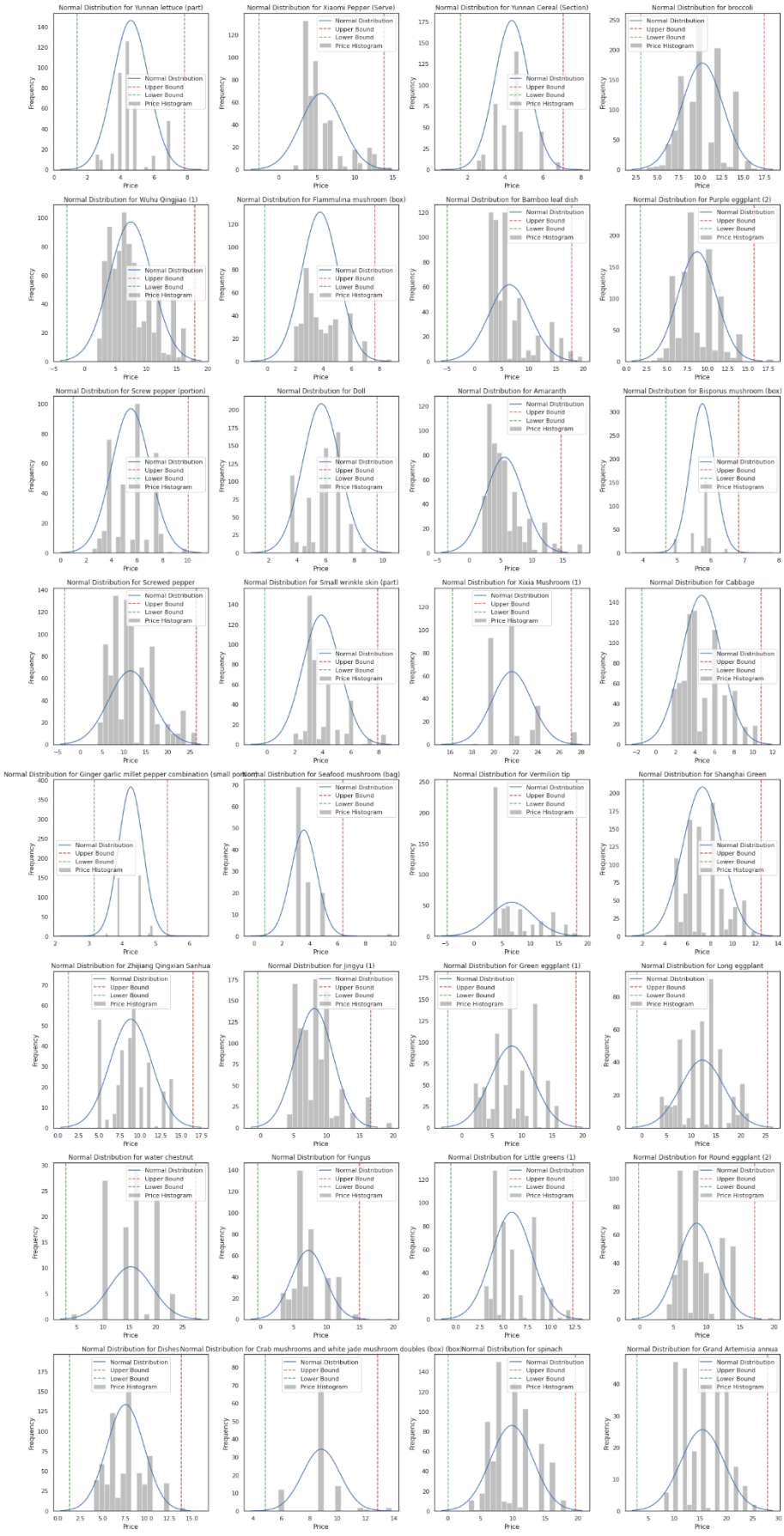
特征融合模型训练过程



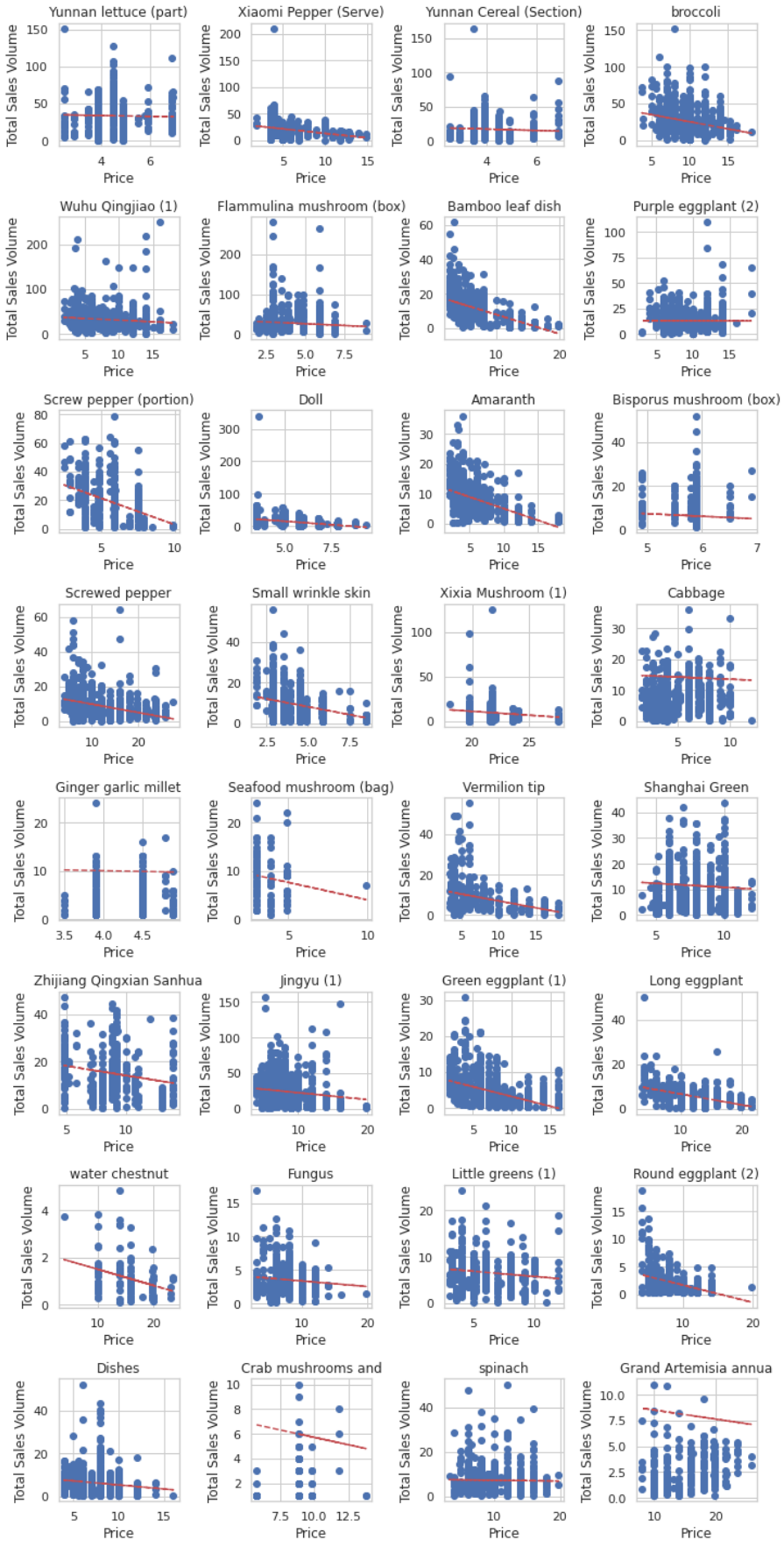
品类 Bootstrap sampling



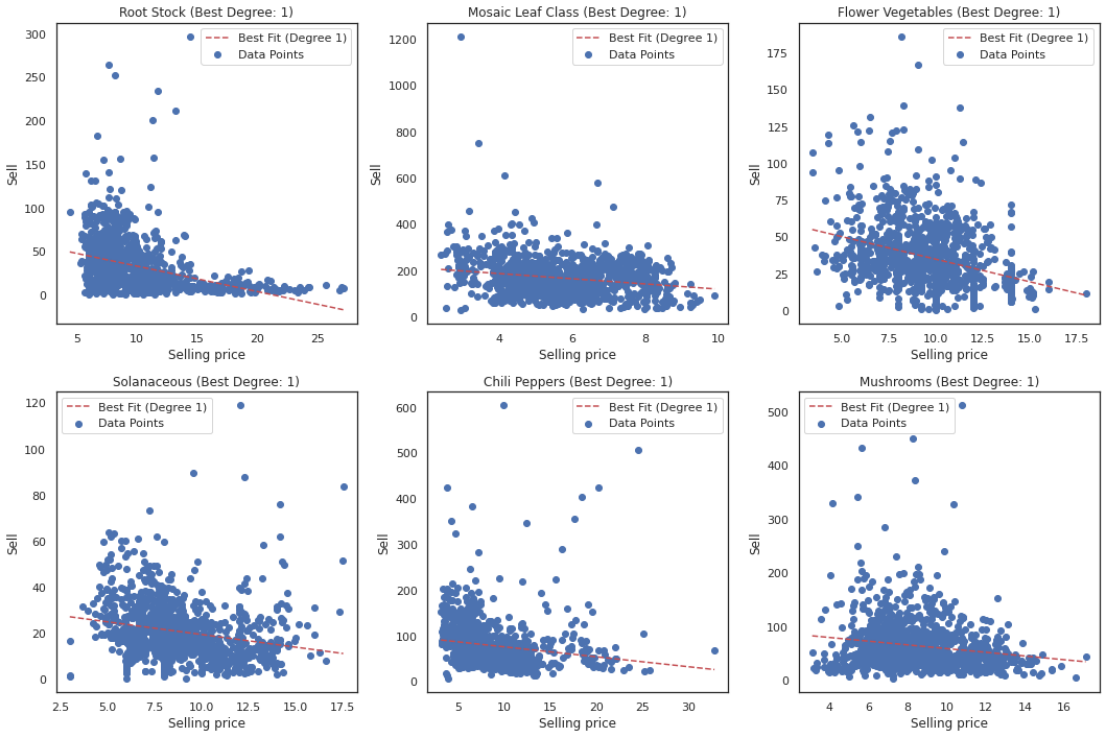
33 品类 Bootstrap sampling 分布



售价-单品销量线性关系图



售价-品类销量线性回归图像



利润率-品类销量线性回归图