生产企业原材料的订购与运输

摘要

近年，随着世界全球化加深，传统的供应关系已经不再适应于当今生产需要，其中原材料供应是企业供应链的主要环节。本文提出了原材料供应数学模型的基础假设，给出企业对供应过程中订购与运输方案的建议。

对于问题一，我们通过对供货商供货特征的定量分析，取得描述供应商重要程度的评价指标，建立了基于熵权逼近理想解方法的供应商排序模型。其中，我们主要从供应商的产业规模、供货能力、违约率、波动情况，来衡量此供应商对该企业正常生产的重要性。为了定量并客观地评价每个供应商的重要程度，我们在模型中使用了熵权，以有效避免多因素评价模型当中确定权重时的主观性。通过比较对象与理想解和负理想解的距离，得到出对象与理想解的相似度，最后排序优选确定最重要的 50 家供应商。

对于问题二，我们建立线性规划模型以对订购方案进行优化，并制定出未来 24 周原材料最经济的订购方案。其中，在供应商的二次优选中，我们注意到每个供应商仅有选择与否两种状态，故采用 0–1 规划取得满足供应需求的最少的 44 家供应商；考虑到原材料储备对保障企业生产的重要意义，制定订购方案前，我们首先应用 LSTM 模型预测选定的供应商的供应水平，确保企业供应链稳定与生产安全，进一步优化订购方案。根据得到的订购方案制定损耗最小的转运方案，最后通过计算机程序仿真，以模拟优化后方案的实施效果。

对于问题三，由于企业为压缩生产成本，尽量多地采购 A 类和尽量少地采购 C 类原材料。我们对问题二中线性规划模型进行扩展，减少生产企业对 C 类原材料供应商的依赖，并将 A 类原材料的采购优先度提高，构建线性规划模型，解出最优采购方案和转运方案。最后由计算机仿真体现方案的实施效果。

对于问题四，我们不再考虑因企业生产技术有限产生的产量上限，而通过优化订购方案、转运方案最大化供应链的供应能力。而通过分析供货数据我们发现，每周转运商的总转运能力远少于供给商的供给能力。因此，我们以对转运商转运方案的优化为主，通过对模型的不断反思与改进，得出了未来 24 周的订购和转运方案。

最后我们对模型进行了中肯的评价和适当的推广。

关键字： TOPSIS 法　熵权法　LSTM 模型　线性规划

原材料是企业重要的经营性资产，是企业维持正常生产经营活动的基础，原材料管理的好坏直接关系到企业的营运资金利用效率，也关系到企业生产经营的持续性[1]。改良生产企业的原材料管理方案，能提高原材料的周转率，有效降低仓储成本，也有利于保障生产流程的正常运行，在较大程度上加强企业竞争力[2]。近年来，随着经济的稳步发展，企业在面对新冠疫情冲击的同时，还面临着更大规模、更高强度的竞争，优化原材料管理方案也较以往更具现实意义。

一、 问题分析

制造商为了达到每日生产指标，以满足市场需求，需要向供货商（提供企业所需的原材料）提交订单。由于运输、交付等问题，订购的原材料不能直接用于生产，而需先交付物流转运至仓库，以便企业生产时取用。

现实中，供应商在接到订单后，常因为生产能上限和供应优先度等问题调整原材料产量，导致实际按时提供的原料量与生产需要存在出入。而此时制造商已经完成原料采购，不能通过更改之前的订购方案来弥补缺少原料，只能使用已经转移至仓库中的原料进行加工。若原材料库存不足，原材料的缺口会导致生产线停摆，造成严重的经济损失。为了减轻原料供应量波动对生产的影响，生产企业可提前储存一些原料，减轻生产停摆带来的损失。但与此同时，过多的原材料积存会显著增加企业的仓储成本。因此，为了改良生产企业的原材料管理方案，可根据以往交易数据，推测该供应商供应量的波动幅度，以制定出更加灵活的订购方案，削弱仓储成本，为企业带来更大收益。

供应商提供 A、B、C 三种原材料，容易发现，A、B、C 三种类型的原材料在采购单价不相等的同时，生产单位商品的消耗量也是不同的。其中，A 类型和 C 类型原材料生产单位商品的采购费用相等，且略低于 B 类型原材料（由于相差小于 1%，本文暂且忽略），进而认为三者的本征采购单价相等。为了便于研究，本文先对原材料做了等价代换：将其供应量以原材料的商品生产能力表示，采购单价也以本征采购单价表示。为了最大化经济效益，帮助生产企业做出决策，本文建立了供应量可预测的原材料订购、库存方案优化模型。

1.1 问题一

题目中要求对附件一中 402 家供应商进行量化分析，并建立供应商选择的评价指导体系，确定 44 家最重要的供应商。供应商的重要程度受多种因素影响，通过查阅相关文献，利用定量分析的方法，我们选取出适当的指标并根据熵值赋予权重，建立基于熵权逼近理想解方法的供应商排序模型，最后对供应商进行排序优选。

1.2 问题二

题目中要求在问题一结果的基础上，选出能满足企业生产需求的最少供应商，针对这些供应商分别制定最经济的订购方案和损耗最小的转运方案。我们首先利用 0-1 规划选出必要的供应商，同时为了保证原材料供应量充足，我们运用 LSTM 模型预测出转运商的供应水平，并建立了原材料订购方案的线性规划模型。通过对供应过程的计算机仿真，分析订购和转运模型的实施效果。

1.3 问题三

题目中要求在尽量多地采购 A 类和尽量少地采购 C 类原材料的前提下，制定最经济的原材料订购方案和损耗最小的转运方案，并分析方案的实施效果。本文发现，对原材料种类的限制，能通过对问题二模型进行扩展，在订购量分配时体现企业对原材料的偏好，进而通过优化模型制定了订购和转运方案。模型求解后，仍经过计算机仿真，分析模型的优化效果。

1.4 问题四

题目中要求在供应商和转运商有限的情况下，评估企业每周产能的上限。本题取消生产企业每周产能上限的同时，也消除了原材料的库存问题。而通过分析供货数据不难发现，向企业输送的最大供给量主要取决于转运商的转运能力。本问题也转化为在转运商以最大能力运输原材料时，一家供应商每周供应的原材料尽量由一家转运商运输。因此，本文主要对转运商的转运方案进行优化，并据此给出未来 24 周的订购和转运方案。

最后我们对模型进行了中肯的评价和适当的推广。

二、 供应商的选择

通过分析过去五年内，生产企业向 402 家原材料供货商发起的订单数据，可发现其中一些更受该企业青睐，该企业在相同周期里向这些供应商发出更多订单。与之相反，也有一些供应商参与原材料供应较少。因此，为了制定该企业的最优订货策略，有必要筛选出对原材料供应影响较大的供应商，并减少对供应影响较少供应商的依赖。

与此同时，订货方案还受到供货商数量的影响，过多的供应商需要花费企业更多的成本，而且到货的不稳定因素增多，进而有可能影响企业的生产计划；供应商数量过少，虽然能够降低管理成本和订货成本，但是到货的时间和质量的风险增大，从风险规避的角度来说，供应商数量应该维持在高水平状态[3]。

综上所述，筛选出供应影响力较强的供应商，并结合生产需求选择合适数量的供应商，与之建立长期、稳定的合作供给关系，能够显著降低该企业的管理成本和资金链风

险。因此，本文通过量化分析供货特征和提出假设，建立并求解相应评价模型，为生产企业订购决策提供基础和指导。

2.1 指标的选取

为了对供应商的的供货特征进行量化分析，建立合理的评价模型，本文首先选取了以下的指标：

1. 订货次数 N_j:

   订货次数表示供应商过去 5 年向生产企业供货的次数，直接反映出该企业对其的依赖程度。订货次数 N_j 越大，表示企业对该供应商越认可。

$$\nu_{ij} = \begin{cases} 0, & \text{供应商 S}j \text{ 在第 } i \text{ 周没有订单,} \\ 1, & \text{供应商 S}j \text{ 在第 } i \text{ 周接到订单;} \end{cases} \tag{1}$$

   上式中，ν_{ij} 是表示某供应商在该周是否接到订单的逻辑变量。通过累加求和，得到定货次数

$$N_j = \sum_{i=001}^{240} \nu_{ij}. \tag{2}$$

2. 供应商 S$j(j = 001, 002, \cdots, 402)$ 近五年企业的订货总量 $\varkappa_j$

   企业会向重要的供货商订购大量货物。$\varkappa_j$ 大的供应商更受企业欢迎，对企业的重要性强。对于供应商 Sj，有

$$\varkappa_j = \sum_{i=001}^{240} \chi_{ij}, \tag{3}$$

   其中，χ_{ij} 是第 i 周，供货商 Sj 接到的订货量。

3. 供应商 S$j(j = 001, 002, \cdots, 402)$ 近五年企业的供货总量 Ψ_j

   供应商的供货量能反映供货量的供应能力，供应能力强的供货商能有力地保障企业生产，所以 Ψ_j 越大，相应的供货商越重要，与 $\varkappa_j$ 类似

$$\Psi_j = \sum_{i=001}^{240} \psi_{ij}, \tag{4}$$

   其中，ψ_{ij} 是第 i 周，供货商 Sj 对企业的供货量。

4. 平均供货偏差 ε_j:

   对企业来讲，越能按照订货量进行供货的供应商是越优质的。我们选择供货量和订货量的差占订货量的比值，表征供货偏差，即供应商 Sj 在第 i 周的供货偏差为

$$\varepsilon_{ij} = \frac{\chi_{ij} - \psi_{ij}}{\chi_{ij}}, \tag{5}$$

而平均供货偏差 ε_j 应为 ε_{ij} 的绝对值的平均值，即

$$\varepsilon_j = \frac{\sum_{i=001}^{240} |\varepsilon_{ij}|}{N_j}. \tag{6}$$

进而本文得到了供应商在供货时绝对偏差量的期望值

$$E_j = \chi_{ij} \times \varepsilon_j. \tag{7}$$

5. 单次最大供应量 Ψ'_j:

如果供应商能一次性向企业提供大量原材料，其就能在一定程度上向企业证明其的供应能力强大，更适合成为对于企业重要的供应商。即单次最大供应量 Ψ'_j 越大，相对的供应商越重要

$$\Psi'_j = max\{\psi_{1j}, \psi_{2j}, \ldots, \psi_{240j}\}. \tag{8}$$

2.2 基于熵权的 TOPSIS 方法选优

根据过去五年的交易数据，本文建立了基于熵权法的理想解法供应商评价体系，并根据评价得分选出供应影响力较强的供应商。

理想解法亦称 TOPSIS 法，是一种有效的多指标评价方法。理想解法通过构造评价问题的正、负理想解，根据每个供应商到理想供应商的相对贴近度，即控制靠近正理想解或远离负理想解的程度，来对方案进行排序，以选出最优方案。

TOPSIS 法对样本数据无特殊要求，并且能清晰地反映各方案的差距，被广泛应用于多方案多目标的决策评价[4]，但使用 TOPSIS 法之前必须确定各评价指标的权重系数。

在传统评价模型的应用过程中，由于指标权重难以确定，使用场景有很大的局限性。而常见的多要素问题中指标权重的确定方法有：专家评测法、层次分析法、二向系数法、熵值法、环比分析法、模糊聚类分析法等。由于前三种方法存在着较大的主观因素，而最后一种方法主要用于模糊指标的重要程度分类，为提高综合评价的准确性和客观性，本文故采用熵权法来解决供应商评价选优模型中的权重赋值问题[5]。

熵权法是一种客观赋权方法。在具体使用过程中，熵权法根据各指标的变异程度，利用信息熵计算出各指标的熵权，从而得出较为客观的指标权重。

2.2.1 权重计算

1. 构造原始数据矩阵

$\mathbf{A} = (a_{jk})_{402\times5}$（用 a_{jk} 指代各供应商的评价值），将以上五个指标放入以下矩

阵。

$$\begin{pmatrix} W_1 & W_2 & \dots & W_{402} \\ \varkappa_1 & \varkappa_2 & \dots & \varkappa_4 02 \\ \Psi_1 & \Psi_2 & \dots & \Psi_4 02 \\ \varepsilon_1 & \varepsilon_2 & \dots & \varepsilon_4 02 \\ N_1 & N_2 & \dots & N_4 02 \end{pmatrix} \tag{9}$$

2. 规范化属性值

得到数据矩阵 $\mathbf{B} = (b_{jk})_{402\times 5}$，为了使每个属性变换后的最优值为 1，且最差值为 0，这里先进行 0—1 标准化

$$b'_{jk} = \frac{b_{jk} - b_{min}}{b_{max} - b_{min}}. \tag{10}$$

此外，为了减小矩阵中的订货量和供货量波动过大的影响，对其进行对数化处理。重新赋值，得

$$b''_{jk} = \ln b'_{jk} (k = 1, 2, 5). \tag{11}$$

3. 利用数据矩阵

计算 $p_{jk}(k = 1, 2, 3, 4, 5)$，即第 j 家供应商关于第 k 个指标值的比重

$$p_{jk} = \frac{b''_{jk}}{\sum_{j=1}^{402} b''_{jk}}. \tag{12}$$

4. 计算第 k 项指标的熵值

熵本源于热力学与统计物理，后由香农引入信息论。根据熵的定义，有

$$e_k = -\frac{1}{ln402} \sum_{j=1}^{402} p_{jk} ln p_{jk}. \tag{13}$$

5. 计算第 k 指标的变异系数

$$g_k = 1 - e_k, \tag{14}$$

对于第 k 项指标，e_k 越大，指标值的变异程度就越小。

6. 计算第 k 指标的权重

$$\omega_k = \frac{g_k}{\sum_{k=1}^{5} g_k}. \tag{15}$$

2.2.2 供应商评价

1. 用向量规范化的方法求得规范决策矩阵

$$\breve{\mathbf{B}} = (\breve{b}_{jk})_{402\times 5}, \tag{16}$$

其中

$$\breve{b}_{jk} = \frac{b''_{jk}}{\sqrt{\sum_{j=1}^{402} b''^2_{jk}}}. \tag{17}$$

2. 构造加权规范阵 $\breve{\mathbf{B}} = (\breve{b}_{jk})_{402\times 5}$

凭借熵权法给定的各属性的权重，得权重向量 $\omega = (\omega_1, \omega_2, \omega_3, \omega_4, \omega_5)$，则有

$$\check{b}_{jk} = \omega_k \breve{b}_{jk}. \tag{18}$$

3. 确定正、负理想解 $\mathbf{C}^+ = (c_1^+, c_2^+, c_3^+, c_4^+, c_5^+)$ 、$\mathbf{C}^- = (c_1^-, c_2^-, c_3^-, c_4^-, c_5^-)$

正理想解

$$c_k^+ = \begin{cases} \max\limits_j \check{b}_{jk}, & k \text{ 为效益型属性}, \\ \min\limits_j \check{b}_{jk}, & k \text{ 为成本型属性}; \end{cases} \tag{19}$$

负理想解反之。

4. 计算各供应商与正、负理想解的距离

与正理想解的距离

$$s_j^+ = \sqrt{\sum_{k=1}^{5} (\check{b}_{jk} - c_k^+)^2}, \tag{20}$$

与负理想解的距离, 同理

$$s_j^- = \sqrt{\sum_{k=1}^{5} (\check{b}_{jk} - c_k^-)^2}, \tag{21}$$

5. 计算各供应商对理想解的相对接近程度作排序指标值（即综合评价指数）

$$f_j = \frac{s_j^-}{s_j^- + s_j^+}. \tag{22}$$

6. 按 f_j 由大到小排列供应商的优劣次序

2.3 供应商的评价结果

以上介绍了本文建立的评价模型的算法，本节将首先展示使用本模型评价的过程和结果。进而确定了 50 家最重要的供应商，如下图所示。

S151	S361	S356	S131	S374	S364	S031	S080	S218	S123
S108	S348	S340	S306	S194	S365	S338	S126	S291	S150
S330	S282	S139	S268	S307	S367	S346	S395	S074	S201
S308	S140	S275	S352	S055	S040	S086	S244	S003	S007
S229	S143	S329	S284	S247	S037	S114	S294	S210	S189

图 1　前 50 家供应商.

根据各个供应商的评价得分特点，本文经过初筛和复选最终认为该企业应至少选择 44 家供应商供应原材料才可能满足生产的需求。

2.3.1 各个指标的权重

运用熵权法，对过去交易数据进行分析，得到各个特征指标的权重，如图 2 所示。

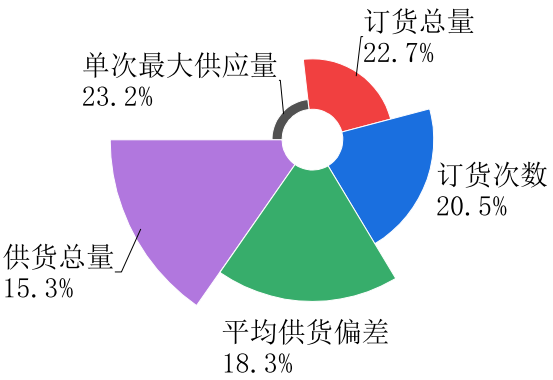

图 2　各指标权重.

2.3.2 供应商综合评价指数

根据评价指标和上文计算出的权重，对供应商进行评价。

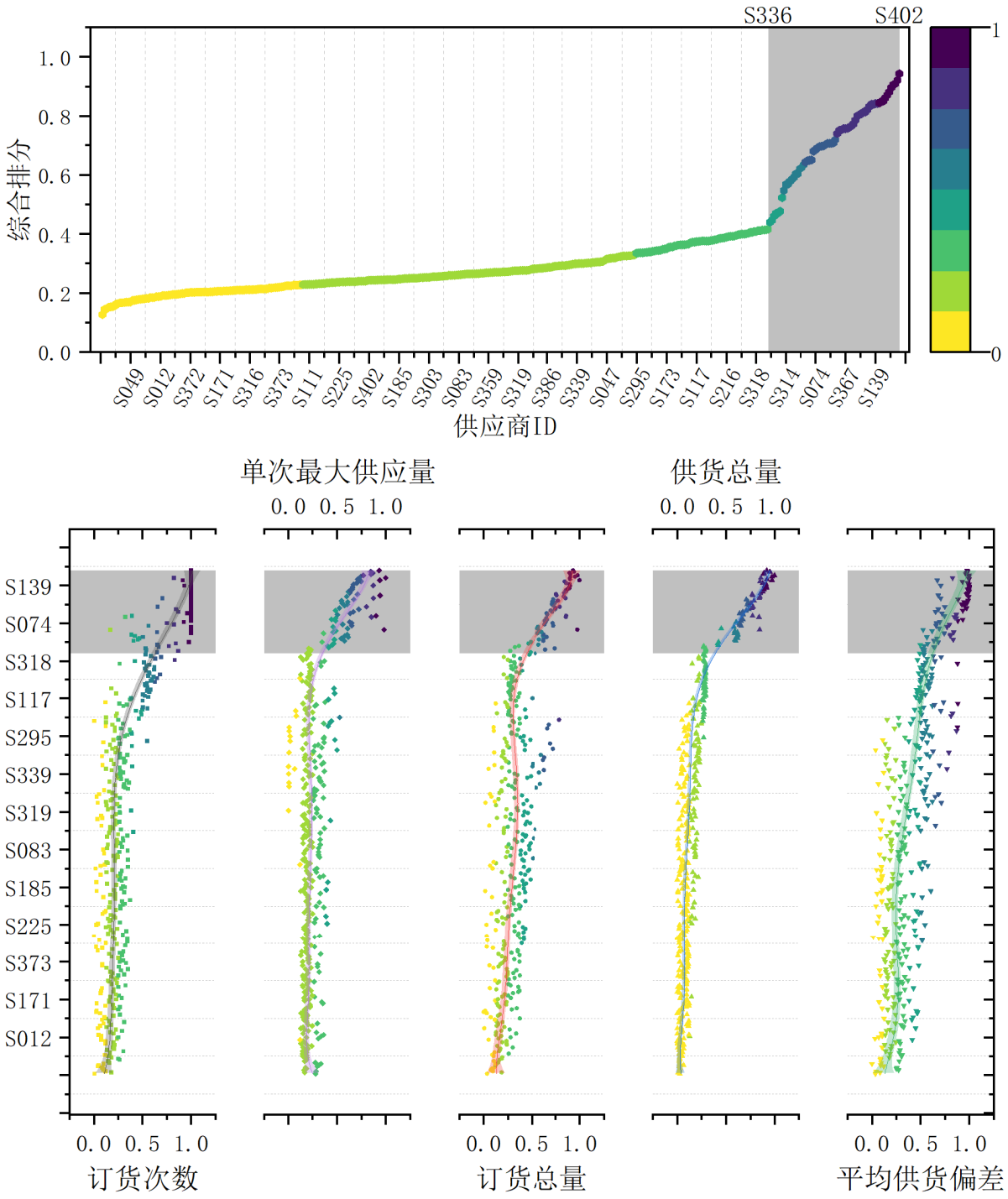

图 3　各供应商评价指数.

2.3.3 供应商的预选

根据以上评价模型对于各供应商的得分，将供应商由高到低排序，并分别以各供应商及其评价分数为、横坐标，绘制图像。由图 2 可见，供应商的得分状况呈积聚分布，并且在 0.416 分处，有显著的拐点。因此，本文先根据评价分数，首先筛选出 66 家高分供应商。

2.4 供应商数量优化

通过预选供应商，本文已筛选出供应能力较突出的供应商。为了进一步得到合适数量的供应商，满足企业每周生产需求，并使供货最稳定，本文建立供应商选择 0–1 规划模型，以决定实际供货商选取：

$$\lambda_j = \begin{cases} 0, & \text{第 } j \text{ 家供应商未选上,} \\ 1, & \text{第 } j \text{ 家供应商被选上;} \end{cases} \tag{23}$$

$$\begin{aligned} \textbf{min} \quad & \Lambda = \sum_{j=1}^{66} \lambda_j \varepsilon_j \\ \textbf{s.t.} \quad & \sum_{j=1}^{66} \lambda_j \bar{y}_j > 2.84 \times 10^4 \end{aligned} \tag{24}$$

其中，λ_j 是表示第 j 家供应商是否被选上的 0–1 变量，而 $\bar{y}_j$ 表示第 j 家供应商的供货能力，计算方法见于后文。通过对模型的求解，得到最终选定的 44 家供应商。

S348	S139	S352	S247	S031	S126	S074	S007
S282	S275	S284	S364	S338	S395	S003	S189
S140	S329	S374	S365	S346	S244	S210	
S143	S131	S194	S367	S086	S294	S123	
S356	S306	S307	S040	S114	S218	S150	
S340	S268	S055	S037	S080	S291	S201	

图 4　选定的 44 家供应商.

综上所述，本文认为该企业应至少选择 44 家供应商供应原材料才可能满足生产的需求。

三、 预测未来供货量

3.1 基于神经网络的供货能力预测

为了优化订购方案，首先需要确定可能的供应量，因此对筛选后的供货商的供应量进行预测，再据此制定相应的订货方案。供应商接受订单后，会在自身供货能力范围内，调整供货量以满足订单需要。当订货量过大时，供应商会仅能提供其最大供应量，因此往年供应数据可以反映出供应商的供货能力上限，本文依据供应商历史供应情况来预测其未来供应情况。

订货商的供应量受多种不可控因素影响，若仅根据时间变量构建时间序列预测模型，难以准确拟合供应量曲线，预测效果差。若将其做主要影响因素的回归分析，由于影响因素众多且相互耦合，无法综合分析，与此同时，预测过程中也有预测函数难以构建的问题[6]。

对于复杂的长时间序列预测的问题，相比于传统时间序列预测手段，深度学习方法有着更好的拟合与预测能力。深度学习模型是一种拥有多个非线性映射层级的深度神经

网络模型，能够对输入信号逐层抽象并提取特征，挖掘出更深层次的潜在规律[7]。其中，LSTM 模型弥补了循环神经网络的梯度消失和梯度爆炸、长期记忆能力不足等问题，使得循环神经网络能够真正有效地利用长距离的时序信息[8]。

因此，本文使用 LSTM 分别对选中的 44 家公司的供货量进行预测。本文在过去五年的交易数据中，以前四年训练神经网络，取第五年为测试集。

LSTM 是由一系列节点（又称 LSTM 单元）组成的。如图 5 所示，LSTM 单元内部主要有三个阶段。

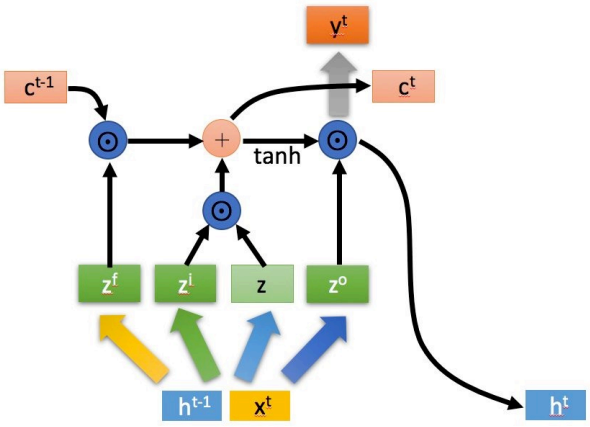

图 5　LSTM 网络结构图

在第 t 个节点中：

(1) 忘记阶段——对上一个节点的输入进行选择性忘记，即遗忘门

$$f^t = \sigma\left(\omega^f \begin{matrix} x^t \\ h^{t-1} \end{matrix}\right). \tag{25}$$

该阶段通过 Sigmoid 函数，计算得到的 f^t 是一个向量，且每个分量均属于 $[0,1]$。一般分量的值会极其接近 0 或 1，为后续选择 c^{t-1} 中用于计算 c^t 的特征做准备。

(2) 记忆阶段——选择有效的输入值，即输入门

$$i^t = \sigma\left(\omega^i \begin{matrix} x^t \\ h^{t-1} \end{matrix}\right). \tag{26}$$

该阶段工作原理与忘记阶段类似，将用于对输入的 x^t 进行选择。

(3) 选择阶段——选择输出并放缩，即输出门

$$o^t = \sigma\left(\omega^o \begin{matrix} x^t \\ h^{t-1} \end{matrix}\right). \tag{27}$$

本阶段得到的 o^t 会放缩 c^t，进而决定节点的输出

LSTM 的优势是前一节点的状态将随数据一起传入后一节点，但并不向外输出，具体来说也有三个：

(1) 单元状态更新值

$$\check{c}^t = \sigma\left(\omega^c \begin{matrix} x^t \\ h^{t-1} \end{matrix}\right). \tag{28}$$

事实上，单元状态更新值与遗忘门、输入门和输出门的作用类似，而且并不传入后一节点。

(2) 单元状态

$$c^t = f^t c^{t-1} + i^t \check{c}^t. \tag{29}$$

(3) 隐藏状态

$$h^t = o^t tanh(c^t). \tag{30}$$

h^t 和 c^t 虽然并不输出，但会作为本节点状态的标志传入下一节点，使 LSTM 具有较好的长时间序列预测能力。最终节点的输出 y^t 也是由 h^t 变化而成的，本文使用的是

$$y^t = \sigma(\omega' h^t), \tag{31}$$

此处得到的输出 y^t 即为神经网络对于第 i 周第 j 家供应商供应量的预测值（ψ_{ij}）。严格来说，由于此处并未考虑订货量的影响，故此处预测的供货量事实上仅仅表征了第 i 周第 j 家供应商的最大供货能力。此外，本文为了使用神经网络预测供货量，所以在条件的假设上做了一定让步（假设并不严格恒成立），导致后文还要对此处得到的结果继续进行修正。

本文的 LSTM 是由 PyTorch 实现的，结构如图 6 所示：

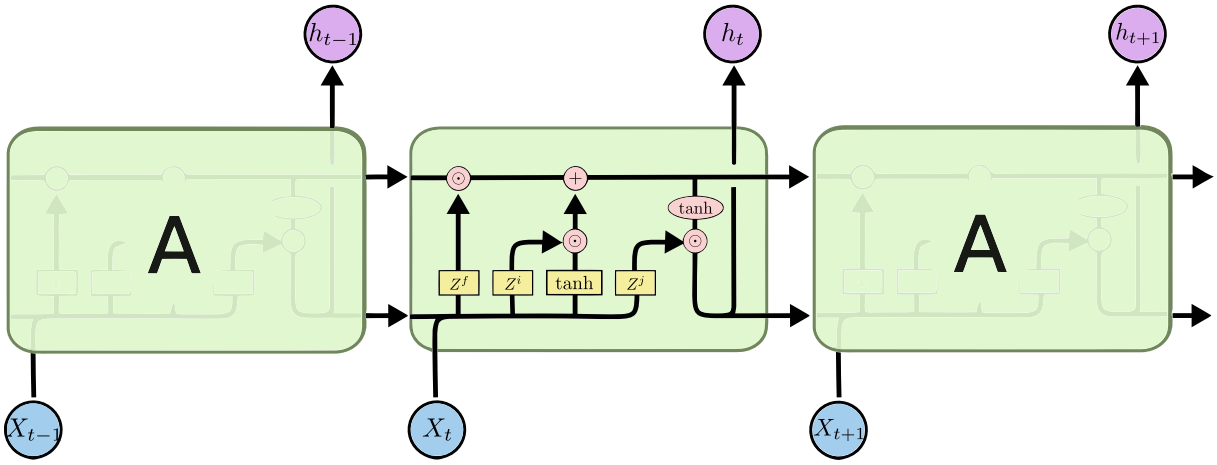

图 6　LSTM 网络结构图

本文在实践上的创新在于将数据传入 LSTM 前的格式处理。日常生产生活中的数据往往只是简单的 2 维表格，需要按照时间窗口切分为特征向量才能使用 LSTM 进行

预测。本文自主实现了按照时间窗口切分特征向量的算法，并完成了封装，有望帮助其他工作者在需要时进行调用，具有泛用性和现实价值。为方便读者理解（也作为上述算法的实例），此处附上和 LSTM 联用的伪代码供参考：

Algorithm 1 Long short-term memory for out model.

Input: 从原始数据中获取原始的订购与供应数据; 从原始数据中挑取前一步筛选的 44 家供应商信息;

Output: 第 i 周第 j 家供应商的预测值 ψ_{ij};

对数据的 A、B、C 分类，做均一化预处理;

将数据按供应商拆分为一维时间序列数据;

将一维时间序列数据先按时间窗口切分为二维数据，以同样的方法，再将每个时间窗口中的数据切分为特征向量;

对升维后的数据做一次正态标准化缩放，保存缩放参数;

将处理后的数据送入 LSTM 网络，训练完毕后，返回神经网络;

每次预测 1 周，将预测的得到的 1 周数据重新送入神经网络，迭代指定此时以得到相应的预测步数;

使用保存的缩放数据，将预测数据复原。

return ψ_{ij}.

通过分析往年交易数据，供应商总是根据企业订货量来供应原材料。由于没有考虑提高订购量对供应量的影响，仅根据往年交易数据进行预测，模型对供应商实际供应的预测值会偏小。通过对预测结果的检验，预测中筛选得到的供应商的总供应量确实小于预期。

3.2 供应量修正

为了提高预测的准确性，本文为模型添加了反映订货量影响的修正。数据显示，一定范围内，供应商的供应量会随订货量的增加而增多。而当订货量超出其的供应能力后，供应商的供货量限于生产条件达到其最大值，即最大供应量。

进而定义该家供货商的修正比 α_j：

$$\alpha_j = \frac{\beta_j - \gamma_j}{\gamma_j}, \tag{32}$$

在第 j 家供应商的历史数据中，当供货量超过订货量时供货量的平均值记为 β_j，而 γ_j 为当供货量小于订货量时供货量的平均值。

修正后第 j 家供货商在第 i 周的实际供货能力 $\hat{\psi}_{ij} = \alpha_j y_j^i$。

3.3 预测模型检验

本文基于企业前四年的供应数据，建立如上原材料供应量预测模型，进而预测了第五年的原材料供应情况。在测试集上，将模型预测结果与真实数据比较供应量的变化情况以检验模型预测准确性，如下图所示：

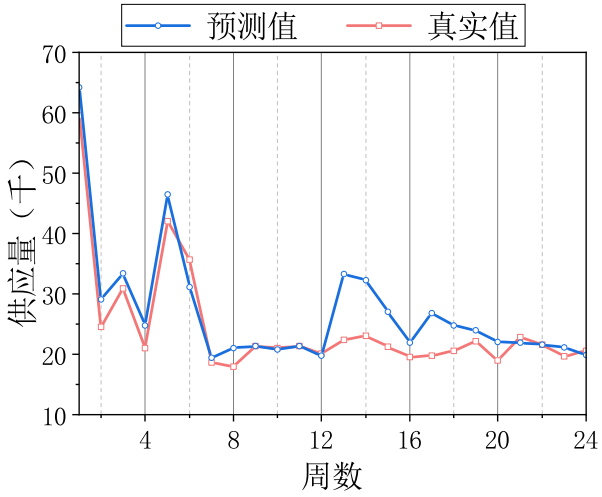

图 7　拟合验证图

由图可知，预测模型前期工作效果极佳；在后期的预测效果相较前期有所下降，但仍然和实际数据较为接近，且在变化趋势上为出现错误，拟合程度较好。预测能力随预测时间延长而下降是自然的，也是 LSTM 的特点。这是由于尽管 LSTM 进行了选择性遗忘和记忆，但前期预测偏差在积累后仍会逐渐放大，使预测误差扩大导致的。测试集上的总体预测效果较好，进而本文认为可以用 LSTM 预测供应商的供货量变化。

四、 制定订购、库存方案

4.1 基于订货量的供货量确定

在生产厂家选择确定的供应商并发出订单后，供应商在其供货能力范围内将尽可能按照订货量进行供货，实际供货量受其自身供货能力的影响；为了模拟现实中供货量的波动情况，本文令实际供货量偏差符合该供货商过去的供货偏差情况。

由式 (7) 可知，预测的供货量为：

$$u_{ij} = \begin{cases} \varepsilon_j v_{ij}, & v_{ij} \leq \psi_{ij}, \\ \varepsilon_j \psi_{ij}, & v_{ij} \geq \psi_{ij}; \end{cases} \tag{33}$$

当订货量高于供应商的能力时，供货量不再随订货量增长，可令此时订货量等于供应能力。在此基础上，本文在后文中通过约束 $v_{ij} \leq \psi_{ij}$，解决了分段函数线性化的问题。

4.2 制定订购方案

为获得最高收益，在制定订购方案时，需要平衡原料库存量。在供应充足时适当扩大库存，以填补未来原材料不足造成的的缺口，使其生产活动稳定进行。为了在尽可能保证产量的前提下，使存储成本最小，本文建立了如下线性规划模型以确定每周的订货量。

分析预测结果，虽然平均每周的供货总量充足（大于 $2.84 \times 10^4 m^3$），但仍有部分周供应不足，记缺货量为 $D_i = 2.84 \times 10^6 - u_i$（供货量充足时，$D_i = 0$）。为了在供应不足时也能尽可能地发挥产能，该企业会在供货总量充足的时候中增加进货，即额外订购量 $d_i = u_i - 2.84 \times 10^6$（供货总量低于 $2.84 \times 10^4 m^3$ 时，$d_i = 0$）。

为了保证订购的原材料均被加工为产品，要求 $\sum_{i=1}^{24} D_i \geq \sum_{i=1}^{24} d_i$。由于此时总可通过调整周期起点的方式，使得在周期结束时订购的原材料均被加工为产品，这为后文通过累加计算总产量并转化为企业盈利奠定了基础。

令目标函数为企业利润，使企业收益最大，构建线性规划模型：

$$\begin{aligned} \textbf{max} \quad & p_1\eta - p_2\mu_i \\ \textbf{s.t.} \quad & \sum_{i=1}^{24} D_i \geq \sum_{i=1}^{24} d_i \end{aligned} \tag{34}$$

其中，η 为 24 周内的总产量，$\eta = 24 \times 2.84 \times 10^6 - \sum_{i=1}^{24} D_i + \sum_{i=1}^{24} d_i$; 第 t 周的原材料库存量 $\mu_t = 2.84 - \sum_{i=1}^{t} D_i + \sum_{i=1}^{t} d_i$。$p_1$ 为单位产品的售价，则 $p_1\eta$ 为企业 24 周的生产所得；p_2 为单位产品所需原料的平均运输和储存成本。该模型中的 p_1 和 p_2 是关键参数：当 p_1 较大时，企业会倾向多投入存储费用来保证产能不被浪费，以生产更多产品赚取高额利润；当 p_2 较大时，企业则会牺牲产量来避免支付昂贵的仓储费用。考虑到企业生产的是建筑和装饰板材、所用的原材料抓哟平时木质纤维和其他植物素纤维材料，本文在测试时，产品的售价约比仓储成本高一个数量级以和市场情况类似。需要泛用时，p_1 和 p_2 应根据实际情况灵活取值。

以上模型的决策变量是订购量 v_{ij}，得到的最优解代表着每周最经济的订货量。此处由于三种类型原材料的本征采购单价相等，因此在进行运筹时并没有在目标函数中计算。另外，通过对结果的检验，本文发现最优解得订购方案在转运商的能力范围之内，因此忽略了转运商的能力对订货量的限制。

经过以上的优化，本文已经确定了每周的订货总量，需根据各供货商供货能力分配订单。本文依据2.2中的各供应商选优顺序，向筛选后的供应商依次订货，直到预计供货量达到预期。

4.3 制定转运方案

为了保证生产的经济效应，企业不仅需要制定订货方案，计划转运商的运输方案也是必要的，本文希望根据上文制定的订货方案进一步计划损耗最少的转运方案。在实际转运过程中，原材料会有一定的损耗，损耗量占供货量的百分比称为“损耗率”。通常情况下，一家供应商每周供应的原材料尽量由一家转运商运输。为体现供应商 Sj 选择的转运商 Tl，引入 0—1 变量

$$\rho_{jl}=\begin{cases}0, & \text{第}i\text{ 周供应商}S_j\text{ 未选择转运商}T_l\text{ 运输原材料},\\ 1, & \text{第}i\text{ 周供应商}S_j\text{ 选择转运商}T_l\text{ 运输原材料};\end{cases} \tag{35}$$

假设同一家转运商 Tl 的损耗率总是恒定的 ι_l，则有：

$$\begin{aligned}&\min \sum_{l=1}^{8}\left(\iota_l\sum_{j=1}^{44}\rho_{ijl}\psi_{ij}\right)p_3\\ &\text{s.t.}\begin{cases}\sum_{i=1}^{8}\rho_{ijl}=1\\ \sum_{j=1}^{44}\rho_{ijl}\psi i_{ij}\le 6000\end{cases}\end{aligned} \tag{36}$$

式中 p_3 是原材料的采购单价，根据供应商贩卖的原材料类型确定。目标函数 L_i 是第 i 周的损失价值，故希望最小。该 0—1 规划模型每周运筹一次，制定该周的转运方案。约束条件表示一家供应商每周供应的原材料只由一家转运商运输，且每家转运商的运输原料的能力为 6000 立方米/周。事实上，根据预测模型，发现有的供应商单周供应量大于 6000 立方米，此时便无法只由一家转运商运输了，不属于所谓的“通常情况”。但这种情况发生次数较少，通过手工分配后，转运效果整体良好。

4.4 模型检验

最后，本文根据前四年的交易数据，基于以上方法制定了优化后的订购方案。并通过计算机按照数据统计分布规律生成随机供货偏差取代优化模型中期望值的方式进行仿真模拟。在仿真情况下，按照前文制定的方案进行订购和转运时，企业每周的订货量如下图所示：

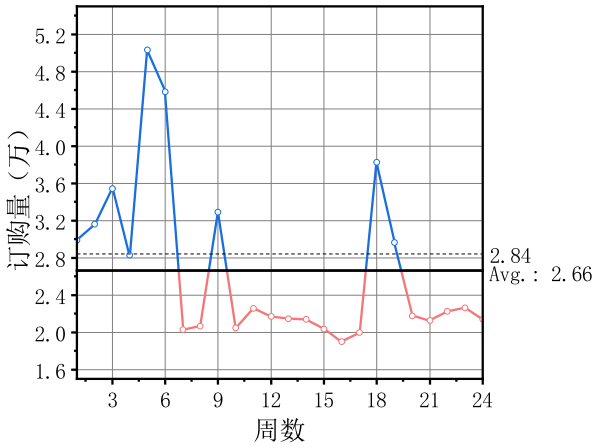

图 8　企业总订货量变化.

由图可知，企业的订货量较平稳，且略高于满足两周生产需求的原材料数量，有效地节约了仓储成本、保证了生产进度。由于后期供应商的总体供货疲软，在预测模型的指导下，企业在前期采取了一定的囤货行为，这虽然花费了较多的存储成本，但使总产量上升，总体有利。本文认为，针对问题 2 制定的订购及转运方案实施效果良好，进一步提高了企业生产的经济效益。

五、 原材料的订购和转运方案扩展

本题要求在尽量多地采购 A 类和尽量少地采购 C 类原材料的前提下，制定最经济的原材料订购方案和损耗最小的转运方案，并分析方案的实施效果。

5.1 制定订购方案

该企产为了压缩生产成本，要减少转运及仓储的成本。三类原材料运输和业为了压缩生存储的单位费用相同，但 A 类原材料的商品生产能力比 C 类原材料强，因此企业会尽量多地采购 A 类和尽量少地采购 C 类原材料。本文忽略了原材料类型偏好对每周订货总量的影响，而主要关注于供应商订购量分配的变化。因此，问题 2 中的每周订货总量计划保持不变。

问题 2 中，原材料的订购数量是直接根据供应商的评价来进行分配的。问题 1 中建立的评价模型并未重点关注供应商贩卖的原材料类型，本文通过依据供应商贩卖的原材料类型，对供应商的评价加分，以尽量多地采购 A 类和尽量少地采购 C 类原材料。考虑到综合评价指标均小于 1 分，本文对售卖 A 类原材料的供应商加 1 分，令企业尽量多地采购 A 类原材料。联系上文，B 类原材料的本征采购单价实际上略高，因此，本文没有对 C 类原材料的供应商扣分。

5.2 制定运输方案

因为 A 类原材料的采购单价较高，因此对转运损耗更加敏感。问题 3 中企业计划尽量多地采购 A 类原材料，所以迫切需要制定合理的运输方案，以确保转运商的转运损耗率尽量少，来达到压缩生产成本的目的。问题 2 中侧重于经济效益，要求损耗最少，而问题 3 要求转运损耗率尽量少，因此对问题 2 中的 0–1 规划模型稍加改动：

$$\begin{aligned} &\min \sum_{l=1}^{8} \left(\iota_l \sum_{j=1}^{44} \rho_{ijl} \psi_{ij} \right) \\ &\text{s.t.} \begin{cases} \sum_{i=1}^{8} \rho_{ijl} = 1 \\ \sum_{j=1}^{44} \rho_{ijl} \psi i_{ij} \leq 6000 \end{cases} \end{aligned} \tag{37}$$

式中 L_i' 代表第 i 周的转运损耗量，由于希望转运损耗率尽量少，故此处转运损耗量应取最小值。

5.3 仿真模拟分析方案实施效果

仿照问题 2，此处本文也进行了仿真模拟，绘制了企业仓库储存量随周数的变化情况关系，如下图 9 所示：

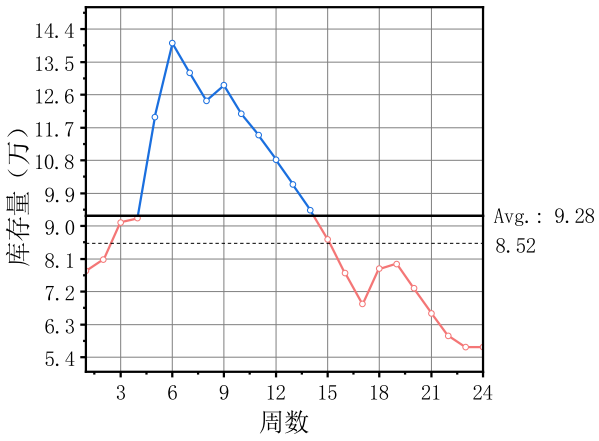

图 9　企业库存量变化.

由图可知，在问题 2 的基础上库存量总体向下平移，节约了仓储费用，模型效果显著。相比问题 2，此处调高了仓储成本与采购单价或售价之比，符合现实中企业偏爱 A 类原材料的原因，这也使企业的囤货倾向有所收敛。

六、 企业产能上限优化

题目中要求在供应商和转运商有限的情况下，评估企业每周产能的上限。本题在取消生产企业每周产能上限的同时，也消除了原材料的库存问题，这使得供应商和转运商的供应量可直接由生产企业转化为产能。本问题也转化为在供应商和转运商有限的情况下，求供应链可向企业输送的最大供给量。

联系附件数据，我们得知近 5 年来，共有 402 家供应商通过 8 家转运商向企业供货。而通过分析供货数据我们发现，每周转运商的总转运能力远少于供给商的供给能力，故向企业输送的最大供给量主要取决于转运商的转运能力。因此，本模型主要对转运商的转运能力进行优化。

本文引申一家供应商每周供应的原材料尽量由一家转运商运输，提出运筹目标为参与转运的转运商次最少：

$$\begin{aligned} &\min \sum_{j=1}^{402} \sum_{l=1}^{8} \left[\frac{\theta_{ijl}}{\psi_{ij}}\right] \\ &\text{s.t.} \begin{cases} \sum_{i=1}^{8} \theta_{ijl} = \psi_{ij} \\ \sum_{j=1}^{402} \theta_{ijl} = 6000 \end{cases} \end{aligned} \tag{38}$$

式中，[] 是高斯函数，代表向下取整；θ_{ijl} 是第 i 周转运商 Tl 为供应商 Sj 运输的供货量。为保证企业产能充分发挥，约束了每家转运商均取得其最大转运能力 6000 立方米。

事实上，由于 A 类原材料生产商品的能力能力最强，企业此时仍对其表现出明显的偏好。

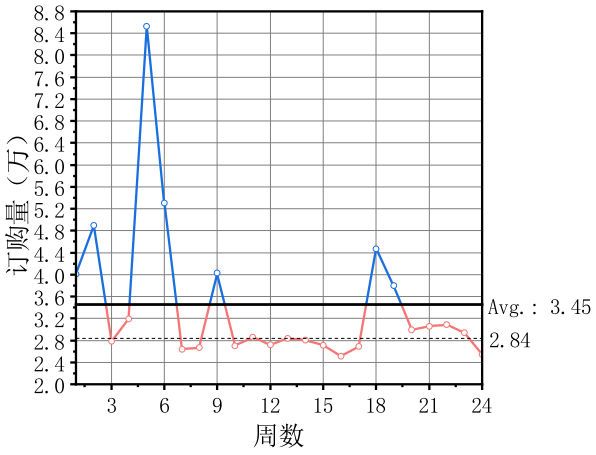

图 10　企业总订购量变化.

由图可知，企业每周对 A 类原材料的收购量均为 48000 立方米，即为转运商的最大转运能力，符合转运商是生产限速步骤的预期。而库存量严格维持为刚好满足企业两周生产需求的原材料库存量，说明提高产能还可以降低仓储成本，体现了科技作为第一生产力的重要地位。但每周的转运商次变化较大，说明本模型能够有效安排转运商的运输工作，实施效果良好。

七、 模型的推广

7.1 模型的创新点

1. 建立了基于熵权逼近理想解方法的供应商排序模型，提高了综合评价的准确性和客观性。
2. 运用 LSTM 模型预测供货商的供应情况，从更深层次反映供货商实际供货情况。
3. 设计了使用计算机模拟供应过程的相关程序，运用仿真方法体现出优化方案的实际效果。

7.2 模型的缺点

1. 指定订购和转运方案相对独立，几乎没有考虑相互作用的影响。
2. 认为转运商的损耗率一定，而没有进行预测或检验，过于粗糙。

本文中，对合作厂商先挑选再优化的办法可推广到其他商业合作中，进一步解决经济学和管理学中的许多问题，同时为社会学中的人际关系研究提供全新思路和不同视角；对企业生产成本进行优化来得到最经济方案的规划方法能给企业经营以理论指导，有很强的实践价值。

参考文献

[1] 杨庆军, 孙满. 浅谈 ERP 环境下的原材料管理[J]. 中国外资, 2012(7): 196-196.

[2] 李红燕. 论企业原材料的管理和研究[J]. 财会学习, 2018(2): 181-181.

[3] 顾丽娟. 基于最优供应商数量的补货策略研究[J]. 洛阳理工学院学报: 社会科学版, 2014(6): 21-24.

[4] 金王莉. 基于熵权 TOPSIS 法的产业综合优势分析——以苏州制造业为例[J]. 中国商论, 2018(28): 159-162.

[5] 马丽娟. 基于供应链管理的供应商选择问题初探[J]. 工业工程与管理, 2002(06): 23-25.

[6] 李鹏, 何帅, 韩鹏飞, 等. 基于长短期记忆的实时电价条件下智能电网短期负荷预测[J]. 电网技术, 2018, 42(12): 4045-4052.

[7] LECUN Y, BENGIO Y, HINTON G. Deep learning[J]. Nature, 2015, 521(7553): 436-444.

[8] 王鑫, 吴际, 刘超, 等. 基于 LSTM 循环神经网络的故障时间序列预测[J]., 2018.

附录 A　源程序

1.1　ELOL.py

```
# %%
#引入序列长度
from dataclasses import dataclass
import numpy as np
import pandas as pd
import torch
from PyEMD import EMD
import matplotlib.pyplot as plt
from IPython.display import clear_output
import pickle
import os


#时间序列类
#传入数据，返回一个指定长度的
class DataSeq:
    def __init__(self, dataSet:np.array, step:int):
        self.data = dataSet
        self.step = step
        self.len = len(self.data) - self.step + 1

    def __getitem__(self,index,step=None):
        if step == None:
            step = self.step
        if isinstance(index,slice):
            return self.getkeys(index,step)
        return self.getkey(index,step)

    def getkey(self,index,step):
        data = self.data[index : index + step]
        assert len(data) == self.step, f'detaData out of index! length is {self.len} but index
            is {index}'
        return data

    def getkeys(self,indexSlice,step):
        start,stop = indexSlice.start, indexSlice.stop
        if start == None :
            start = 0
        if stop == None:
            stop = self.len
        else:
            stop = stop - 1
```

```
        ls = []
        for index in range(start,stop+1):
            ls.append(self.getkey(index,step))
        datas = np.array(ls)
        return datas

    def __len__(self):
        return self.len

    def __str__(self):
        return str(self.data)



class DateData(torch.utils.data.Dataset):

    def __init__(self,dataSet:np.array,length = 256, imf_num = 4-1, pre_num=1,
         dataSetWindows=1000):
        super().__init__()

        self.rawData = dataSet.copy()
        self.rawLen = len(self.rawData)
        step = min(self.rawLen, dataSetWindows) - length
        self.step = step
        dataSet = dataSet[-dataSetWindows:]

        #获取 imf_num数
        self.X = DataEMD(dataSet[:-pre_num],length,step,imf_num)
        self.imf_num = min(self.X.imf_num - 1, imf_num)
        self.Data = DataEMD(dataSet,length,step,self.imf_num)

        self.len = len(self.Data) - 1
        self.params = [length , self.imf_num, pre_num, dataSetWindows]

    def __getitem__(self,index):
        if isinstance(index,slice):
            assert index.stop is None or len(self.X) >= index.stop, 'detaData out of index!'
        else:
            assert self.len >= index, f'detaData out of index! length is {self.len} but index is
                {index}'
        return self.Data[index].astype(np.float32), self.Data[index+1].astype(np.float32)

    def __len__(self):
        return self.len

    def update(self,newData):
        dataSet = np.concatenate([self.rawData,newData],axis=0)
```

```
        self.__init__(dataSet,*self.params)

    def copy():
        pass

# %%
class DataEMD(DataSeq):
    def __init__(self, dataSet:np.array,length:int, step:int,imf_num=-1,emd = EMD()):
        self.rawData = dataSet
        data = emd(self.rawData,max_imf=imf_num).transpose(1,0)
        all_length,self.imf_num = data.shape
        super().__init__(DataSeq(data,length), step)
        self.emd = emd

    def update(self,data):
        pass


# %%
def re_EMD(data):
    batch, step, length, imf_num = data.shape
    ls = []
    for i in range(imf_num):
        chose = Chose_Y(i)
        temp_y = get_num(chose(data))
        ls.append(temp_y)
    d = np.concatenate(ls,axis=1)

    return d.sum(axis=1)


# %%
def get_num(Y):
    return Y.cpu().detach().numpy()

# %%
class Chose(torch.nn.Module):
    def __init__(self, imf):
        super().__init__()
        self.imf = imf

    def forward(self,X):
        #input = b * s * l * imf
        y = X[:,:,:,self.imf]
        O = y.transpose(1,0)
        return O
```

```
class Chose_Y(torch.nn.Module):
    def __init__(self, imf,length=1):
        super().__init__()
        self.imf = imf
        self.length = length

    def forward(self,X):
        #input = b * s * l * imf
        b,s,l,imf = X.shape
        y = X[:,:,-self.length:,self.imf]
        O = y.reshape(b*s,-1)
        return O


class MyLstm_reg(torch.nn.Module):
    def __init__(self,length,hidden, layer=2,out_num = 1):
        super().__init__()
        self.LSTM = torch.nn.LSTM(length,hidden,num_layers=layer)
        self.state = None
        self.linear = torch.nn.Linear(hidden,out_num)

    def forward(self,X):

        y, self.states= self.LSTM(X)

        s,b,l = y.shape
        h = y.reshape(s*b,l)
        o = self.linear(h)
        return o


class add_net(torch.nn.Module):
    def __init__(self,axis = 2):
        super().__init__()
        self.axis = axis

    def forward(self,X):
        """
        X = b * s * imf * hidden
        """
        return X.sum(axis = self.axis)

# %%
class Trans:

    # 标准化类，默认使用正态标准化
    def __init__(self, trans_fn = None, re_trans_fn = None):
```

```
        self.re_state = False
        if trans_fn == None:
            self.trans_fn = self._stand
            self.re_trans_fn = self._re_stand
        elif re_trans_fn != None:
            self.trans_fn = trans_fn
            self.re_trans_fn = re_trans_fn
        else:
            RuntimeError('没有传入恢复函数！')

    def _stand(self,data):
        if self.re_state == False:
            self.re_trans_params = [data.mean(),data.std()]
            self.re_state = True
        new_data = (data - self.re_trans_params[0]) / (self.re_trans_params[1])
        return new_data
    def _re_stand(self,data):
        temp_data = data * (self.re_trans_params[1])
        re_data = temp_data + self.re_trans_params[0]
        return re_data

    def _max_min(self, data):
        if self.re_state == False:
            self.re_trans_params = [data.min(),data.max()]
            self.re_state = True
        new_data = (data - self.re_trans_params[0]) / (self.re_trans_params[1] -
            self.re_trans_params[0])
        return new_data

    def _re_max_min(self,data):
        temp_data = data * (self.re_trans_params[1] - self.re_trans_params[0])
        re_data = temp_data + self.re_trans_params[0]
        return re_data

# %%
def grad_clipping(net, theta): #@save
    """裁剪梯度"""
    if isinstance(net, torch.nn.Module):
        params = [p for p in net.parameters() if p.requires_grad]
    else:
        params = net.params
    norm = torch.sqrt(sum(torch.sum((p.grad ** 2)) for p in params))
    if norm > theta:
        for param in params:
            param.grad[:] *= theta / norm

# %%
```

```
def train_begin(net, data_iter, epoch, imf, device, lr=0.01,opim_fn = torch.optim.Adam,out_num
    = 1000, show_pic = True):
    # 初始化网络，在初次拟合训练时使用
    ls = []
    #迁移至GPU
    net.to(device)
    #初始 chose_Y 并选择 imf
    chose = Chose_Y(imf)
    opimter = opim_fn(net.parameters(),lr)

    loss = torch.nn.MSELoss()
    X,Y = next(iter(data_iter))

    trans = Trans()
    y = chose(Y)
    y_new = trans.trans_fn(y)
    out_time = torch.log(y_new.std() * y_new.abs().mean()/(500) + 1)

    for i in range(epoch):
        one_temp = []
        for X,Y in data_iter:
            X_new = trans.trans_fn(X)
            Y_new = trans.trans_fn(Y)
            yhat = net(X_new.to(device))
            y1 = chose(Y_new.to(device))
            l = loss(yhat[5:],y1[5:])
            if i> out_num and get_num(out_time) > get_num(l):
                return ls,trans
            opimter.zero_grad()
            l.backward()
            grad_clipping(net,1)
            opimter.step()
            one_temp.append(l)
            #ls.append(l)
        one_temp = torch.stack(one_temp,dim=0)
        ls.append(one_temp.max())

        if i % 20 == 0 and show_pic == True:
            clear_output(wait=True)
            print('out_time is loss less than',out_time,'and i is',i)
            plt.cla()
            temp = torch.stack(ls,dim=0)
            plt.plot(get_num(temp)[-100:])
            plt.show()

            plt.plot(get_num(y1)[:],'r')
            plt.plot(get_num(yhat)[:],'b',alpha=0.4)
```

```
            plt.show()
    return ls,trans

# %%
def get_net(imf,length = 64 ,hidden=256):
    #返回一个网络
    net = torch.nn.Sequential(
    Chose(imf),
    MyLstm_reg(length,hidden=hidden)
    )
    return net

# %%
def load_nets_and_trans(nets_path_ls, trans_path,length=64,hidden=256):
    nets = []
    imf_num = len(nets_path_ls)
    with open(trans_path,'rb') as f:
        trans_ls = pickle.load(f)
    for i in range(imf_num):
            net = get_net(i,length=length,hidden=hidden)
            net.load_state_dict(torch.load(nets_path_ls[i]))
            nets.append(net)
    return nets, trans_ls

# %%
def train_all_net(data_iter, imf_num, device, lr = 0.0005, min_epoch = 800, max_epoch = 10000,
     root_path='net',net_suffix='_lstm.pkl',trans_suffix= 'trans_ls.info',
              opim_fn = torch.optim.Adam,show_pic = False, length=64, hidden=256):
    nets= []
    trans_ls = []
    for i in range(imf_num):
        temp_net = get_net(i,length, hidden)
        ls, trans_one = train_begin(temp_net, data_iter, max_epoch,
                                i,device, lr = lr, out_num = min_epoch,opim_fn=opim_fn,
                                show_pic = show_pic)
        trans_ls.append(trans_one)
        torch.save(temp_net.state_dict(),root_path+ f'\\{i}'+net_suffix)
        net = get_net(i,length, hidden)
        net.load_state_dict(torch.load(root_path+ f'\\{i}'+net_suffix))
        nets.append(net)
    dump = pickle.dumps(trans_ls)
    with open(root_path+ '\\'+trans_suffix,'wb') as f:
        f.write(dump)
    return nets, trans_ls


# %%
```

```
def update_net(data_iter, nets, imf_num, device, lr = 0.0005, min_epoch = 800, max_epoch =
    10000, root_path='net',net_suffix='_lstm.pkl',trans_suffix= 'trans_ls.info',
              opim_fn = torch.optim.Adam, show_pic = False, dump_local = False, length=64,
                  hidden=256):
    for net in nets:
        net.train()
    trans_ls = []
    for i in range(imf_num):
        ls, trans_one = train_begin(nets[i], data_iter, max_epoch, i,device,
                                lr = lr,out_num = min_epoch,opim_fn=opim_fn, show_pic= show_pic)
        trans_ls.append(trans_one)
        if dump_local == True:
            torch.save(nets[i].state_dict(),root_path+ f'\\{i}'+net_suffix)
            net = get_net(i, length, hidden)
            net.load_state_dict(torch.load(root_path+ f'\\{i}'+net_suffix))
            nets[i] = net

    if dump_local == True:
        dump = pickle.dumps(trans_ls)
        with open(root_path+ '\\'+trans_suffix,'wb') as f:
            f.write(dump)
    return nets, trans_ls

# %%
def predict_one(nets,trans_ls,data_iter):
    for net in nets:
        net.eval()
        net.to('cpu')
    X,Y = next(iter(data_iter))
    new_in = torch.cat([X,Y[0:1,-2:-1]],dim=1)
    imf_num = len(nets)

    pred_Y = torch.zeros_like(nets[0](new_in))

    for i in range(imf_num):
        trans_in_i = trans_ls[i].trans_fn(new_in)
        trans_Y = nets[i](trans_in_i)
        pred_Y += trans_ls[i].re_trans_fn(trans_Y)
    return pred_Y



# %%
class ELOL:
    """
    Emd LSTM OnLine Learning Module
    """
```

```
def __init__(self,length,imf_num,hidden,rawData, device, pre_num=1, dataSetWindows=1000):

    self.length = length

    self.hidden = hidden
    self.data = DateData(rawData, length, imf_num-1, pre_num=pre_num, dataSetWindows =
        dataSetWindows)
    self.imf_num = self.data.imf_num
    self.data_iter = torch.utils.data.DataLoader(self.data, batch_size = 1)
    self.device = device

def init_nets(self, lr = 0.0005, min_epoch = 2000,
              max_epoch = 10000, root_path='net',
              net_suffix='_lstm.pkl',trans_suffix= 'trans_ls.info',
              opim_fn = torch.optim.Adam,show_pic = True):

    try:
        os.mkdir(root_path)
    except:
        print(f'文件夹 {root_path} 已经存在……开始训练网络')
    self.nets, self.trans_ls = train_all_net(self.data_iter,self.imf_num + 1, self.device,
                                      lr = lr, min_epoch = min_epoch,
                                      max_epoch = max_epoch, root_path = root_path,
                                      net_suffix = net_suffix,trans_suffix= trans_suffix,
                                      opim_fn = opim_fn,show_pic = show_pic,
                                          length=self.length, hidden = self.hidden)


def load_nets_and_trans(self, nets_path, trans_path):
       self.nets, self.trans_ls = load_nets_and_trans(nets_path,trans_path,
                                               length=self.length, hidden= self.hidden)


def update_data_and_net(self,data , lr = 0.005,
                min_epoch = 500, max_epoch = 10000,
                root_path='net',net_suffix='_lstm.pkl',
                trans_suffix= 'trans_ls.info',
                opim_fn = torch.optim.Adam,
                show_pic = False, dump_local = False):
    self.data.update(data)
    self.data_iter = torch.utils.data.DataLoader(self.data, batch_size = 1)
    self.nets, self.trans_ls = update_net(self.data_iter, self.nets, self.imf_num+1,
        self.device,
                                  lr=lr, min_epoch= min_epoch, max_epoch=max_epoch,
                                  root_path=root_path, net_suffix=net_suffix, trans_suffix=
                                      trans_suffix,
                                  opim_fn=opim_fn, show_pic= show_pic,
```

```
                                        dump_local=dump_local,length=self.length,
                                        hidden=self.hidden)


    def predict(self):
        self.pred_Y = predict_one(self.nets,self.trans_ls,self.data_iter)
        return get_num(self.pred_Y[-1])
```

1.2 header.py

```
from ELOL import MyLstm_reg,pd,torch,Trans,get_num,grad_clipping,clear_output,plt,np
import pickle

def pre_ABC(x,data_index = 2):
    if x['材料分类'] == "A":
        x[data_index:] = x[data_index:] / 0.6
    if x['材料分类'] == "B":
        x[data_index:] = x[data_index:] / 0.66
    if x['材料分类'] == "C":
        x[data_index:] = x[data_index:] / 0.72
    return x

def re_pre_ABC(x,data_index = 2):
    if x['材料分类'] == "A":
        x[data_index:] = x[data_index:] * 0.6
    if x['材料分类'] == "B":
        x[data_index:] = x[data_index:] * 0.66
    if x['材料分类'] == "C":
        x[data_index:] = x[data_index:] * 0.72
    return x

# %%
class DataSeq:
    def __init__(self, dataSet:np.array, step:int):
        self.data = dataSet
        self.step = step
        self.len = len(self.data) - self.step + 1

    def __getitem__(self,index,step=None):
        if step == None:
            step = self.step
        if isinstance(index,slice):
            return self.getkeys(index,step)
        return self.getkey(index,step)
```

```
    def getkey(self,index,step):
        data = self.data[index : index + step]
        assert len(data) == self.step, f'detaData out of index! length is {self.len} but index
            is {index}'
        return data

    def getkeys(self,indexSlice,step):
        start,stop = indexSlice.start, indexSlice.stop
        if start == None :
            start = 0
        if stop == None:
            stop = self.len
        else:
            stop = stop - 1
        ls = []
        for index in range(start,stop+1):
            ls.append(self.getkey(index,step))
        datas = np.array(ls)
        return datas

    def __len__(self):
        return self.len

    def __str__(self):
        return str(self.data)

# %%
class PureData(torch.utils.data.Dataset):
    def __init__(self,X,Y):
        super().__init__()
        self.X = X
        self.Y = Y
        self.len = len(X)

    def __getitem__(self,index,step=None):
        return self.X[index],self.Y[index]
    def __len__(self):
        return self.len


# %%
def train_begin(net, data_iter, epoch, device, lr=0.01,opim_fn = torch.optim.Adam,out_num =
     1000, show_pic = True):
    # 初始化网络，在初次拟合训练时使用
    ls = []
    #迁移至GPU
    net.to(device)
```

```
    #初始 chose_Y 并选择 imf
    opimter = opim_fn(net.parameters(),lr)

    loss = torch.nn.MSELoss()
    X,Y = next(iter(data_iter))

    trans = Trans()
    y_new = trans.trans_fn(Y)
    out_time = torch.log(y_new.std() * y_new.abs().mean()/(500) + 1)

    for i in range(epoch):
        one_temp = []
        for X,Y in data_iter:
            X_new = trans.trans_fn(X)
            Y_new = trans.trans_fn(Y)
            yhat = net(X_new.to(device))
            y1 = Y_new.to(device)
            l = loss(yhat[5:],trans_shape(y1)[5:])
            if i> out_num and get_num(out_time) > get_num(l):
                return ls,trans
            opimter.zero_grad()
            l.backward()
            grad_clipping(net,1)
            opimter.step()
            one_temp.append(l)
            #ls.append(l)
        one_temp = torch.stack(one_temp,dim=0)
        ls.append(one_temp.max())

        if i % 20 == 0:
            clear_output(wait=True)
            print('out_time is loss less than',out_time,'and i is',i)
            if show_pic == True:
                plt.cla()
                temp = torch.stack(ls,dim=0)
                plt.plot(get_num(temp)[-100:])
                plt.show()

                plt.plot(get_num(trans_shape(y1))[:],'r')
                plt.plot(get_num(yhat)[:],'b',alpha=0.4)
                plt.show()
    return ls,trans

# %%
def trans_shape(Y):
    b,s,l = Y.shape
    y = Y.reshape(b*s,-1)[:,-1].reshape(-1,1)
```

```
    return y


# 使用选定的条目筛选项目
def filter_item(data, filter_list,key='供应商ID'):
    temp_data = data.copy()
    indexs = data[key]
    indexs.name=None
    temp_data.index = indexs
    output_data = temp_data.loc[filter_list,:]
    return output_data.reset_index(drop=True)

#传入一个数据框和总和，按顺序取到所有累加值，直至等于总和
def sort_and_sub(data,total):
    def cumsum_to_sub(temp_one):
        temp_index = temp_one[1:].cumsum() > temp_one['temp']
        temp_two = temp_one[1:].copy()
        temp_two.loc[temp_index] = 0
        idx_max = len(temp_two[temp_index==False])
        if idx_max < len(temp_two):
            temp_two.loc[idx_max] = temp_one['temp'] - temp_two.sum()
        return temp_two
    temp_data = data[:]
    temp_total = total[:]
    temp_total.index = temp_data.columns
    temp_total.name='temp'
    new_temp = pd.concat([pd.DataFrame(temp_total).T,temp_data],axis=0)
    temp_temp = new_temp.apply(cumsum_to_sub)
    return temp_temp

#删除Nan
def drop_nan(serise):
    temp_serise = serise.copy()
    nan_index = (pd.isna(temp_serise)!=True)
    return temp_serise.loc[nan_index]

def set_plt_size(long=12,high=8):
    plt.rcParams['figure.figsize'] = (long,high)



def predict_product(data,pred_step = 48, length = 48 , step = 24, batchSize = 12, hidden = 256
    , max_epoch = 1000, min_epoch = 150, device = 'cpu',show_pic = True):
    temp_data = np.array(data,dtype=np.float32)
    dataX = DataSeq(DataSeq(temp_data[:-1],length),step)
    dataY = DataSeq(DataSeq(temp_data[1:],length),step)
    data = PureData(dataX,dataY)
```

```
    data_iter = torch.utils.data.DataLoader(data,batchSize,shuffle=True)
    X,Y = next(iter(data_iter))

    net = MyLstm_reg(length, hidden)
    ls,trans = train_begin(net,data_iter,max_epoch,device,out_num=min_epoch,show_pic=show_pic)
    new_temp = torch.tensor(temp_data[-(length+step):])
    new_temp = trans.trans_fn(new_temp).numpy()

    for i in range(pred_step):
        dataX = DataSeq(DataSeq(new_temp[:-1],length),step)
        dataY = DataSeq(DataSeq(new_temp[1:],length),step)
        data = PureData(dataX,dataY)
        data_iter = torch.utils.data.DataLoader(data,batchSize)
        X,Y = next(iter(data_iter))
        X_new = Y
        yhat = net(X_new.to(device))
        new_temp = np.concatenate((new_temp,get_num(yhat[-1])))[-(length+step):]

    one = torch.tensor(new_temp)
    two = trans.re_trans_fn(one)
    return two,ls

#常数
gongying = "附件1 近5年402家供应商的相关数据.xlsx"
data_order = pd.read_excel(gongying,'企业的订货量')
data_supply = pd.read_excel(gongying,'供应商的供货量')

device = 'cuda:0'
```

1.3 main.py

```
# %%
from header import *
import pulp
from random import choice

# %%
df_pred = pd.read_csv('pred.csv').drop(columns='Unnamed: 0')

# %%
#获取分类数据
data = pd.read_csv('temp.csv')
index = data[data['选中'] == 1]['Unnamed: 0'].values

df_class = filter_item(data_order.loc[:,['供应商ID','材料分类']],index)
```

```
new_df_pred = pd.concat([df_class['材料分类'],df_pred],axis=1)

# %%
new_df_pred.apply(re_pre_ABC,axis=1).sum(axis=0)

# %%
#new_df_pred = new_df_pred.apply(re_pre_ABC,axis=1)

# %%
new_df_pred = new_df_pred.iloc[:,:2+24]

# %%
#预测的最大供货量的和
pred_sum = new_df_pred.iloc[:,2:].sum(axis=0)
#去除供应商id和类别后预测的最大供应量
pred_pure = new_df_pred.iloc[:,2:]
#大于2.84的索引
low_index = (pred_sum -2.84e4 < 0)
#小于2.84的索引
big_index = (pred_sum -2.84e4 > 0)

# %%
Storage_cost = 0.1
gross_profit = 1
# 开始准备线性规划数据

big_num_index = pred_sum.index.to_list()

myProblem = pulp.LpProblem('订购数量规划',sense=pulp.LpMaximize)

D_w = pred_sum.copy()
D_w.loc[big_index] = 0
D_w.loc[low_index] = 2.84e4 - D_w[low_index]
D_w = D_w.tolist()

d_w_constraint = pred_sum.copy()
d_w_constraint.loc[low_index] = 0
d_w_constraint.loc[big_index] = d_w_constraint[big_index] - 2.84e4
d_w_constraint = d_w_constraint.tolist()

d_var = pulp.LpVariable.dicts(name='d_w',indices=big_num_index,lowBound=0)

u = pulp.LpVariable.dicts(name='u_w',indices=big_num_index,lowBound=0)
for i in range(len(big_num_index)):
    u[str(i)] = 2.84e4 + pulp.lpSum([d_var[str(j)] for j in range(i+1)]) - sum(D_w[:i+1])
```

```
#定义目标
myProblem += gross_profit * pulp.lpSum(d_var) - Storage_cost * pulp.lpSum(u)

#定义约束

for i in range(len(big_num_index)):
    myProblem += (d_var[str(i)] <= d_w_constraint[i])

for i in range(len(big_num_index)):
    myProblem += (u[str(i)] >= 2.84e4*2)

myProblem += pulp.lpSum(d_var) <= sum(D_w) + 2.84e4

#solve
myProblem.solve()

# %%
#检查输出
ls = []
for v in u.values():
    print(v.name, "=", v.value())
    ls.append(v.value())
print(sum(ls))

for v in d_var.values():
    print(v.name, "=", v.varValue)

# %%
ls = []
for v in d_var.values():
    #print(v.name, "=", v.varValue)
    ls.append(v.value())
#预测的供货量
pred_order = pd.Series(ls)
#预测的最大供货量的和的复制
pred_order_temp = pred_sum.copy()
#经过加上2.84e4的遮蔽运算，得到的是每周预计供货量
pred_order_temp.loc[big_index] = 2.84e4
pred_order = pd.Series(pred_order_temp.values + pred_order.values)

# %%
pred_order

# %%
#倒序排列后的预测最大供应量数据，包含供应商ID和类别
temp = new_df_pred.iloc[::-1,:].reset_index(drop=True)
temp_pure = temp.iloc[:,2:]
```

```
out = pd.concat([temp.iloc[:,:2],sort_and_sub(temp_pure,pred_order)],axis=1)
out.to_csv('订购方案.csv',index=None)



# %%
#获取分数
index = data[data['选中'] == 1]['Unnamed: 0'].values
Fraction = pd.read_csv('总表.csv').drop(columns='Unnamed: 0')
Fraction = filter_item(Fraction,index)
Fraction = Fraction['综合排分'].values

# %%
pd.read_csv('总表.csv').drop(columns='Unnamed: 0')

# %%

#获取偏差
index = data[data['选中'] == 1]['Unnamed: 0'].values

data_order = filter_item(data_order, index)
data_supply = filter_item(data_supply, index)

pure_race_data = ((data_supply.iloc[:,2:] - data_order.iloc[:,2:]) / data_order.iloc[:,2:])
pure_race_data = pure_race_data.fillna(0)

race_mean = pure_race_data.sum(axis=1) / (data_order.iloc[:,2:] !=0).sum(axis=1)
#取偏差绝对值
race_mean_abs = race_mean

pred_order_index = pred_order[(pred_order != 0)].index.to_list()
pred_order_pure = pred_pure.loc[:,[str(i) for i in pred_order_index]]
pred_order_lp = pred_order[pred_order_index]

# %%
# 开始准备线性规划数据

def get_week_order(week):

    pred_order_pure_one_week = pred_order_pure[str(week)].tolist()

    pred_order_lp_one_week = pred_order_lp[week].tolist()

    myProblem1 = pulp.LpProblem('订购数量分配规划',sense=pulp.LpMaximize)

    z_var =
        pulp.LpVariable.dicts(name='z_w_'+str(week)+'_n',indices=range(len(pred_order_pure_one_week)),lowBound=0
```

```
    #定义目标
    myProblem1 += pulp.lpSum([Fraction[i] * z_var[i] for i in
        range(len(pred_order_pure_one_week))])

    #定义约束

    #myProblem1 += pulp.lpSum([z_var[i]*(1/(1+race_mean[i])) for i in
        range(len(pred_order_pure_one_week))]) == 2.84e4 + pred_order_lp_one_week
    myProblem1 += pulp.lpSum([z_var[i]*((1+race_mean[i])) for i in
        range(len(pred_order_pure_one_week))]) ==pred_order_lp_one_week

    #print(pred_order_lp_one_week)

    for i in range(len(pred_order_pure_one_week)):
        myProblem1 += z_var[i]*(1/(1+race_mean[i])) <= pred_order_pure_one_week[i]

    myProblem1.solve()

    ls = []
    i=0
    for v in z_var.values():
        #print(v.name, "=", v.varValue,'pred_order = ',pred_order_pure_one_week[i],'race
            =',race_mean[i])
        ls.append(v.value())
        i += 1

    #print(pulp.lpSum([z_var[i]*((1+race_mean[i])) for i in
        range(len(pred_order_pure_one_week))]).value() )

    out = pd.Series(ls,name=f'W{week}')

    return out

# %%
out = []
for i in range(len(big_num_index)):
    if i in pred_order_index:
        out.append(get_week_order(i))
    else:
        out.append(pred_pure[str(i)].rename(f'W{i}'))

# %%
out_order = pd.DataFrame(out).T
out_order.index = data[data['选中'] == 1]['Unnamed: 0'].values

# %%
```

```
out_order.values.sum()

# %%
(out_order.values*(race_mean.values+1).reshape(44,-1)).sum()

# %%
# 添加分类数据，方便还原
out_order.insert(0,"材料分类",new_df_pred["材料分类"].values)
out_order

# %%
#temp = (out_order.values*(race_mean.values+1).reshape(44,-1))
#pd.DataFrame(temp)

# %%
# 先还原，后导出
out_order.apply(lambda x:re_pre_ABC(x,data_index=1),axis=1).to_csv('order_24_week.csv')

# %%
#获取比率new_pred_pure_race_data
index = data[data['选中'] == 1]['Unnamed: 0'].values

data_order = filter_item(data_order, index)
data_supply = filter_item(data_supply, index)

pred_pure_race_data = (data_supply.iloc[:,2:] - data_order.iloc[:,2:]) / data_order.iloc[:,2:]

new_pred_pure_race_data = pred_pure_race_data.copy()

#储存到 dict中，方便调用
supplyer, week = new_pred_pure_race_data.shape
race_dict = {}
for i in range(supplyer):
    race_dict[i] = drop_nan(new_pred_pure_race_data.loc[i] + 1).tolist()

#设置函数方便抽取随机数

def get_all_race_random():
    ls = []
    for i in race_dict.values():
        ls.append(choice(i))
    return ls

# %%
race_mean.values+1

# %%
```

```
def set_plt_size(long=12,high=8):
    plt.rcParams['figure.figsize'] = (long,high)

# %%
2**16

# %%
pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)).sum(axis=0).cumsum() +
    2.84e4 - pd.Series([x*2.84e4 for x in range(1,24+1)])

# %%
pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)).sum(axis=0).cumsum() +
    2.84e4

# %%
out_order.iloc[:,1:].values * (race_mean.values+1).reshape(44,-1)

# %%
def 动态库存量(df):
    sum_temp = df.sum(axis=0).cumsum()+2.84e4
    temp = pd.Series([x*2.84e4 for x in range(1,len(sum_temp) +1)])
    return sum_temp - temp
动态库存量(pd.DataFrame(out_order.iloc[:,1:].values * (race_mean.values+1).reshape(44,-1)))

# %%
动态库存量(pd.DataFrame(out_order.iloc[:,1:].values *
    (race_mean.values+1).reshape(44,-1))).values

# %%



# %%
# 绘制预测供货量与实际供货量的仿真
sum_ls = []
set_plt_size()
for ssjds in range(32):
    weeks = out_order.columns
    simulation = []
    for week in weeks:
        race_week = get_all_race_random()
        simulation.append(out_order[week] * race_week)

    out_simulation = pd.DataFrame(simulation).T

    #out_simulation.sum(axis=0).plot(alpha=0.4,color='gray',linewidth=0.05)
    out_simulation.sum(axis=0).plot(alpha=0.1,color='gray',linewidth=0.5)
```

```
    sum_ls.append(out_simulation.sum().sum()/24)
pd.DataFrame(out_order.values *
    (race_mean.values+1).reshape(44,-1)).sum().plot(color='r',linewidth=2)
#plt.savefig('fig/预测仿真.png',dpi=320)

# %%
max(12,1,122)

# %%
def 动态库存量(df):
    sum_temp = df.sum(axis=0)
    length = len(sum_temp)
    temp = 2.84e4
    ls = []
    for i in range(length):
        temp = max((temp + sum_temp[i] - 2.84e4) , 0)
        ls.append(temp)
    return pd.Series(ls)
sum_ls = []
set_plt_size()
for ssjds in range(3200):
    weeks = out_order.columns
    simulation = []
    for week in weeks:
        race_week = get_all_race_random()
        simulation.append(out_order[week] * race_week)

    out_simulation = pd.DataFrame(simulation).T

    #out_simulation.sum(axis=0).plot(alpha=0.4,color='gray',linewidth=0.05)
    #动态库存量(out_simulation).plot(alpha=0.1,color='gray',linewidth=0.5)
    动态库存量(out_simulation).plot(alpha=0.4,color='gray',linewidth=0.05)

    sum_ls.append(out_simulation.sum().sum()/24)
动态库存量(pd.DataFrame(out_order.values *
    (race_mean.values+1).reshape(44,-1))).plot(color='r',linewidth=2)
plt.plot([0 for i in range(24)],color='blue',linewidth=1)
plt.savefig('fig/预测库存量仿真.png',dpi=320)

# %%
动态库存量(pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)))

# %%
[0 for i in range(24)]

# %%
sum_ls = []
```

```
set_plt_size()
for ssjds in range(320):
    weeks = out_order.columns
    simulation = []
    for week in weeks:
        race_week = get_all_race_random()
        simulation.append(out_order[week] * race_week)

    out_simulation = pd.DataFrame(simulation).T

    #out_simulation.sum(axis=0).plot(alpha=0.4,color='gray',linewidth=0.05)
    out_simulation.sum(axis=0).plot(alpha=0.1,color='gray',linewidth=0.5)

    sum_ls.append(out_simulation.sum().sum()/24)
pd.DataFrame(out_order.values *
     (race_mean.values+1).reshape(44,-1)).sum().plot(color='r',linewidth=2)
plt.savefig('fig/预测仿真.png',dpi=320)

# %%
(pd.DataFrame(out_order.values * (race_mean.values+1).reshape(44,-1)).sum().sum())/24

# %%
(sum(sum_ls))/320
```

1.4 work1.py

```
# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# %%
def get_bool(data,bool):
    return data[bool]
def sort_and_plot(data):
    return data.sort_values().reset_index(drop=True).plot()

# %%
#归一化方案
def Normal(x):
    return (x-x.mean()) / x.std()
def Min_Max(x):
    return (x-x.min()) / (x.max() - x.min())

# %%
```

```
def plot_and_save(data:pd.DataFrame, path:str, size=(24.0, 16.0)):
    plt.rcParams['figure.figsize'] = size
    sort_and_plot(data)
    plt.savefig(path)
    plt.cla()

# %%
gongying = "附件1 近5年402家供应商的相关数据.xlsx"
转运商 = "附件2 近5年8家转运商的相关数据.xlsx"

# %%
data_order = pd.read_excel(gongying,'企业的订货量')
data_supply = pd.read_excel(gongying,'供应商的供货量')

# %%
test_data = data_order.copy()

# %%
data_item_num = test_data.iloc[:,2:]

# %%
test_data['订货次数']=(data_item_num>0).sum(axis=1)
test_data['订货总量'] = data_item_num.sum(axis=1)
test_data['供货总量'] = data_supply.iloc[:,2:].sum(axis=1)

# %%
data_sub = data_item_num-data_supply.iloc[:,2:]

# %%
test_data['平均供货偏差'] = (((data_sub / data_item_num).abs().fillna(0)).sum(axis=1) /
    test_data['订货次数'])
test_data['单次最大供应量'] = data_supply.iloc[:,2:].max(axis=1)

# %%
test_data.iloc[:,-5:]

# %%
targets = test_data.columns[-5:]

# %%
for target in targets:
    plot_and_save(test_data[target],'fig/'+target+'.png',size=(12.0,8))

# %%
data_change = test_data.iloc[:,-5:]

# %%
```

```
for i in [ '订货总量', '供货总量', '单次最大供应量']:
    data_change[i] = np.log(data_change[i].values)

# %%
data_change = data_change.apply(Min_Max)

# %%
data_change

# %%
data_temp = np.asarray(data_change[['订货次数', '订货总量', '供货总量', '平均供货偏差',
    '单次最大供应量']])
#计算熵值
k = -1/np.log(402)
data_log= data_temp*np.log(data_temp)
data_log = pd.DataFrame(data_log)
data_log=data_log.fillna(0)
data_log=data_log.values
ls=[]
#计算变异指数
for i in range(5):
    e_j=k*data_log.sum(axis=0)[i]
    ls.append(e_j)
temp_list =[]
for i in ls:
    temp_list.append(1-i)
#计算权重
ls=[]
#删除错误定义
#del(sum)
for i in temp_list:
    ls.append(i/sum(temp_list))


# %%
print(ls,targets)
```

1.5 linear.jl

```
### A Pluto.jl notebook ###
# v0.19.9

using Markdown
using InteractiveUtils
```

```
# This Pluto notebook uses @bind for interactivity. When running this notebook outside of
    Pluto, the following 'mock version' of @bind gives bound variables a default value
    (instead of an error).
macro bind(def, element)
   quote
      local iv = try
          Base.loaded_modules[Base.PkgId(Base.UUID("6e696c72-6542-2067-7265-42206c756150"),
          "AbstractPlutoDingetjes")].Bonds.initial_value catch; b -> missing; end
      local el = $(esc(element))
      global $(esc(def)) = Core.applicable(Base.get, el) ? Base.get(el) : iv(el)
      el
   end
end

#    5928cfb0-1923-11ed-3ace-2bfad8956c7c
using JuMP,Gurobi,DataFrames,CSV,Parsers,Tables,PlutoUI

#    274a1543-f320-4705-8e98-55d6a3fbea74
function Base.filter((title,name)::Tuple, df::DataFrame)
  return filter(title=>x -> x == name, df)
end

#    dda52e10-674b-47ab-b765-aa4acf90780a
function Base.filter((title,name)::Tuple, df::DataFrame,week::Integer)
  return filter(title=>x -> x == name, df)[:,"W"*string(week-1)]
end

#    eb6d766a-bc18-4f21-9945-d2bb387a5e6d
function Base.filter((title,name)::Tuple, df::DataFrame,week::String)
  return filter(title=>x -> x == name, df)[:,week]
end

#    d2e2fc85-3b17-4fce-af9b-5f1d38109691
function Base.sum(l::Vector{Vector{AffExpr}})
  temp = zero(l[1][1])
  for i in l
    temp += sum(i)
  end
  return temp
end

#    243d047b-920d-451c-946e-1fcd45f227b4
function num2weekStr(num)
  return "W"*string(num-1)
end

#    2fd96ad3-0367-49ea-846e-07f6722f82ff
```

```
@bind week Slider(1:24)

#     36ca59d9-2891-43af-bdbb-0a2a4eee5853
md"""
 目前在计算的周:  $(week)
"""

#     9cea8136-3a62-4776-ba94-0f1aa0eaaf42
weekStr = num2weekStr(week)

#     500d4a9c-0879-4738-bf22-ad726d90fe9d
@bind limit Slider(1:5)

#     7ec11938-e35a-478c-9af3-5b8a8e0108b6
md"""
目前的转运商允许量: $(limit)
"""

#     4669b138-a94b-43f2-a4c2-0a803f1ca3d0
@bind Transshipment_capacity Slider(6000:500:22000)

#     b1530cee-90e3-483d-86e8-cc54e7c24338
md"""
目前的转运能力: $(Transshipment_capacity)
"""

#     a9ed9185-630c-4c6e-aac5-155900de98f3
md"""
## 导入数据
"""

#     4c41b978-fdd4-4b03-b621-f0753b727eb5
begin
   df_order_24_week = CSV.read("order_24_week (2).csv",DataFrame)
   df_Average_loss_of_forwarders = CSV.read("转运商平均损耗.csv",DataFrame)
   forwarders_id = df_Average_loss_of_forwarders[:,:转运商ID]
end

#     ef66f393-4a3b-4d0a-8d20-dd521b5529cd
filter((:材料分类,"C"),df_order_24_week,weekStr)

#     ccb7017d-eaaf-451e-96f0-76b729ba799c
md"""
# 线性方程求解函数
"""

#     7f084915-1fb5-46aa-bc15-e1d73ed5960e
```

```
function solve_chose_forwarders(week::String,
                    df_order::DataFrame,
                    df_Average_loss::DataFrame,
                    rate_list::Vector,
                    limit::Int64,
                    Transshipment_capacity::Int64)
  #每个材料类别的损耗量
  function sum_model(u ,
               len_of_u::Int64,
               num_of_forwarder::Int64,
               Average_loss::Vector{Float64},
               order::Vector{Float64})
     sum_i = sum([[u [i,j] * Average_loss[j] * order[i] for i in 1:len_of_u] for j in
         1:num_of_forwarder])
     return sum_i
  end

  # 全部损耗量
  function sum_model(U::Vector,
               length_list::Vector,
               num_of_forwarder::Int64,
               Average_loss::Vector{Float64},
               order_list::Vector{Vector{Float64}},
               rate_list::Vector)
     temp=0
     for (u ,len_of_u,order,rate) in zip(U,length_list,order_list,rate_list)
        temp += rate* sum_model(u ,
               len_of_u,
               num_of_forwarder,
               Average_loss,
               order)
     end
     return temp
  end

  #每一个供应商的转运量
  function sum_forwarders(index_of_forwarders::Int64,
                 U::Vector,
                 length_list::Vector,
                 Average_loss::Vector{Float64},
                 order_list::Vector{Vector{Float64}})
     temp = 0
     j = index_of_forwarders
     for (u ,len_of_u,order) in zip(U,length_list,order_list)
        temp += sum([u [i,j] * order[i] for i in 1:len_of_u])
     end
     return temp
```

```
end

# 01约束
function sum_bin(u ,len::Int64,model)
   for i in 1:len
      @constraint(model, sum(u [i,:])<=limit)
   end
end

#运货量等于订货量
function sum_bin(u ,len::Int64,
           model,
           order::Vector{Float64},
           num_of_forwarder::Int64)
   for i in 1:len
      @constraint(model, sum([u [i,j] * order[i] for j in 1:num_of_forwarder]) == order[i])
   end
end

#定义模型
model = Model(Gurobi.Optimizer)

#定义变量
num_of_forwarder=length(df_Average_loss_of_forwarders[:,1])
begin
   length_A = length(filter((:材料分类,"A"),df_order,week))
   length_B = length(filter((:材料分类,"B"),df_order,week))
   length_C = length(filter((:材料分类,"C"),df_order,week))
   @variable(model, u [i = 1:length_A, j= 1:num_of_forwarder],Bin)
   @variable(model, u [i = 1:length_B, j= 1:num_of_forwarder],Bin)
   @variable(model, u [i = 1:length_C, j= 1:num_of_forwarder],Bin)
end

#定义目标函数
@objective(model, Min,
      sum_model([u ,u ,u ],
         [length_A,length_B,length_C],
         num_of_forwarder,
         df_Average_loss[:,:平均损耗],
         [filter((:材料分类,"A"),df_order,week),
            filter((:材料分类,"B"),df_order,week),
            filter((:材料分类,"C"),df_order,week)],
         rate_list))

#定义约束
#01约束
for (u ,len_of_u) in zip([u ,u ,u ],[length_A,length_B,length_C])
```

```
    sum_bin(u ,len_of_u,model)
  end
  #供应商的转运量小于6000
  for j in 1:num_of_forwarder
    @constraint(model,
    sum_forwarders(j,[u ,u ,u ],[15,14,15],
            df_Average_loss[:,:平均损耗],
            [filter((:材料分类,"A"),df_order,week),
            filter((:材料分类,"B"),df_order,week),
            filter((:材料分类,"C"),df_order,week)])
    <=Transshipment_capacity)
  end
  #运货量等于订货量
  for (u ,len_of_u,order) in zip([u ,u ,u ],
                [length_A,length_B,length_C],
                [filter((:材料分类,"A"),df_order,week),
                  filter((:材料分类,"B"),df_order,week),
                  filter((:材料分类,"C"),df_order,week)])
    sum_bin(u ,len_of_u,model,order,num_of_forwarder)
  end
  optimize!(model)
  return model,(u ,u ,u )
end

#    407fe9e3-c524-4b51-a0dd-0229e398f535
function solve_chose_forwarders(week::String,
                    df_order::DataFrame,
                    df_Average_loss::DataFrame)
  solve_chose_forwarders(weekStr,df_order,df_Average_loss,[1.2,1.1,1,1,1],limit,Transshipment_capacity)
end

#    3737fa99-4b4d-46f3-bdd4-106e589aa847
(model,var) = solve_chose_forwarders(weekStr,df_order_24_week,df_Average_loss_of_forwarders)

#    febc35df-f986-43e6-a98a-6a8107ef7dd9
termination_status(model)

#    1f51645c-fa66-45d0-a2a3-846554264525
termination_status(model)

#    14a0e20c-b79d-4d0f-aed9-e31753f2ddfa
md"""
# 把解转化为坐标
"""

#    02bd0f1c-4c41-4a3a-8928-d578cccec15a
# 解矩阵转化为数字
```

```
function mat2sym(mat::Matrix)

  function var2sym(sym_list::Vector,
              var::Vector)
    temp = missing
    for (i,sym) in zip(var,sym_list)
      if i == 1.0
        temp = sym
        break
      end
    end
    return temp
  end

  function mat2vec(mat::Matrix)
    x,_ = size(mat)
    return [mat[i,:] for i in 1:x]
  end

  return mat |> x->value.(x) |> mat2vec |> x->(x-> var2sym(forwarders_id,x)).(x)
end

#    9851ee35-f131-48e7-b87e-1c03d27b5263
# 解转化为坐标数字元组
function solve2location(供应商ID表_list::Vector,
               solve_list::Tuple,
               data_list::Vector)
  #辅助函数
  function delete(str::AbstractString,
            del::String)
    return replace(str,del=>"")
  end

  function sym2num(str::AbstractString)
    return str |> x->delete(x,"T") |> x->delete(x,"S") |> x->Parsers.parse(Int64,x)
  end

  function sym2num(str::Missing)
    return -1
  end

  function mat2location(供应商ID表::Vector,
                solve::Matrix,
                data::Vector)
    x = 供应商ID表 |> x->sym2num.(x)
    y = solve |> mat2sym |> x->sym2num.(x)
    return zip(x,y,data)
```

```
  end

  temp = []
  for (供应商ID表,solve,data) in zip(供应商ID表_list,solve_list,data_list)
    output = mat2location(供应商ID表,solve,data)
    for i in output
      temp = cat(temp,i,dims=1)
    end
  end
  return temp
end

#    a083a5da-0536-4bd9-9c5f-2ec66678d7f2
begin
  供应商ID表_list = [filter((:材料分类,i),df_order_24_week)[:,:Column1] for i in ["A","B","C"]]
  周订购数据_list = [filter((:材料分类,i),df_order_24_week)[:,week+2] for i in ["A","B","C"]]
  location_list = solve2location(供应商ID表_list, var, 周订购数据_list)
end

#    393930a9-26f6-4a25-9648-97b90d21eb51
md"""
# 写入表格，准备复制
"""

#    c9dee99e-9a55-4092-b2e4-e48dc1dcccba
#temp = DataFrame(fill!(Matrix{Float64}(undef, 402, 8),-114.514), :auto)

#    c7764ec3-6fc2-455a-8fdf-b1a319d4c0f6
for i in location_list
  (y,x,data) = i
  if x == -1
    continue
  end
  temp[y,x] =data
end

#    a9df1cd9-bfe1-461d-bac3-a91589c2163f
CSV.write("temp_location-$(weekStr).csv",temp)

#    7e218375-87e3-44a7-afa3-d5f4052375fc
#read("temp_location-$(weekStr).csv",String) |> x->replace(x,"-114.514"=>"") |> x->
    write("temp_location-$(weekStr).csv",x)

#    606088ae-d999-42ea-aa98-308412ac779d
function write_table(location_list::Vector,weekStr::String)
  temp = DataFrame(fill!(Matrix{Float64}(undef, 402, 8),-114.514), :auto)
  for i in location_list
```

```
    (y,x,data) = i
    if x == -1
       continue
    end
    temp[y,x] =data
  end
  path = joinpath("location","temp_location-$(weekStr).csv")
  CSV.write(path,temp)
  read(path,String) |> x->replace(x,"-114.514"=>"") |> x-> write(path,x)
end

#    2927eb71-9b9e-4026-8857-3331e715fc39
joinpath("location","temp_location-$(weekStr).csv")

#    618e342c-f035-4337-8705-91c10eaffbfd
write_table(location_list,weekStr)
```