队伍编号	MCB2401653
赛道	B

基于数据挖掘与优化算法的电商库存管理及分仓规划研究

摘 要

随着电子商务的快速发展，电商企业面临着日益增长的商品存储和分发需求。为了提高仓库管理效率和降低仓储成本，需要对商品进行合理的分类和存储规划。本文对各商品的存货量以及销量数据进行挖掘与分析，构建了准确的货量预测和品类分仓规划模型旨在协助电商平台优化库存管理，更好地规划和定制促销活动，提高库存周转率。

针对问题一，首先进行数据预处理，引入峰度系数、偏度系数对原序列进行描述性统计，用**分布特性与序列趋势特性结合**的方法进行缺失值、异常值的处理。对于库存量预测，构造滞后项，平均变化率等多种特征，接着采用MA、ES、Linear Trend、SA、WMA、LR作为基线模型进行性能比较，基线最优为ES，其类别平均R^2为0.73，类别平均MAE为11195.53，接着通过对平滑系数进行调优，**调优ES模型的类别平均R^2为0.98，类别平均MAE为4596.3。**针对销量预测，采用复杂度较指数平滑模型更高的模型来进行拟合，**对于自回归模型，我们采用销量数据的23年序列进行预测，对于LSTM系列模型，我们通过提取22年同时期序列的趋势特征作为特征列增强模型的表征能力，**经过比较分析：ARIMA为最优基线模型，其类别平均R^2为0.6，类别平均MAE为2610.1。然后通过对ARIMA模型进行**残差检验**，验证其基本符合线性模型的残差假设，接着采用**变分模态分解**将原始序列分解为多个IMF分量，结合**最小化重构序列与原序列均方误差的K值选择策略**，使用最优基线模型对每个IMF分量进行预测，再将预测结果汇总，最终**VMD+ARIMA模型的类别平均R^2为0.91，类别平均MAE为961.19。**

针对问题二，首先对问题一的预测结果进行预处理，包括取整以及去除小于0的部分以符合实际情况，接着构建一个**整数线性规划**（ILP）模型，将所有目标进行加权，构成一个总的目标函数，以品类是否放入仓库来构建决策变量，由于传统求解器计算效率低下，故采用遗传算法（GA）提高求解效率，并通过加入**惩罚项**来确保满足仓库容量和生产能力的限制条件。经过多次迭代后，得到合理的“一品一仓”分配方案。

针对问题三，放宽了每个品类只能存放在单一仓库的约束，我们在原有的遗传算法框架基础上**调整个体表示方式**，使得每个品类可以对应于多个仓库编号，并增加了两个新的约束条件：件型约束以及高级品类约束。**通过调整权重参数，强调了品类关联度的重要性**，同时也不忽视其他关键指标的表现。接着采用遗传算法进行求解，最后运用**蒙特卡洛模拟**进行敏感性分析并绘制结果的热力图，验证了调整不同目标的权重系数来体现对目标的关注度方法的合理性。

最后对所建立的模型进行了讨论和分析，**本文模型的主要优点为算法简单易于实现，且相较于深度学习模型更透明，可解释性更强。**

关键词:时间序列预测; 变分模态分解; 分仓规划; 遗传算法

目录

1 问题重述

1.1 问题背景

随着电子商务的快速发展，电商企业面临着日益增长的商品存储和分发需求。为了提高仓库管理效率和降低仓储成本，电商企业需要对商品进行合理的分类和存储规划。在实际操作中，商品通常按照品类、件型等属性进行分类和标记，以便于库存管理。商品品类繁多，数量庞大，需要分散到不同的仓库中存储。

准确的货量预测是品类分仓规划的重要依据。通过预测未来的库存量和销量，企业可以提前规划仓储资源，减少不必要的资源投入。库存量预测涉及对每个品类在所有仓库中所需存放的总库存量的预测，而销量预测则涉及对每个品类在所有仓库中所需打包出库的总量的预测。这些预测结果将受到仓库的仓容和产能限制。

在进行品类分仓规划时，需要考虑多个业务目标，包括仓容利用率、产能利用率、总仓租成本、品类分仓数和品类关联度。合理的分仓方案应该在满足仓容和产能约束的同时，最大化品类关联度，同时兼顾其他指标。

1.2 问题提出

基于所给数据，解决以下三个问题：

问题1： 建立货量预测模型，预测未来3个月（7-9月）每个月的库存量和销量。库存量预测需要给出月均库存量，而销量预测需要给出未来每天的预测结果。

问题2： 在限定每个品类只能放在一个仓库中的条件下，基于问题1的预测结果，建立规划模型，求得品类的分仓方案。方案需要包括应使用的仓库以及这些仓库需要存放哪些品类的库存。

问题3： 放开一品一仓的假设，允许一个品类存放于多个仓库，但同一品类存放的仓库数量不能超过3个。同时，希望同件型、同高级品类尽量放在一个仓库中。基于问题1的预测结果，建立规划模型，求得新的品类分仓方案，并分析不同方案中各业务指标的表现。

2 问题分析

2.1 问题一的分析

在问题一中，目标是建立一个货量预测模型，以预测未来三个月（7月至9月）每个月的库存量及每天的销量。为了达成这一目标，首先需要对原始数据进行了清洗与预处理，包括缺失值和异常值的处理。接着，通过可视化趋势分析以及描述性统计来理解不同品类的历史库存量和销量的变化特性。基于这些特征，采用较为简单，复杂度不高的时间序列预测模型来进行库存量预测，同时，对于销量预测，数据点更多的情况则可以采用更为复杂的ARIMA模型结合VMD分解的方法。ARIMA模型能够捕捉时间序列中的自回归、差分和移动平均成分，而VMD分解则可以将复杂的时间序列分解成多个简单的子序列，从而提

高预测精度。此外，还可以引入LSTM模型，利用其强大的长期依赖关系捕捉能力进一步优化销量预测。最终，通过对各种方法的性能评估，确定最佳预测方案，并将结果用于后续的分仓规划问题。

2.2 问题二的分析

在问题二中，任务是在每个品类只能存放在一个仓库的前提下，根据问题一得到的预测结果，制定最优的品类分仓方案。这涉及到综合考虑多个业务指标，包括但不限于仓容利用率、产能利用率、总仓租成本、品类分仓数以及品类关联度。为了解决这个问题，可以构建一个整数线性规划（ILP）模型，旨在最小化总仓储成本的同时最大化仓库效率和品类关联度。然而，考虑到使用传统求解器可能会面临计算效率低下的问题，可以采用遗传算法（GA）。遗传算法模拟自然选择的过程，通过迭代选择、交叉和变异等操作逐步优化解决方案。在具体实现过程中，通过加入惩罚项来确保满足仓库容量和生产能力的限制条件。

2.3 问题三的分析

针对问题三，放宽了前一个问题中每个品类只能存放在单一仓库的约束，允许同一品类分散到最多三个不同的仓库中存储。与此同时又希望同件型或同高级品类的商品尽可能被安排在同一仓库内，以此减少订单处理时的包裹数量，降低履约成本。为此，可以在原有的遗传算法框架基础上做了相应的调整，改变个体表示方式，使得每个品类可以对应于多个仓库编号。此外，通过增加两个新的约束条件：一是尽量让具有相同件型的商品位于同一个仓库；二是尽量让属于同一高级类别的商品也集中在一起。通过调整权重参数，来提高品类关联度的重要性，同时也不忽视其他关键指标的表现。最后，可以运用蒙特卡洛模拟技术进行敏感性分析，探究不同权重设置下模型输出的变化情况，从而帮助更好地理解和优化最终的分仓策略。

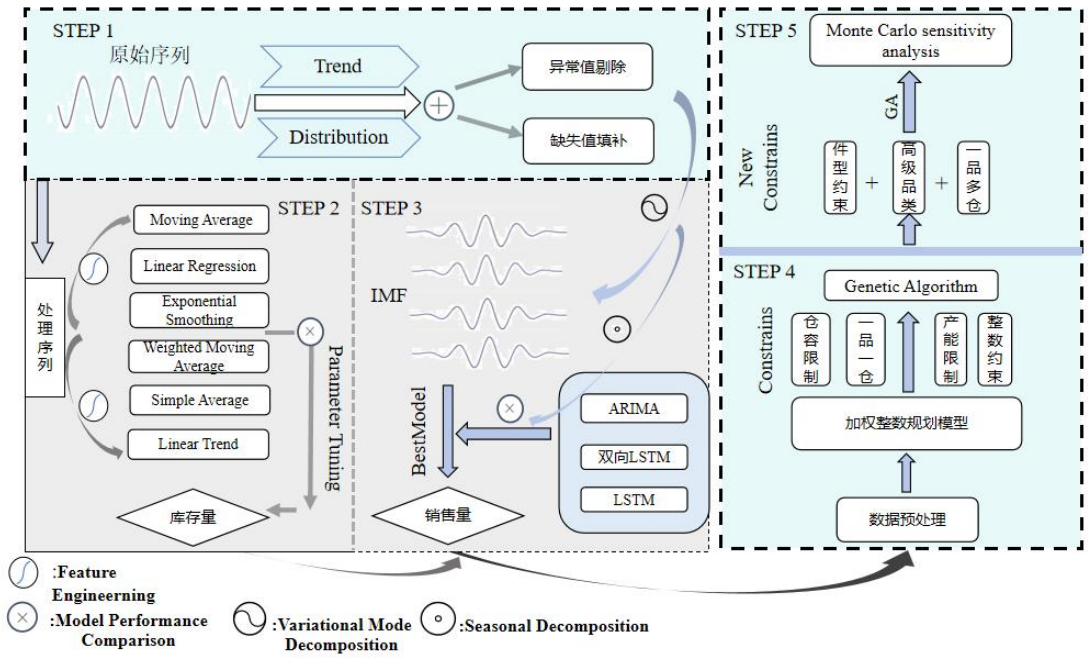


图1 总体思路图

3 模型假设

1、假设所调查的商品库存量和销量序列的数据都比较准确；

2、假设搜索空间中的解在不同时间段具有相同的适应度分布，即数据是同质的。

3、假设优化问题中的噪声（如测量误差）是独立的，不会对搜索过程产生系统性偏差。

4、假设噪声在分解过程中是独立的，且对每个模态的影响是均匀的。

4 符号说明

符号	意义
S_k	偏度系数
K_u	峰度系数
CV	变异系数
y_{t-1}	滞后项
β_1	斜率
MA_t	移动平均
α	平滑系数
$\mu_K(t)$	模态分量
$e^{-j\omega_K t}$	中心频率
$\lambda(t)$	拉格朗日乘子
f	适应度函数
r	重构信号
P	惩罚项
R	关联度矩阵
x_{ij}	决策变量

5 问题一模型建立与求解

5.1 数据清洗与预处理

5.1.1 原始数据可视化与趋势分析

问题一主要涉及附件一与附件二数据，我们首先对原始数据进行可视化来初步分析序列的趋势，由于品类数目较多，故选择前五个品类进行具体分析，下面是这五个品类的库存量以及销量变化趋势：

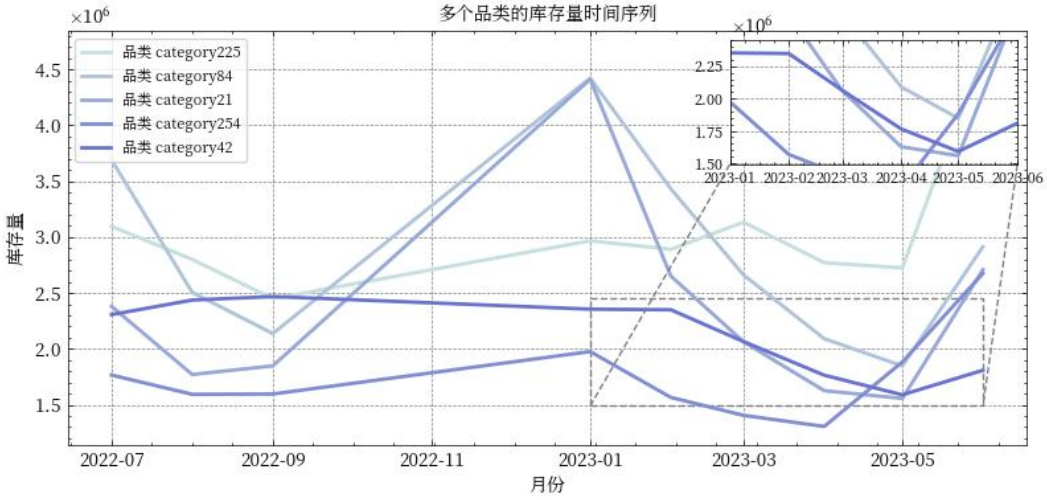


图2 前五类别的库存量时间序列图

根据库存量时间序列趋势图，我们可以观察到五个类别在2022年7月至2023年6月期间的库存变化情况。整体而言，这些类别的库存量经历了不同程度的波动。category225的库存量在2022年下半年逐渐增加至峰值后，在2023年初急剧下降，并随后保持在一个较低水平上波动；而category84的库存则相对稳定，但在2023年初出现了一次显著的增长，之后又回到了接近初始的水平。对于category21来说，其库存量在2022年下半年持续增长并在2023年初达到顶峰，紧接着迅速减少，直到2023年中期才有所回升。相比之下，category254的库存量在整个考察期内基本保持平稳，仅有一些小幅度的波动；category42则显示出较为稳定的库存水平，但在进入2023年后出现了明显的下降趋势。从共同特征来看，大多数类别都在2023年初经历了一定程度上的库存量变动，这可能是由于季节性因素、市场需求的变化或是市场策略调整所导致的结果。此外，不同类别之间也展现出各自独特的库存管理特点，例如category225和category21表现出较大的库存量波动，而category254和category42则相对更加稳定。

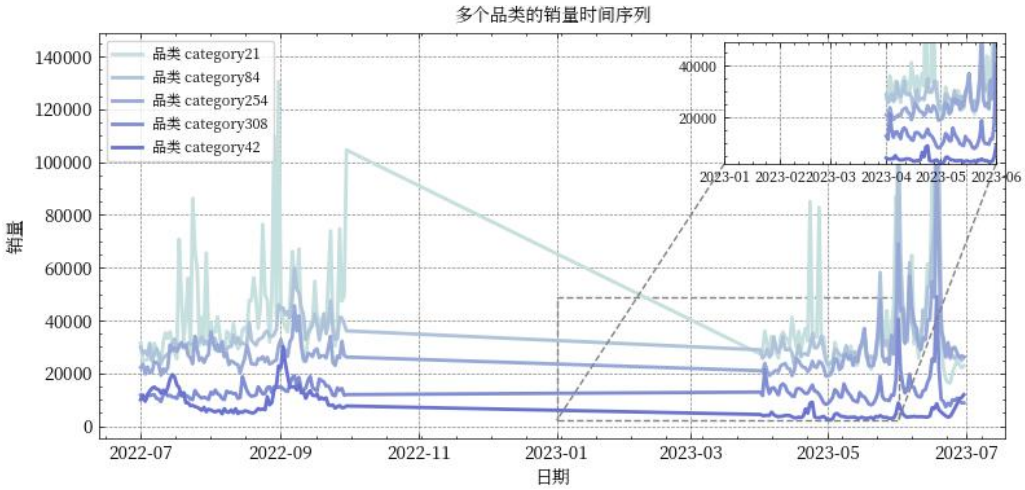


图3 前五类别的销量时间序列图

根据销量时间序列趋势图，我们可以观察到五个不同类别从2022年7月至2023年7月期间的销售表现。category84的销量在整个时间段内表现出高度波动的特点，特别是在2022年下半年达到高峰后，在2023年开始逐渐回落，第一季度末再次出现一次高峰，随后销量继续下滑。相比之下，category21的销量相对平稳，但在2022年末至2023年初有一段时间销量明显提升，之后回归正常水平。category225的销量曲线较为平缓，尽管在2022年第四季度和2023年上半年初期经历了一次轻微的上升，整体上显示出稳定的销售表现。category254的销量走势与category21相似，整体平稳但存在周期性波动，尤其是在2023年前两个月销量有所上升，之后趋于稳定。最后，category2的销量最低且波动最小，仅在2022年第三季度末至第四季度初有一次小幅上升，总体保持在较低水平。这些销量变化可能受到季节性因素、促销活动或行业事件的影响，同时也反映出各产品在市场中的竞争地位及消费者偏好的差异。例如，category84的高波动性可能表明这是一个竞争激烈的市场，而category2的低销量则暗示其市场份额有限。

5.1.2 描述性统计

上面我们对原始数据进行了粗略的趋势分析，下一步我们对这五个类别的数据进行详细的描述性统计，本文引入最大值、最小值、中位数、标准差、偏度系数、峰度系数，变异系数来描述统计数据：

(1)偏度系数

偏度系数(Skewness)用于衡量数据分布的偏斜程度。$S_k = 0$表示数据近似对称分布，$S_k > 0$表示数据呈右偏分布，$S_k < 0$表示数据呈左偏分布。偏度系数的绝对值越大，数据分布的偏斜程度越明显。其计算公式如下：

$$S_k = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu^3}{\sigma^3}$$

其中S_k为偏度系数，$E(X)$为均值，μ^3为三阶中心距

(2)峰度系数

峰度系数(Kurtosis)用于衡量数据分布的峰度程度。$K_u = 0$表示数据分布为正态分布，$K_u < 0$表示数据分布的峰度较小，数据更分散，$K_u > 0$表示数据分布的峰度较大，数据更集中。峰度系数的绝对值越大，数据分布的峰度程度越明显。其计算公式如下：

$$K_u = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\mu^4}{\sigma^4}$$

其中K_u为峰度系数，$E(X)$为均值，μ^4为三阶中心距

(3)变异系数

变异系数是一个统计学中的度量指标，用来衡量一组数据的离散程度与平均值的比例关系。它通过将标准差与均值进行比较，提供了一个无单位的度量方式，因此特别适用于比较不同单位或不同尺度数据的离散程度。其计算公式如下：

$$CV = \frac{\sigma}{\mu} \times 100\%$$

其中σ是数据的标准差，μ是数据的均值。

统计结果如下：

表1 前五类别的库存量描述性统计结果

品类	标准差	最大值	最小值	偏度	峰度	变异系数
类别225	641611.2	467605	24489	1.980	5.847	20.994
类别84	848816.5	4421974	1846707	0.591	2.243	29.730
类别21	886943.4	4411095	1555956	1.499	4.408	37.988
类别254	406775.4	2675098	1305514	1.294	4.042	23.225
类别42	330224.83	2467824	1588575	-0.507	1.660	15.527

通过对前五个类别的库存量进行描述性统计分析，我们可以得出以下结论：

• 类别225的库存量表现出较高的波动性，其数据分布呈现出明显的右偏和尖峭特征，表明该品类可能存在一些极端高值。

• 类别84的库存量变化范围非常大，但数据分布相对较为平坦且右偏程度较低。这说明尽管该类别的库存量在不同时间点上有显著差异，但整体上并没有出现极端的异常值。

• 类别21的库存量波动最大，数据分布不仅右偏而且非常尖峭，存在大量极端值。这表明该类别的库存量非常不稳定，可能会频繁出现极高的库存水平或极低的库存水平

• 类别254的库存量同样显示出较大的波动性，其数据分布也呈现右偏和尖峭的特点，意味着该品类的库存量也可能受到某些特定因素的影响而出现极端值。

• 类别42的库存量变化最小，数据分布左偏且较为平坦，是五个品类中相对最稳定的。这表明该类别的库存量在不同时间点上保持了较好的一致性，较少出现极端值。

总体来看，类别21和类别254的库存量波动较大且存在较多极端值；类别42的库存量最为稳定，变化较小且分布较为平坦；类别84和类别225的库存量也存在一定的波动，但相对于其他品类来说较为中等。

表2 前五类别的销量描述性统计结果

品类	标准差	最大值	最小值	偏度	峰度	变异系数
类别225	13872.101	110234	8226	3.150	15.835	61.427
类别84	13951.381	141914	22966	4.469	28.970	40.117
类别21	18200.216	130556	16318	2.104	8.289	46.207
类别254	9476.035	100254	18916	3.735	23.524	33.460
类别42	4808.175	30050	2256	1.593	5.960	65.338

通过对前五个类别的销量进行描述性统计分析，我们可以得出以下结论：

• 类别225的销量表现出较高的波动性，其数据分布呈现出明显的右偏和尖峭特征。这表明该类别的销量可能存在一些极端高值，且整体变化较大。

• 类别84的销量同样具有较高的波动性，数据分布也呈现显著的右偏和尖峭特征。这说明尽管该类别的销量在不同时间点上有显著差异，但整体上并没有出现极端的异常值。

• 类别21的销量波动较大，数据分布不仅右偏而且非常尖峭，存在大量极端值。这表明该类别的销量非常不稳定，可能会频繁出现极高的销量或极低的销量。

• 类别254的销量波动相对较小，数据分布虽然仍呈现右偏和尖峭的特点，但整体上较为稳定。这表明该类别的销量在不同时间点上保持了一定的一致性，较少出现极端值。

• 类别42的销量波动性较高，数据分布呈现右偏和尖峭的特点。这表明该类别的销量在不同时间点上变化较大，且存在极端值。

总体来看，类别21和类别42的销量波动较大，且存在较多极端值；类别225和类别84的销量也显示出较高的波动性，需要注意极端值的影响；类别254的销量波动相对较小。

5.1.3 缺失值与异常值处理

首先我们统计了原始数据的缺失值：

• 针对库存量数据：每一个类别都没有存在缺失值，这为我们后续的处理提供了便利。

• 针对销量数据：我们发现每一个类别从2022年9月到2023年4月都有长时间段的数据缺失问题，我们尝试了随机森林以及prophet模型填充，效果均不佳，**原因是缺失数据时间段太长，模型无法学习到数据的模式，所以我们不进行数据的填充，**同时有近十五个仓库2023年4月1日到2023年6月30日的序列存在部分缺失，我们设定阈值为10，如果与标准长度(91)差距大于10，则我们将不处理该列，反之，我们将其填补至正常长度。

接着我们处理原始数据的异常值：

• 针对库存量数据：我们观察到对于每一个类别，库存量的数据仅有不到十个左右，**所以为了保证数据量不过低，我们不选择对库存量数据进行异常值的剔除。**

• 针对销量数据，我们结合实际情况分析，**销量的极端值可能是由于市场特性、季节性因素、促销活动或突发事件等引起的，这些极端值有时反映了真实的市场情况。**在这种情况下，完全剔除这些极端值可能会导致信息丢失，并且可能会影响后续的分析和预测结果，故我们也不进行异常值剔除。

5.2 库存量预测

经过我们上面的统计分析，每个类别的库存量数据点很少，所以我们选择采用较为简单的解释性强的模型，下面是我们的总体库存量预测流程：

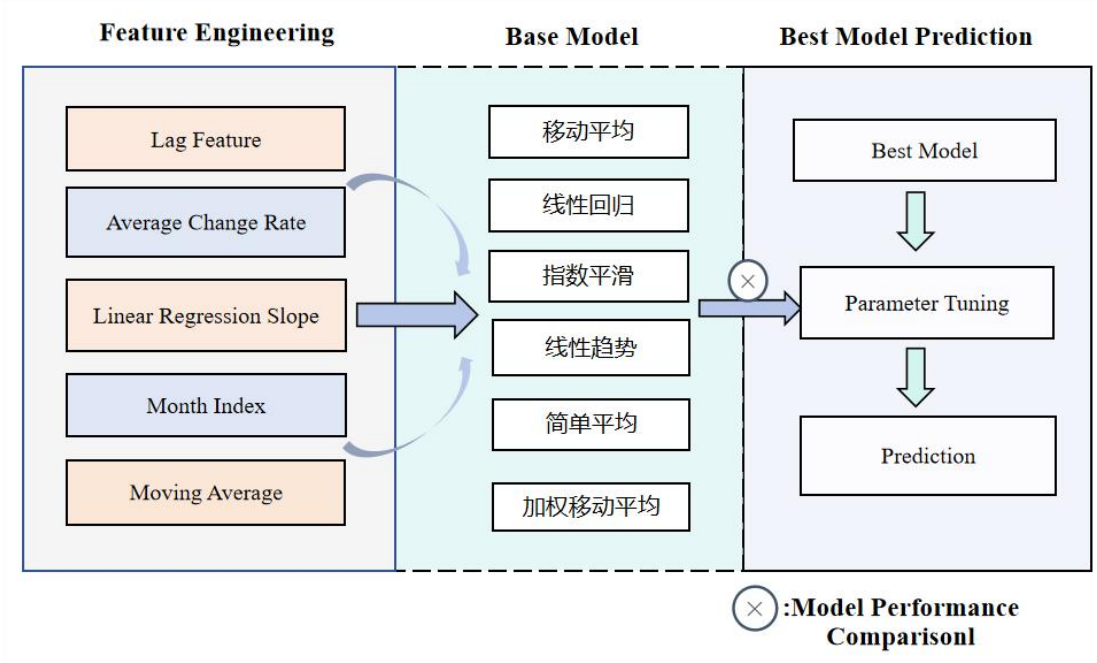


图3 库存量预测总体思路

同时为了较快的评价模型的性能，我们自定义如下指标：

假设我们有三个类别 (类别1、类别2、类别3),并且我们已经计算了每个类别的R^2、MSE、RMSE和MAE值。类别平均评价指标的计算如下：

·类别平均R^2:

$$\text{Average } R^2 = \frac{R^2_{\text{类别1}} + R^2_{\text{类别2}} + R^2_{\text{类别3}}}{3}$$

·类别平均 MSE:

$$\text{Average MSE} = \frac{\text{MSE}_{\text{类别1}} + \text{MSE}_{\text{类别2}} + \text{MSE}_{\text{类别3}}}{3}$$

·类别平均 RMSE:

$$\text{Average RMSE} = \frac{\text{RMSE}_{\text{类别1}} + \text{RMSE}_{\text{类别2}} + \text{RMSE}_{\text{类别3}}}{3}$$

·类别平均 MAE:

$$\text{Average MAE} = \frac{\text{MAE}_{\text{类别1}} + \text{MAE}_{\text{类别2}} + \text{MAE}_{\text{类别3}}}{3}$$

通过这种方式，我们可以得到每个评价指标的类别平均值，从而更好地评估模型在不同类别上的性能。

5.2.1 库存量特征构造

根据预测流程，我们首先来进行特征的构造：

• 一阶滞后项(Lag Feature)

一阶滞后项是指将时间序列数据向前移动一个时间步长，从而创建一个新的特征。这个特征可以帮助模型捕 捉时间序列中的短期依赖关系，即当前时间点的值与前一个时间点的值之间的关系。

假设时间序列数据为y_t,则一阶滞后项y_{t-1}可以表示为：

$$y_{t-1} = y_t$$

• 平均变化率 (Average Change Rate)

平均变化率是指在特定时间段内，时间序列数据的变化量除以时间段的长度。这个特征反映了在特定时间段 内时间序列的平均变化趋势。

假设时间段为$[t_1, t_2]$,时间序列数据为y_t,则平均变化率change_rate可以表示为：

$$change\_rate = \frac{y_{t_2} - y_{t_1}}{t_2 - t_1}$$

• 线性回归斜率(Linear Regression Slope)

线性回归斜率是指通过对特定时间段内的数据进行线性回归，得到的回归直线的斜率。这个特征反映了时间 序列在特定时间段内的线性增长或下降趋势。

假设时间段为$[t_1, t_2]$,时间序列数据为y_t ,线性回归模型为$y = \beta_0 + \beta_1 t$,则斜率β_1可以表示为：

$$\beta_1 = \frac{\sum_{i=1}^{n} (t_i - t)\,(y_i - \bar{y})}{\sum_{i=1}^{n} (t_i - \bar{t})^2}$$

其中$\bar{t}$和$\bar{y}$分别是时间点和时间序列数据的均值。

• 移动平均(Moving Average)

移动平均是指在时间序列数据上计算一定窗口大小内的平均值。这个特征可以帮助模型捕捉时间序列的长期 趋势，减少短期波动的影响。

假设窗口大小为k,时间序列数据为y_t ,则移动平均MA_t可以表示为：

$$MA_t = \frac{1}{k} \sum_{i=t-k+1}^{t} y_i$$

• 月份序号(Month Index)

月份的序号是指将时间序列数据中的每个时间点表示为一个序号，表示其在时间序列中的位置。这个特征可 以帮助模型捕捉时间序列的季节性或周期性模式。

假设时间序列数据为y_t,则月份的序号t可以表示为：

$$t = 1,2,3,\ldots,n$$

其中n是时间序列的长度。

5.2.2 线性回归初步拟合

由于数据点太少，所以我们首先选择最简单的模型—线性回归来进行拟合：

线性回归是一种广泛使用的统计方法，用于建立自变量（特征）和因变量（目标）之间的线性关系。线性回归的目标是找到一条直线（在多维情况下是一个超平面），使得预测值与实际值之间的误差最小，其流程如下：

假设我们有一个数据集，其中包含n个样本，每个样本有m个特征。对于第i个样本，其特征向量为:

$$x_i = [x_{i1}, x_{i2}, \ldots, x_{im}]$$

对应的因变量为:

$$y_i$$

。

线性回归模型的数学表达式为：

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_m x_{im} + \epsilon_i$$

其中：

y_i是第i个样本的因变量。 β_0是截距,表示当所有特征值为零时的预测值。$\beta_1, \beta_2, \ldots, \beta_m$是特征的系数,表示每个特征对因变量的影响。 $x_{i1}, x_{i2}, \ldots, x_{im}$是第$i$个样本的特征值。·$\epsilon_i$是误差项，表示模型无法解释的部分。

预测结果如下：

图4 线性回归预测结果

经验证，其拟合平均R^2为负，且数值很大，说明线性回归对于库存量的预测模型复杂度还是太简单了，并没有学习到数据的模式，所以下面我们采用一些相较于线性回归复杂度高一些的模型来进行预测：

• 移动平均 (Moving Average, MA)

移动平均是一种简单的时间序列平滑技术，它通过取一定数量的最近观测值的平均来减少数据中的随机波动。移动平均可以帮助识别数据的趋势或周期性模式。

对于一个时间序列$\{y_t\}$,k期的移动平均可以表示为：

$$MA_t = \frac{1}{k}\sum_{i=0}^{k-1} y_{t-i}$$

• 指数平滑 (Exponential Smoothing, ES)

指数平滑是另一种平滑技术，它对近期的数据赋予更高的权重，而对较远的数据赋予较低的权重。这使得模型能够更快地响应最近的变化。

单指数平滑 (Simple Exponential Smoothing, SES) 可以表示为：

$$S_t = \alpha y_t + (1-\alpha)S_{t-1}$$

其中，S_t是时间t的平滑值，α是平滑系数,y_t是时间t的实际观测值。

• 线性趋势 (Linear Trend)

线性趋势模型假设时间序列有一个固定的斜率，即随着时间的推移，数据以恒定的速度增加或减少。这种模型适合于具有明显线性增长或下降趋势的数据。

线性趋势模型可以表示为：

$$y_t = a + b \cdot t + \epsilon_t$$

其中，y_t是时间t的观测值，a是截距，b是斜率，t是时间点，ϵ_t是误差项。

• 简单平均 (Simple Average, SA)

简单平均是对整个时间序列的所有观测值求平均，适用于没有明显趋势或季节性的稳定时间序列。

对于一个时间序列$\{y_t\}$,简单平均可以表示为：

$$SA = \frac{1}{n}\sum_{t=1}^{n} y_t$$

其中，n是观测值的数量。

• 加权移动平均 (Weighted Moving Average, WMA)

加权移动平均与简单的移动平均类似，但它给不同的观测值赋予不同的权重。通常是最近的观测值赋予更高的权重。

对于一个时间序列$\{y_t\}$,k期的加权移动平均可以表示为：

$$_{WMAt} = \sum_{i=0}^{k-1} w_i \cdot y_{t-i}$$

其中，w_i是第i个观测值的权重，且$\sum_{i=0}^{k-1} w_i = 1$。

基线模型预测结果如下：

表3 库存量基线模型预测结果

模型	平均R^2	平均MSE	平均RMSE	平均MAE
移动平均	-1.09	6167091070	22723.99	17895.75
指数平滑	**0.73**	**2469744520**	**15014.63**	**11195.53**
线性趋势	0.47	6040465797	20906.23	17244.44
简单平均	0.01	7961016353	27350.45	21469.17
加权移动平均	-3.66	10135516857	28322.05	22577.03

从结果中我们看出，指数平滑模型的表现最优，其不仅具有最高的决定系数（R^2），表明它能够很好地解释数据的变化，还具有最低的均方误差（MSE）、均方根误差（RMSE）和平均绝对误差（MAE），表明其预测误差最小。相比之下，其他模型如移动平均、简单平均和加权移动平均的表现较差，它们的R^2接近或低于0，且误差指标显著高于指数平滑。这说明这些模型在数据集上未能有效捕捉到数据的模式，可能是由于它们对数据的假设与实际情况不符，或者参数设置不够合理。

另外，指数平滑模型相对简单且稳健，不容易过拟合，特别是在数据点较少的情况下。这意味着即使数据量有限，指数平滑也能提供稳定的预测结果。**综上所述，指数平滑在数据上表现出色，是因为其能够灵活地适应短期波动,适用于无趋势和季节性的数据，并且在小样本条件下保持良好的稳定性和准确性**

5.2.3 指数平滑模型调优

调优指数平滑模型的过程通常包括以下步骤：

表4 指数平滑调优步骤

***Step* 1** 初始化：选择一个初始的平滑系数α和初始平滑值S_0。初始平滑值S_0通常设置为第一个观测值y_1或者所有观测值的平均值。
***Step* 2** 生成预测：使用选定的α生成整个时间序列的平滑值S_t。
***Step* 3** 计算评估指标：计算模型的评估指标，如均方误差(MSE)、均方根误差(RMSE)、平均绝对误差(MAE)和决定系数(R2)等。
***Step* 4** 调整α:尝试不同的α值 (例如从0.1到0.9,步长为0.1),并重复步骤2和3，记录每个α值下的评估指标。
***Step* 5** 选择最佳α: 根据评估指标选择最佳的α值。通常会选择使评估指标最小的α值，例 如使MSE或MAE最小的α值。

于是我们根据上述步骤对每个类别的平滑系数进行调优，拟合结果如下：

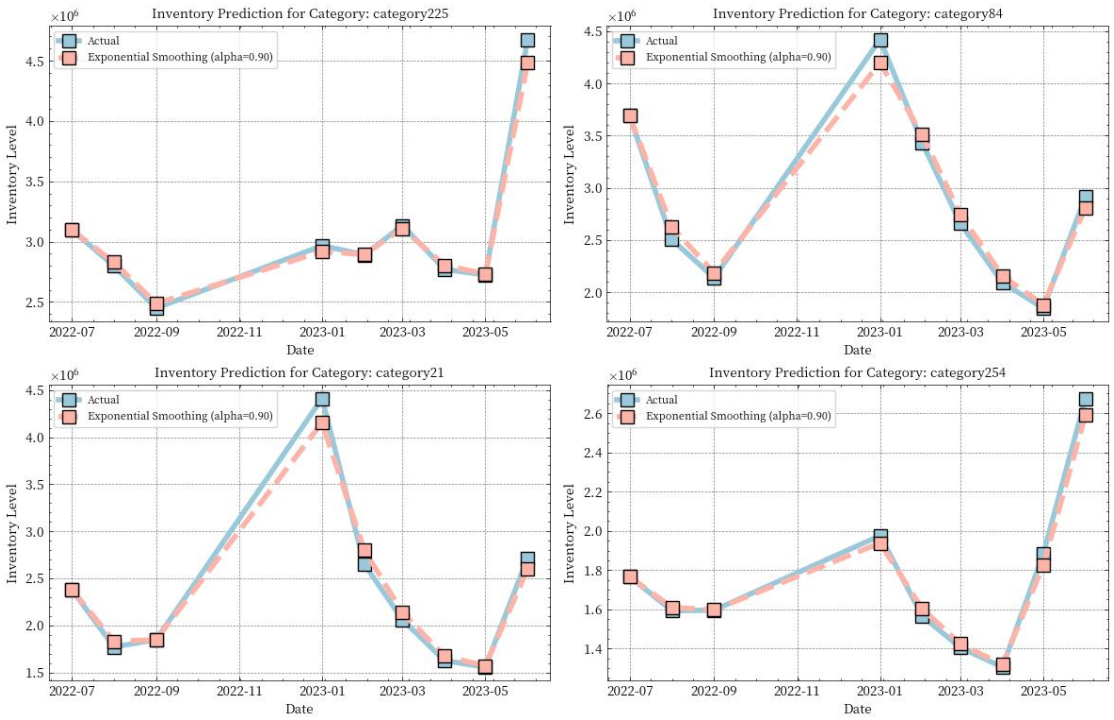


图5 调优指数平滑预测结果

接下来我们汇总了所有的拟合结果并进行比较分析：

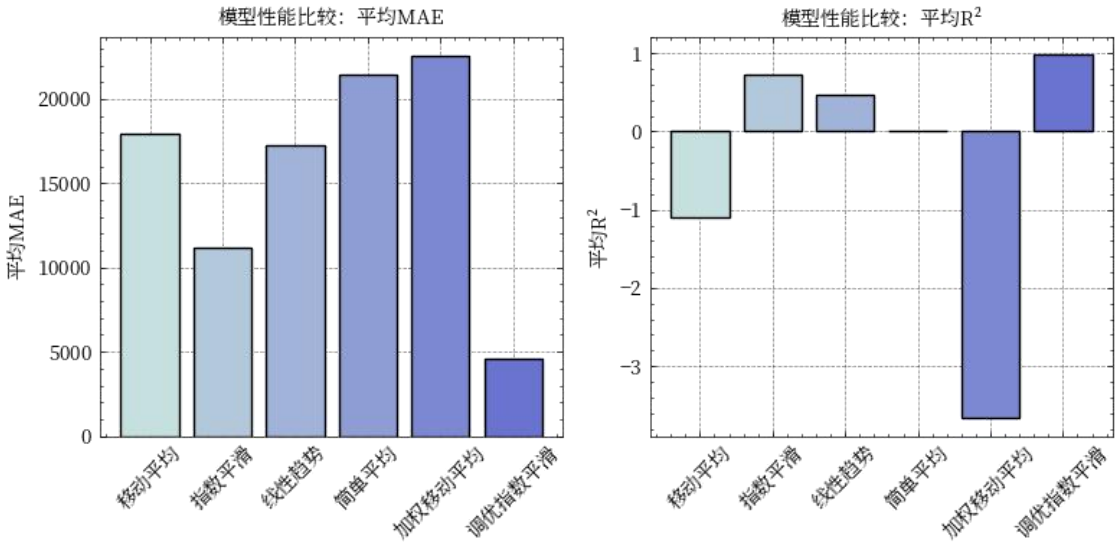


图6 模型性能比较图

表5 模型性能表

模型	平均R^2	平均MSE	平均RMSE	平均MAE
移动平均	-1.09	6167091070	22723.99	17895.75
指数平滑	0.73	2469744520	15014.63	11195.53
线性趋势	0.47	6040465797	20906.23	17244.44

简单平均	0.01	7961016353	27350.45	21469.17
加权移动平均	-3.66	10135516857	28322.05	22577.03
调优指数平滑	**0.98**	**1245326412**	**6073.2**	**4596.3**

这些结果表明，调优指数平滑不仅在解释数据变化方面优于其他模型，还在减少预测误差方面表现出色，所以我们最终选择调优指数平滑模型来进行库存量的预测。
预测结果如下：

表6 库存量预测结果

	7月库存量	8月库存量	9月库存量
category1	6163.33722	6169.940717	6175.883865
category31	39.69760659	42.12545253	44.31051387
category61	289299.9081	288382.7253	287557.2609
category91	6720.070835	6817.334586	6904.871963
category121	97629.86342	96948.44051	96335.15988
category151	141903.9588	141003.9216	140193.8882
category181	815.1370026	806.260305	798.2712771
category211	7638.257001	7648.388301	7657.506472
category241	58460.02704	58210.75137	57986.40327
category271	20465.56789	20336.47899	20220.29898
category301	3388.94922	3356.503518	3327.302387
category331	52677.68465	52172.50083	51717.8354

5.3 日销量预测

日销量相较于库存量数据点更多，所以我们选择采用复杂度较指数平滑模型更高的模型来进行拟合，对于自回归模型，我们采用销量数据的23年序列进行预测，对于LSTM系列模型，我们通过提取22年同时期序列的趋势特征作为特征列增强模型的表征能力，最后我们选择出Base模型中的SOTA模型，再将原始序列使用变分模态分解为多个简单子序列，使用SOTA模型对每个子序列进行预测，最后将子序列预测结果进行汇总，得出最终预测结果，思路示意图如下：

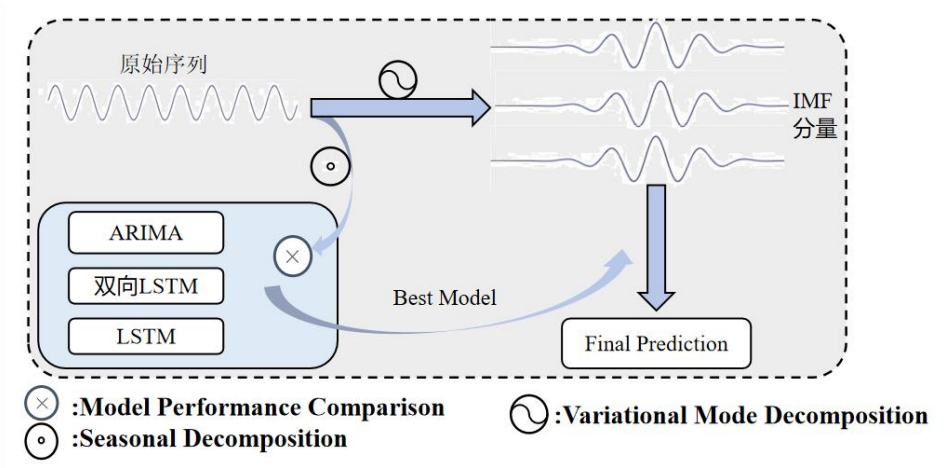


图7 销量预测思路

5.3.1 ARIMA单序列预测

ARIMA（AutoRegressive Integrated Moving Average）模型是一种常用的时间序列预测方法。它综合了自回归模型（AR）、差分法（I）和移动平均模型（MA）的特性，可以有效地捕捉时间序列数据中的各种模式。ARIMA模型的全称是自回归积分滑动平均模型。

・ADF检验

ADF（Augmented Dickey-Fuller）检验是一种用于检测时间序列数据是否存在单位根的统计方法。单位根是时间序列分析中的一个重要概念，若时间序列存在单位根，则
表示该序列是非平稳的，无法通过简单的均值和方差描述其统计特性。*ADF*检验通过引入滞后项来解决原始 Dickey-Fuller检验中存在的自相关问题，从而更加准确地判断时间序列的平稳性。

ADF检验的原理是对以下假设进行检验：

• 原假设 H_0：时间序列存在单位根，即时间序列是非平稳的。

• 备择假设 H_1：时间序列不存在单位根，即时间序列是平稳的

ADF检验的基本模型是自回归模型，可以表示为：

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \epsilon_t$$

ADF检验的关键是检验γ是否显著小于1。如果γ显著小于1，则拒绝原假设(即时间序列存在单位根),认为时间序列是平稳的，我们对原始销量序列进行检验，结果如下：

ADF Statistic: -3.867287 p-value: 0.002289
Critical Values:1%: -3.505 5%: -2.894 10%: -2.584

结果分析：

ADF统计量：这个值表示检验统计量的量度。通常，如果ADF统计量绝对值较大（即更负），则更倾向于拒绝原假设，认为序列是平稳的。

p值：这个值表示检验结果的显著性水平。如果p值小于某个显著性水平，则认为结果在统计上是显著的，从而拒绝原假设，认为序列是平稳的。

最终结果的ADF统计量的绝对值较大，远小于0，这意味着序列具有平稳性。p值远小于0.05，这意味着检验结果在统计上是显著的，我们可以拒绝原假设，认为**序列是平稳的。**

・ARIMA阶数确定

ARIMA模型：ARIMA模型是AR(自回归)、I(差分以实现平稳性)、MA(移动平均)三个部分的组合，可以表示为ARIMA(p,d,q)

使用ACF和PACF确定ARIMA模型阶数：

ACF: 如果ACF在滞后p时迅速衰减至0，而在滞后q时仍然显著不为0，这表明MA项的阶数可能是q。

PACF:如果PACF在滞后p时迅速衰减至0，这表明AR项的阶数可能是p。

由此我们绘制出了序列的ACF与PACF图：

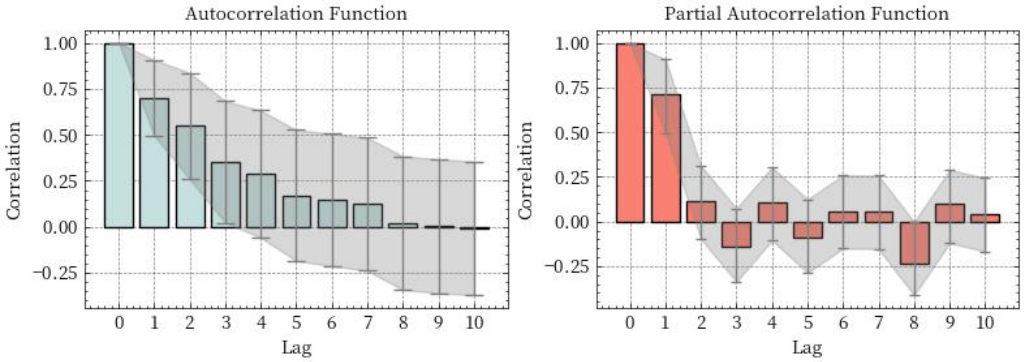


图8 ACF与PACF图

从图中我们不太好分析出具体的阶数，于是我们尝试了许多参数组合，最终选择p, d, q为(5,0,5)拟合效果最好，预测结果如下：

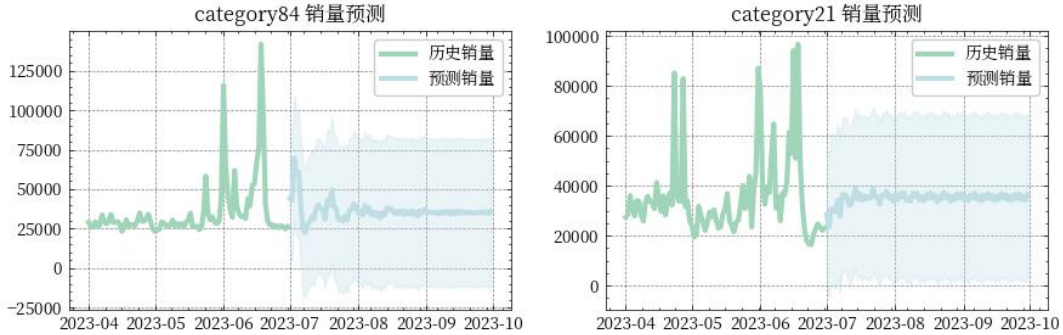


图9 ARIMA销量预测结果

5.3.2 LSTM序列预测

上面我们采用了ARIMA模型来进行预测，下面我们来尝试第二种基线模型：

长短期记忆网络（LSTM，Long Short-Term Memory）是一种特殊的循环神经网络（RNN），专门设计用于解决传统RNN在处理长序列数据时遇到的梯度消失和梯度爆炸问题。LSTM通过引入门控机制来控制信息的流动，从而有效地捕捉序列中的长期依赖关系，其核心是以下四点：

• 遗忘门 (Forget Gate) :

假设在时间步t,LSTM的输入为x_t,前一个时间步的隐藏状态为h_{t-1},前一个时间步的细胞状态为C_{t-1} 。 LSTM的计算过程如下：

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

其中，σ是sigmoid函数，W_f和b_f是遗忘门的权重和偏置。

• 输入门(Input Gate) :

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

其中，i_t是输入门的输出，$\tilde{C}_t$是候选细胞状态。

• 更新细胞状态(Cell State):

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

其中，C_t是更新后的细胞状态。

• 输出门(Output Gate):

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot tanh(C_t)$$

其中，o_t是输出门的输出，h_t是当前时间步的隐藏状态。

根据我们前面对销量描述性统计的分析可知，数据存在很长一段时间的缺失，而且是不好填充的，所以**我们选择仅采用2023,4,1到2023,6,30的数据做训练，然后我们提取2022,7,1到2022,9,30日数据的趋势性，季节性，残差特征来作为特征列代入训练，**这样就可以对数据的利用率最大，又可以避免因长时间的数据缺失导致的模型性能问题。

季节性分解是一种时间序列分析方法，用于将时间序列数据分解为三个主要组成部分：趋势（Trend）、季节性（Seasonality）和残差（Residuals）。
季节性分解的组成部分：

• 趋势（Trend）：表示时间序列的长期变化趋势，通常是数据随时间逐渐增加或减少的模式。

• 季节性（Seasonality）：表示时间序列中周期性重复的模式，通常与特定的时间段（如月、季度、年）相关。

• 残差（Residuals）：表示时间序列中无法由趋势和季节性解释的部分，通常被认为是随机噪声。

我们仅展示第一个类别的分解结果：

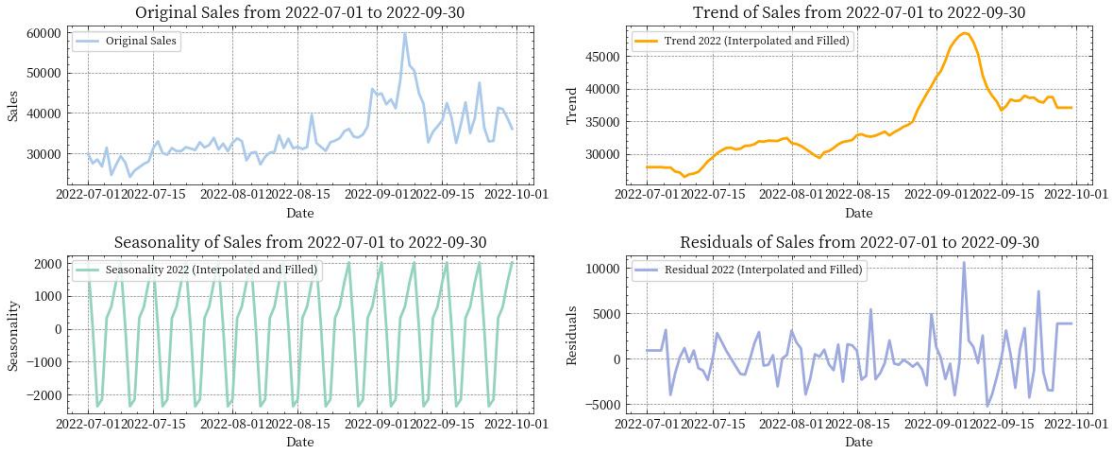


图10 季节性分解结果

从季节性分解的结果来看，原始销售额在2022年7月1日至2022年9月30日期间呈现出显著的波动趋势，特别是在8月底至9月初期间出现了一个明显的增长峰值。整体上，销售额随时间逐渐增加，尤其是在9月份显示出更大幅度的变化。趋势成分揭示了去除季节性和随机因素后的长期变化模式，显示销售额在整个时间段内总体呈上升态势，尤其在8月中旬开始加速增长，并在9月初达到顶峰后略有下降，这表明可能存在某种外部或内部因素导致这段时间内的需求激增。季节性成分则展示了重复出现的周期性模式，存在明显的每周一次的周期性波动，这可能是由于工作日与周末之间的差异造成的；此外，在特定日期附近出现了更大的波动幅度，例如在8月的最后一周和9月的第一周之间，这可能对应于某个特殊事件或者节假日的影响。残差成分代表了无法归因于趋势或季节性的剩余变动部分，大多数情况下保持相对稳定的小幅波动，但在某些时期如8月末至9月初则表现出较大的离群值，这些大的残差可能指示着未捕捉到的重要事件或者是数据中的噪声。

接下来我们构建了LSTM模型以及其延展模型双向LSTM来进行预测：

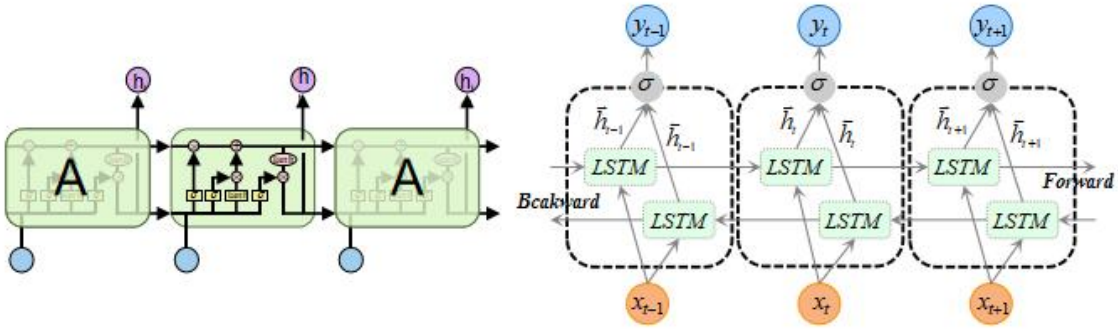


图11 LSTM系列模型

5.3.3 基线模型性能比较

我们统计了所选的三种基线模型的性能以供比较：

表7 销量基线模型性能表

模型	平均R^2	平均MSE	平均RMSE	平均MAE
ARIMA	**0.60**	**1953530.71**	**4220.95**	**2610.58**
LSTM	-0.03	3512360.21	4503.6	3567.3
双向LSTM	0.13	2898635.3	4501.3	3451.2

根据提供的预测性能数据，ARIMA模型在三种模型（ARIMA、LSTM和双向LSTM）中表现最佳。具体来说，ARIMA模型的平均R^2值为0.60，表明它能够较好地解释数据中的变异；同时，其平均MSE为1,953,530.71，平均RMSE为

4220.95，平均MAE为2610.58，这些误差指标均显著低于其他两种模型，显示出较高的预测精度。相比之下，LSTM模型的性能较差，表现为负的R^2值（-0.03），意味着其预测效果甚至不如简单使用数据集平均值作为预测，且其MSE、RMSE和MAE分别高达3,512,360.21、4,503.6和3,567.3，显示了非常大的预测误差。双向LSTM虽然在R^2值上有所改进（0.13），但其MSE、RMSE和MAE仍然非常高（分别为2,898,635.3、4,501.3和3,451.2），表明其预测精度仍有很大提升空间。总体而言，ARIMA模型不仅在解释数据变异方面表现出色，而且在预测准确性上也明显优于LSTM和双向LSTM，是当前数据集下更优的选择，于是我们接下来专注于ARIMA模型进行最终的预测。

5.3.4 VMD+ARIMA序列预测

要使用ARIMA模型进行最终预测，我们要先证明模型的有效性，下面以两个类别为例：

• Ljung-Box 检验（LB检验）：

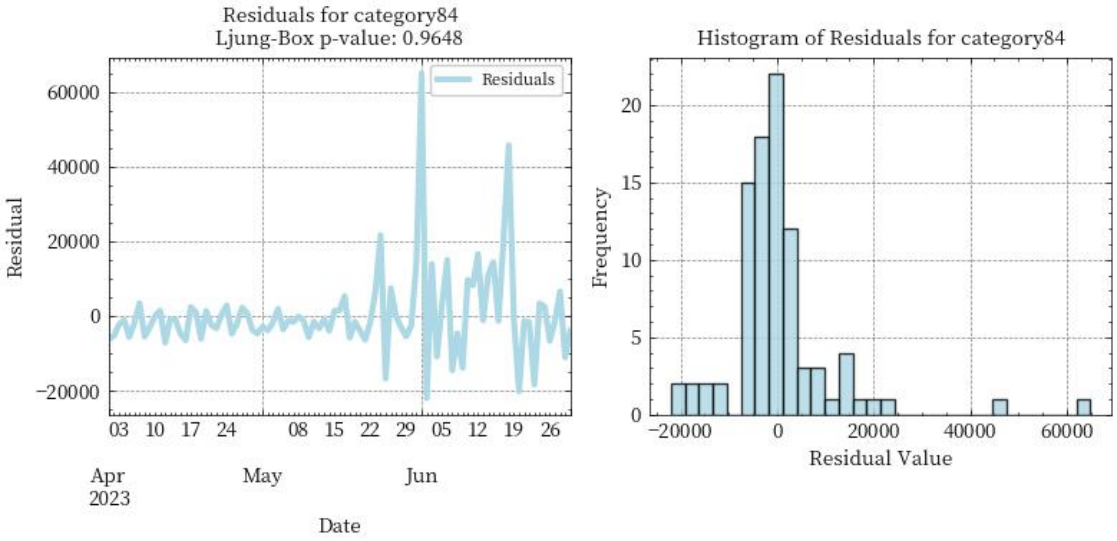
Ljung-Box 检验是一种用于检验时间序列数据中是否存在自相关性的统计检验方法。它主要用于检验残差（Residuals）是否独立，即残差中是否存在显著的自相关性。

Ljung-Box 检验的原假设和备择假设：

原假设H0：残差中不存在自相关性，即残差是独立的。

备择假设H1：残差中存在自相关性，即残差不是独立的。

于是我们对残差列进行了LB检验，同时我们绘制出了残差的直方图：



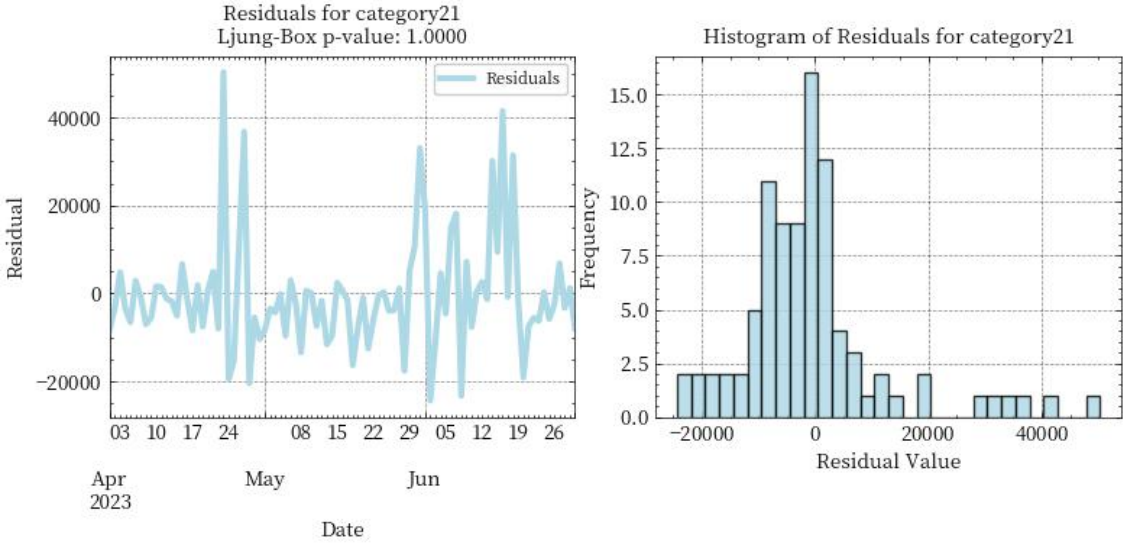


图12 ARIMA模型残差检验图

从类别84的残差分析结果来看，残差时序图显示了大部分时间内的残差值围绕零点波动，表明模型在这些时间段内具有相对稳定的预测性能。然而，在某些时间节点（如大约5月29日左右）出现了显著偏离零点的高峰，这可能暗示了在这些特定时期内存在异常事件或模型预测能力较弱。残差直方图则揭示了残差值的分布情况，从图中我们看出，残差集中于零点，近似服从正态分布，此外，Ljung-Box检验的p值表明残差序列没有显著的自相关性，这是一个积极的迹象，因为理想的残差应该是无关联的白噪音。**综合来看，模型可以在大多数情况下能够较好地预测销售数据，且模型的残差是近似服从线性模型的残差假设的，**我们接下来继续改良模型的性能。

• 变分模态分解

VMD 算法是一种时频分析算法，能够将信号一次性分解成多个单分量调幅调频信号，避免了迭代过程中遇到的端点效应和虚假分量问题。该算法可以有效处理非线性、非平稳的GNSS 高程时间序列，但由于它对噪声敏感，处理存在噪声的 GNSS 高程时间序列时，可能出现模态混叠现象。VMD 的分解过程即变分问题的求解过程，在该算法中，本征模态函数(intrinsic mode function,IMF)被定义为一个有带宽限制的调幅-调频函数，VMD 算法的功能便是通过构造并求解约束变分问题，将原始信号分解为指定个数的 IMF 分量。

假设将一个信号分解为K个IMF 分量， VMD 算法分解的具体流程如下 (1)通过 Hilbert 变换，得到每个模态分量 $\mu_K(t)$的解析信号，进而得到其单边频谱为

$$\left[\delta(t)+\frac{j}{\pi t}\right]\cdot\mu_\kappa(t)$$

对各模态解析信号预估一个中心频率$e^{-j\omega_K t}$,将每个模态的频谱调制到相应的基频带

$$\left[\left(\delta(t)+\frac{j}{\pi t}\right)\cdot\mu_K(t)\right]e^{-j\omega_K t}$$

计算上述解调信号梯度平方L 的范数， 估计出各模态信号带宽，受约束的变分问题为

$$\min_{\langle\mu_K\rangle,\langle\omega_K\rangle}\left\{\sum_K \|d,\left[\left(\delta(t)+\frac{j}{\pi t}\right)\cdot\mu_K(t)\right]e^{-j\omega_K t}\|_2^2\right\}$$

$$\sum_\kappa \mu_\kappa = f$$

式中，μ_κ代表分解得到的K个 IMF 分量；ω_κ表示各模态对应的中心频率。为了求解该约束性变分问题，引人二次惩罚因子α和拉格朗日乘法算子$\lambda(t)$,将约束性变分问题变为非约束性变分问题。扩展的拉格朗日表达式为

$$L(\{\mu_\kappa\},\{\omega_\kappa\},\lambda)=$$

$$\alpha\sum_K \|\partial_t\left[\left(\delta(t)+\frac{j}{\pi t}\right)\bullet\mu_\kappa(t)\right]e^{-j\omega_K t}\|_2^2+$$

$$\|f(t)-\sum_\kappa \mu_\kappa(t)\|_2^2+\left\langle\lambda(t),f(t)-\sum_\kappa \mu_\kappa(t)\right\rangle$$

式中，α 为二次惩罚因子；$\lambda(t)$为拉格朗日乘法算子。其中，α 可在高斯噪声存在的情况下，保证信号的重构精度，通常取拉格朗日算子使得约束条件保持严格性。利用乘法算子交替方向法解决以上无约束变分问题，通过交替更新$\mu_K^{n+1},\omega_K^{n+1}$和$\lambda^{n+1}$寻求扩展拉格朗日表达式的“鞍点”。

在时间序列分析中，使用 Variational Mode Decomposition (VMD) 对原始信号进行分解时，需要选择合适的分解模式数量K。选择合适的K值可以提高分解效果，从而提高后续预测模型的性能，于是我们设计了以下的K值选择策略

表8 最优K值选择策略

***Step* 1** 定义K值的范围：选择一个K值的范围$K_{range}=\{K_1,K_2,...,K_n\}$,其中$K_i$是整数。

***Step* 2** 初始化最小误差和最优K 值：初始化最小误差min_error为一个较大的值。初始化最优K值 *best_K*为空。

***Step* 3** 对于每个K 值K_i 在K_{range} 中：对原始信号x进行 VMD 分解，得到分解后的信号u,其中u是一个$K_i\times N$的矩阵，N是信号长度。

***Step* 4** 确保分解后的信号长度与原始信号一致：如果u的长度不等于原始信号长

度N,则截取u的前N个元素。

***Step* 5** 计算所有 IMF 的和：计算所有 IMF 的和$r = \sum_{k=1}^{K_i} u_k$,其中u_k是第k个IMF

。

***Step* 6** 计算误差：计算原始信号 x 和重构信号 r 之间的均方误差(MSE):

$$error = MSE(x[:-1], r)$$

其中$x[:-1]$表示原始信号去掉最后一个元素。

***Step* 7** 更新最优K值：如果当前误差 *error* 小于最小误差$min\ error$,则更新最小误差和最优K值：

$$min\ error = error \qquad best\ \mathrm{K}{=}K_i$$

接下来我们展示前两个类别的分解结果：

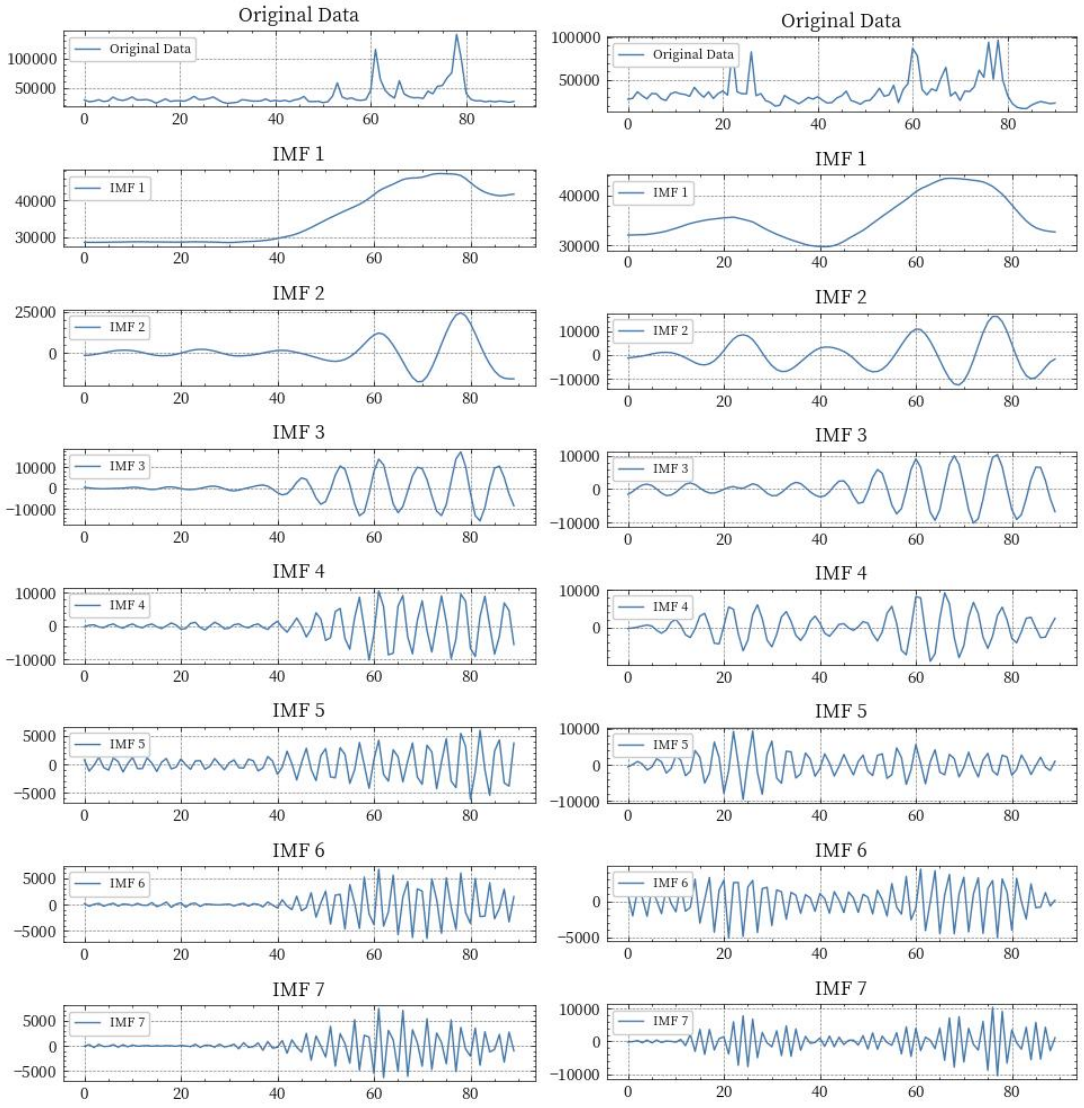


图13 变分模态分解的IMF分量

我们通过VMD将原始的复杂的时间序列分解为了简单的时间序列，然后再采用ARIMA对每个时间序列，最终预测结果如下：

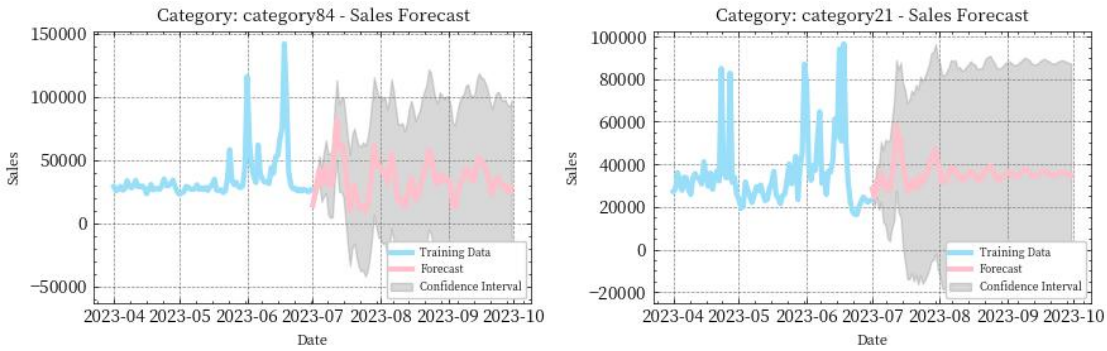


图14 VMD+ARIMA预测结果

表9 销量预测模型性能表

模型	平均R^2	平均MSE	平均RMSE	平均MAE
ARIMA	0.60	1953530.71	4220.95	2610.58
LSTM	-0.03	3512360.21	4503.6	3567.3
双向LSTM	0.13	2898635.3	4501.3	3451.2
VMD+ARIMA	**0.91**	**472201.41**	**1469.18**	**961.19**

结果显示，VMD+ARIMA模型在所有指标上都表现出了最佳的预测性能，具有最高的R^2值和最低的MSE、RMSE、MAE值。这表明VMD+ARIMA模型能够最准确地捕捉和解释数据中的变异性，提供更为可靠的预测结果。

VMD+ARIMA模型的优异表现可能归因于以下几个原因：首先，**变分模态分解（VMD）作为一种预处理技术，能够有效地分解原始时间序列信号，提取内在的模态分量，从而增强了ARIMA模型对非线性和非平稳性的适应能力。**其次，ARIMA模型本身在处理具有明显季节性和趋势性的时间序列数据时表现出色，而VMD的分解过程可能进一步突出了这些特性，使得ARIMA模型能够更好地拟合数据。相比之下，LSTM和双向LSTM模型虽然在理论上能够处理复杂的时间序列数据，但在本研究中并未展现出预期的性能。这可能是因为LSTM模型对数据和参数的敏感性较高，需要更精细的调优和更充分的训练。此外，双向LSTM虽然在结构上考虑了时间序列的前后信息，但在本研究中并未显著提升预测性能，这可能与数据量不够有关。

总体而言，VMD+ARIMA模型在预测时间序列数据方面的优势表明，结合先进的信号处理技术和经典的统计模型可以有效提升预测精度。

最终预测结果如下

表10 销量预测结果

类别	7.1	7.11	7.21	7.31	8.11	8.21	8.31	9.11	9.21
category 1	23.97	25.06	24.71	24.59	24.65	24.68	24.71	24.73	24.77
category 31	0	0	0	0	0	0	0	0	0
category 61	1682.05	1722.84	1771.67	1809.36	1843.89	1870.96	1893.97	1915.40	1931.63
category 91	0	0.05	0.24	0.40	0.55	0.67	0.75	0.84	0.91
category 121	734.89	1039.93	1080.54	1097.86	1106.18	1108.93	1109.39	1108.58	1107.26
category 151	1188.57	1784.09	1799.51	1800.42	1797.45	1793.27	1788.55	1783.19	1778.36
category 181	2.95	2.40	2.65	2.82	2.88	2.97	3.03	3.11	3.15
category 211	10.85	16.58	18.68	19.85	20.63	21.05	21.33	21.52	21.65
category 241	844.91	774.37	763.28	764.04	767.86	771.38	774.27	776.67	778.11
category 271	114.47	166.17	174.25	179.29	179.50	180.07	180.16	180	179.79
category 301	24.33	24.47	25.58	26.40	26.95	27.25	27.44	.27.57	27.64
category 331	483.19	507.26	517.58	521.81	523.42	523.52	522.99	522.06	521.06

6 问题二模型建立与求解

6.1 数据预处理

由于问题二使用的是问题一预测的数据，而预测的库存量和销量大都为浮点数，且有些为负数，这显然是不符合要求的，所以我们对预测数据进行如下处理：

- 提取平均销量

设Z_{ij}表示第i个品类在第j天的销售预测值，则日均销量A_i可以表示为：

$$A_i = round\left(\frac{\sum_{j=1}^{n-1} Z_{ij}}{n-1}\right)$$

其中，n是除去品类列和最后一个非数值列后的总列数。

- 提取平均月库存量

设X_{ij}表示第i个品类在第j个月份的库存量，则平均库存量Y_i可以表示为：

$$Y_i = round\left(\frac{X_{i7} + X_{i8} + X_{i9}}{3}\right)$$

• 创建完整关联度矩阵

设R是一个关联度矩阵，其中R_{ij}表示品类i与品类j之间的关联度。初始关联度矩阵可以表示为：

$$R_{ij} = \sum \text{关联度}_k, \text{如果存在记录使得 品类}1_k = i \text{且品类}2_k = j$$

$$\text{否则} R_{ij} = 0$$

确保包含所有品类后的关联度矩阵R'可以表示为： C是所有品类的集合。$C_f = sorted(C)$是完整品类列表：

$$R' = reindex(R, rows = C_f, cols = C_f)$$

$$R'_{ij} = 0, \text{如果}(i,j)\text{在原始矩阵中不存在}$$

最后我们将销量以及库存量都进行取整，小于0的部分我们置0，下面我们可视化一些处理结果：

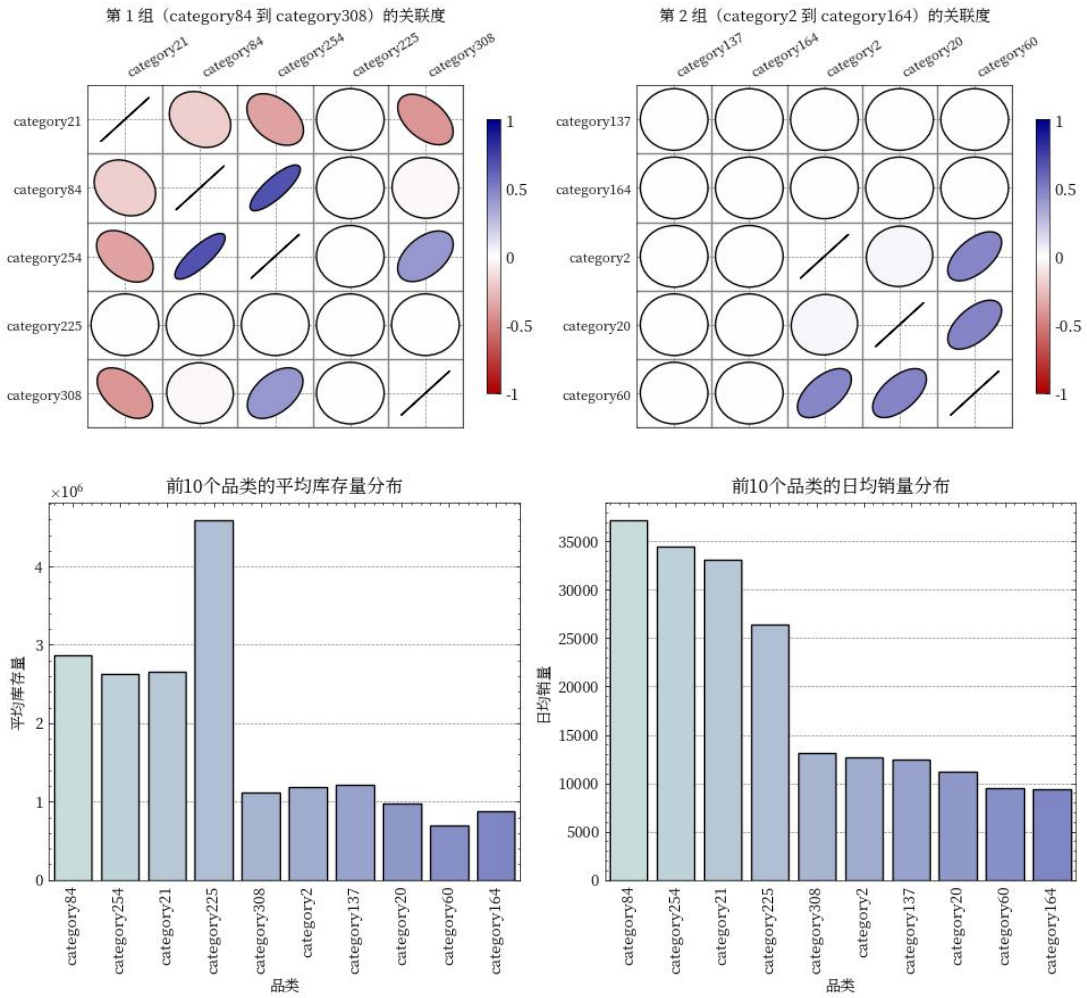


图15 预处理结果可视化

在数据预处理过程中，处理负数、取整以及构建品类关联度矩阵是确保数据质量和一致性的重要步骤。首先，处理负数是为了确保销售量和库存量符合

现实意义。在实际业务场景中，这些数值通常是非负的，负数可能表示数据输入错误或系统异常。去除或修正负数可以提高数据质量，避免因异常值导致的误判或错误结论，并且满足许多统计模型和优化算法对非负数据的假设。其次，取整操作确保了数据与实际操作的一致性，因为在实际中销售量和库存量通常是整数。取整不仅可以简化后续计算，尤其是在涉及离散决策变量时，还能避免小数点后的误差累积，从而提高数据的一致性和准确性。最后，构建品类关联度矩阵有助于发现不同品类之间的潜在关系，**综上所述，这些预处理步骤有利于提高数据的质量和一致性**。

6.2 ILP模型的建立

问题2的目标是基于问题1的预测结果，为每个品类找到一个合适的仓库进行存放，即实现“一品一仓”的分仓方案。这个问题的核心在于如何在满足仓容和产能限制的前提下，综合考虑多个业务目标，如仓容利用率、产能利用率、总仓租成本、品类分仓数和品类关联度，来确定最优的分仓方案。

我们需要综合考虑以下几个目标：最大化仓容利用率。最大化产能利用率。最小化总仓租成本。最小化品类分仓数（即每个品类只放在一个仓库中）。最大化品类关联度。

于是我们建立了如下的整数规划模型（ILP）来进行求解：

目标函数：

$$\textbf{Minimize } \alpha_1 \cdot \text{仓租成本} + \alpha_2 \cdot \text{品类分仓数} - \alpha_3 \cdot \text{仓容利用率} - \alpha_4 \cdot \text{产能利用率} + \alpha_5 \cdot \text{品类关联度}$$

其中，$\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$是权重系数，用于平衡各个目标的重要性。

决策变量：$\cdot x_{ij}$:表示品类i是否存放在仓库j,如果是则为1，否则为0。

约束条件：

• 仓容限制：每个仓库的库存量不能超过其仓容上限:

$$\sum_i \text{预测库存量}_i \cdot x_{ij} \leq \text{仓容上限}_j$$

• 产能限制：每个仓库的出库量不能超过其产能上限:

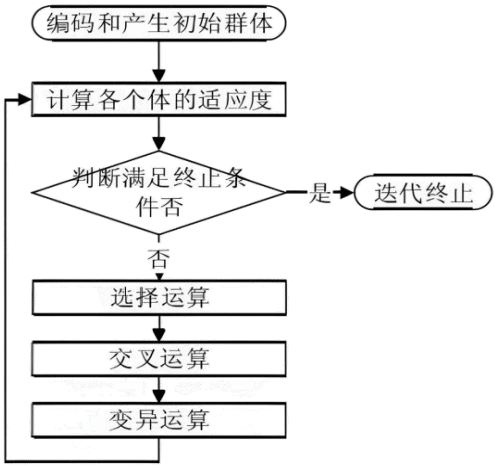
$$\sum_i \text{预测销量}_i \cdot x_{ij} \leq \text{产能上限}_j$$

• 一品一仓：每个品类只能放在一个仓库中:

$$\sum_j x_{ij} = 1 \quad \forall i$$

• 非负整数约束：决策变量x_{ij}为0或1:

遗传算法（Genetic Algorithm，GA）是一种模拟自然界中生物进化过程的优化技术，它利用了生物遗传和自然选择的原理来解决复杂的优化问题。在遗传算法中，每个潜在解被视为一个“个体”，而问题的解集则构成了一个“种群”。通过迭代过程，种群中的个体经过选择、交叉（或称为杂交）、变异等遗传操作，不断进化，以期找到问题的最优解或近似最优解。右图为遗传算法流程的可视化图：



$$x_{ij} \in \{0,1\} \quad \forall i,j$$

6.3 ILP模型的求解

由于deap库进行的是无约束优化，所以我们将约束条件转化为惩罚项加入到目标函数中，具体如下：

设f为原始的适应度函数，P为惩罚值，$penalty_flag$为一个布尔变量，表示是否触发了惩罚条件。

原始适应度函数f可以表示为：

$$f = \alpha_1 \cdot warehouse_cost + \alpha_2 \cdot category_spread - \alpha_3 \cdot warehouse_capacity_utilization - \alpha_4 \cdot production_capacity_utilization + \alpha_5 \cdot category_association_utilization$$

惩罚函数F可以表示为：

$$F = \begin{cases} f & \text{if } penalty\_flag = False \\ f + P & \text{if } penalty\_flag = True \end{cases}$$

其中，P是一个预设的较大的惩罚值，用于在触发惩罚条件时加到原始适应度函数f上

最终我们进行如下的参数设置：

表11 遗传算法参数设置

参数	值
penalty_value	10000
Alpha1	1
Alpha2	0.5
Alpha3	1
Alpha4	1
Alpha5	0.5
n	300
indpb	0.05

tournsize	3
gen	500
mate probability	0.5
mutate probability	0.2

接着我们使用这些参数来对优化模型进行求解，适应度变化曲线如下：

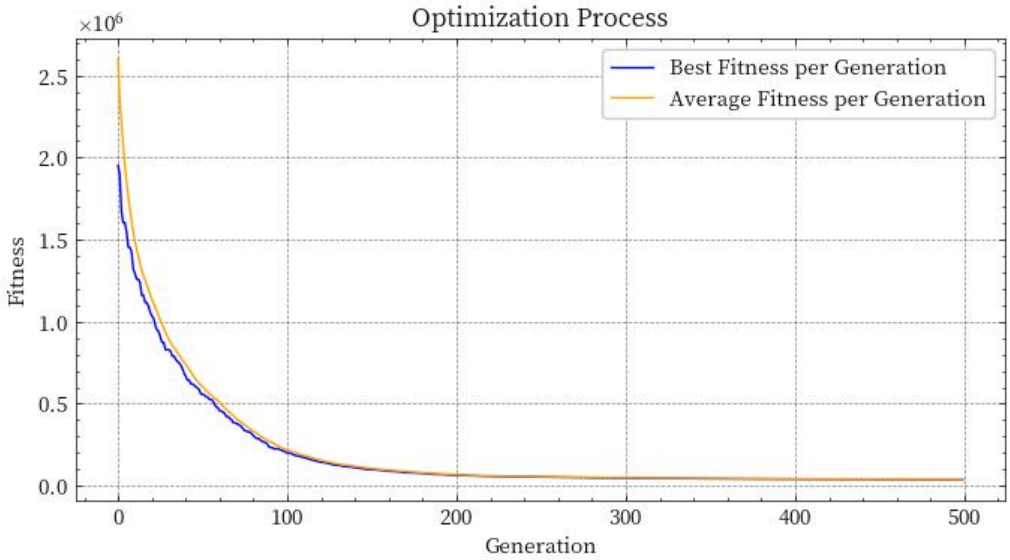


图16 遗传算法适应度变化曲线

从适应度曲线图中可以看到，在开始的几代内，无论是最佳适应度还是平均适应度都迅速下降，这意味着在早期，算法快速找到了比随机初始化更好的解决方案。随着代数的增加，这两条曲线逐渐趋于平稳，表明算法正在接近最优解或局部最优解。具体来说，在大约第100代之后，曲线的变化幅度显著减小，说明此时算法已经进入了收敛状态。下面是优化的分配方案：

表12 一品一仓分配方案

类别	Warehouse
category1	warehouse6
category31	warehouse6
category61	warehouse92
category91	warehouse93
category121	warehouse92
category151	warehouse70
category181	warehouse6
category211	warehouse84
category241	warehouse82
category271	warehouse92
category301	warehouse138
category331	warehouse84

7 问题三模型建立与求解

7.1 一品多仓分配模型

由于第二问和第三问类似，所以我们着重分析第三问相较于第二问有哪些改进点：

在第二问中，individual 是一个一维列表，每个元素代表某个品类被分配到的单个仓库编号。而在第三问中，为了让每个品类可以分配到多个仓库，我们修改individual 为一个二维列表，每个子列表包含某个品类被分配到的所有仓库编号：

·第二问 (单仓库分配):

$$\text{individual}=[w_0, w_1, ..., w_{N-1}]$$

其中w_i表示第i个品类被分配到的仓库编号。

·第三问 (多仓库分配):

$$\text{individual}=\left[[w_{0,0}, w_{0,1}, ...], [w_{1,0}, w_{1,1}, ...], ..., [w_{N-1,0}, w_{N-1,1}, ...]\right]$$

其中$w_{i,j}$表示第i个品类被分配到的第j个仓库编号。

同时第三问相较于第二问又多了以下两个约束条件：

条件 1:同件型商品尽量放在同一个仓库中：

$$\text{penalty flag item}=\bigvee_{j=0}^{M-1}\big(\exists i,k \in \text{warehouse categories}_j: \text{category info.loc}[i,\text{'件型'}] \neq \text{category info.loc}[k,\text{'件型'}]\big)$$

条件 2: 同高级品类商品尽量放在同一个仓库中:

$$\text{penalty flag\_category}=\bigvee_{j=0}^{M-1}(\exists i,k \in \text{warehouse categories}_j: \text{category info.loc}[i,\text{'高级品类'}] \neq \text{category info.loc}[k,\text{'高级品类'}])$$

所以我们修改了第二问的惩罚机制：

$$penalty\_flag=penalty\_flag\_item \vee penalty\_flag\_category$$

详细解释

• 条件 1:同件型商品尽量放在同一个仓库中:对于每个仓库j,检查所有被分配到该仓库的品类。如果存在任意两个品类i和k,它们的件型不同，则设置 penalty_flag_item 为 True。

• 条件 2: 同高级品类商品尽量放在同一个仓库中 ：对于每个仓库j,检查所有被分配到该仓库的品类。如果存在任意两个品类i和k ,它们的高级品类不同，则设置 penalty_flag_category 为 True

我们将最终的优化模型进行汇总：
目标函数：

$$\textbf{Minimize } \boldsymbol{\alpha_1} \cdot \text{仓租成本} + \boldsymbol{\alpha_2} \cdot \text{品类分仓数} - \boldsymbol{\alpha_3} \cdot \text{仓容利用率} - \boldsymbol{\alpha_4} \cdot \text{产能利用率} + \boldsymbol{\alpha_5} \cdot \text{品类关联度}$$

其中，$\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$是权重系数，用于平衡各个目标的重要性。

决策变量：$\cdot x_{ij}$:表示品类i是否存放在仓库j,如果是则为1，否则为0。

约束条件：

• 仓容限制：每个仓库的库存量不能超过其仓容上限:

$$\sum_i \text{预测库存量}_i \cdot x_{ij} \leq \text{仓容上限}_j$$

• 产能限制：每个仓库的出库量不能超过其产能上限:

$$\sum_i \text{预测销量}_i \cdot x_{ij} \leq \text{产能上限}_j$$

• 一品多仓：每个品类只能放在三个仓库以内：

$$\sum_j x_{ij} <= 3 \quad \forall i$$

• 非负整数约束：决策变量x_{ij}为0或1:

$$x_{ij} \in \{0,1\} \quad \forall i, j$$

• 件型约束：同件型商品尽量放在同一个仓库中

$$\text{penalty flag item} = \bigvee_{j=0}^{M-1} \left(\exists i, k \in \text{warehouse categories}_j : \text{category info.loc}[i, \text{'件型'}] \neq \text{category info.loc}[k, \text{'件型'}]\right)$$

• 高级品类约束：同高级品类商品尽量放在同一个仓库中

$$\text{penalty flag\_category} = \bigvee_{j=0}^{M-1} (\exists i, k \in \text{warehouse categories}_j : \text{category info.loc}[i, \text{'高级品类'}] \neq \text{category info.loc}[k, \text{'高级品类'}])$$

同时，为了最大化品类关联度，我们进行如下的参数设置：

表13 遗传算法参数设置

参数	值
penalty_value	10000
Alpha1	1
Alpha2	1
Alpha3	1
Alpha4	1
Alpha5	**2**
n	300
indpb	0.05
tournsize	3
gen	500
mate probability	0.5
mutate probability	0.2

遗传算法优化结果如下：

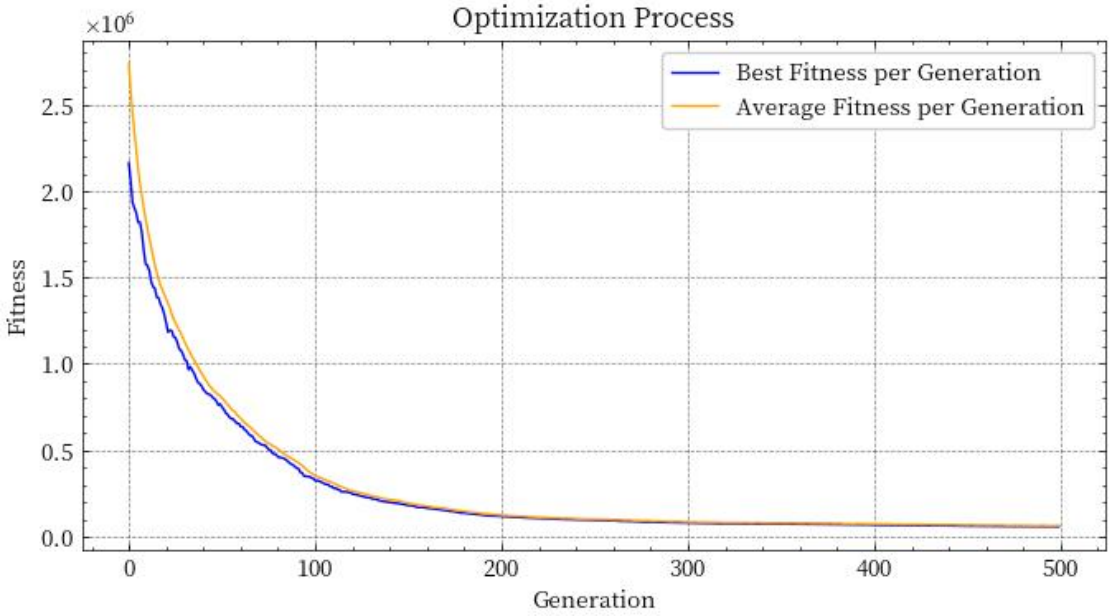


图17 遗传算法适应度变化曲线

与一品一仓类似，在早期，算法快速找到了比随机初始化更好的解决方案。随着代数的增加，这两条曲线逐渐趋于平稳，表明算法正在接近最优解或局部最优解。具体来说，在大约第100代之后，曲线的变化幅度显著减小，说明此时算法已经进入了收敛状态。下面是优化的分配方案：

表14 一品多仓分配方案

类别	Warehouse	Warehouse	Warehouse
category1	warehouse98		
category31	warehouse55		
category61	warehouse70	Warehouse67	
category91	warehouse71	warehouse82	
category121	warehouse59		
category151	warehouse94	warehouse107	
category181	warehouse70,	Warehouse58	
category211	warehouse8	Warehouse131	warehouse125
category241	warehouse80		
category271	warehouse80	warehouse82	
category301	warehouse30		
category331	warehouse48	Warehouse91	

7.2 一基于蒙特卡洛模拟的灵敏度分析

蒙特卡洛模拟是一种计算方法，它通过重复随机抽样来获得可能的结果范围，并用于风险分析。在敏感性分析中，蒙特卡洛方法可以用来评估模型输出对输入参数不确定性的敏感性。这种方法通过模拟输入参数的随机变化来观察这些变化如何影响模型的输出。

由于计算资源有限，所以我们仅对五个指标的系数进行敏感性分析：
权重参数范围：

alpha1：(0.5, 1.5)
alpha2：(0.2, 1.0)
alpha3：(0.5, 1.5)
alpha4：(0.5, 1.5)
alpha5：(0.2, 1.0)

蒙特卡洛模拟次数:20

每次模拟都会随机抽取一组权重参数，并运行遗传算法以找到最佳适应度,部分结果如下：

表15 灵敏度分析结果

alpha1	alpha2	alpha3	alpha4	alpha5	best_fitness
1.256210369	0.271115227	1.494339289	1.176503077	0.726385062	1660124.842
0.512184909	0.736354531	0.584493711	0.62123793	0.974020268	1464214.518
0.710087352	0.805308642	0.59664148	1.262650228	0.807789204	1704009.203
0.818638015	0.244715425	0.950318973	1.427265454	0.606987801	1848571.179
0.615739553	0.696287108	1.361453748	1.107950859	0.63661838	1639431.061
0.674187401	0.465679336	0.901470756	0.91174434	0.437310175	1673230.684

0.908948825	0.764678443	0.635012542	1.162397265	0.730012075	1615869.604
1.310280474	0.843232557	1.244356158	0.676899655	0.308686737	1468135.556

我们绘制出了模拟结果的热力图如下：

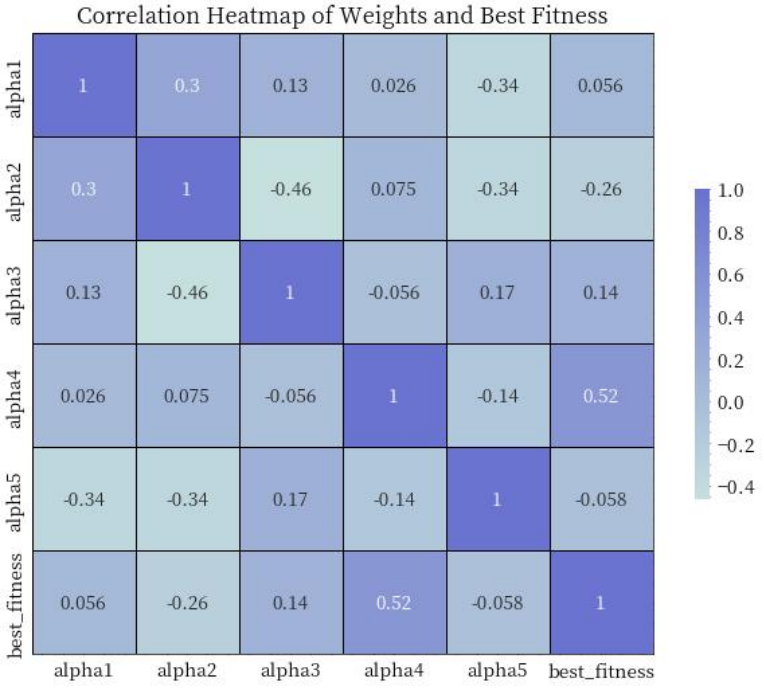


图18 参数相关性图

根据相关性热力图可以看出，Alpha1与Alpha2呈弱正相关（r=0.3），与Alpha3呈弱正相关（r=0.13），与Alpha4呈极弱正相关（r=0.026），与Alpha5呈弱负相关（r=-0.34），并且与Best Fitness呈极弱正相关（r=0.056）。Alpha2与Alpha1呈弱正相关（r=0.3），但与Alpha3呈强负相关（r=-0.46），与Alpha4呈弱正相关（r=0.075），与Alpha5呈弱负相关（r=-0.34），并且与Best Fitness呈弱负相关（r=-0.26）。Alpha3与Alpha1呈弱正相关（r=0.13），与Alpha2呈强负相关（r=-0.46），与Alpha4呈极弱负相关（r=-0.056），与Alpha5呈弱正相关（r=0.17），并且与Best Fitness呈弱正相关（r=0.14）。Alpha4在所有参数中表现突出，它与Alpha1呈极弱正相关（r=0.026），与Alpha2呈弱正相关（r=0.075），与Alpha3呈极弱负相关（r=-0.056），与Alpha5呈弱负相关（r=-0.14），并且与Best Fitness呈较强正相关（r=0.52），这表明Alpha4的增加可能会显著提升最佳适应度。Alpha5与Alpha1呈弱负相关（r=-0.34），与Alpha2呈弱负相关（r=-0.34），与Alpha3呈弱正相关（r=0.17），与Alpha4呈弱负相关（r=-0.14），并且与Best Fitness呈极弱负相关（r=-0.058）。

总体来看，Alpha4为一个与最佳适应度呈现较强正相关的参数，这意味着其值的增加可能会导致最佳适应度的提升。其他参数也都与最佳适应度之间的存在一定相关性，我们认为这是一个好的现象，有以下原因：

• 关键参数识别：如果某个权重参数与最佳适应度有显著的相关性，这表明该参数对最终结果有重要影响。通过这种相关性，我们可以识别出哪些参数是优化过程中的关键因素。

• 参数间的协同作用：当权重参数之间存在相关性时，这可能意味着它们之间存在某种协同效应。例如，某些参数组合可能会带来更好的整体性能。理解这些协同效应有助于设计更有效的优化策略。

• 鲁棒性：理解参数之间的相关性有助于构建更加鲁棒的模型。例如，如果某些参数在一定范围内变化时，结果仍然保持稳定，这表明模型具有较好的鲁棒性。

8 模型评价与推广

8.1 模型的优点

1、对数据集进行充分挖掘，建立特征工程，同时考虑了数据的趋势特征和统计特性，**对数据集的处理比较完整。**

2、对销量以及库存量的预测分别采用了指数平滑以及ARIMA模型，**模型易于实现且可解释性强。**

3、采用多种优化算法进行比较，且通过灵敏度分析验证了加权模型的有效性。

8.2 模型的缺点

1、特征工程的数理特征提取仍有改进空间，后续可以对特征工程更好的处理。

2、受限于时间，各时序预测模型参数可以进一步寻优。

8.3 模型的推广

本研究提出的库存预测与分仓规划模型具有较强的通用性和灵活性，可广泛应用于各类电子商务平台以及其他涉及大量商品仓储管理的企业中。特别是对于那些正在经历快速增长阶段或面临季节性销售波动挑战的企业而言，这些模型能够帮助它们更好地理解市场需求趋势，合理安排资源分配，从而实现成本控制与服务品质之间的平衡。此外，随着物联网技术和自动化设备在物流领域的普及，未来可以进一步将本文中的理论成果与智能仓储系统相结合，推动整个供应链体系向智能化方向发展。

参考文献

[1] 单玉庭, 刘韬, 褚惟, 缪护. 遗传算法优化变分模态分解在轴承故障特征提取中的应用[J]. 噪声与振动控制, 2024, 44(1): 148-153.

[2] 王俊，李霞，周昔东，等. 基于VMD和LSTM的超短期风速预测[J]. 电力系统保护与控制，2020，48（11）：45-52.

[3] 基于分形优化的VMD和GA-BP的短期风速预测. 植物生态学报, 2022.

[4] 基于VMD-ARIMA-HGWO-SVR组合模型的港口集装箱吞吐量预测. 兰州大学图书馆，2018.

[5] 唐甜甜，周伟. 基于ARIMA和BP神经网络的供应链需求预测模型及其对比分析. 应用数学进展, 2021.

[6] 陈祥, 杨志强, 田镇, 等. GA-VMD 与多尺度排列熵结合的 GNSS 坐标时序降噪方法[J]. 武汉大学学报（信息科学版）, 2021.

[7] Adhikari, R., & Agrawal, R. K. (2013). Time series forecasting using ARIMA model. International Journal of Computer Applications, 70(16), 23-28

附录

No: 1	Filename: 库存量预测

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# 选择前三个品类的数据
selected_categories = inventory_df['品类'].unique()[:3]

# 定义要使用的指数平滑方法
def exponential_smoothing(data, alpha):
    smoothed = [data[0]]
    for i in range(1, len(data)):
        smoothed.append(alpha * data[i] + (1 - alpha) * smoothed[-1])
    return pd.Series(smoothed, index=data.index)

# 初始化用于存储所有品类的评估指标的列表
all_es_metrics = []

# 对每个选定的品类绘制时间序列图并进行预测
for i, category in enumerate(selected_categories):
    # 筛选出当前品类的数据并排序
    category_data = inventory_df[inventory_df['品类'] == category].sort_index()

    # 初始化评估指标列表
    es_metrics = []

    # 尝试不同的alpha值
    alphas = np.arange(0.1, 1.0, 0.1)
    for alpha in alphas:
        # 指数平滑
        es_predictions = exponential_smoothing(category_data['库存量'], alpha)
        y_true_es = category_data['库存量'][es_predictions.index]

        # 计算评估指标
        mse_es = mean_squared_error(y_true_es, es_predictions)
        rmse_es = np.sqrt(mse_es)
        mae_es = mean_absolute_error(y_true_es, es_predictions)
        r2_es = r2_score(y_true_es, es_predictions)

        # 存储评估指标
        es_metrics.append({'Alpha': alpha, 'MSE': mse_es, 'RMSE': rmse_es, 'MAE': mae_es, 'R2': r2_es})

    # 将评估指标转换为DataFrame
    es_metrics_df = pd.DataFrame(es_metrics)

    # 找到最佳的alpha值
    best_alpha = es_metrics_df.loc[es_metrics_df['R2'].idxmax()]['Alpha']
    print(f"对于品类 {category}，最佳的alpha值是: {best_alpha:.2f}")

    # 使用最佳alpha值进行指数平滑
    best_es_predictions = exponential_smoothing(category_data['库存量'], best_alpha)
    y_true_best_es = category_data['库存量'][best_es_predictions.index]

    # 计算最佳alpha下的评估指标
    mse_best_es = mean_squared_error(y_true_best_es, best_es_predictions)
    rmse_best_es = np.sqrt(mse_best_es)
    mae_best_es = mean_absolute_error(y_true_best_es, best_es_predictions)
    r2_best_es = r2_score(y_true_best_es, best_es_predictions)

    # 存储当前品类的最佳评估指标
    all_es_metrics.append({'Category': category, 'MSE': mse_best_es, 'RMSE': rmse_best_es, 'MAE': mae_best_es,
'R2': r2_best_es})
```

```
    # 打印最佳alpha下的评估指标
    print(f"\n对于品类 {category}，使用最佳alpha值时的评价指标:")
    print(f"最大R²: {r2_best_es:.2f}")
    print(f"平均MSE: {mse_best_es:.2f}")
    print(f"平均RMSE: {rmse_best_es:.2f}")
    print(f"平均MAE: {mae_best_es:.2f}")

    # 绘制实际库存量与预测值的图表
    plt.figure(figsize=(12, 6))
    plt.plot(category_data.index, category_data['库存量'], label='Actual')
    plt.plot(best_es_predictions.index, best_es_predictions, label=f'Exponential Smoothing
(alpha={best_alpha:.2f})', linestyle='--')
    plt.title(f'Inventory Prediction for Category: {category}')
    plt.xlabel('Date')
    plt.ylabel('Inventory Level')
    plt.legend()
    plt.show()

    # 预测未来三个月
    last_known_value = best_es_predictions.iloc[-1]
    future_months_predictions = []
    for _ in range(3):
        next_month_prediction = best_alpha * last_known_value + (1 - best_alpha) * last_known_value
        future_months_predictions.append(next_month_prediction)
        last_known_value = next_month_prediction

    print(f"\n未来三个月的库存量预测 (基于最佳alpha值 {best_alpha:.2f}):")
    print(future_months_predictions)

# 将所有品类的评估指标转换为DataFrame
all_es_metrics_df = pd.DataFrame(all_es_metrics)

# 计算所有品类的平均评估指标
avg_mse = all_es_metrics_df['MSE'].mean()
avg_rmse = all_es_metrics_df['RMSE'].mean()
avg_mae = all_es_metrics_df['MAE'].mean()
avg_r2 = all_es_metrics_df['R2'].mean()

# 输出所有品类的平均评估指标
print("\n所有品类的平均评估指标:")
print(f"平均MSE: {avg_mse:.2f}")
print(f"平均RMSE: {avg_rmse:.2f}")
print(f"平均MAE: {avg_mae:.2f}")
print(f"平均R²: {avg_r2:.2f}")
```

No: 2	Filename: 销量预测

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import warnings
from vmdpy import VMD

# 忽略警告
warnings.filterwarnings('ignore')

# 选择用于训练的时间范围
start_date = '2023-04-01'
end_date = '2023-06-30'

# 获取所有类别的列表
categories = sales_df['品类'].unique()

# 选择 ARIMA 参数 (p, d, q)
p, d, q = 1, 0, 1  # 减少 p 和 q 的值

# 创建一个字典来存储每个类别的评价指标
metrics_dict = {}

# 创建一个DataFrame来存储预测结果
forecast_df = pd.DataFrame()

# 对每个类别进行拟合
for category in categories:
    # 只保留当前类别的数据
    category_data = sales_df[sales_df['品类'] == category].sort_values(by='日期')

    # 选择指定时间范围的数据用于训练
    train_df = category_data[(category_data['日期'] >= start_date) & (category_data['日期'] <= end_date)]

    # 将 '日期' 列设置为索引
    train_df.set_index('日期', inplace=True)

    # 提取销量列
    sales_series = train_df['销量'].values
    print(len(sales_series))
    # 定义 K 值的范围
    K_range = np.arange(3, 8)

    # 初始化最小误差和最优 K 值
    min_error = float('inf')
    best_K = None

    # 进行误差分析
    for K in K_range:
        # 使用当前 K 值进行 VMD 分解
        u, u_hat, omega = VMD(sales_series, alpha, tau, K, DC, init, tol)

        # 确保分解后的信号长度与原始信号一致
        if u.shape[1] != len(sales_series):
            u = u[:, :len(sales_series)]

        # 计算所有 IMF 的和
        reconstructed_signal = np.sum(u, axis=0)

        # 调整长度：如果reconstructed_signal比sales_series短，则填充；如果长，则截断
        if len(reconstructed_signal) < len(sales_series[:-1]):
```

```
            # 填充
            reconstructed_signal = np.pad(reconstructed_signal, (0, len(sales_series[:-1]) - len(reconstructed_signal)),
mode='edge')
        elif len(reconstructed_signal) > len(sales_series[:-1]):
            # 截断
            reconstructed_signal = reconstructed_signal[:len(sales_series[:-1])]

        # 计算误差
        error = mean_squared_error(sales_series[:-1], reconstructed_signal)

        # 更新最优 K 值
        if error < min_error:
            min_error = error
            best_K = K

    print(f"Category: {category}, Best K: {best_K}, Min Error: {min_error}")

    # 初始化预测结果
    forecast_values = np.zeros(92)

    # 对每个 IMF 进行预测
    for i in range(best_K):
        imf = u[i, :]

        # 拟合 ARIMA 模型
        try:
            model = ARIMA(imf, order=(p, d, q))
            model_fit = model.fit()  # 使用不同的优化方法
        except Exception as e:
            print(f"Error fitting ARIMA model for category {category}, IMF {i}: {e}")
            continue

        # 进行未来销量的预测
        forecast_steps = 92
        forecast = model_fit.get_forecast(steps=forecast_steps)
        forecast_values += forecast.predicted_mean

    # 创建一个新的DataFrame来保存当前类别的预测结果
    future_dates = pd.date_range(start=train_df.index[-1] + pd.Timedelta(days=1), periods=forecast_steps, freq='D')
    category_forecast = pd.DataFrame({'日期': future_dates, '品类': category, '预测销量': forecast_values})

    # 将当前类别的预测结果添加到总预测结果DataFrame中
    forecast_df = pd.concat([forecast_df, category_forecast], ignore_index=True)
    # 提取'品类'列并去掉'category'前缀，转换为整数
    forecast_df['品类_排序'] = forecast_df['品类'].str.replace('category', '').astype(int)

    # 按照处理后的'品类_排序'列进行排序
    forecast_df_sorted = forecast_df.sort_values(by='品类_排序')

    # 删除辅助列，因为我们只在排序时需要它
    forecast_df_sorted.drop(columns=['品类_排序'], inplace=True)

    # 再次创建forecast_pivot，这次基于已排序的forecast_df_sorted
    forecast_pivot = forecast_df_sorted.pivot(index='品类', columns='日期', values='预测销量')
    forecast_pivot_copy = forecast_pivot.copy()

    # 修改行索引，去掉'category'前缀并转换为整数
    new_index = [int(idx.replace('category', '')) for idx in forecast_pivot_copy.index]
    forecast_pivot_copy.index = new_index

    # 按照行索引（即修改后的品类编号）进行排序
    forecast_pivot_copy.sort_index(inplace=True)

# 打印排序后的数据框
print(forecast_pivot_copy)
```

No: 3	Filename: 第二问

```
import pandas as pd
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpStatus
import chardet
# 读取文件的一部分内容以检测编码
with open('C:/Users/30766/Desktop/附件/附件1.csv', 'rb') as f:
    result = chardet.detect(f.read(10000))  # 读取前10000个字节进行检测
    encoding = result['encoding']
    print(f"Detected encoding: {encoding}")

# 读取数据
inventory_prediction = pd.read_csv('C:/Users/30766/Desktop/11.csv', encoding=encoding)
sales_prediction = pd.read_csv('C:/Users/30766/Desktop/exported_data.csv')
warehouse_info = pd.read_csv('C:/Users/30766/Desktop/附件/附件3.csv', encoding=encoding)
category_association = pd.read_csv('C:/Users/30766/Desktop/附件/附件4.csv', encoding=encoding)
category_info = pd.read_csv('C:/Users/30766/Desktop/附件/附件5.csv', encoding=encoding)

# 将仓库名称设置为索引
warehouse_info.set_index('仓库', inplace=True)
# 处理库存预测数据
# 重命名列
inventory_prediction.rename(columns={'Unnamed: 0': '品类'}, inplace=True)

# 计算三个月的平均库存量，并进行取整
inventory_prediction['平均库存量'] = inventory_prediction[['7月库存量', '8月库存量', '9月库存量
']].mean(axis=1).apply(lambda x: round(x))

# 处理销量预测数据
# 转置数据，使每行代表一个品类，每列代表一天的销量
sales_prediction = sales_prediction.set_index('Unnamed: 0').T
sales_prediction.index.name = '日期'
sales_prediction.reset_index(inplace=True)

# 将第一列设置为品类
sales_prediction.columns = ['品类'] + list(sales_prediction.columns[1:])

# 处理负数和取整
# 只对数值列进行取整处理
numeric_cols = sales_prediction.select_dtypes(include=['number']).columns
sales_prediction[numeric_cols] = sales_prediction[numeric_cols].applymap(lambda x: max(0, round(x)))

# 计算日均销量并取整
sales_prediction['日均销量'] = sales_prediction.iloc[:, 1:-1].mean(axis=1).apply(round)

# 处理品类关联度
# 创建一个关联度矩阵
association_matrix = pd.crosstab(
    index=category_association['品类1'],
    columns=category_association['品类2'],
    values=category_association['关联度'],
    aggfunc='sum'
).fillna(0)

# 确保 association_matrix 包含所有品类
all_categories = set(inventory_prediction['品类'])

# 创建一个完整的索引和列列表
full_categories = sorted(all_categories)

# 重新构建 association_matrix，确保所有品类都包含在内
association_matrix = association_matrix.reindex(index=full_categories, columns=full_categories).fillna(0)
```

```
import random
import pandas as pd
from deap import base, creator, tools, algorithms
import matplotlib.pyplot as plt
# 定义适应度函数
def evaluate(individual):
    # 初始化惩罚标志
    penalty_flag = False
    penalty_value = 10000  # 惩罚值
    # 计算仓租成本
    warehouse_cost = sum(warehouse_info.loc[warehouse_info.index[j], '仓租日成本'] for j in individual)

    # 计算品类分仓数
    category_spread = len(set(individual))

    # 计算仓容利用率
    warehouse_capacity_utilization = 0.0
    for j in range(len(warehouse_info)):
        capacity_used = sum(inventory_prediction.loc[i, '平均库存量'] for i in range(len(individual)) if individual[i]
== j)
        if capacity_used > warehouse_info.loc[warehouse_info.index[j], '仓容上限']:
            #print(f"Generation {gen}: 仓容限制被违反 (仓库 {j})")
            penalty_flag = True
            break
        warehouse_capacity_utilization += capacity_used / warehouse_info.loc[warehouse_info.index[j], '仓容上限']

    production_capacity_utilization = 0.0
    for j in range(len(warehouse_info)):
        capacity_used = sum(sales_prediction.loc[i, '日均销量'] for i in range(len(individual)) if individual[i] == j)
        if capacity_used > warehouse_info.loc[warehouse_info.index[j], '产能上限']:
            # 如果超过产能限制，加上惩罚
            #print(f"Generation {gen}: 产能限制被违反 (仓库 {j})")
            penalty_flag = True
            break
        production_capacity_utilization += capacity_used / warehouse_info.loc[warehouse_info.index[j], '产能上限']

    # 计算品类关联度
    category_names = inventory_prediction['品类'].tolist()
    category_association_utilization = sum(
        association_matrix.loc[category_names[i], category_names[k]]
        for i in range(len(individual))
        for k in range(len(individual))
        if individual[i] == individual[k]
    )

    # 检查每个品类是否只被分配到一个仓库
    unique_categories = set(individual)
    if len(unique_categories) != len(individual):
        # 如果某个品类被分配到多个仓库，则对适应度进行惩罚
        #print(f"Generation {gen}: 品类分配到多个仓库")
        penalty_flag = True

    # 综合目标函数
    alpha1 = 1
    alpha2 = 0.5
    alpha3 = 1
    alpha4 = 1
    alpha5 = 0.5
    fitness_value = (
        alpha1 * warehouse_cost +
        alpha2 * category_spread -
        alpha3 * warehouse_capacity_utilization -
        alpha4 * production_capacity_utilization +
        alpha5 * category_association_utilization
    )
    # 如果有惩罚标志，返回一个较大的惩罚值
```

```
    if penalty_flag:
        return (fitness_value + penalty_value,)

    return (fitness_value,)

# 创建适应度函数和个体
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# 初始化工具箱
toolbox = base.Toolbox()

# 定义采样方法
def sample_indices():
    indices = []
    for _ in range(len(inventory_prediction)):
        indices.append(random.randint(0, len(warehouse_info) - 1))
    return indices

# 注册属性生成器
toolbox.register("indices", sample_indices)

# 使用属性生成器初始化个体
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indices)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# 定义遗传操作
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate)

# 创建种群
pop = toolbox.population(n=300)  # 指定种群大小

# 记录每一代的最佳适应度和平均适应度
best_fitnesses = []
average_fitnesses = []

# 运行遗传算法
for gen in range(500):
    # 选择、交叉和变异
    offspring = toolbox.select(pop, len(pop))
    offspring = list(map(toolbox.clone, offspring))
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        if random.random() < 0.5:
            toolbox.mate(child1, child2)
            del child1.fitness.values
            del child2.fitness.values
    for mutant in offspring:
        if random.random() < 0.2:
            toolbox.mutate(mutant)
            del mutant.fitness.values

    # 评估新个体
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    # 替换种群
    pop[:] = offspring

    # 计算并记录最佳适应度和平均适应度
    fits = [ind.fitness.values[0] for ind in pop]
    best_fit = min(fits)
    avg_fit = sum(fits) / len(pop)
```

```
    best_fitnesses.append(best_fit)
    average_fitnesses.append(avg_fit)

    print(f"Generation {gen}: Best Fitness = {best_fit}, Average Fitness = {avg_fit}")

# 绘制优化过程
plt.figure(figsize=(10, 6))
plt.plot(best_fitnesses, label='Best Fitness per Generation', color='blue')
plt.plot(average_fitnesses, label='Average Fitness per Generation', color='orange')
plt.xlabel('Generation')
plt.ylabel('Fitness')
plt.title('Optimization Process')
plt.legend()
plt.grid(True)
plt.show()

# 输出结果
best_individual = tools.selBest(pop, k=1)[0]
for i, j in enumerate(best_individual):
    print(f"品类 {inventory_prediction['品类'][i]} 存放在仓库 {warehouse_info.index[j]}")

import optuna

# 定义 Optuna 的目标函数
def objective(trial):
    # 建议超参数
    cxpb = trial.suggest_uniform('cxpb', 0.1, 1.0)
    mutpb = trial.suggest_uniform('mutpb', 0.1, 1.0)
    ngen = trial.suggest_int('ngen', 10, 300)
    pop_size = trial.suggest_int('pop_size', 100, 500)

    # 创建种群
    pop = toolbox.population(n=pop_size)

    # 运行遗传算法
    algorithms.eaSimple(pop, toolbox, cxpb=cxpb, mutpb=mutpb, ngen=ngen, verbose=False)

    # 获取最佳个体的适应度
    best_individual = tools.selBest(pop, k=1)[0]
    best_fitness = best_individual.fitness.values[0]

    return best_fitness

# 创建 Optuna 研究
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)

# 输出最佳参数和结果
best_params = study.best_params
best_value = study.best_value
print(f"Best parameters: {best_params}")
print(f"Best value: {best_value}")
```

No: 4	Filename: 第三问

```
import pandas as pd
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpStatus
import chardet
# 读取文件的一部分内容以检测编码
with open('C:/Users/30766/Desktop/附件/附件1.csv', 'rb') as f:
    result = chardet.detect(f.read(10000))  # 读取前10000个字节进行检测
    encoding = result['encoding']
    print(f"Detected encoding: {encoding}")

# 读取数据
inventory_prediction = pd.read_csv('C:/Users/30766/Desktop/11.csv', encoding=encoding)
sales_prediction = pd.read_csv('C:/Users/30766/Desktop/exported_data.csv')
warehouse_info = pd.read_csv('C:/Users/30766/Desktop/附件/附件3.csv', encoding=encoding)
category_association = pd.read_csv('C:/Users/30766/Desktop/附件/附件4.csv', encoding=encoding)
category_info = pd.read_csv('C:/Users/30766/Desktop/附件/附件5.csv', encoding=encoding)

# 将仓库名称设置为索引
warehouse_info.set_index('仓库', inplace=True)
# 处理库存预测数据
# 重命名列
inventory_prediction.rename(columns={'Unnamed: 0': '品类'}, inplace=True)

# 计算三个月的平均库存量，并进行取整
inventory_prediction['平均库存量'] = inventory_prediction[['7月库存量', '8月库存量', '9月库存量']].mean(axis=1).apply(lambda x: round(x))

# 处理销量预测数据
# 转置数据，使每行代表一个品类，每列代表一天的销量
sales_prediction = sales_prediction.set_index('Unnamed: 0').T
sales_prediction.index.name = '日期'
sales_prediction.reset_index(inplace=True)

# 将第一列设置为品类
sales_prediction.columns = ['品类'] + list(sales_prediction.columns[1:])
```

```
# 处理负数和取整
# 只对数值列进行取整处理
numeric_cols =
sales_prediction.select_dtypes(include=['number']).colum
ns
sales_prediction[numeric_cols] =
sales_prediction[numeric_cols].applymap(lambda x:
max(0, round(x)))

# 计算日均销量并取整
sales_prediction['日均销量'] = sales_prediction.iloc[:, 1:-
1].mean(axis=1).apply(round)



# 处理品类关联度
# 创建一个关联度矩阵
association_matrix = pd.crosstab(
    index=category_association['品类1'],
    columns=category_association['品类2'],
    values=category_association['关联度'],
    aggfunc='sum'
).fillna(0)

# 确保 association_matrix 包含所有品类
all_categories = set(inventory_prediction['品类'])

# 创建一个完整的索引和列列表
full_categories = sorted(all_categories)

# 重新构建 association_matrix，确保所有品类都包含在
内
association_matrix =
association_matrix.reindex(index=full_categories,
columns=full_categories).fillna(0)

# 确保品类是字符串类型
category_info['品类'] = category_info['品类'].astype(str)

# 确保高级品类和件型是字符串类型
category_info['高级品类'] = category_info['高级品类
'].astype(str)
category_info['件型'] = category_info['件型'].astype(str)

# 合并品类信息和库存预测数据
inventory_prediction =
inventory_prediction.merge(category_info, on='品类',
how='left')

# 合并品类信息和销量预测数据
```

```
sales_prediction = sales_prediction.merge(category_info,
on='品类', how='left')


def evaluate(individual):
    # 初始化惩罚标志
    penalty_flag = False
    penalty_value = 10000  # 惩罚值


    # 计算仓租成本
    warehouse_cost =
sum(warehouse_info.loc[warehouse_info.index[j], '仓租
日成本'] for sublist in individual for j in sublist)


    # 计算品类分仓数
    category_spread = sum(len(set(sublist)) for sublist in
individual)


    # 计算仓容利用率
    warehouse_capacity_utilization = 0.0
    for j in range(len(warehouse_info)):
        capacity_used = sum(inventory_prediction.loc[i, '平
均库存量'] for i in range(len(individual)) for k in
individual[i] if k == j)
        if capacity_used >
warehouse_info.loc[warehouse_info.index[j], '仓容上限']:
            penalty_flag = True
            break
        warehouse_capacity_utilization += capacity_used /
warehouse_info.loc[warehouse_info.index[j], '仓容上限']


    # 计算产能利用率
    production_capacity_utilization = 0.0
    for j in range(len(warehouse_info)):
        capacity_used = sum(sales_prediction.loc[i, '日均销
量'] for i in range(len(individual)) for k in individual[i] if
k == j)
        if capacity_used >
warehouse_info.loc[warehouse_info.index[j], '产能上限']:
            penalty_flag = True
            break
        production_capacity_utilization += capacity_used /
warehouse_info.loc[warehouse_info.index[j], '产能上限']


    # 计算品类关联度
    category_names = inventory_prediction['品类'].tolist()
    category_association_utilization = sum(
        association_matrix.loc[category_names[i],
category_names[k]]
        for i in range(len(individual))
        for j in individual[i]
        for k in range(len(individual))
```

```
        for l in individual[k]
        if i != k and j == l
    )

    # 检查每个品类是否只被分配到不超过3个仓库
    category_count = {}
    for sublist in individual:
        for j in sublist:
            if j not in category_count:
                category_count[j] = 0
            category_count[j] += 1
    for count in category_count.values():
        if count > 3:
            penalty_flag = True
            break

    # 检查同件型和同高级品类的商品是否尽量放在同
一个仓库中
    for j in range(len(warehouse_info)):
        warehouse_categories = [i for i in
range(len(individual)) for k in individual[i] if k == j]
        if len(warehouse_categories) > 1:
            for i in warehouse_categories:
                for k in warehouse_categories:
                    if i != k:
                        if (category_info.loc[i, '件型'] !=
category_info.loc[k, '件型']) or (category_info.loc[i, '高级
品类'] != category_info.loc[k, '高级品类']):
                            penalty_flag = True
                            break

    # 综合目标函数
    alpha1 = 0.5
    alpha2 = 0.3
    alpha3 = 0.5
    alpha4 = 0.5
    alpha5 = 2.0
    fitness_value = (
        alpha1 * warehouse_cost +
        alpha2 * category_spread -
        alpha3 * warehouse_capacity_utilization -
        alpha4 * production_capacity_utilization +
        alpha5 * category_association_utilization
    )

    # 如果有惩罚标志，返回一个较大的惩罚值
    if penalty_flag:
        return (fitness_value + penalty_value,)

    return (fitness_value,)
```

```
# 创建适应度函数和个体
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list,
fitness=creator.FitnessMin)

# 初始化工具箱
toolbox = base.Toolbox()

# 定义采样方法
def sample_indices():
    indices = []
    for _ in range(len(inventory_prediction)):
        indices.append([random.randint(0,
len(warehouse_info) - 1) for _ in range(random.randint(1,
3))])
    return indices

# 注册属性生成器
toolbox.register("indices", sample_indices)

# 使用属性生成器初始化个体
toolbox.register("individual", tools.initIterate,
creator.Individual, toolbox.indices)
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

# 定义遗传操作
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes,
indpb=0.05)
toolbox.register("select", tools.selTournament,
tournsize=3)
toolbox.register("evaluate", evaluate)

# 创建种群
pop = toolbox.population(n=300)  # 指定种群大小

# 记录每一代的最佳适应度和平均适应度
best_fitnesses = []
average_fitnesses = []

# 运行遗传算法
for gen in range(500):
    # 选择、交叉和变异
    offspring = toolbox.select(pop, len(pop))
    offspring = list(map(toolbox.clone, offspring))
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        if random.random() < 0.5:
            toolbox.mate(child1, child2)
```

```
            del child1.fitness.values
            del child2.fitness.values
    for mutant in offspring:
        if random.random() < 0.2:
            toolbox.mutate(mutant)
            del mutant.fitness.values

    # 评估新个体
    invalid_ind = [ind for ind in offspring if not
ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    # 替换种群
    pop[:] = offspring

    # 计算并记录最佳适应度和平均适应度
    fits = [ind.fitness.values[0] for ind in pop]
    best_fit = min(fits)
    avg_fit = sum(fits) / len(pop)

    best_fitnesses.append(best_fit)
    average_fitnesses.append(avg_fit)

    print(f"Generation {gen}: Best Fitness = {best_fit},
Average Fitness = {avg_fit}")

# 绘制优化过程
plt.figure(figsize=(10, 6))
plt.plot(best_fitnesses, label='Best Fitness per Generation',
color='blue')
plt.plot(average_fitnesses, label='Average Fitness per
Generation', color='orange')
plt.xlabel('Generation')
plt.ylabel('Fitness')
plt.title('Optimization Process')
plt.legend()
plt.grid(True)
plt.show()

# 输出结果
best_individual = tools.selBest(pop, k=1)[0]
for i, warehouses in enumerate(best_individual):
    print(f"品类 {inventory_prediction['品类'][i]} 存放在
仓库 {', '.join([warehouse_info.index[j] for j in
warehouses])}")
```