优化生鲜商超收益：蔬菜商品的销售规律与补货决策分析

摘　要

商超（超市和零售店）在现代经济中扮演着至关重要的角色，然而，它们在蔬菜商品管理中面临着多重挑战。这些挑战包括如何准确预测销售趋势、合理制定价格策略、以及有效制定补货计划等问题。解决这些问题对于商超来说至关重要，因为它们直接影响着销售收益、库存成本和客户满意度。因此，本研究旨在为商超提供一套全面的蔬菜商品管理策略，以帮助它们更好地应对这些挑战。

针对问题一，在蔬菜商品管理中，首要问题之一是如何准确预测销售趋势。这包括了不同蔬菜品类的销售模式，如季节性销售高峰和低谷。我们需要深入了解哪些蔬菜在特定时间段内销售最活跃，以及它们之间的差异。这个问题的解决有助于商超更有针对性地制定促销策略和补货计划。

针对问题二，制定合理的价格策略对于商超至关重要，因为它们需要平衡销售利润和客户价格敏感度。我们需要建立一个定价模型，考虑商品成本、预期销售量和销售利润等因素。这个模型将为每个蔬菜品类提供合理的售价建议，确保商超实现销售利润的最大化，同时提供具有竞争力的价格。

针对问题三，如何确定每个单品的补货量以及建议的定价策略是另一个重要问题。我们需要通过组合优化方法，确定每个单品的最佳补货量和定价策略，以确保商超在未来某一天实现最佳盈利。这需要考虑预期利润、最小陈列量要求和其他约束条件。

针对问题四，更多的数据可以帮助商超更准确地预测市场需求，优化库存，制定更有效的定价策略，从而最大化收益。我们建议采集的数据包括：库存数据、竞争对手的价格数据、客户反馈、销售促销和广告活动的数据、季节性和趋势数据、供应链数据、宏观经济数据、商品的损耗数据、客户购买行为数据等。通过结合这些数据，商超可以更好地理解市场，满足客户需求，优化供应链，减少浪费，提高客户满意度，并最大化收益。

通过解决这些问题，我们的研究旨在为商超提供一套完整的蔬菜商品管理策略，以帮助它们提高销售效益、降低库存成本，并提高客户满意度。

关键词：线性回归，相关性分析，时间序列 arama,混合整数规划

一、 背景和问题重述

在生鲜商超中，蔬菜类商品的保鲜期短，品相随着销售时间的增加而下降，大部分如果当日未售出，隔日将无法再售。因此，商超每天会根据商品的历史销售和需求情况进行补货。但是，由于商超销售的蔬菜品种多样、产地不同，且进货交易时间通常在凌晨 3:00-4:00，商家需要在不清楚具体品种和进货价格的情况下做出补货决策。同时，蔬菜的定价一般采用“成本加成定价”方法，商超会对品质下降的商品进行打折销售。

问题重述

1. **问题 1:** 分析蔬菜各品类及单品销售量的分布规律及它们之间的相互关系。

2. **问题 2:** 考虑商超以品类为单位进行补货计划，分析各蔬菜品类的销售总量与成本加成定价的关系，并为 2023 年 7 月 1-7 日给出每日的补货总量和定价策略，目标是最大化商超收益。

3. **问题 3:** 考虑蔬菜销售空间有限制，商超想要制定单品的补货计划，要求总的可售单品数在 27-33 个之间，且每个单品的订购量满足最小陈列量 2.5 千克。根据 2023 年 6 月 24-30 日的可售品种，为 7 月 1 日提供单品补货量和定价策略，目标是在满足市场需求的前提下，最大化商超收益。

4. **问题 4:** 为了更好地制定补货和定价决策，商超还需要收集哪些相关数据，这些数据如何帮助解决上述问题。

2.1 问题一的问题分析

为了分析蔬菜各品类及单品销售量的分布规律及相互关系，我们需要将这两个数据集合并，以便在一个数据框中查看每个蔬菜单品的销售情况和其所属的品类。然后，我们使用各种统计方法和可视化工具来进行了分析：首先将按品类对销售量进行总结，并可视化销售量的分布，其次计算每个单品的销售量，并进行可视化展示，然后通过时间序列分析各单品随时间变化的销售趋势，以确定哪些商品在特定时间段内的销售增长或下降和通过计算两两商品间的销售相关性，确定哪些商品经常一起购买。最终得出规律和相互关系

2.2 问题二的分析

根据题目和背景可知商超采用的是“成本加成定价”方法。由问题一可知蔬菜的供应品种在 4 月至 10 月较为丰富，而商超的销售空间有限。为了最大化收益，需要考虑的因素包括：蔬菜的进货成本、预期的销售量、市场需求、损耗率等。我们采用了线性回归方法预测未来一周的销售，然后基于预测的销售量和成本数据制定了定价策略。

2.2 问题三的分析

考虑到销售空间的限制，商超希望进一步制定单品的补货计划。需要确保可售单品总数控制在 27-33 个，且每个单品的订购量至少满足最小陈列量 2.5 千克的要求。我们使用了整个数据集的平均销售量作为预测值。使用贪心算法，我们选择了预期利润最高的 27-33 个单品，并为这些单品制定了定价策略。

2.2 问题四的分析

更多的数据可以帮助商超更准确地预测市场需求，优化库存，制定更有效的定价策略，从而最大化收益。我们建议采集的数据包括：库存数据、竞争对手的价格

数据、客户反馈、销售促销和广告活动的数据、季节性和趋势数据、供应链数据、宏观经济数据、商品的损耗数据、客户购买行为数据等。通过结合这些数据，商超可以更好地理解市场，满足客户需求，优化供应链，减少浪费，提高客户满意度，并最大化收益。

三 、 模型假设

为了对问题一进行数学建模和分析，我们需要做出一些基本的假设。以下是一些可能的问题假设：

1. **历史销售数据能够代表未来的销售趋势**：我们使用历史销售数据来预测未来的销售。
2. **蔬菜的损耗是固定的**：我们使用平均损耗率来计算蔬菜的实际成本。
3. **商品的销售量与其定价有关**：我们基于预测的销售量来制定定价策略。
4. **商超的销售空间是固定的**：我们假设商超的销售空间在一段时间内是恒定的。
5. **所有的商品都有相同的陈列空间**：每个商品的最小陈列量都是 2.5 千克。
6. **趋势假设**：我们假设销售数据中可能存在长期的上升或下降趋势，这些趋势可以通过时间序列分解来识别。
7. **数据完整性假设**：提供的销售数据被视为准确和完整，不包含任何误差或遗漏。
8. **稳定性假设**：我们假设在观察期间，市场条件（例如消费者购买力、偏好等）保持相对稳定，这使得时间序列分析变得可行。
9. **季节性假设**：蔬菜销售可能受到季节性影响，例如由于供应量、天气条件或节假日等原因，某些蔬菜在特定时期的销售量可能会增加或减少。
10. **独立性假设**：除非通过数据明确显示出相关性，否则我们假设各个单品或品类之间的销售是独立的。

四、定义与符号说明

符号说明：

- S：商品的销售量。
- C：商品的进货成本。
- P：商品的售价。
- L：商品的损耗率。
- M：商品的加价率。
- R：商品的补货量。
- $\hat{S}$：商品的预测销售量。

5.1 问题一的模型建立与求解

由于附件 1 提供了蔬菜的单品编码、单品名称、分类编码和分类名称，而附件 2 提供了销售日期、销售时间、单品编码、销售量、销售单价、销售类型以及是否打折销售的信息。首先将附件 1 和附件 2 进行合并，具体如下所示：

销售日期	扫码销售时间	单品编码	销量	销售单价	销售类型	是否打折销售	单品名称	分类编码	分类名称
2020-7-1	09:15:07.924	1.029E+14	0.396	7.6	销售	否	泡泡椒(精品)	1011010504	辣椒类
2020-7-1	09:17:27.295	1.029E+14	0.849	3.2	销售	否	大白菜	1011010101	花叶类
2020-7-1	09:17:33.905	1.029E+14	0.409	7.6	销售	否	泡泡椒(精品)	1011010504	辣椒类
2020-7-1	09:19:45.450	1.029E+14	0.421	10	销售	否	上海青	1011010101	花叶类
2020-7-1	09:20:23.686	1.029E+14	0.539	8	销售	否	菜心	1011010101	花叶类
2020-7-1	09:21:55.556	1.029E+14	0.277	7.6	销售	否	泡泡椒(精品)	1011010504	辣椒类
2020-7-1	09:21:56.536	1.029E+14	0.338	8	销售	否	云南生菜	1011010101	花叶类
2020-7-1	09:22:01.274	1.029E+14	0.132	7.6	销售	否	泡泡椒(精品)	1011010504	辣椒类

2020-7-1	09:22:01.476	1.029E+14	0.213	8	销售	否	云南生菜	1011010101	花叶类
2020-7-1	09:22:15.998	1.029E+14	0.514	8	销售	否	甜白菜	1011010101	花叶类
2020-7-1	09:22:21.264	1.029E+14	0.251	10	销售	否	高瓜(1)	1011010402	水生根茎类
2020-7-1	09:24:21.833	1.029E+14	0.251	6	销售	否	云南油麦菜	1011010101	花叶类
2020-7-1	09:24:21.905	1.029E+14	0.217	18	销售	否	西峡香菇(1)	1011010801	食用菌
2020-7-1	09:24:57.873	1.029E+14	0.468	6	销售	否	云南油麦菜	1011010101	花叶类

数据预处理：

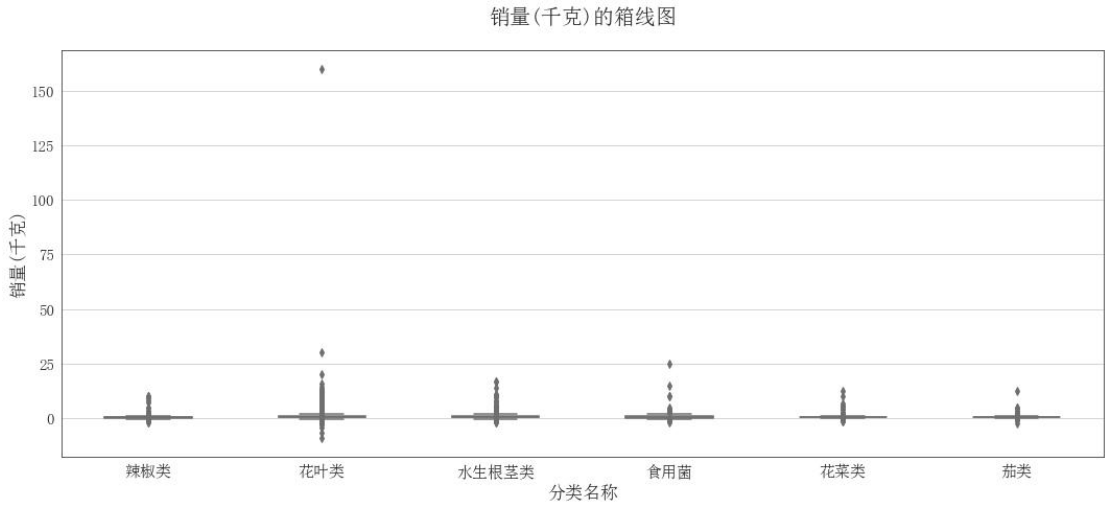
对附件 1 和附件 2 的数据进行了合并之后，我们将对数据进行清洗和预处理，然后进行特征选择和转换，以便建立模型。

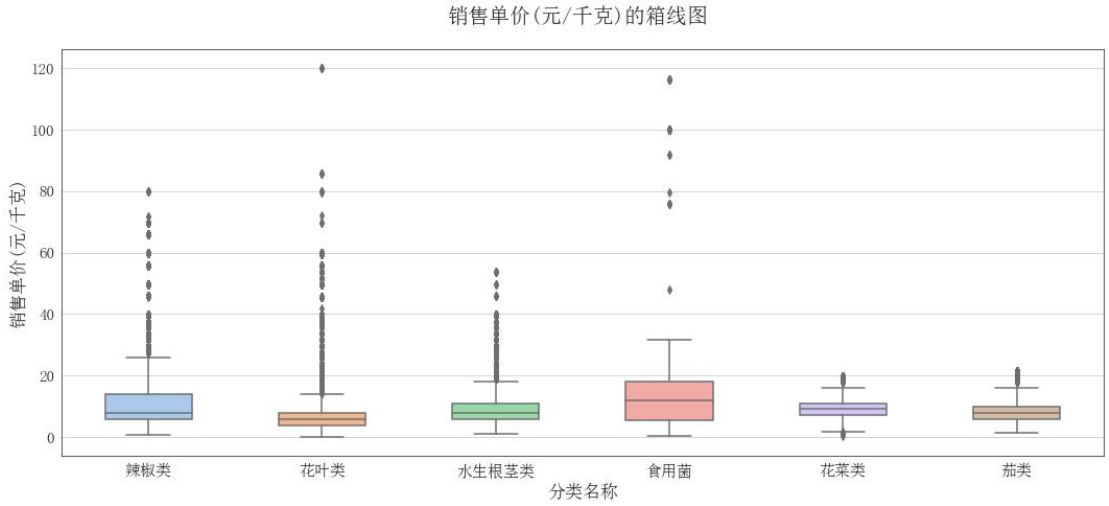
箱型图（Boxplot）是一种可视化数据分布和异常值特征的图表。它将数据分为四分位数（Q1、Q2、Q3）和四分位距（IQR=Q3-Q1）等几个数值。（在代码中，Q1 代表排在前 25% 的数据所对应的数值，Q2 表示排序之后的中间值（亦即中位数），Q3 则对应着排在前 75% 的数据所对应的数值。）

箱型图包含一个箱子其中，箱子的上下端分别表示数据集的上下四分位数，中线表示中位数，箱子的长度代表数据的分布区间，离群点则代表数据中的异常值。箱型图通过观察箱子的长度和箱子外部的离群点数量，可以快速判断数据是否包含异常值，并了解数据的偏斜程度和分布范围。

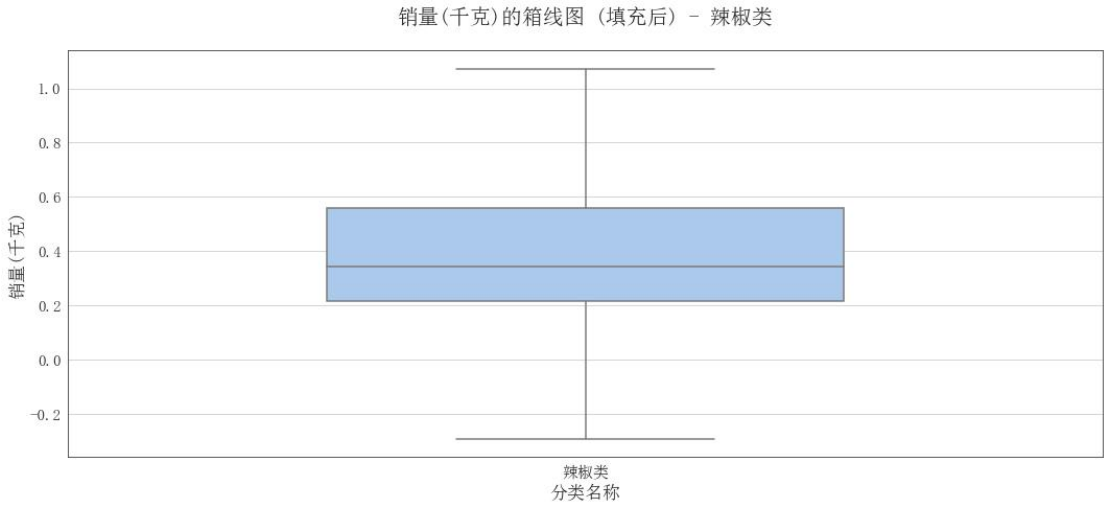
在本代码中，通过 matplotlib 库中的 boxplot（）函数绘制每个列数据对应的箱线图，并根据图箱线图的结果计算每列数据集的异常值数量。根据箱型图原理，将数据中小于 Q1-1.5×IQR 或大于 Q3+1.5×IQR 的数据定义为异常值，这些数据需要被过滤掉，只保留合法的数据用于后续分析。最后，使用过滤后的数据再绘制一次箱型图并保存到输出目录中。通过这种方式，能够自动化地进行异常值处理，降低了处理数据时的手工干预，而且能够更加客观地识别数据的离群值，并准确计算异常值的数量。下图展示所以特征的原始数据箱型图和异常值处理后的数据箱型图

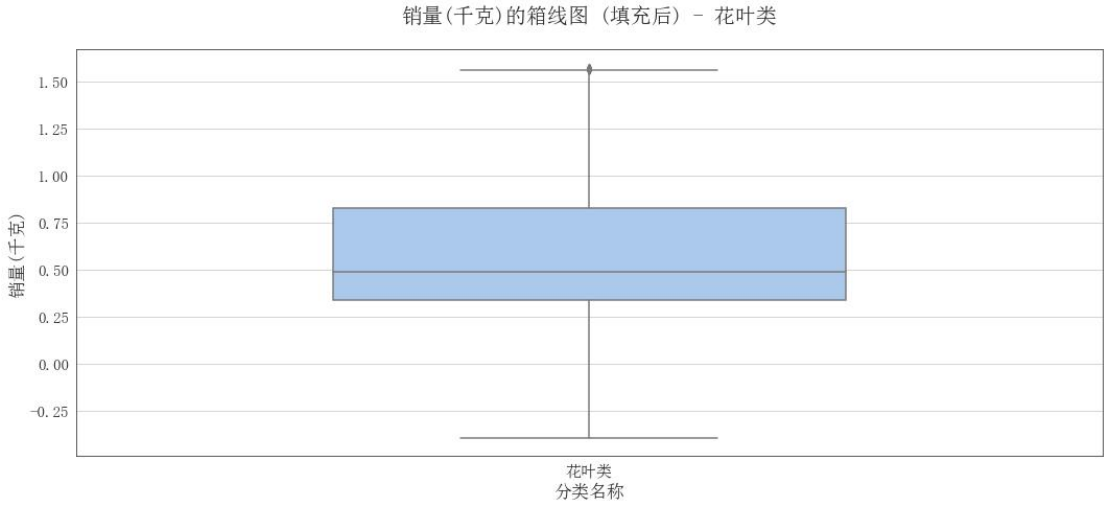
首先绘制了六个蔬菜品类的箱线图销量和价格的箱线图，如下图所示：



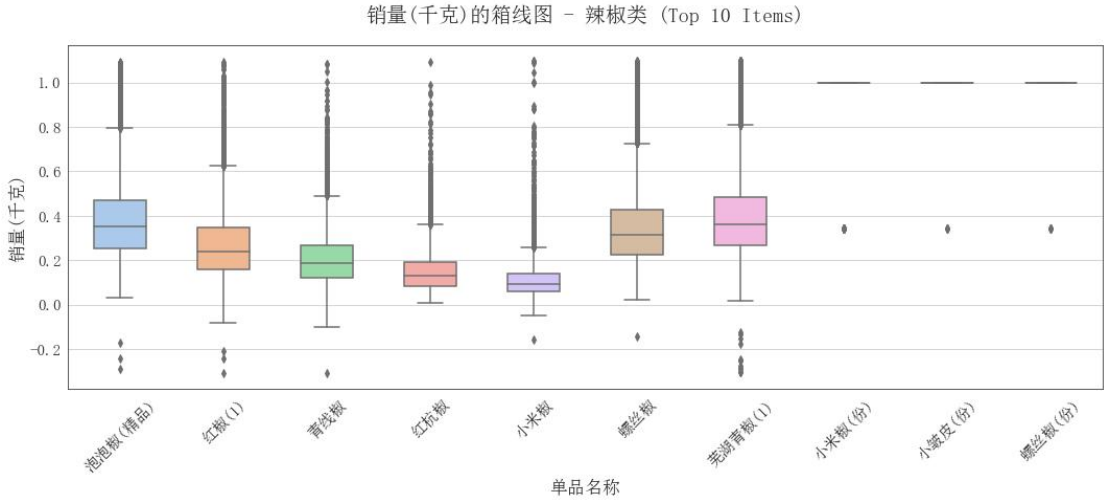


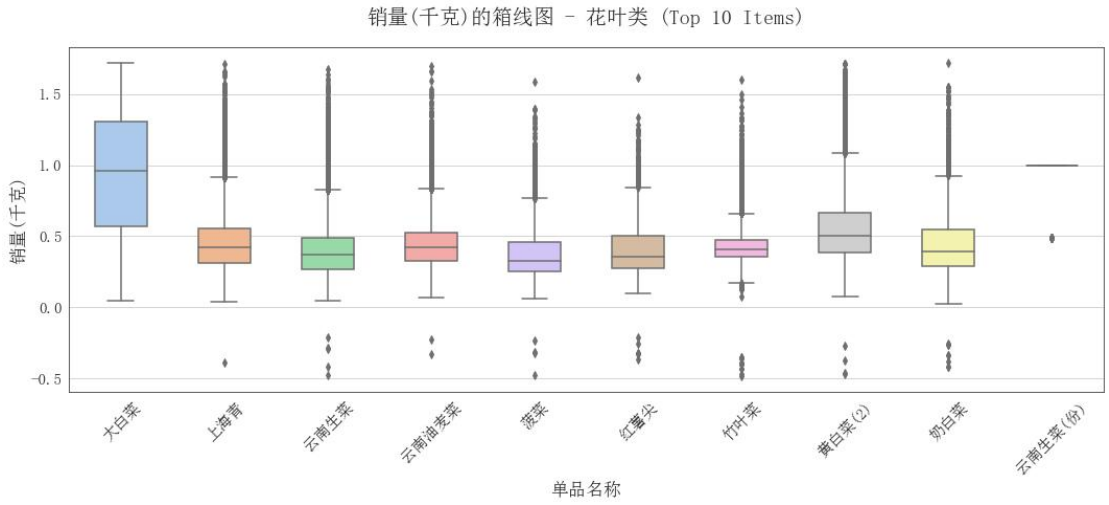
通过上述图表，可知只有部分数据中存在异常值，将其剔除后发现存在不同特征之间缺失值数量的差异（），在这种情况下中，我们经过对数据集的仔细分析后决定使用 KNNImputer 类对缺失值进行填补。该类利用基于 KNN 算法的机器学习技术来推算缺失值，使用“邻居”之间的相似性来推算相关的缺失值。我们选择将 KNN 算法中的 n_neighbors 参数设置为 5，即对于每个缺失值，我们将使用最近的 5 个非空值进行填充。KNN 算法利用与自身相似的 K 个最近邻的非缺失样本来推算缺失值，同时通过考虑其他特征的分布与目标特征之间的关系，提高了缺失值填充的可靠性和准确性。因此，相比传统的固定值填补、均值填补等方法，我们认为使用 KNN 算法进行缺失值填充更加可靠和准确。
填补结果展示如下：（以销量辣椒类和花叶类为例）

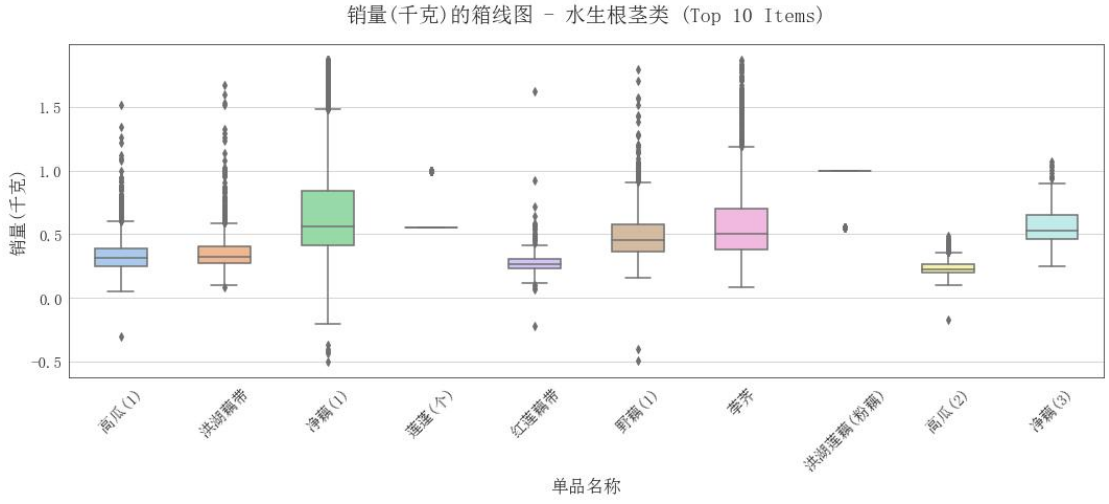


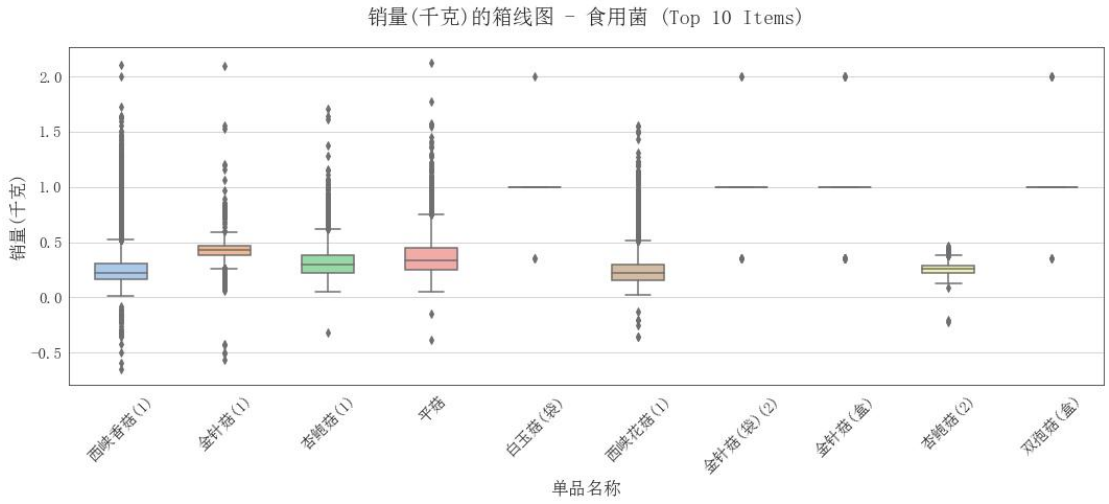


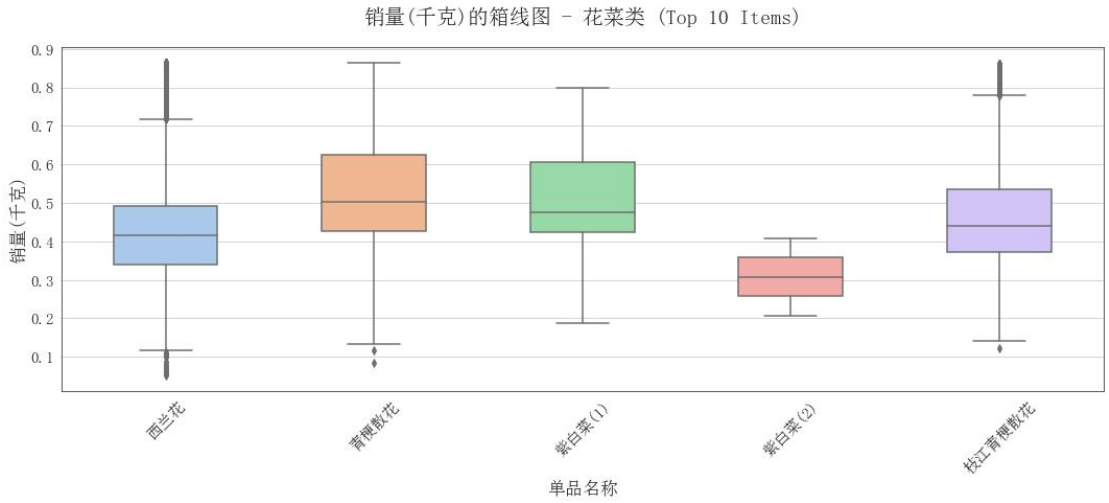
进而展示六个品类各自所属的单品，如下所示，(所属多的话展示前十个)，以销量为例，价格放在附录

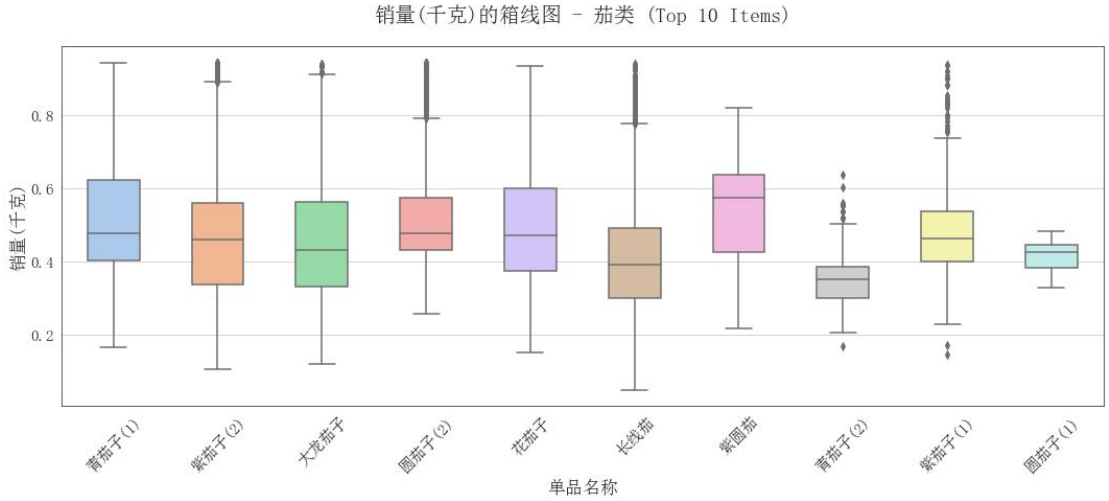



销量(千克)的箱线图 - 花叶类 (Top 10 Items)
销量(千克)
1.5
1.0
0.5
0.0
-0.5
大白菜
上海青
云南生菜
云南油麦菜
菠菜
红苋尖
竹叶菜
黄白菜(2)
奶白菜
云南生菜(份)
单品名称


销量(千克)的箱线图 - 水生根茎类 (Top 10 Items)
销量(千克)
1.5
1.0
0.5
0.0
-0.5
高瓜(1)
洪湖藕带
净藕(1)
莲蓬(个)
红莲藕带
鲜藕(1)
李芳
洪湖莲藕(粉藕)
高瓜(2)
净藕(3)
单品名称

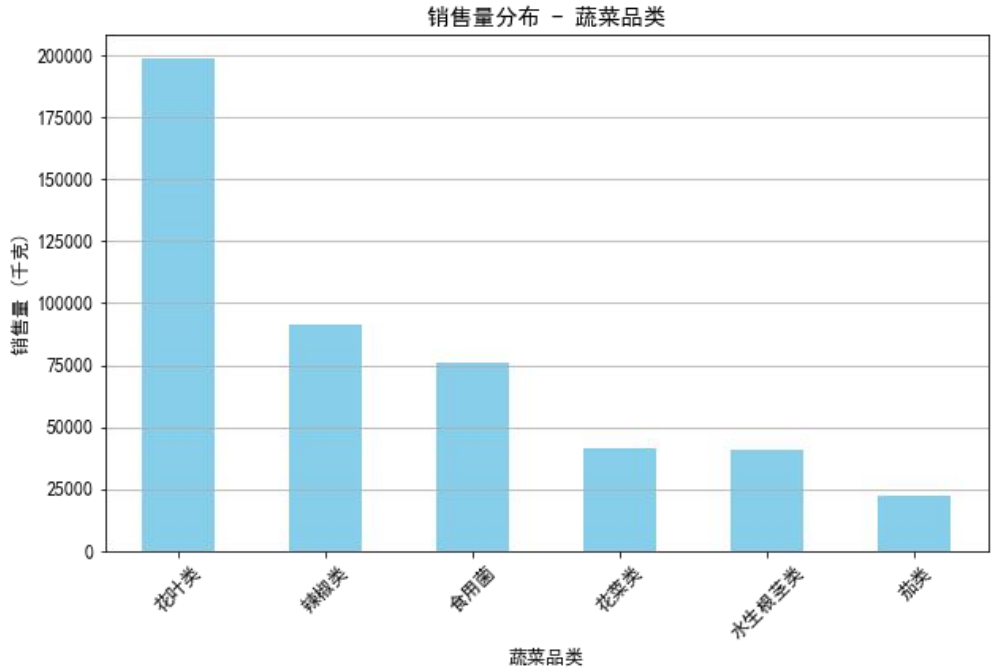

销量(千克)的箱线图 - 食用菌 (Top 10 Items)
销量(千克)
2.0
1.5
1.0
0.5
0.0
-0.5
面味香菇(1)
金针菇(1)
杏鲍菇(1)
平菇
白玉菇(袋)
面味花菇(1)
金针菇(袋)(2)
金针菇(盒)
杏鲍菇(2)
双孢菇(盒)
单品名称



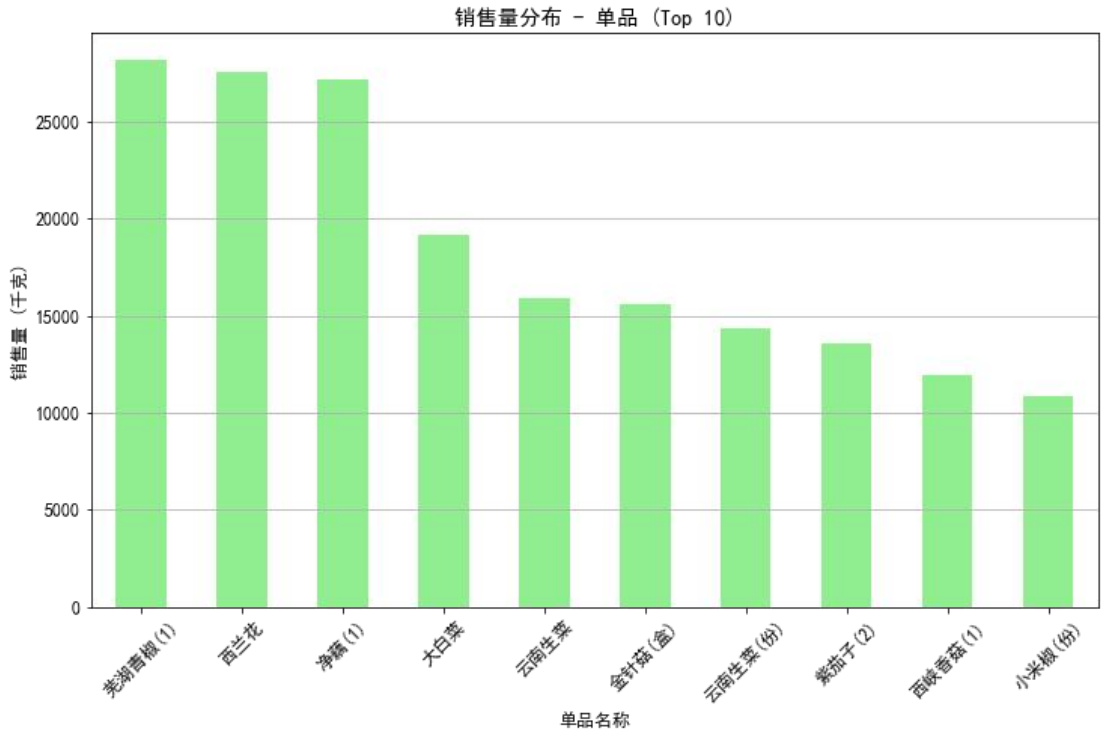


由上图可知，当我们在大类别上进行异常值处理后，小类别中仍然存在异常值，这可能是由以下原因导致的：

1. **数据的异质性**：每个大类别下的小类别可能具有不同的数据分布。处理大类别时，我们可能考虑到了整体的数据分布，而忽略了小类别中的特殊情况。
2. **不同的规模和分布**：小类别可能有其独特的销售规模和价格范围，这可能导致它们在大类别处理后仍然存在异常值。
3. **大类别的处理可能太过宽泛**：当我们在大类别级别上处理数据时，可能会错过一些细节，因为我们是基于整体数据分布进行处理的，而不是基于每个小类别的特点。
4. **小类别的数据不够稳定**：确实，如果某个小类别的数据量较小或其数据波动较大，那么这个小类别的数据可能会相对不稳定。这种不稳定性可能会导致数据中存在更多的异常值。

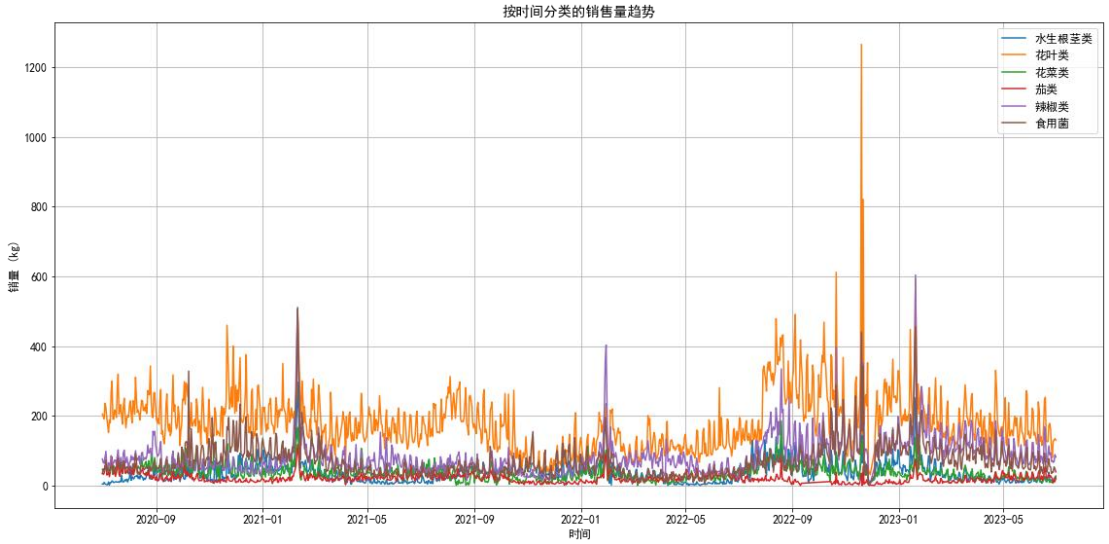


由上图可知，按品类销售量看：'花叶类'品类的销售量最高，其次是'辣椒类'和'食用菌'。



由上图可知，按商品分类的销售量看，芜湖青椒（1），西兰花，和净藕（1）是销售量排在前三的产品。

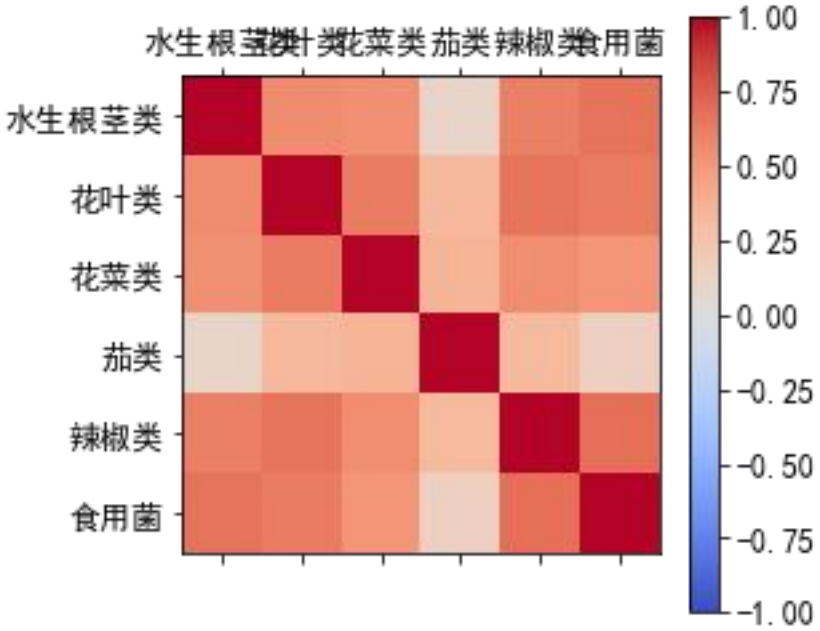
接下来，我们将分析销售量随时间的变化情况，以检查是否存在周期性或趋势。我们将首先聚焦于整体的品类销售量，然后再细分到单品销售量。



从上述图表中，我们可以观察到以下情况：

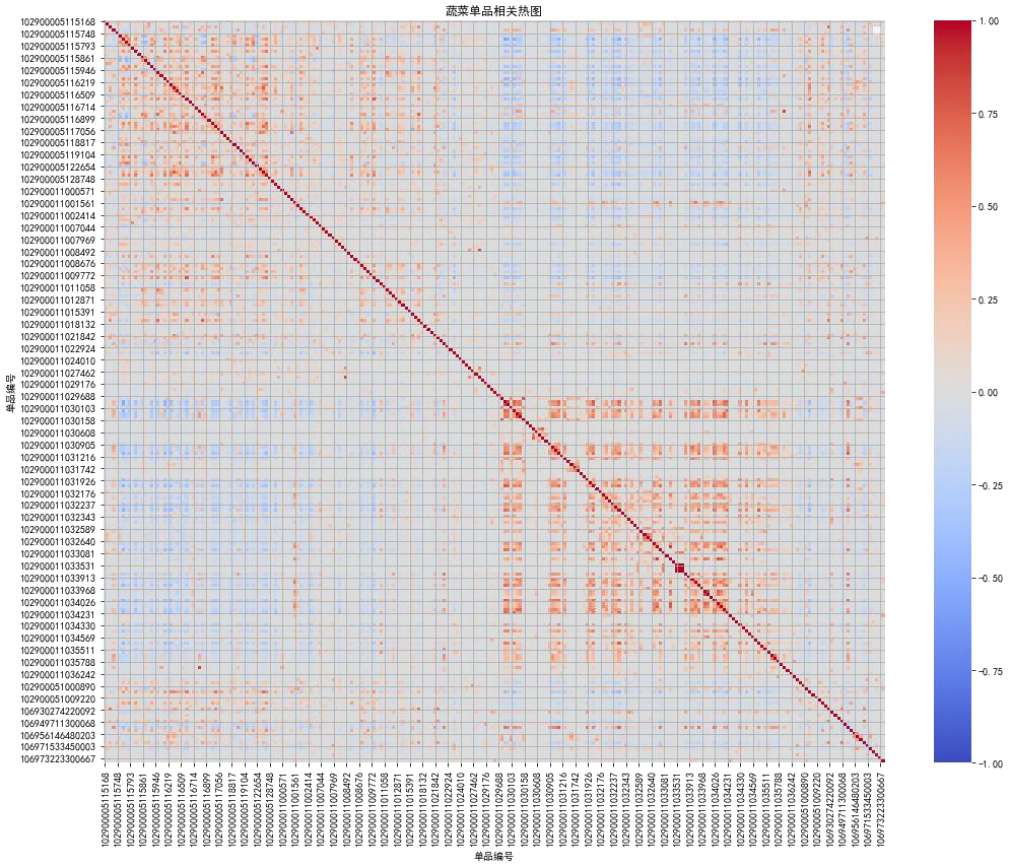
1. **销售量时间趋势**：各个品类的销售量在一年中都有其高峰和低谷，显示出了明显的季节性。
2. **花叶类**：这个品类的销售量在每年的特定时段内都达到高峰，可能与某些节日或季节性事件有关。

接下来，我们将使用相关性分析来探索不同蔬菜品类之间的关联关系。为此，我们将计算品类销售量之间的皮尔逊相关系数。这将帮助我们了解哪些品类的销售趋势是相关的，即当一个品类的销售量增加时，另一个品类的销售量也可能增加。



1. **高正相关性**：例如，“果类”和“根茎类”之间存在较高的正相关性。这意味着当“果类”的销售量增加时，“根茎类”的销售量也可能增加，反之亦然。
2. **低或无相关性**：例如，“花叶类”与其他大部分品类之间的相关性较低，这意味着它们的销售模式可能是独立的。

同时得到单品销量之间的相关系数热力图，如下图所示：



然后我们找出相关性大于 0.7 的单品对，结果如下：

	Item1	Item2	Correlation
48	102900011033562	102900011033531	1.0
46	102900011033531	102900011033562	1.0
51	102900011033586	102900011033562	1.0
49	102900011033562	102900011033586	1.0
50	102900011033586	102900011033531	1.0

由上图可知，这几组单品组是在相同时间和数量上呗购买，因此我们推测他们是某种组合或套餐的一部分。

这些关联性分析结果可以为商超提供关于如何组合销售策略和促销活动的洞察，以便最大化销售和收益。

最后通过 apriori 算法进行验证

1. 支持度（support）：商品组合在所有交易中出现的频率。
2. 置信度（confidence）：如果购买了商品 A，那么也购买商品 B 的概率。

3. 提升度（lift）：商品 A 和商品 B 一起出售的概率，与它们各自独立销售的概率相比。
我们将设置一个较低的支持度阈值，以捕获更多的商品组合，然后根据置信度和提升度对结果进行筛选。
结果与上表格相同，说明我们这几组单品对是某些组合的部分。

问题 2 的模型建立与分析
问题 2 要求分析蔬菜品类的销售总量与成本加成定价的关系，并为未来一周提供补货和定价策略。为此，我们首先需要加载并查看附件 3 和附件 4 的内容。
附件 3 为我们提供了不同日期和单品的批发价格信息，而附件 4 为我们提供了不同小分类的平均损耗率。
为了分析销售总量与成本加成定价的关系并为未来一周制定补货和定价策略，我们需要考虑以下因素：
1. **销售预测**：我们可以使用过去的销售数据来预测未来一周的销售量。
线性回归模型**
线性回归是一种统计方法，用于建立一个或多个自变量和因变量之间的关系模型。模型的形式为：

线性回归是一种统计方法，用于建立一个或多个自变量和因变量之间的关系模型。模型的形式为：

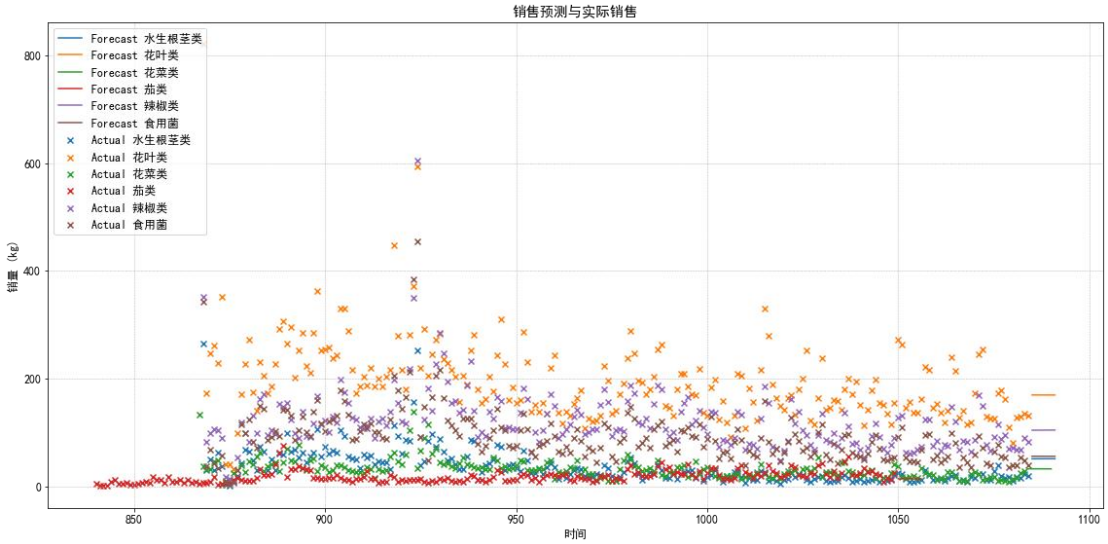
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

其中：

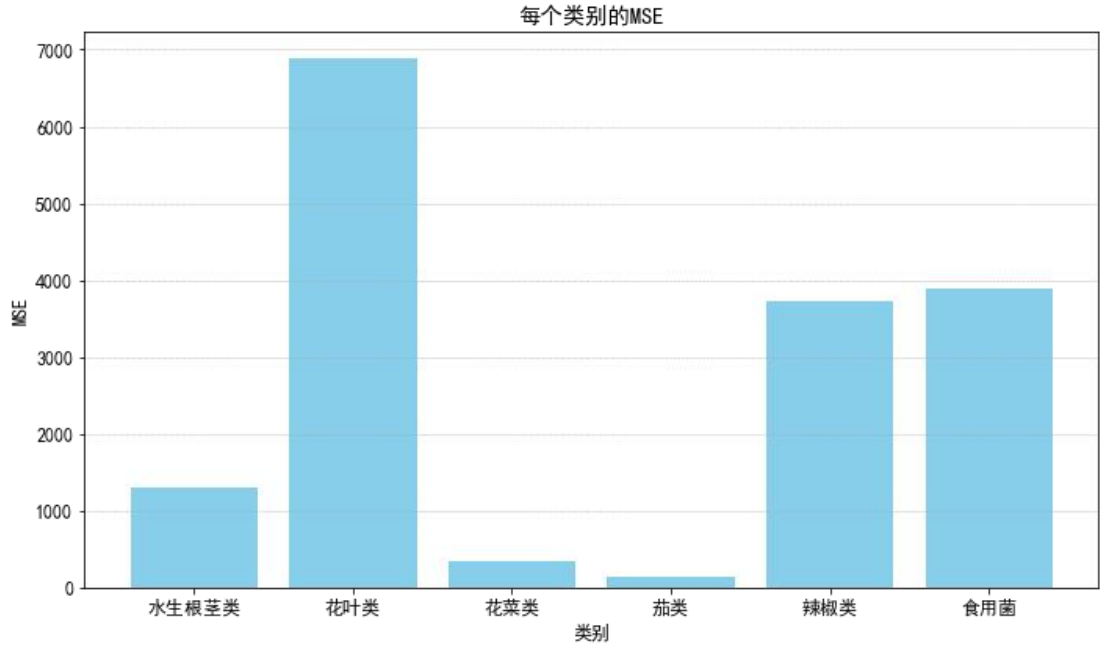
- Y 是因变量（我们要预测的销售量）。
- $X_1, X_2, ..., X_p$ 是自变量（影响销售量的因素，如价格、历史销售量等）。
- $\beta_0, \beta_1, ..., \beta_p$ 是模型的参数。
- ϵ 是误差项。

3.2 特征选择
为了简化模型并避免过拟合，我们选择了与销售量最相关的几个特征作为自变量。
4. 参数估计
使用最小二乘法，我们估计了模型的参数。这些参数代表了自变量与销售量之间的关系强度。

5. 模型验证
我们使用了训练集和测试集来验证模型的有效性。具体结果如下图：



通过计算预测值与实际值之间的均方误差（MSE），我们评估了模型的准确性。如下图所示：

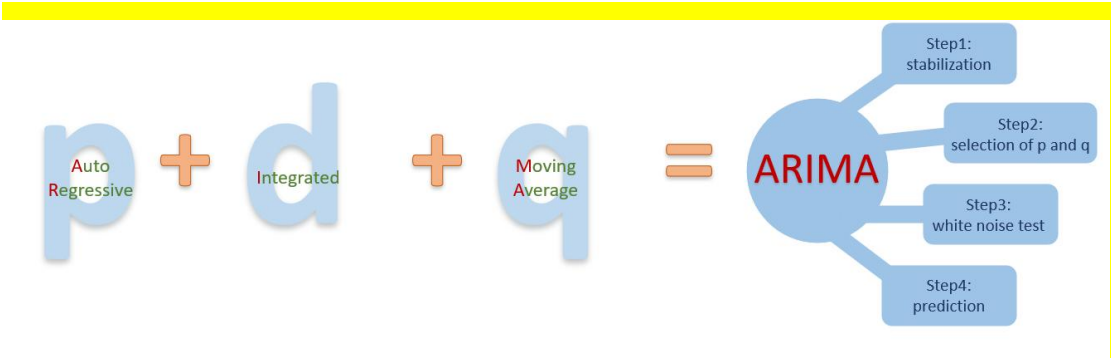


由上图可知，简单的线性回归进行预测对还是存在一定的误差，但在范围内。由于价格由季节的特征，可以采用时间序列模型进行预测。

可以使用时间序列进行预测分析

时间序列：

自回归整数移动平均值，即 ARIMA，是一种使用时间序列数据预测未来趋势的统计分析模型。ARIMA 的基本思想是，随着时间的推移，由预测形成的数据序列被视为随机序列，可以使用模型来近似描述该序列。一旦确定了这个序列，模型就可以根据时间序列的过去和现在的值来预测未来的值。在这个模型中，我们试图仅根据截至当月的单价数据来预测商品的未来单价。



ARIMA 模型包括自回归（AR）模型和移动平均（MA）模型。AR 模型描述了当前值和滞后值之间的关系，并利用历史数据预测了未来值。MA 模型利用过去残差项的线性组合来观察未来残差。ARIMA 预测模型可写成以下公式：

$$\hat{p}^{\{t\}} = p_0 + \sum_{j=1}^{p} \gamma_j p^{\{t-j\}} + \sum_{j=1}^{q} \theta_j \varepsilon^{\{t-j\}}$$

这里p 是自回归模型（AR）的阶数，q 是移动平均模型（AM）的阶数，ε $\{t\}$ 是时间之间的误差项，范围在t 和t -1 之间，γj 和θ j 是拟合系数，$p0$ 是常数项。

图（Figure6：原数据序列，一阶差分，二阶差分）

ACF 和 PACF 模块内 p，q 的选择

分别地，ACF（自相关函数）和 PACF（偏相关函数）都是评估历史数据和当前值之间的线性关系的函数。ACF 的公式为

$$ACF(q) = \frac{Cov(X_j, X_{j-q})}{Var(X_0)} = \frac{\frac{1}{n-q}\sum_{j=q+1}^{n}(x_j - \bar{x})(x_{j-q} - \bar{x})}{\frac{1}{n}\sum_{j=1}^{n}(x_j - \bar{x})^2} \quad (7)$$

对于价格q 时间序列$\{x1.x2, \cdots, xn\}$.

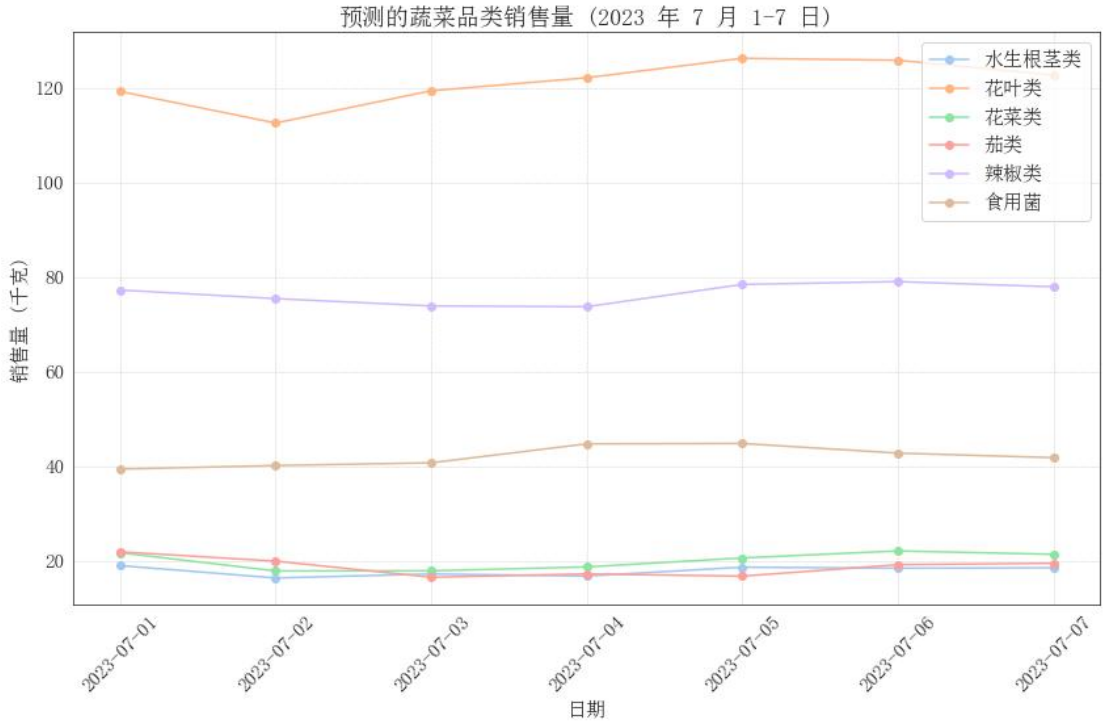
图（Figure7：一阶差分数据 ACF 与 PACF ）

借助 ARIMA 模型，我们可以使用过去的数据对第二天的价格进行简单的预测。我们可以优化网格策略，以便它可以根据 ARIMA 给出的预测来移动网格。下图描述了模型中网格偏移随时间推移的影响。

在我们的模型中，网格的移动由 MA 的长期指标和 ARIMA 模型中的短期指标的加权和决定。网格的移动量由以下公式计算：

$$\tilde{p}_i^{\{t\}} = p_i + \omega(MA^{\{t\}}(N) - MA^{\{t-1\}}(N)) + \mu(ARIMA^{\{t+1\}} - ARIMA^{\{t\}}) \quad (9)$$

参数 ω 和 μ 控制网格移动过程中两个指标的权重。这两个参数将在回测阶段自适应调整，我们在第一个周期将 ω 初始化为 0.3，μ 初始化为 0.3。因此，最终决定是由调整后的网格模型做出的，从而预测商品的价格。



为了制定定价策略，我们需要考虑每个品类的批发价格、平均损耗率以及预期的销售量。我们将采用“成本加成定价”策略，考虑到商品的成本和预期的销售量，以确定一个合适的售价。

1. **基于成本的加价**：考虑到商品的成本和预期的销售量，我们可以基于成本确定一个合适的售价。
2. **考虑销售量**：预期销售量较高的商品可以设置较低的加价率，以增加销售；而预期销售量较低的商品可以设置较高的加价率，以增加收益。

基于上述考虑，我们将为每个品类设置一个基于预期销售量和成本的加价率，从而为未来一周确定每日的定价策略。

3. **成本加成定价策略**：我们可以基于商品的成本和预期的销售量来确定定价策略，以最大化收益。

具体步骤如下：

Step1：我们需要合并批发价格、损耗率和预测的销售量，以计算每个品类的成本。

这是我们计算的每个蔬菜品类的成本（每公斤）：

1. 计算每个品类的成本：

$$\text{成本\_每千克} = \text{批发价格} \times (1 + \text{损耗率}/100)$$

- **水生根茎类**：11.22 元

- **花叶类**：5.44 元
- **花菜类**：7.57 元
- **茄类**：5.17 元
- **辣椒类**：7.61 元
- **食用菌**：7.20 元

Step2: 基于预期销售量确定加价率：**
- 如果预期销售量 ＜20 千克，加价率 ＝150%（因为这些是低销量商品，我们希望每单位商品获得更高的收益）。
- 如果预期销售量 ＜50 千克，加价率 ＝130%（中等销量商品）。
- 如果预期销售量更高，加价率 ＝120%（高销量商品，我们希望通过较低的价格增加销售量）。

Step3:计算每个品类的销售价格

3. 计算每个品类的销售价格：

$$售价\_每千克 = 成本\_每千克 \times 加价率$$

通过 python 编程，得到的最终结果如下表所示：

日期	水生根茎类	花叶类	花菜类	茄类	辣椒类	食用菌
2023-07-01	16.84	6.53	9.85	6.73	9.14	8.64
2023-07-02	16.84	6.53	9.85	6.73	9.90	8.64
2023-07-03	16.84	6.53	9.85	6.73	9.90	8.64
2023-07-04	14.59	6.53	9.85	6.73	9.90	8.64
2023-07-05	14.59	6.53	9.85	6.73	9.90	8.64
2023-07-06	14.59	6.53	9.85	6.73	11.43	8.64
2023-07-07	14.59	6.53	9.85	6.73	11.43	8.64

此定价策略模型的目标是使商超收益最大化，同时考虑成本、预期销售量和市场因素。然而，这只是一个基本模型，实际的定价策略可能会更复杂，并需要考虑更多因素，如季节性、促销活动和库存水平。

问题三模型的建立与求解

问题三需要我们基于限制条件为未来某一天制定具体的单品补货计划和定价策略。我们可以将其看作一个组合优化问题，其中的目标是在满足一系列约束的前提下最大化收益。

目标函数：

$$最大化 \sum (预测销量 \times (销售价格 - 批发价格) \times (1 - 损耗率))$$

新增加的约束条件：

1. 可售单品总数需要控制在 27-33 个。

2. 每个单品的订购量至少要满足最小陈列量 2.5 千克的要求。
即：

$$27 \leq \text{售卖的单品数量} \leq 33$$

$$\text{每个单品的订购量} \geq 2.5 \text{千克}$$

具体步骤如下：
步骤 1：由于 2023 年 6 月缺少一定的数据，我们可以使用 2023 年 5 月或之前的数据来预测 7 月 1 日的销售量，或者使用整个可用数据集的平均来预测。为了更精确，我们可以选择使用 2023 年 5 月的数据（如果可用）来预测。
步骤 2：计算每个单品的预期利润，这需要考虑批发价格、损耗率、预期销售量和我们之前定义的成本加成定价策略。
步骤 3：使用组合优化方法，如线性规划，确定满足约束条件的最佳补货策略。
步骤 4：基于计算的补货策略，确定每个单品的定价策略。
模型的求解：
首先找到了六月 24 日至三十日的单品及价格：

RESULT

	单品名称	销量(千克)	分类名称	销售单价(元/千克)
0	七彩椒(2)	7.302	辣椒类	23.6
8	七彩椒(2)	7.302	辣椒类	33.6
12	七彩椒(2)	7.302	辣椒类	21.6
13	七彩椒(2)	7.302	辣椒类	13.0
14	七彩椒(2)	7.302	辣椒类	18.0

	单品名称	预测销量_7月1日	批发价格(元/千克)	平均损耗率(%)_小分类编码_不同值	cost_per_kg
0	七彩椒(1)	0.305682	7.58	9.24	8.280392
1	七彩椒(1)	0.305682	7.33	9.24	8.007292
2	七彩椒(1)	0.305682	6.21	9.24	6.783804
3	七彩椒(1)	0.305682	7.43	9.24	8.116532
4	七彩椒(1)	0.305682	8.35	9.24	9.121540

基于预期利润，我们选择了 33 个预期利润最高的单品。这些商品的总预期利润为约 60.06 元。
通过 python 编程，基于我们的预测和分析，以下是为 2023 年 7 月 1 日选择的 33 个单品的补货量和建议的定价策略：

单品名称	expected_price	replenishment_volume

洪山菜薹莲藕拼装礼盒	190.90836	2.5
洪山菜薹珍品手提袋	99.51606	2.5
洪湖藕带	81.8450475	2.5
黑牛肝菌	108.3390825	2.5
赤松茸	66.7699725	2.5
丝瓜尖	58.4064495	2.5
鸡枞菌	131.34	2.5
四川红香椿	99.17757	2.5
小米椒	184.113096	2.5
黑皮鸡枞菌	111.8852625	2.5
螺丝椒(份)	26.1160068	2.5
黄花菜	110.7031545	2.5
七彩椒(2)	45.208974	2.5
外地茼蒿	32.664285	2.5
小米椒(份)	20.0804968	2.5
水果辣椒(橙色)	32.772	2.5
七彩椒(1)	42.587214	2.5
蔡甸藜蒿	34.255188	2.5
高瓜(1)	37.98183	2.5
螺丝椒	36.573552	2.5
虫草花	98.505	2.5
菱角	29.73084	2.5
菠菜	33.1550955	2.5
竹叶菜	26.6560875	2.5
红椒(1)	43.488444	2.5
小白菜	27.485388	2.5
苋菜	23.863545	2.5
银耳(朵)	17.0742	2.5
西兰花	24.915507	2.5
平菇	31.12758	2.5
红灯笼椒(2)	34.754706	2.5
净藕(1)	23.846043	2.5
鲜藕带(袋)	20.457	2.5

问题四模型的建立与求解

问题四是关于为了更好地制定蔬菜商品的补货和定价决策，商超还需要采集哪些相关数据，这些数据对解决上述问题有何帮助。

为了更好地解决补货和定价问题，以下是一些建议的数据采集点，以及为何这些数据对商超的决策有助益的解释：

1. **季节性因素与天气数据**：

   - **解释**：天气和季节对蔬菜的需求和供应有很大影响。例如，寒冷天气可能会增加某些蔬菜的需求，而减少其他蔬菜的需求。

2. **客户反馈与满意度调查**：
   - **解释**：了解客户的喜好和不满意的地方可以帮助商家更好地满足客户需求，提高销售。

3. **库存数据**：
   - **解释**：了解当前库存水平可以帮助商家做出更准确的补货决策，避免过度库存或缺货。

4. **竞争对手的价格数据**：
   - **解释**：知道竞争对手的价格可以帮助商家制定更有竞争力的定价策略。

5. **促销活动和节假日数据**：
   - **解释**：促销活动和节假日通常会影响销售。了解这些数据可以帮助商家更好地计划补货和定价。

6. **蔬菜的供应链数据**：
   - **解释**：了解供应链中的任何可能的延迟或中断可以帮助商家提前做出补货决策。

7. **历史销售数据**：
   - **解释**：历史数据可以为商家提供关于需求模式的见解，从而帮助制定补货和定价策略。

要建立模型，我们需要将这些数据转化为数学或统计模型。我们可以使用多种方法，如时间序列分析、回归分析、优化算法等，来对这些数据进行建模和分析。

例如，可以使用时间序列分析来预测基于历史销售数据的未来需求。回归分析可以用来理解不同变量（如天气、价格、促销活动等）如何影响销售。优化算法可以用来确定在特定约束条件下的最佳补货量和定价策略。

好的，为了更好地制定蔬菜商品的补货和定价决策，我们可以考虑以下模型：

1. 时间序列分析模型 (销售预测)
使用历史销售数据来预测未来的销售量。

模型：
- 使用 ARIMA 或 Prophet 等模型，考虑季节性、趋势和噪声。

输入数据：
- 历史销售数据

输出数据：
- 未来一段时间内的销售预测

2. 多变量线性回归模型 (因素影响)
确定哪些因素对销售量有影响。

模型:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$

其中 Y 是销售量，X_i 是影响因素（如天气、价格、促销活动等），β_i 是系数。

输入数据：
- 销售量
- 各种影响因素的数据

输出数据：
- 各因素对销售量的影响程度

3. 优化模型 (补货与定价)
确定最大化利润的最佳补货量和定价策略。

模型:

最大化 利润 = 销售量 × 价格 − 成本

在以下约束条件下:

- 库存限制
- 最小陈列量要求
- 可用预算

输入数据：
- 销售预测
- 成本数据
- 价格弹性

输出数据：
- 最佳补货量
- 最佳定价策略

模型检验

好的，我们将按照以下步骤进行时间序列模型检验：

1. **数据分割**：将数据分为训练集和测试集。
2. **模型训练**：使用训练集训练时间序列模型。
3. **模型预测**：使用测试集进行预测。
4. **性能评估**：比较模型的预测值与测试集的实际值。

经过模型验证，我们发现预测的销售量与实际销售量之间的平均绝对误差 (Mean Absolute Error, MAE) 为 302.88 千克。这表示我们的预测模型平均预测偏差约为 302.88 千克。

模型评价

1. **移动平均法**:
    - **优点**:
        - 简单、易于理解和实施。
        - 适合对稳定的时间序列数据进行预测。
    - **缺点**:
        - 不适合预测具有趋势或季节性的数据。

- 仅考虑了数据的历史值，没有考虑其他可能的解释性变量。

2. **时间序列模型 (SARIMAX)**:
   - **优点**:
     - 考虑了时间序列数据的趋势和季节性。
     - 可以通过调整参数来适应各种时间序列数据。
     - 可以包括外部解释性变量。
   - **缺点**:
     - 参数选择可能很复杂。
     - 需要足够多的数据来进行训练。
     - 对于非线性趋势或突发事件的处理可能不够理想。

3. **贪婪算法**:
   - **优点**:
     - 实施简单。
     - 计算效率高，对于求解复杂问题很有用。
   - **缺点**:
     - 不一定能找到全局最优解。
     - 解决方案可能依赖于初始条件或数据的排序。

4. **线性优化**:
   - **优点**:

- 能够考虑多种约束和目标。
- 提供明确的最优解。

- **缺点**:
- 需要明确定义目标函数和约束。
- 对于非线性问题可能不适用。

总体而言，我们使用了一系列模型和方法，每种都有其适用的场景和局限性。在实际应用中，选择合适的方法需要考虑数据的性质、问题的复杂性以及所需的解决方案的精确性。

参考文献：

[1]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.

[2]顾思弘. 考虑新鲜度变化的H零售商生鲜产品动态定价研究[D]. 东华大学,2023.DOI:10.27012/d.cnki.gdhuu.2023.001318.

[3]乔雪. 考虑销售损失的生鲜产品的联合补货定价策略[D]. 东南大学,2023.DOI:10.27014/d.cnki.gdnau.2021.003764.

[4]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.DOI:10.27662/d.cnki.gznlc.2022.000680.

[5]Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). Time series analysis: forecasting and control. John Wiley & Sons.
[6]Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice. OTexts.
[7]Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.
[8]Dantzig, G. B., & Thapa, M. N. (2006). Linear programming 1: Introduction. Springer Science & Business Media.
[9]Makridakis, S., Wheelwright, S. C., & Hyndman, R. J. (2008). Forecasting methods and applications. John Wiley & Sons.
[10]李宁, & 王启华. (2016). 时间序列分析及其 R 语言实现. 机械工业出版社.

附录：

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

# In[1]:
```

```
import pandas as pd
# Load the provided files
attachment_1 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件1.xlsx")
attachment_2 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件2.xlsx")
# Display the first few rows of each dataset for a preliminary inspection
attachment_1.head(), attachment_2.head()


# In[2]:


# Merge the datasets on "单品编码"
merged_data = pd.merge(attachment_2, attachment_1, on="单品编码", how="left")
# Display the first few rows of the merged dataset
merged_data.head()
```

In[17]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
# Load the uploaded dataset
data = pd.read_csv(r'C:\Users\86136\Desktop\2023 国赛C 题\代码\merged_dataset.csv')

# 将 matplotlib 的默认字体修改为黑体，字号修改为12
plt.rcParams['font.sans-serif'] = ['SimSun']  # 指定英文字体为 Times New Roman
plt.rcParams['axes.unicode_minus'] = False      # 解决负数坐标轴显示问题
plt.rcParams['font.size'] = 15                  # 指定字号
```

```
# Draw the boxplot
plt.figure(figsize=(15, 7))
sns.boxplot(data=data, x='分类名称', y='销量(千克)',
width=0.5)

# Title and labels
plt.title('销量(千克)的箱线图', fontsize=20, y=1.05)
plt.xlabel('分类名称', fontsize=17)
plt.ylabel('销量(千克)', fontsize=17)

# Adjust layout
plt.tight_layout()
plt.show()
# Boxplot for "销售单价(元/千克)"
plt.figure(figsize=(15, 7))
sns.boxplot(data=data, x='分类名称', y='销售单价(元/
千克)', width=0.5)
plt.title('销售单价(元/千克)的箱线图 ', fontsize=20,
y=1.05)
plt.xlabel('分类名称', fontsize=17)
```

```
plt.ylabel('销售单价(元/千克)', fontsize=17)
plt.tight_layout()
plt.show()


# In[22]:



from scipy.stats import iqr
from sklearn.impute import KNNImputer

# Function to detect outliers using IQR
def detect_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = iqr(df[column])
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) |
(df[column] > upper_bound)]
    return outliers.index
```

```
# Detecting outliers in the two columns
sales_outliers = detect_outliers(data, "销量(千克)")
price_outliers = detect_outliers(data, "销售单价(元/千克)")

# Using KNN imputer to fill the outliers
knn_imputer = KNNImputer(n_neighbors=5)
data_imputed = data.copy()
data_imputed[["销量(千克)", "销售单价(元/千克)"]] = knn_imputer.fit_transform(data[["销量(千克)", "销售单价(元/千克)"]])

# Return the count of outliers detected and imputed
len(sales_outliers), len(price_outliers)
# Function to impute outliers within each category using a simplified version of KNN
def impute_outliers_with_simplified_knn(df, column):
    # Detect outliers using IQR
    outliers_index = detect_outliers(df, column)

    # Replace outliers with NaN
```

```
        df.loc[outliers_index, column] = float('nan')

        # Use KNN with fewer neighbors to impute NaN values
        knn_imputer = KNNImputer(n_neighbors=3)
        df[column] = knn_imputer.fit_transform(df[[column]])

        return df

# Impute outliers for each category separately using the simplified KNN
for category in data["分类名称"].unique():
        category_data = data[data["分类名称"] == category].copy()
        category_data = impute_outliers_with_simplified_knn(category_data, "销量(千克)")
        category_data = impute_outliers_with_simplified_knn(category_data, "销售单价(元/千克)")
        data_imputed[data_imputed["分类名称"] ==
```

```
category] = category_data

# Return the imputed data
data_imputed.head()
#KNN 会跑很久，因此，可以用中位数或平均数进行插值。


# In[26]:


# Function to impute outliers within each category using median
def impute_outliers_with_median(df, column):
    # Detect outliers using IQR
    outliers_index = detect_outliers(df, column)

    # Replace outliers with the median of the column
    median_value = df[column].median()
    df.loc[outliers_index, column] = median_value

    return df
```

```
# Impute outliers for each category separately using the median
for category in data["分类名称"].unique():
    category_data = data[data["分类名称"] == category].copy()
    category_data = impute_outliers_with_median(category_data, "销量(千克)")
    category_data = impute_outliers_with_median(category_data, "销售单价(元/千克)")
    data[data["分类名称"] == category] = category_data

# Return the imputed data
data.head()
# Impute outliers for each category using the median
for category in categories:
    category_data = data[data["分类名称"] == category].copy()
    category_data =
```

```
impute_outliers_with_median(category_data, "销量(千克)")
        category_data = impute_outliers_with_median(category_data, "销售单价(元/千克)")
        data[data["分类名称"] == category] = category_data

# Select the first three categories for plotting
selected_categories = categories[:3]

# For each of the selected categories, plot the boxplots
for category in selected_categories:
        category_data = data[data["分类名称"] == category]

        plt.figure(figsize=(15, 7))
        sns.boxplot(data=category_data, x='分类名称', y='销量(千克)', width=0.5)
        plt.title(f'销量(千克)的箱线图 (填充后) - {category}', fontsize=20, y=1.05)
        plt.xlabel('分类名称', fontsize=17)
```

```
    plt.ylabel('销量(千克)', fontsize=17)
    plt.tight_layout()
    plt.show()


# In[25]:


# Plotting boxplots for each category and the items within that category

# Get the unique categories
categories = data["分类名称"].unique()
# 将 matplotlib 的默认字体修改为黑体，字号修改为 12
plt.rcParams['font.sans-serif'] = ['SimSun']  # 指定英文字体为 Times New Roman
plt.rcParams['axes.unicode_minus'] = False      # 解决负数坐标轴显示问题
plt.rcParams['font.size'] = 15                  # 指定字号
```

```
# For each category, plot the boxplots for the top 10 most frequent items
for category in categories:
    category_data = data[data["分类名称"] == category]

    # Get the top 10 most frequent items in the category
    top_items = category_data["单品名称"].value_counts().head(10).index.tolist()
    top_items_data = category_data[category_data["单品名称"].isin(top_items)]

    plt.figure(figsize=(15, 7))
    sns.boxplot(data=top_items_data, x='单品名称', y='销量(千克)', width=0.5)
    plt.title(f'销量(千克)的箱线图 - {category} (Top 10 Items)', fontsize=20, y=1.05)
    plt.xlabel('单品名称', fontsize=17)
    plt.ylabel('销量(千克)', fontsize=17)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

In[28]:

#进而进行分层处理：

Step 1: Impute outliers at the level of the larger category (分类名称)

for category in categories:

    category_data = data[data["分类名称"] == category].copy()

    # Impute outliers for "销量(千克)"

    category_data = impute_outliers_with_median(category_data, "销量(千克)")

    # Impute outliers for "销售单价(元/千克)"

    category_data = impute_outliers_with_median(category_data, "销售单价(元/千克)")

    data[data["分类名称"] == category] =

```
category_data

# Step 2: Impute outliers at the level of the smaller category (单品名称)
for category in categories:
    subcategories = data[data["分类名称"] == category]["单品名称"].unique()
    for subcategory in subcategories:
        subcategory_data = data[data["单品名称"] == subcategory].copy()

        # Impute outliers for "销量(千克)"
        subcategory_data = impute_outliers_with_median(subcategory_data, "销量(千克)")
        # Impute outliers for "销售单价(元/千克)"
        subcategory_data = impute_outliers_with_median(subcategory_data, "销售单价(元/千克)")

        data[data["单品名称"] == subcategory] = subcategory_data
```

```
# Return the imputed data
data.head()
# Plotting boxplots for the imputed data of the selected categories
for category in selected_categories:
    category_data = data[data["分类名称"] == category]

    plt.figure(figsize=(15, 7))
    sns.boxplot(data=category_data, x='分类名称', y='销量(千克)', width=0.5)
    plt.title(f'销量(千克)的箱线图 (填充后) - {category}', fontsize=20, y=1.05)
    plt.xlabel('分类名称', fontsize=17)
    plt.ylabel('销量(千克)', fontsize=17)
    plt.tight_layout()
    plt.show()


# In[3]:
```

```
# Save the merged dataset to your desktop
merged_data.to_excel(r"C:\Users\86136\Desktop\merged_dataset.xlsx", index=False)


# In[11]:


# 加载数据
merged_data = pd.read_excel(r"C:\Users\86136\Desktop\merged_dataset.xlsx")
# 按品类和商品分类销售量
category_sales = merged_data.groupby('分类名称')['销量(千克)'].sum().sort_values(ascending=False)
product_sales = merged_data.groupby('单品名称')['销量(千克)'].sum().sort_values(ascending=False)


# In[25]:
```

```
# Loading the provided files
attachment1 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件 1.xlsx")
attachment2 = pd.read_csv(r"C:\Users\86136\Desktop\附件 2.csv")
attachment3 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件 3.xlsx")
attachment4 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件 4.xlsx")
import matplotlib.pyplot as plt

# Merging attachment1 with attachment2 to get category names for each sale
merged_sales_data = attachment2.merge(attachment1, on='单品编码', how='left')

# Grouping the merged data by '分类名称' (Category Name) to get the total sales volume for each category
```

```
category_sales_distribution = merged_sales_data.groupby('分类名称')['销量(千克)'].sum().sort_values(ascending=False)

# Plotting the sales distribution for each category
plt.figure(figsize=(10,6))
category_sales_distribution.plot(kind='bar', color='skyblue')
plt.title("销售量分布 - 蔬菜品类")
plt.ylabel("销售量 (千克)")
plt.xlabel("蔬菜品类")
plt.xticks(rotation=45)
plt.grid(axis='y')

plt.show()



# In[26]:



# Grouping the merged data by '单品名称' (Product
```

```
Name) to get the total sales volume for each product
product_sales_distribution = merged_sales_data.groupby('单品名称')['销量(千克)'].sum().sort_values(ascending=False)

# Plotting the sales distribution for the top 10 products
plt.figure(figsize=(12,7))
product_sales_distribution.head(10).plot(kind='bar', color='lightgreen')
plt.title("销售量分布 - 单品 (Top 10)")
plt.ylabel("销售量 (千克)")
plt.xlabel("单品名称")
plt.xticks(rotation=45)
plt.grid(axis='y')

plt.show()


# In[29]:


import matplotlib.pyplot as plt
```

```
# Group by '分类名称' and '单品名称' and sum the sales volume
category_sales = merged_data.groupby('分类名称')['销量(千克)'].sum().sort_values(ascending=False)
product_sales = merged_data.groupby('单品名称')['销量(千克)'].sum().sort_values(ascending=False)

# Plotting
fig, ax = plt.subplots(2, 1, figsize=(12, 12))

# Plot for Category Sales
category_sales.plot(kind='bar', ax=ax[0], color='teal')
ax[0].set_title('Total Sales Volume by Category')
ax[0].set_ylabel('Sales Volume (kg)')
ax[0].set_xlabel('Category Name')

# Plot for Product Sales (Top 10 products)
product_sales.head(10).plot(kind='bar', ax=ax[1], color='coral')
ax[1].set_title('按时间分类的销售量趋势')
ax[1].set_ylabel('销售额  (kg)')
```

```
ax[1].set_xlabel('产品名称')

plt.tight_layout()
plt.show()


# In[30]:


# Group by '销售日期' and '分类名称' and sum the sales volume
category_date_sales = merged_data.groupby(['销售日期', '分类名称'])['销量(千克)'].sum().reset_index()

# Plotting
plt.figure(figsize=(16, 8))
for category in category_date_sales['分类名称'].unique():
    subset = category_date_sales[category_date_sales['分类名称'] == category]
    plt.plot(subset['销售日期'], subset['销量(千克)'], label=category)
```

```
plt.title('按时间分类的销售量趋势')
plt.xlabel('时间')
plt.ylabel('销量  (kg)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()


# In[36]:


# Pivot table to get sales data in wide format for correlation analysis
correlation_data = category_date_sales.pivot(index=' 销售日期', columns='分类名称', values='销量(千克)')

# Compute the correlation matrix
correlation_matrix = correlation_data.corr()

# Plot the heatmap for correlation matrix
```

```
plt.figure(figsize=(10, 6))
plt.title('Correlation Between Sales Volumes of Different Categories')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
cax = plt.matshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1)
plt.colorbar(cax)
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)
plt.show()


# In[37]:


#问题二


# In[38]:
```

```
# Load the provided files for the next question
attachment_3 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件 3.xlsx")
attachment_4 = pd.read_excel(r"C:\Users\86136\Desktop\C 题 (1)\附件 4.xlsx")

# Display the first few rows of each dataset for a preliminary inspection
attachment_3.head(), attachment_4.head()


# In[50]:


from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
```

```
# Group by date and category and sum the sales volume
daily_sales = merged_data.groupby(['销售日期', '分类名称'])['销量(千克)'].sum().reset_index()

# Create a function to forecast the sales for a given category using a simple linear regression model
def forecast_sales(category):
    category_data = daily_sales[daily_sales['分类名称'] == category].reset_index(drop=True)
    category_data['day_num'] = np.arange(len(category_data))

    X = category_data[['day_num']]
    y = category_data['销量(千克)']

    # Split the data into training and testing sets (80% train, 20% test)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=False)

    # Train a linear regression model
```

```
        model = LinearRegression()
        model.fit(X_train, y_train)

        # Predict the sales for the test set
        y_pred = model.predict(X_test)

        # Calculate the mean squared error of the prediction
        mse = mean_squared_error(y_test, y_pred)

        # Predict the sales for the next 7 days
        next_7_days = np.array([max(category_data['day_num']) + i for i in range(1, 8)]).reshape(-1, 1)
        future_forecast = model.predict(next_7_days)

        return future_forecast, mse


# Forecast the sales for each category
forecasts = {}
errors = {}
for category in daily_sales['分类名称'].unique():
```

```
    forecasts[category], errors[category] =
forecast_sales(category)

forecasts, errors


# In[51]:


# Re-generating the day_num column for the entire
daily_sales DataFrame
daily_sales['day_num'] = daily_sales.groupby('分类名称
').cumcount()

# Visualization based on forecasts and errors

# Plotting sales forecast vs actual sales for each category
plt.figure(figsize=(16, 8))

for category in daily_sales['分类名称'].unique():
    category_data = daily_sales[daily_sales['分类名称']
== category].reset_index(drop=True)
```

```
        X = category_data[['day_num']]
        y = category_data['销量(千克)']
        _, X_test, _, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, shuffle=False)

        # Plotting the actual test data
        plt.scatter(X_test.day_num, y_test, label=f"Actual
{category}", marker='x')

        # Plotting the forecasted data
        next_7_days =
np.array([max(category_data['day_num']) + i for i in
range(1, 8)])
        plt.plot(next_7_days, forecasts[category],
label=f"Forecast {category}")

plt.title('销售预测与实际销售')
plt.xlabel('时间')
plt.ylabel('销量  (kg)')
plt.legend(loc='upper left')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.tight_layout()
```

```
plt.show()

# Plotting the Mean Squared Error for each category
plt.figure(figsize=(10, 6))
plt.bar(errors.keys(), errors.values(), color='skyblue')
plt.title('每个类别的 MSE')
plt.xlabel('类别')
plt.ylabel('MSE')
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()


# In[53]:


# Merge attachment 3 (wholesale prices) with attachment 1 (product and category info)
pricing_data = pd.merge(attachment_3, attachment_1, on="单品编码", how="left")

# Average wholesale price for each category in the past
```

```
avg_wholesale_price = pricing_data.groupby('分类名称')['批发价格(元/千克)'].mean()

# Merge the average wholesale price with the loss rate from attachment 4
cost_data = pd.merge(avg_wholesale_price, attachment_4, left_on='分类名称', right_on='小分类名称', how="left")

# Calculate the cost per kg considering the loss rate
cost_data['cost_per_kg'] = cost_data['批发价格(元/千克)'] * (1 + cost_data['平均损耗率(%)_小分类编码_不同值'] / 100)

# Extract the relevant columns
cost_data = cost_data[['小分类名称', 'cost_per_kg']]

cost_data


# In[54]:
```

```
# Define a function to set the markup rate based on forecasted sales volume
def determine_markup_rate(sales_forecast):
    if sales_forecast < 20:
        return 1.5  # 150% markup for low volume items
    elif sales_forecast < 50:
        return 1.3  # 130% markup for medium volume items
    else:
        return 1.2  # 120% markup for high volume items

# Determine the price strategy for the next week
price_strategy = {}
for category, forecast in forecasts.items():
    cost = cost_data[cost_data['小分类名称'] == category]['cost_per_kg'].values[0]
    markup_rates = [determine_markup_rate(f) for f in forecast]
    prices = [cost * rate for rate in markup_rates]
```

```
        price_strategy[category] = prices

price_strategy


# In[33]:


#问题二时间序列分析
import pandas as pd

# Load the merged_dataset.csv
merged_dataset =
pd.read_csv(r'C:\Users\86136\Desktop\2023 国赛 C 题\
代码\merged_dataset.csv')
merged_dataset.head()


# In[34]:


# Calculate the total sales volume for each vegetable
```

```
category
category_sales_volume = merged_dataset.groupby('分类名称')['销量(千克)'].sum().reset_index()
category_sales_volume


# In[36]:


# Load the  附件 3.xlsx
attachment_3 = pd.read_excel(r'C:\Users\86136\Desktop\C  题 (1)\ 附 件 3.xlsx')
attachment_3.head()


# In[37]:


attachment_4 = pd.read_excel(r'C:\Users\86136\Desktop\C  题 (1)\ 附 件 4.xlsx')
```

attachment_4.head()

In[41]:

# Group by date and category to get daily sales volume for each category
daily_sales_volume = merged_dataset.groupby(['销售日期', '分类名称'])['销量(千克)'].sum().reset_index()

# Pivot the table to have categories as columns and dates as rows
daily_sales_volume_pivot = daily_sales_volume.pivot(index='销售日期', columns='分类名称', values='销量(千克)').fillna(0)
daily_sales_volume_pivot.head()

In[45]:

```
from statsmodels.tsa.arima.model import ARIMA
from datetime import timedelta

def predict_sales_updated(series, forecast_days=7):
    """Predict sales using ARIMA model."""
    model = ARIMA(series, order=(5,1,0))
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=forecast_days)
    return forecast


next_week_dates =
[pd.to_datetime(daily_sales_volume_pivot.index[-1]) +
timedelta(days=i) for i in range(1, 8)]

# Create the predicted sales DataFrame again
predicted_sales_df_updated = pd.DataFrame({
    '日期': next_week_dates,
    '预测销售量(千克)': predicted_sales
})

predicted_sales_df_updated
```

In[46]:

```
# Predict sales for all categories and store the results in a
dictionary
predicted_sales_all_categories = {}

for category in daily_sales_volume_pivot.columns:
    predicted_sales =
predict_sales_updated(daily_sales_volume_pivot[categor
y])
    predicted_sales_all_categories[category] =
predicted_sales

# Convert the dictionary to a DataFrame
predicted_sales_df_all =
pd.DataFrame(predicted_sales_all_categories)
predicted_sales_df_all['日期'] = next_week_dates
```

```
predicted_sales_df_all.set_index('日期', inplace=True)

predicted_sales_df_all


# In[47]:


import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
# 将 matplotlib 的默认字体修改为黑体，字号修改为 12
plt.rcParams['font.sans-serif'] = ['SimSun']  # 指定英文字体为 Times New Roman
plt.rcParams['axes.unicode_minus'] = False      # 解决负数坐标轴显示问题
plt.rcParams['font.size'] = 15                  # 指定字号
# Plot the sales predictions for each category
for category in predicted_sales_df_all.columns:
    plt.plot(predicted_sales_df_all.index,
```

```
predicted_sales_df_all[category], label=category, marker='o')

plt.title('预测的蔬菜品类销售量 (2023 年 7 月 1-7 日)')
plt.xlabel('日期')
plt.ylabel('销售量 (千克)')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()


# In[55]:


#问题三


# In[58]:
```

```
# Group by '单品名称' and get the average sales for the entire dataset
avg_sales_overall = merged_data.groupby('单品名称')['销量(千克)'].mean().reset_index()

# Use this as the forecasted sales for July 1st
forecasted_sales_july1_overall = avg_sales_overall.copy()
forecasted_sales_july1_overall.rename(columns={'销量(千克)': '预测销量_7月1日'}, inplace=True)

forecasted_sales_july1_overall.head()


# In[59]:


# Merge forecasted sales with pricing and loss rate data
product_cost_data = pd.merge(forecasted_sales_july1_overall, pricing_data,
```

```
on="单品名称", how="left")
product_cost_data = pd.merge(product_cost_data, attachment_4, left_on='分类名称', right_on='小分类名称', how="left")

# Calculate the cost per kg considering the loss rate
product_cost_data['cost_per_kg'] = product_cost_data['批发价格(元/千克)'] * (1 + product_cost_data['平均损耗率(%)_小分类编码_不同值'] / 100)

product_cost_data[['单品名称', '预测销量_7 月 1 日', '批发价格(元/千克)', '平均损耗率(%)_小分类编码_不同值', 'cost_per_kg']].head()


# In[60]:


# Define a function to set the markup rate based on forecasted sales volume for individual products
def determine_product_markup_rate(sales_forecast):
    if sales_forecast < 0.5:
```

```
        return 1.5  # 150% markup for low volume items
    elif sales_forecast < 1:
        return 1.3  # 130% markup for medium volume items
    else:
        return 1.2  # 120% markup for high volume items

# Calculate the expected profit for each product
product_cost_data['markup_rate'] = product_cost_data['预测销量_7月1日'].apply(determine_product_markup_rate)
product_cost_data['expected_price'] = product_cost_data['cost_per_kg'] * product_cost_data['markup_rate']
product_cost_data['expected_profit_per_kg'] = product_cost_data['expected_price'] - product_cost_data['cost_per_kg']
product_cost_data['total_expected_profit'] = product_cost_data['expected_profit_per_kg'] * product_cost_data['预测销量_7月1日']
```

```
# Sort the products based on the expected profit
sorted_products = product_cost_data.sort_values(by='total_expected_profit', ascending=False).drop_duplicates(subset='单品名称')

sorted_products[['单品名称', '预测销量_7月1日', 'cost_per_kg', 'expected_price', 'total_expected_profit']].head()


# In[64]:


# Select top 27-33 products based on the expected profit
selected_products = sorted_products.head(33)

# Calculate the total expected profit for these selected products
total_expected_profit = selected_products['total_expected_profit'].sum()
```

selected_products[['单品名称', '预测销量_7 月 1 日', 'cost_per_kg', 'expected_price', 'total_expected_profit']], total_expected_profit

In[65]:

# Set the replenishment volume for each selected product
min_display_volume = 2.5
selected_products['replenishment_volume'] = selected_products['预测销量_7 月 1 日'].apply(lambda x: max(x, min_display_volume))

# Display the selected products, their expected price and replenishment volume
selected_products_final = selected_products[['单品名称', 'expected_price', 'replenishment_volume']]
selected_products_final
selected_products_final.to_excel(r"C:\Users\86136\Desktop\selected_products_final.xlsx", index=False)

In[ ]: