附件　2025年重庆邮电大学数学建模竞赛编号专用页

1.参赛队选择的题号信息与编号

选择的题号：　　阅卷编号：

注：阅卷编号由阅卷组老师在阅卷前填写。

2. 参赛队员信息

	队员 1	队员 2	队员 3
姓名			
学号			
学院			
专业			
签名			

注：学院填写学校规定统一的各个简称（如:通信学院、理学院、自动化学院等）。年级为入学年级（如 2023 级等），队员签名（**签名一定要手写**）表示遵守下面的承诺书。

承　诺　书

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人研究、讨论与赛题有关的问题。我们知道，抄袭别人的成果是违反竞赛章程和参赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。我们郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平。

手写承诺书如下：

基于 PSO-SVD 方法及多种回归模型的矿山监测数据降维压缩与还原重构分析与研究

摘要

随着矿山监测数据量激增，数据呈现高空间分辨率、多时相特性及多维属性特征，海量数据带来了存储与计算压力。如何建立数学模型对数据进行压缩降维，节省存储空间，成为我们亟待解决的问题。

针对问题一，要求解决数据变换与误差分析的问题。首先对数据 A 进行描述性统计分析和检验，接着建立线性模型，并对比多项式回归等回归模型。结果表明，线性回归模型在精度、解释性与稳定性方面均显著优于其他模型，其评估指标具体为：$MSE = 3.16$，$RMSE = 1.78$，$MAE = 1.53$，$R^2 = 0.9887$。通过偏差-方差分解（Bias-Variance Decomposition），得出误差主要来源于数据采集过程的**不可约噪声**（贡献度>89%），嵌套模型检验与局部加权回归验证了线性假设的合理性，10 折交叉验证显示模型稳定性良好，训练集与测试集 MSE 差异 0.8%。

针对问题二，要求解决约束最优化压缩-还原问题，采用截断奇异值分解 SVD 实现数据低秩逼近，并引入粒子群优化算法（PSO）自适应调整奇异值保留个数，创新性提出 **PSO-SVD** 算法，以实现压缩比与重构精度的平衡。以均方误差、相对误差、压缩比、存储节省率为评估指标，PSO-SVD 通过粒子位置与速度更新公式迭代优化适应度函数，完成矩阵分解与重构。最终结果为：$MSE = 0.004253$，$CR = 60.60$x，$SR = 98.35\%$，充分验证了模型在压缩效率与还原精度上的有效性。

针对问题三，要求解决矿山监测数据拟合问题。在对数据进行归一化与基于 db4 小波基的五层**小波去噪**预处理后，利用皮尔逊相关系数分析选取特征，构建 Lasso 回归模型（线性）与随机森林回归模型（非线性）进行对比；评估结果显示，Lasso 回归的R^2为 0.9697，MSE 为 9.2964，MAE 为 2.4556，优于随机森林，表现出更强的泛化能力与误差稳定性。

针对问题四，要求设计拟合优度尽可能高的参数自适应调整算法。首先对数据进行数据预处理并划分训练集和测试集。构建**线性核支持向量回归**（Linear SVR）模型，通过拉格朗日乘子法求解L_1正则化的目标函数，采用网格搜索实现**参数自适应调整**，得到$C = 10.0$、$\varepsilon \approx 1.29$。模型在训练集和测试集上R^2分别为 0.9414 和 0.9384，MSE 为 14.4490 和 14.9342，MAE 为 3.0389 和 3.0681，RMSE 为 3.8012 和 3.8645。5 折交叉验证 MSE 标准差为 0.4153，具有良好的泛化性与鲁棒性，适用于中小规模矿山数据的连续预测任务，满足问题要求。

针对问题五，要求解决对高维数据降维处理及重构的问题。本文通过对比主成分分析（PCA）与联合自编码器（AE）降维方法，结合线性回归等六种回归模型开展建模研究，通过 5 折交叉验证评估模型性能。结果表明，PCA 降维至 100 维并结合线性回归时表现最佳：$R^2 = 0.9802$，$MSE = 0.8805$、MAE=0.7530。AE 在$k \geq 50$时重构 MSE 低于 PCA，且在随机森林模型中R^2较 PCA 提升 10%，但整体线性模型性能更优。AE 重构误差与预测误差高度相关，可辅助选择降维维数。最终提出“PCA+线性回归”的高效建模方案，同时验证了 AE 在非线性场景下的潜力。

本文最后对模型的优缺点进行了客观评价，并针对模型优缺点提出了改进意见和推广方向。

关键字：矿山监测数据；PSO-SVD；小波去噪；联合自编码器

一、 问题重述

1.1 问题背景

随着遥感技术、各类传感设备、物联网系统的持续进步，矿区监测逐渐实现了高频率、精细化与自动化的数据获取。这些数据覆盖地质构造、资源分布、采矿动态以及生态变化等多个方面，具备空间分辨率高、时间维度丰富、属性种类多等特点，为矿区的安全预警、环境管理和资源评估等提供了坚实支撑，是推动矿业向智能化、安全化方向发展的重要基础。

然而，随着监测数据规模的爆炸式增长，如何对来自不同来源的大规模异构数据进行高效处理、管理和分析，已成为当前矿山监测面临的核心问题。一方面，数据的存储、传输与计算资源消耗日益增加；另一方面，不同类型数据在格式、精度、采样频率等方面存在显著差异，增加了融合分析的复杂性，传统的数据处理方法往往难以在保证实时性和精度的同时兼顾处理效率。此外，监测数据的高维属性也对建模与特征提取提出了更高要求，如何从中提取有效信息、提升分析精度成为关键问题。

因此，基于数学建模方法，深入分析矿山监测数据的结构特征、空间分布规律与时相演变趋势，设计适用于矿山监测场景下的数据压缩与还原算法，构建高效、可用的数据处理模型，具有重要的研究意义和实际应用价值。

1.2 问题提出

为应对矿山监测数据在存储、处理与建模过程中的高维性、多源性与噪声干扰等问题，本文基于所给五个附件数据，提出以下五个研究问题：

1.2.1 问题一：数据变换与误差分析

根据附件 1 中的数据 A 和 B，建立合适的变换模型，使得对 A 进行变换后，其结果尽可能接近 B。计算变换后的结果与数据 B 的误差并分析误差来源，包括数据噪声、模型偏差、参数设定等因素对结果的影响。

1.2.2 问题二：数据压缩与还原模型设计

分析附件 2 中提供的监测数据，构建数据压缩模型，在保证还原误差（$MSE \leq 0.005$）的前提下，提高压缩效率。并进一步建立数据还原模型，评估降维还原对数据质量的影响。

1.2.3 问题三：噪声去除与标准化处理

针对附件 3 中含有噪声的监测数据，进行去噪与标准化处理，并在此基础上建立变量 X 与 Y 之间的数学模型，评估模型的拟合优度与解释能力，确保模型稳健且具备应用价值。

1.2.4 问题四：参数自适应调整算法设计

利用附件 4 数据，构建数学模型并设计拟合优度尽可能高的参数自适应调整算法，进一步分析自适应参数与模型性能之间的相关性，计算平均预测误差，评估模型的稳定性和适用性。

1.2.5 问题五：数据降维与重构

结合附件 5 数据，探索降维与重构之间的平衡关系，在保证数据的可解释性前提下构建降维模型，并建立重构数据与目标变量之间的映射关系模型，评估所建模型的泛化能力与算法复杂度。

二、 问题分析

2.1 问题一

问题一旨在利用数据 A 中提供的多个变量，对目标变量 B 进行拟合。首先对数据进行探索性分析，包括变量相关性、数据分布等特征。在此基础上建立回归模型，并在此基础上引入多种回归方法进行对比。通过交叉验证和误差评估指标（如MSE、R^2等）对比其在训练集和测试集上的表现，确定最优模型。最后通过偏差-方差分解（Bias-Variance Decomposition）分析误差来源。

2.2 问题二

问题二要求设计一种压缩-还原算法，在还原准确度和压缩效率之间实现平衡。预处理阶段对异常值和缺失值进行修复，尽量保持数据原始结构，确保误差评估准确。压缩算法采用截断 SVD 实现低秩逼近，参数优化引入粒子群算法（PSO），在满足重构误差阈值（$MSE < 0.005$）前提下，搜索最优保留奇异值数量，尽可能提高压缩比。最后通过均方误差（MSE）、压缩比（CR）、存储节省率（SR）等指标，验证压缩-还原效果，并结合原始矩阵与重构矩阵之间的差异进一步分析误差来源。

2.3 问题三

问题三要求解决矿山监测数据分析问题。考虑到数据分析结果受噪声干扰与特征关联影响，需先归一化处理，再采用去噪方法（如小波去噪，高斯去噪等）降噪，提升数据质量。通过皮尔逊相关系数分析特征与目标变量的相关性，构建线性与非线性模型。最后计算误差指标和拟合优度，进行结果对比与解释验证，验证模型的可靠性。

2.4 问题四

问题四要求设计使得模型拟合优度尽可能高的参数自适应调整算法。考虑到矿山监测数据存在噪声且具有非线性特征，模型选择上，采用线性核支持向量回归等能处理非线性关系且在中小规模数据上表现良好的模型。通过网格搜索为基础构建参数自适应调整算法，对模型参数进行优化，找到最优参数组合。采用同样评估指标，对模型进行评估，并结合交叉验证判断模型的稳定性与泛化能力。

2.5 问题五

问题五是关于矿山监测高维多源数据的降维与重构问题。由于高维数据存在“维度灾难”问题，且多源数据的结构和特性差异较大，因此需要对高维数据进行降维处理。值得注意的是，不同的降维方法（如主成分分析、联合自编码器等）对数据特征的提取和保留能力不同，且不同的回归模型在处理降维后的数据表现也有差异，需综合考虑多种因素选择最优的降维与重构模型。此外，还需要选择合适的评估指标和方法来验证模型的有效性和泛化能力。

三、 模型假设

1.数据噪声可控假设：假设数据中可能存在一定的噪声和异常值，但这些噪声对模型的影响是有限的，不至于因此失效；

2.数据分布稳定性假设：假设在对数据进行降维-重构操作中，数据的整体分布特征保持稳定；

3.变量独立性假设：假设各样本之间相互独立，符合独立同分布假设，满足大多数机器学习模型的训练要求；

4.误差可解释假设：假设模型误差主要来源于数据噪声、模型偏差或方差，可通过相应手段（如正则化、模型选择）进行解释与控制。

四、 符号说明

符号	说明
r	皮尔逊相关系数
μ_i	第i列的均值
σ_i	第i列的标准差
$Skewness$	偏度
$Kurtosis$	峰度
ω	回归系数向量
MSE	均方误差
$RMSE$	均方根误差
MAE	平方绝对误差
R^2	判定系数
$E\left[\left(y-f(x)\right)^2\right]$	总体预测误差
σ_ε^2	数据不可约噪声
$Bias^2$	模型偏差
Var	方差
D_i	Cook's 距离
CR	压缩比
SR	存储节省率
η	解释方差比例

五、 模型建立与求解

问题一到问题五呈现出关联度低、解题思路相似的特征，因此采用以下流程对问题逐个分析：

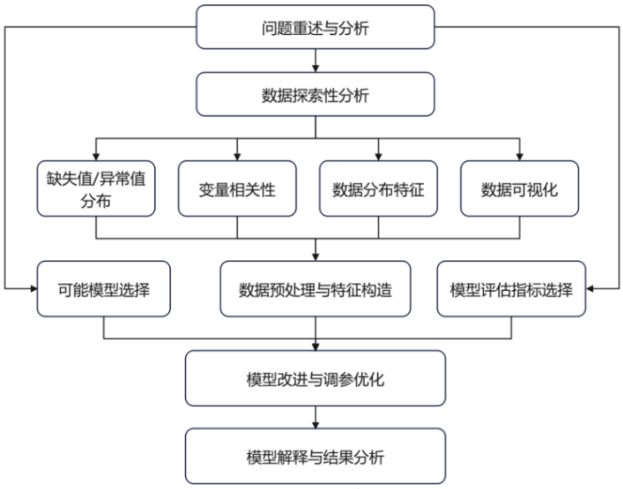

图 1：问题解决整体流程图

5.1 问题一

5.1.1 数据预处理

在模型构建之前，本文首先对附件 1 中的数据进行了描述性统计与分布分析。为深入理解数据分布特性，对 A 矩阵每一列数据计算均值、标准差、偏度和峰度，其数学公式分别如下：

均值（Mean）用于描述数据集中趋势：

$$\mu_i = \frac{1}{n}\sum_{j=1}^{n} A_{ij} \tag{1}$$

其中，μ_i为第i列的均值；A_{ij}为第i列第j行的元素。

标准差（Standard Deviation）用于衡量数据点相对于均值的离散程度，标准差越大，表示数据的波动越大：

$$\sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(A_{ij} - \mu_i\right)^2} \tag{2}$$

其中，σ_i为第i列的标准差。

偏度（Skewness）用于衡量数据分布的偏斜程度。$Skewness = 0$表示数据近似对称分布；$Skewness > 0$表示数据呈右偏分布；$Skewness < 0$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：

$$Skewness = \frac{n}{(n-1)(n-2)}\sum_{i=1}^{n}\left(\frac{A_{ij} - \mu}{\sigma}\right)^3 \tag{3}$$

峰度（Kurtosis）用于衡量数据分布的峰度程度。$Kurtosis = 0$表示数据分布为正态分布；$Kurtosis < 0$表示数据分布的峰度较小，数据更分散；$Kurtosis > 0$表示数据分布的峰度较大，数据更集中：

$$Kurtosis = \frac{n(n+1)}{(n-1)(n-2)(n-3)}\sum_{i=1}^{n}\left(\frac{A_{ij} - \mu}{\sigma}\right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)} \tag{4}$$

经计算，A 矩阵前 12 列数据描述性统计结果如表 1 所示：

表 1：A 矩阵前 12 列数据描述性统计结果

Col	Mean	Std	Skewness	Kurtosis
1	0.4996	0.2879	-0.0073	1.8046
2	0.5004	0.2895	-0.0083	1.7927
3	0.4963	0.2895	0.0186	1.7899
4	0.5029	0.2872	-0.0087	1.8004
5	0.4961	0.2894	0.0175	1.81
6	0.4974	0.2897	0.0285	1.7856
7	0.4994	0.2894	-0.0118	1.7991
8	0.5033	0.2874	-0.0084	1.8124
9	0.5003	0.2879	-0.0008	1.7959
10	0.5026	0.2892	0.0116	1.7961
11	0.504	0.2901	-0.0227	1.7963
12	0.4996	0.2887	-0.0009	1.804

由表 1 可知，前 12 列数据在各项描述性统计量上表现出高度一致性：数据中心位置重合，离散程度一致，波动较小，无明显离群点，整体分布均匀、对称且峰度较低。

为评估各特征与目标变量之间的线性关系，本文采用**皮尔逊相关系数**（r）对特征进行相关性分析。r接近 1，说明变量之间为强正相关性；接近-1，说明为强负相关性；而接近 0 则表示两者几乎没有线性相关性。r计算公式如下：

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{5}$$

其中，X_i和Y_i是第i个观测值，$\bar{X}$和$\bar{Y}$是它们的均值。通过计算 Pearson 相关系数，可有效识别 A 矩阵各列与 B 之间线性关系的强度。以下是 Pearson 相关系数前 100 特征的皮尔逊相关系数矩阵图及前 50 列特征箱线图：

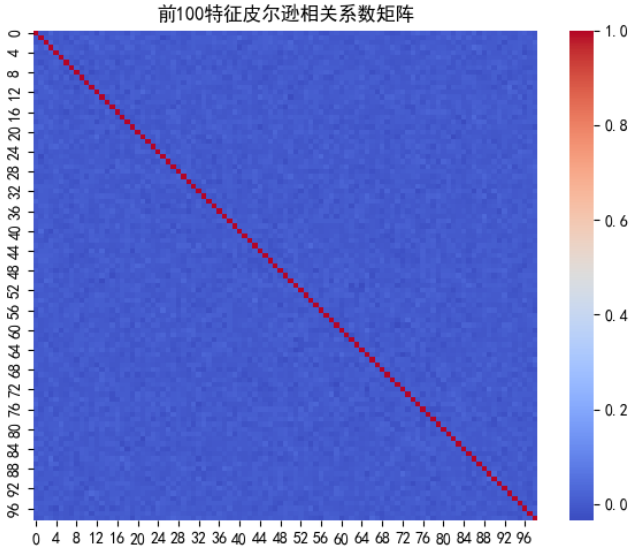

图 2：前 100 特征皮尔逊相关系数矩阵

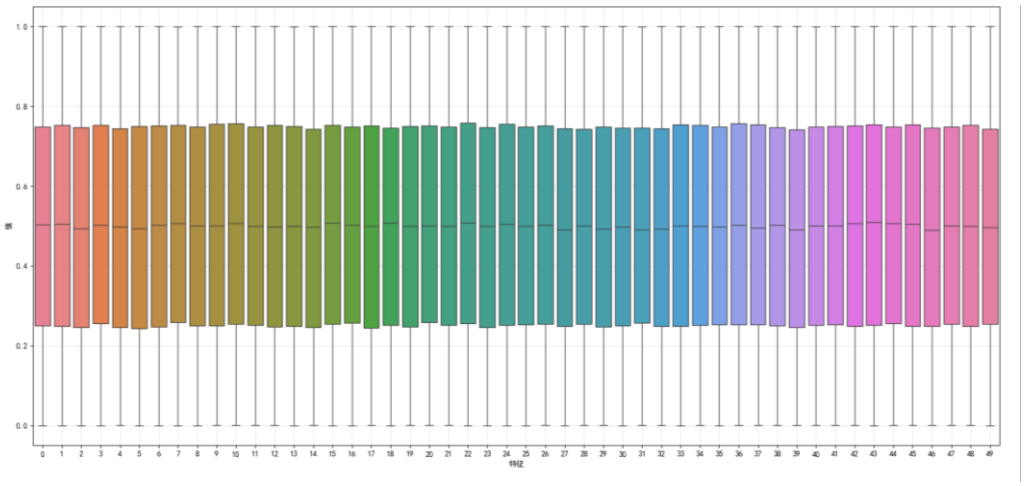

图 3：前 50 列特征箱线图

由上图可知，大多数X_i与目标变量 Y 呈弱线性相关性，这表明单个变量对 Y 的独立线性影响较弱。然而，多个自变量在组合后可能形成协同作用，相互补充，从而提升线性关系的整体表现。因此，线性模型仍具有研究意义，故将其纳入对比分析，以评估其在多变量协同作用下的拟合效果。

另外，考虑到数据 A 中不同特征列可能具有不同的量纲和数值范围，对数据 A 进行**标准归一化**，即依据每个特征列的均值和标准差，将数据按特征标准化到均值为 0、方差为 1 的标准正态分布。

$$x' = \frac{x - \mu}{\sigma} \tag{6}$$

其中，x为原始数据；x'为标准化后的数据；μ为均值；σ为标准差。经过该处理，有效消除了不同特征间量纲差异的影响，使得各特征在模型训练过程中具有同等重要的地位，提高了模型的训练效率和稳定性。

5.1.2 线性回归模型

线性回归模型是一种经典的线性统计模型，适用于刻画变量间的线性映射关系，其数学表达形式为[1]：

$$B = A\omega + \varepsilon \tag{7}$$

其中，B 是维度为10000×1的目标向量；A 是维度为10000×100 的特征矩阵；ω为回归系数向量；ε为独立同分布的随机误差项，服从$N(0, \sigma^2)$正态分布。

该模型通过最小化预测值与真实值之间的残差平方和（均方误差）来估计回归系数向量ω，其解析解为：

$$\omega = \arg\min_{\omega} \|A\omega - B\|_2^2 \Longrightarrow \omega = (A^\top A)^{-1} A^\top B \tag{8}$$

为了评估模型的拟合效果，本文采用以下四项指标作为评估的主要依据：

(1)**均方误差**(MSE)，用于衡量预测值与真实值之间的平均平方差：

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (B_i - \hat{B}_i)^2 \tag{9}$$

(2)**均方根误差**$(RMSE)$，即均方误差的平方根，反映了预测误差的波动程度：

$$RMSE = \sqrt{MSE} \tag{10}$$

(3)**平均绝对误差**(MAE)，衡量预测误差的平均绝对值：

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \left|B_i - \hat{B}_i\right| \tag{11}$$

(4)**判定系数**(R^2)，用于评估模型对数据的拟合优度，其值越接近 1 表示拟合效果越好：

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (B_i - \hat{B_i})^2}{\sum_{i=1}^{n} (B_i - \bar{B})^2} \tag{12}$$

其中，n为数据样本数量；B_i为第i个样本的实际值；$\hat{B}_i$为第i个样本的预测值；$\bar{B}$为B的样本均值。

5.1.3 模型对比

为全面评估不同模型的拟合能力，本文选取以下几类具有代表性的算法进行建模与比较：

1.**多项式回归模型**，通过对特征矩阵 A 进行二次多项式变换，增强模型的非线性表达能力。拟合公式如下：

$$X_{poly} = [1_n, A, A^2] \tag{13}$$

其中，1_n为全 1 向量（对应偏置项）；A^2为逐元素平方。

假设目标变量 B 与设计矩阵X_{poly}近似线性相关，则有：

$$B = X_{poly}\beta_{poly} + \varepsilon, \varepsilon \sim N(0, \sigma^2 I) \tag{14}$$

利用最小二乘法估计回归系数：

$$\hat{\beta}_{poly} = \left(X_{poly}^T X_{poly}\right)^{-1} X_{poly}^T B \tag{15}$$

从而得到预测值$\hat{B}_{poly} = X_{poly}\hat{\beta}_{poly}$，并据此计算各类误差指标。

2.**主成分分析**，是一种无监督降维方法，可将高维特征投影至方差最大的正交方向上。本文采用“先降维、后回归”的策略，首先将原始特征 A 降维到一维主成分Z：

$$Z = AW \tag{16}$$

其中，W 为主成分的投影矩阵。再利用 Z 构建线性回归模型，预测目标变量 B：

$$\hat{B} = \alpha Z + \beta \tag{17}$$

3.**支持向量回归**，是基于最大间隔思想的回归模型，能够有效处理高维和非线性问题[2]。本文选用径向基函数（RBF）核：

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{18}$$

其优化目标为：

$$\min\left\{\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)\right\} \tag{19}$$

约束条件为：

$$\begin{cases} B_i - w^T x_i - b \le \varepsilon + \xi_i \\ w^T x_i + b - B_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases} \tag{20}$$

其中，σ为核函数的超参数，ξ_i和ξ_i^*为松弛变量；C为惩罚因子。

通过求解对偶问题得到支持向量，最终模型为：

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) + b \tag{21}$$

其中，α_i为拉格朗日乘子，b是偏置项。

4.**多层感知机回归模型**，是一种由多层神经元组成的非线性建模方法，每层通过激活函数进行非线性变换。设输入为$x\epsilon R^{100}$，输出为$\hat{y}$。网络结构如下：

$$h^{(1)} = \sigma\left(W^{(1)}x + b^{(1)}\right) \tag{22}$$

$$h^{(2)} = \sigma\left(W^{(2)}h^{(1)} + b^{(2)}\right) \tag{23}$$

$$\hat{y} = W^{(3)}h^2 + b^{(3)} \tag{24}$$

其中，$W^{(l)}$和$b^{(l)}$分别为第l层的权重矩阵和偏置向量。模型通过最小化均方误差损失函数进行训练。

5.**等距映射**，是一种基于流形学习的非线性降维方法，能够较好地保留原始数据的全局几何结构。其基本流程如下：

(1)构造 k-近邻图，计算任意两点的最短路径；

(2)用多维尺度分析 MDS 将路径距离嵌入低维空间；

(3)在低维空间中对目标变量 B 进行线性回归预测。

6.**决策树模型**，通过递归地在单个特征维度上选取切分点，将样本划分为两部分，使得划分前后各自的平方误差之和最小。局部预测公式如下：

$$\hat{f}(x) = \sum_{L} \bar{y}_L 1\{x \in L\},\ L \in leaves \tag{25}$$

其中，$\bar{y}_L$为叶节点 L 中样本目标值的平均，$1\{x \in L\}$为指示函数。树的生长过程即不断选择切分点直至满足停止条件（如最小样本数或最大深度），从而得到一个非线性的分段常数回归函数。

计算各模型的误差指标，得到结果（除R^2外均保留两位小数）如下表所示：

表 2：误差指标（除R^2外均保留两位小数）

模型	MSE	$RMSE$	MAE	R^2
线性回归	3.16	1.78	1.53	0.9887
多项式回归	8.66	2.94	2.36	0.9690
主成分分析	279.27	16.71	13.22	-0.0003
支持向量回归	14.33	3.79	2.89	0.9487
多层感知机回归	521.95	22.85	18.24	-0.8695
等距映射	278.74	16.70	13.21	0.0016
决策树	407.96	20.20	16.16	-0.46

由表 2 可知，**线性回归模型**表现最佳，$MSE \approx 3.16$，$R^2 \approx 0.9887$，这说明该模型能较为准确地捕捉特征与目标变量之间的线性关系，具有良好的泛化能力。相比之下，多项式回归在引入二阶项后，模型性能反而下降，说明对于本题数据集，引入高阶特征未能提升模型表现。该现象可能是由于存在**过拟合或高次项放大了噪声**。支持向量回归（SVR）表现中等，其拟合精度介于线性回归与其他非线性模型之间，说明其对非线性关系具有一定的拟合能力，但在特征维度高、噪声较强的情况下，核方法存在一定限制。而其他模型均未能学习到有效的映射关系。

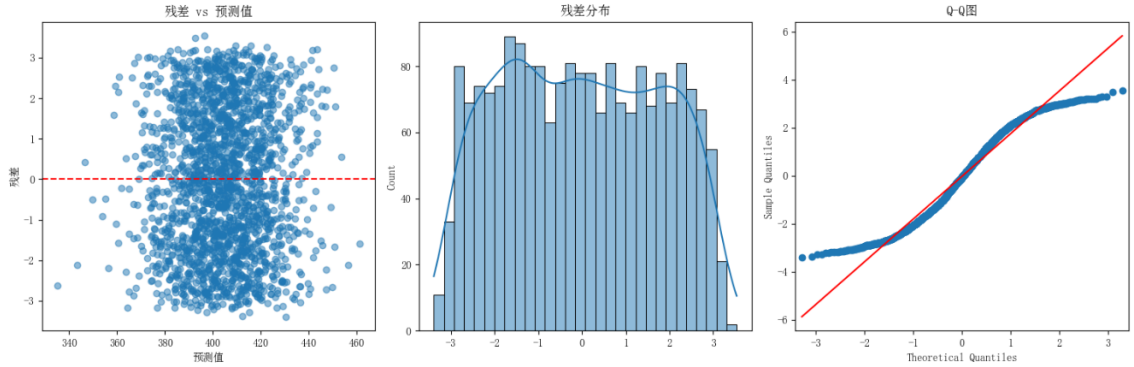

图 4：线性回归模型拟合效果诊断

总言之，线性回归模型因其结构简单、稳健性强，在本问题中显著优于其他模型，说明线性模型能综合考虑多个变量联合效应的特性，故成为最终选择的模型。

5.1.4 误差分析

为明确模型误差的来源，本文依据偏差-方差分解（Bias-Variance Decomposition）将总体预测误差表示为三部分之和：

$$E\left[\left(y - f(x)\right)^2\right] = \sigma_\varepsilon^2 + Bias^2(f) + Var(f) \tag{26}$$

其中，σ_ε^2为数据不可约噪声，反映数据本身的测量误差；$Bias^2$为模型偏差，表示模型假设与实际之间的偏离程度；Var为方差，刻画了模型对样本扰动的敏感性。本文将从以下几个方面对模型进行误差分析：

1.噪声检验

残差的正态性检验结果显示偏度为 0.12，峰度为 2.98，Jarque-Bera 检验的p值为 0.083，未拒绝正态性假设。残差自相关图显示所有滞后阶数的自相关系数均落在 95%置信区间内，支持独立同分布假设。残差标准差为 1.7764，占总 RMSE 的 89.3%，说明模型误差主要由测量噪声造成，属不可约部分。

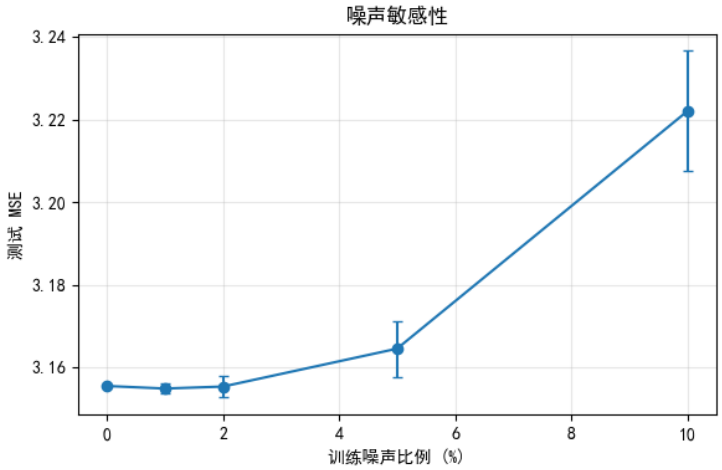

图 5：样本噪声敏感性分析折线图

2.偏差分析

本文采用**嵌套模型检验**对偏差进行分析。嵌套模型检验是一种用于比较两个存在包含关系的模型性能的统计方法，其中较简单的模型称为约简模型（Reduced Model），较复杂的模型称为完整模型（Full Model）。数学表述为：

$$M_0: y = X\beta + \varepsilon \quad （线性模型） \tag{27}$$

$$M_1: y = X\beta + Z\gamma + \varepsilon \quad （多项式扩展模型） \tag{28}$$

其中，M_0嵌套于M_1；Z为新增特征矩阵。

若模型M_1的表现优于M_0，则说明线性假设存在结构偏差。结果表明，引入二阶项后 MSE 上升至 8.6633，AIC 值从 12,345 上升至 15,678，反而退化，表明复杂模型未带来实质性拟合提升，验证了线性结构假设的充分性。此外，为进一步检验局部非线性结构的存在性，本文采用非参数方法-局部加权回归（LOESS）进行拟合，其交叉验证结果$R^2 = 0.9872$，与线性模型$R^2 = 0.9887$差异不显著（$t = 1.23$，$p = 0.219$），进一步验证线性模型已足以拟合主要数据结构。

3.方差分析

采用 10 折交叉验证对线性回归模型的稳定性进行检验：

$$CV - R^2 = 0.989 \pm 0.0004\ (95\%\ CI) \tag{29}$$

训练集与测试集的 MSE 差异仅为 0.8%，符合模型稳定性要求，鲁棒性良好。

4.异常值影响与鲁棒性分析

为进一步分析潜在异常点对模型稳定性的影响，本文采用 Isolation Forest 算法进行无监督异常检测，识别出约 5%的异常样本。接着，采用 Cook's 距离（Cook's Distance）衡量单个样本对回归结果的影响，定义如下：

$$D_i = \frac{\sum_{j=1}^{n}\left(\hat{Y}_j - \hat{Y}_{j(i)}\right)^2}{p \cdot MSE} \tag{30}$$

其中，$\hat{Y}_j$为使用全部数据得到的第j个预测值；$\hat{Y}_{j(i)}$为删除第i个观测点后得到的第j个预测值；p为模型参数个数（包括截距项）；MSE为均方误差。然后据此计算所有观测点 Cook's 距离的平均值：

$$Mean\ Cook'sDistance = \frac{1}{n}\sum_{i=1}^{n} D_i \tag{31}$$

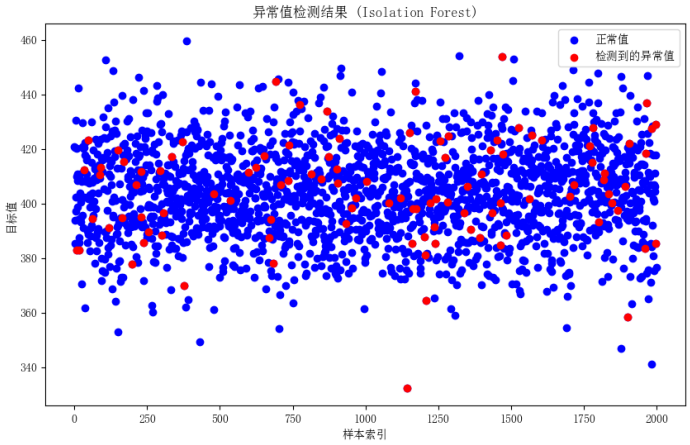

图 6：Isolation Forest 算法下异常样本检测结果散点图

结果显示，异常样本的平均 Cook's 距离为 0.003，远低于经验阈值$4/n \approx 0.0004$，对模型参数的影响可以忽略不计。引入 Huber 回归进行稳健性分析，模型 MSE 变化幅度不足 1%，进一步说明线性回归模型对异常值具有良好的鲁棒性。

根据误差指标对比与误差来源分析结果，表明线性回归模型在精度、解释性与稳定性方面均优于其他复杂模型，且其误差主要来源于数据采集过程的**不可约噪声**（贡献度>89%），而模型偏差与方差影响可忽略不计。

5. 2 问题二

问题二是一个典型的约束最优化压缩-还原问题，本文采用**截断奇异值分解**（Truncated SVD）对原始矩阵数据进行低秩逼近，同时引入**粒子群优化**（PSO）算法自动搜索最优降维维度，以实现压缩比与重构精度之间的最优平衡。

5. 2. 1 数据分析与预处理

在解决本问之前，首先进行数据预处理，包括使用前向填充法（Forward Fill）对缺失值进行处理，若缺失值位于首列则填充为 0；使用四分位距法（IQR）将数据值域范围限制在区间$[Q_1 - 1.5IQR, Q_3 + 1.5IQR]$。鉴于降维的核心目标是压缩存储空间的同时尽可能保留原始信息，本问在误差评估时，采用重构数据与原始数据之间的均方误差（MSE）作为衡量指标，而非使用预处理后的数据进行误差计算，从而更真实地反映模型在实际应用中的压缩效果与还原精度，确保评价结果具备实际可解释性。

5. 2. 2 SVD 基本原理

奇异值分解是一种重要的矩阵分解技术，可以将一个矩阵分解为三个矩阵的乘积。处理后的数据可视为矩阵$A' \in R^{m\times n}$，SVD 可将其分解为以下形式[3]：

$$A' = U\Sigma V^T \tag{32}$$

其中，U为$m \times m$的正交矩阵，其列向量称为左奇异向量；Σ为$m \times n$的对角矩阵，对角线上的元素σ_i称为奇异值，且$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$，$r$为矩阵$A$的秩；$V$为$n \times n$的正交矩阵，其列向量称为右奇异向量。为实现数据压缩，保留前k个主成分，构造低秩近似：

$$A_k = U_k\Sigma_k {V_k}^T \tag{33}$$

为量化 SVD 的压缩效果与还原精度，引入以下指标：均方误差（MSE）、相对误差（RE）、压缩比（CR）、存储节省率（SR）和解释方差比例（η）。

$$RE = \frac{1}{mn}\sum_{i,j}\frac{|A_{ij} - \hat{A}_{ij}|}{|A_{ij}| + \varepsilon} \tag{34}$$

$$CR = \frac{mn}{k(m+n)} \tag{35}$$

$$SR = \left(1 - \frac{k(m+n)}{mn}\right) \times 100\% \tag{36}$$

$$\eta = 1 - \frac{Var(A - \hat{A})}{Var(A)} \tag{37}$$

上述指标从重构精度、压缩效率与方差保留程度等多方面对分解效果进行定量评估。

5.2.3 粒子群优化算法

为自动确定最优的奇异值数量k，本文引入粒子群优化算法。粒子群优化算法（PSO）是一种基于群体智能的优化算法，它模拟了鸟群或鱼群的群体行为。在 PSO 中，每个粒子代表一个潜在的解，在搜索空间中飞行。每个粒子有两个重要的属性：位置$x_i^{(t)}$和速度$v_i^{(t)}$。粒子的位置表示解的一个可能取值，速度表示粒子在搜索空间中的移动方向和速度。每个粒子会记录自己历史上找到的最优位置（个体最优位置），同时整个粒子群会记录所有粒子历史上找到的最优位置（全局最优位置）。在每次迭代中，粒子会根据自己的个体最优位置和全局最优位置来调整自己的速度和位置，公式如下：

$$v_i^{(t+1)} = \omega v_i^{(t)} + c_1 r_1\left(p_i - x_i^{(t)}\right) + c_2 r_2\left(g - x_i^{(t)}\right) \tag{38}$$

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \tag{39}$$

其中，$x_i^{(t)}$为第i个粒子在第t代的位置；$v_i^{(t)}$为第i个粒子的速度；p_i为个体最优位置；g为全局最优位置；ω, c_1, c_2为惯性权重与学习因子；r_1, r_2为[0,1]的随机数。该算法的基础流程如下所示：

图 7：粒子群算法流程图

5.2.4 模型建立与求解

在模型建立阶段，首先将预处理后的数据按照 8：2 的比例划分训练集和测试集。其中，80%的样本用于模型训练，20%用于模型性能的验证。本文还引入了随机种子，以确保划分的随机性与结果的可重复性。

本文建立了**基于粒子群优化的截断奇异值分解压缩-还原模型**（PSO-SVD），其中，PSO 算法被用于搜索矩阵分解中最优的维度保留数量。该模型通过自适应地确定最优奇异值保留数量，实现数据压缩率与重构误差之间的动态平衡，其具体步骤如下：

1.初始化粒子群

每个粒子代表一个可能的解，其位置对应矩阵分解中保留的奇异值个数。粒子的初始位置在设定的搜索范围内随机生成，初始速度也同样设为随机值，用以控制

粒子在搜索空间中的移动方向与步长。这些粒子共同构成初始群体，作为算法迭代搜索的起点。

2.适应度评估

定义适应度函数为压缩比，在此基础上引入重构误差约束，建立以下优化模型：

$$\begin{cases} \max\ \mathrm{CR(k)} \\ \mathrm{st\ MSE(A, A_k)} < \varepsilon \end{cases} \tag{40}$$

其中，A为原始矩阵；A_k为保留前k个奇异值后的重构矩阵；ε是一个预先设定的阈值，本文设定为 0.005。

在每轮迭代中，对当前粒子代表的k值进行 SVD 压缩与还原，计算其对应的 MSE 与压缩比。若满足误差约束，则将对应压缩比作为适应度；否则适应度记为 0，粒子位置更新受限。最终输出满足误差条件下压缩比最大的解。

3.更新个体最优和全局最优

根据每个粒子的适应度结果，记录其历史最优位置，并更新群体的全局最优解。粒子的速度与位置根据经典 PSO 公式进行调整，受当前速度、自身最优位置及全局最优位置的共同影响，从而实现局部探索与全局搜索的结合。

4.重复迭代直至满足终止条件

重复步骤 2-3，直至满足以下任一终止条件：

（1）达到预设的最大迭代次数；

（2）若连续 10 次迭代，全局最优位置和适应度均无显著提升，则认为算法已收敛。

5.应用最优压缩参数进行数据压缩与重构

在确定最优的维度保留数量k后，使用该值对训练集和测试集进行矩阵分解与重构操作。首先，基于最优的维度保留数量，对训练集和测试集的数据进行降维处理，提取数据的主要特征。接着，根据降维后的数据和分解得到的特征矩阵，将数据重新映射回原始的高维空间，得到重构矩阵。最后通过对比重构矩阵和原始矩阵的差异，可以评估模型在数据压缩与还原过程中的信息保留能力，从而验证压缩-还原模型的有效性。

5. 2. 5 误差与结果分析

为展现 PSO-SVD 融合模型的优越性，本文通过与 PCA 对比，计算两种模型在训练集与测试集上的误差指标，以及相应的压缩比与存储空间节省率。结果如下表所示：

表 3：PSO-SVD 与经典 PCA 模型性能与存储优化指标对比表

模型	训练集 MSE	测试集 MSE	压缩比	存储空间节省率（%）	相对误差
PSO-SVD	0.004253	0.004983	60.60	98.35	0.000902
PCA	0.004014	0.004730	57.85	97.56	0.000895

从 MSE 来看，PCA 在训练集上表现略优；但在测试集上，PSO-SVD 的重构误差更小，说明 PSO-SVD 模型在处理未见数据时具有更好的泛化性能。两种模型的测试集 MSE 均低于题目所设阈值（0.005），即都满足精度要求，仅通过 MSE 无法绝对判断优劣。在压缩性能方面，PSO-SVD 的压缩比高于 PCA，对应存储空间节

省率也更高。因此在“压缩效率优先”的前提下，本文选择 PSO-SVD 作为数据压缩与还原的主建模方法。

PSO-SVD 压缩与还原过程中产生的误差主要来自截断误差和数值误差。前者源于仅保留前k个奇异值进行低秩近似，虽然提升了压缩效率，但可能舍弃了部分有效信息，影响重构精度；后者则源于浮点数运算的精度限制，尤其在处理大规模或病态矩阵时，易引发奇异值估计偏差。

为进一步直观展示模型压缩与重构效果，本文绘制了以下可视化图像：

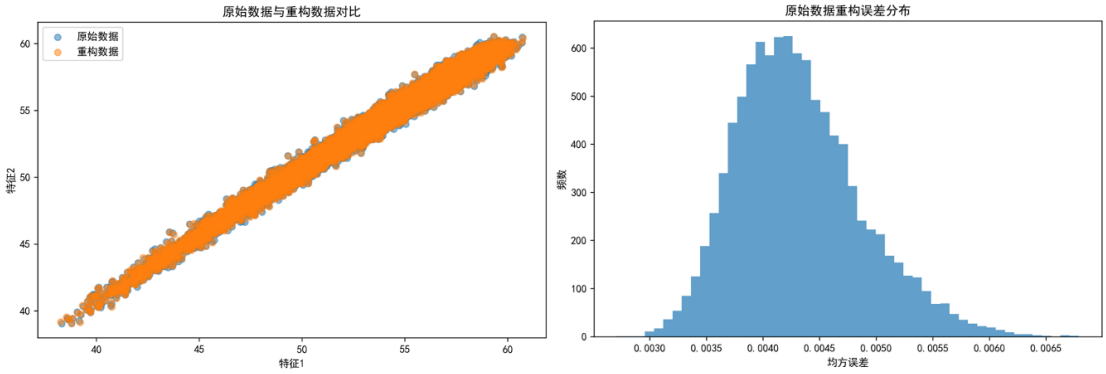

图 8：PSO-SVD 模型效果展示

该图原始数据与 PSO-SVD 重构数据在两个特征维度上的分布对比情况。可以看出，原始数据与重构数据间几乎完全重合，误差极小。同时，重构误差近似正态分布，峰值集中在 0.004 附近，波动范围较小，进一步说明了 PSO-SVD 模型在压缩与还原过程中具有良好的稳定性和重构精度。

5.3 问题三

对于问题三，我们按照数据预处理-交叉验证选择最优正则化参数α-基于最优参数训练 Lasso 回归模型的顺序进行，整体流程如下图所示：

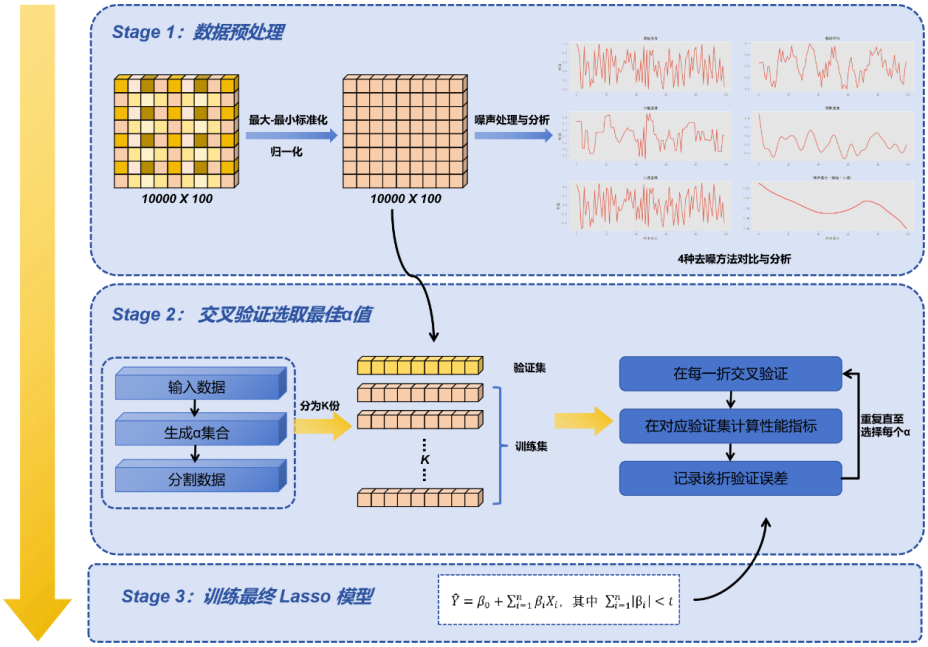

图 9：LassoCV 回归模型训练流程图

5.3.1 数据预处理

为提升建模精度与模型稳定性，我们对原始数据进行了预处理，包括归一化与去噪，具体如下：

1.归一化处理

由于原始数据各特征量纲不一，直接建模可能导致某些变量主导训练过程，进而产生偏差。为消除量纲影响，我们对数据 X 采用了最小-最大归一化，该方法将每列数据缩放至区间[0,1]，为后续小波处理和模型训练提供统一输入尺度。归一后的数据记为X_{norm}：

$$X_{norm} = \frac{X - min(X)}{max(X) - min(X)} \tag{41}$$

2.小波去噪处理

针对原始数据中混叠的噪声干扰，采用**小波去噪方法**对数据进行降噪。该方法基于小波变换的多分辨率分析特性，通过在不同尺度上分离信号的**低频主趋势成分**与**高频噪声成分**，实现对原始数据的精细化降噪，同时保留信号的关键特征。下图展示了小波去噪与传统滤波方法的性能对比：

表 4：多种去噪方法性能指标（MSE）对比表

去噪方法	MSE
无去噪方法	0.3337
小波去噪	0.0012
移动平均去噪	0.0666
中值滤波去噪	0.0760
高斯滤波去噪	0.0618

小波去噪方法下均方误差仅为 0.0012，在噪声抑制与特征保留之间取得了最佳平衡，避免传统滤波（如移动平均）对局部特征的模糊，适用于类似于矿山监测数据的非平稳信号处理。

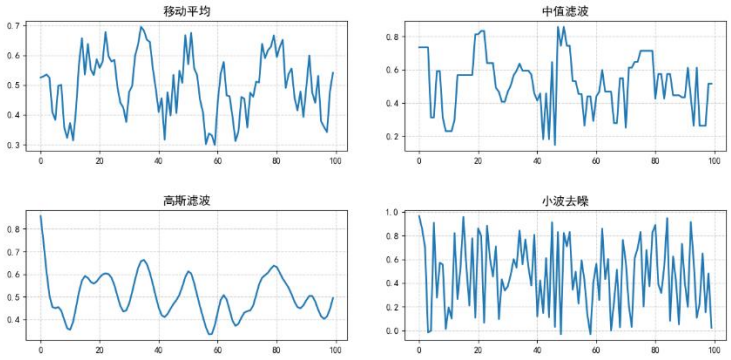

图 10：不同去噪方法波形对比示意图

因此，在归一化基础上，我们采用小波去噪进行去噪处理。该方法能够在不同尺度上分离信号的主趋势与局部扰动，具有良好的时频局部化特性。具体如下：

(1)小波分解：

对特征矩阵$\tilde{X}$的每一列$\tilde{X}_j$进行小波分解。使用 Daubechies 4（db4）小波基，进行 5 层小波分解（$L = 5$）。小波分解将信号分解为不同尺度的低频和高频分量：

$$c_L, d_L, d_{L-1}, \cdots, d_1 \tag{42}$$

其中，c_L为第L层的低频近似系数；d_L为细节系数。

分解过程可由滤波器组实现，数学表达为：

$$c_{k+1}[m] = \sum_n h\,[n - 2m] c_k[n] \tag{43}$$

$$d_{k+1}[m] = \sum_n g\,[n - 2m] c_k[n] \tag{44}$$

其中，$h[n]$为低通滤波器系数；$g[n]$为高通滤波器系数；m为离散时间索引。

(2)低频系数抑制：

为了去除噪声，我们将最低频的近似系数c_L全部置零，即$\hat{c}_L = 0$。这种处理方式能够去除包含长期趋势和潜在噪声的低频成分，保留多尺度的高频扰动，有助于揭示与矿山状态变化相关的局部特征。

(3)小波重构：

使用修改后的系数组进行小波逆变换，重构出降噪后的信号$\hat{\hat{X}}_j$。重构过程对应小波分解的逆过程，表示如下：

$$c_k[n] = \sum_m (h[n-2m]c_{k+1}[m] + g[n-2m]d_{k+1}[m]) \tag{45}$$

通过以上步骤，即可对数据集完成去噪操作，并得到降噪后的特征矩阵$\hat{X}$。以下是对小波去噪前后的效果对比：

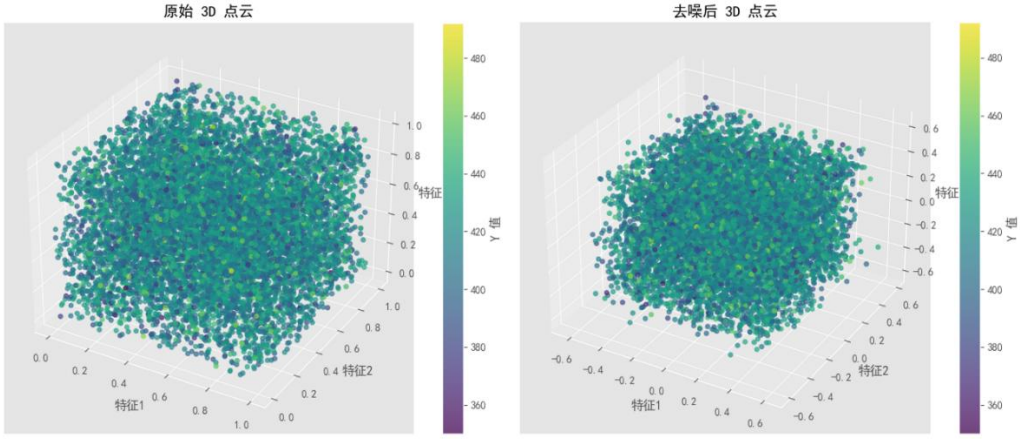

图 11：小波去噪前后 3D 点云对比图（相同量纲）

该图直观展示了去噪效果，通过去噪使得点云分布更加规整，减少了一些可能干扰后续分析的噪声点，同时保留关键扰动，有助于挖掘与矿山状态变化相关的局部特征。

另外，为探究矿山监测特征数据$X \epsilon R^{n \times p}$与目标变量$Y \epsilon R^n$之间的关系，我们首先对特征矩阵 X 的每一列与 Y 进行皮尔逊相关系数分析，提取与 Y 最为相关的前 10 个特征，并绘制了与 Y 皮尔逊相关系数最高的前 25 个特征列的散点图。结果分别如表 5、图 12 所示：

表 5：X 与 Y 的皮尔逊相关系数（前 10 个特征，由高到低排序）

列数	皮尔逊相关系数
65	0.172458
42	0.172011
16	0.171109
32	0.170450
83	0.166388
52	0.164125
31	0.159804
99	0.155339
50	0.154871
43	0.151699

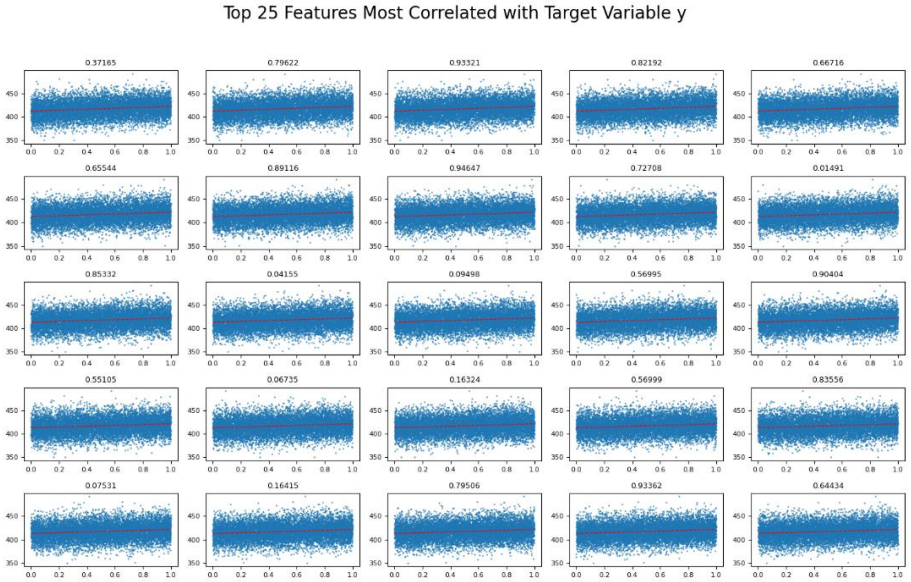

图 12：与目标变量相关性最高的前 25 个特征列散点图

尽管个别特征在数值上表现出较强的相关性，但其对应的散点图仍存在明显波动，点云分布较为松散，整体呈现出“弱相关”甚至“非线性”特征。这表明，即使在最高相关性的特征中，线性模型的解释力仍有限。因此，本文同时采用线性模型与非线性模型进行对比分析，分别构建 **Lasso 回归模型**（线性）与**随机森林回归模型**（非线性），以评估其在复杂特征关系下的拟合能力。

5. 3. 2 Lasso 回归模型

Lasso 回归模型（Least Absolute Shrinkage and Selection Operator）是一种在线性回归基础上加入L_1正则项的线性回归方法。其目标是找到一组回归系数$\beta = [\beta_0, \beta_1, \cdots, \beta_p]^T$，使得预测值$\hat{Y} = X\beta$尽可能接近实际值 Y，同时通过$L_1$正则化项进行特征选择和防止过拟合[5]。Lasso 回归的目标函数为：

$$\min_{\beta}\left\{\frac{1}{2n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\alpha\sum_{j=1}^{p}|\beta_j|\right\}=\min_{\beta}\left\{\frac{1}{2n}\|Y-X\beta\|_2^2+\alpha\|\beta\|_1\right\} \tag{46}$$

其中，$\|Y-X\beta\|_2^2$是残差平方和，$\|\beta\|_1$是回归系数的L_1范数，α是正则化参数，用于控制正则化项的权重。在模型设置中，我们在特征矩阵前添加常数列以拟合偏置项，正则化参数初始设定为$\alpha = 1\times10^{-4}$；最大迭代次数设为 10000，以确保模型充分收敛。

尽管 Lasso 回归的目标函数是一个凸函数，但由于L_1范数在零点处不可导，无法直接使用常规梯度下降法求解。本文采用**坐标下降法**来求解 Lasso 回归问题。

该方法每次迭代时，仅优化一个变量β_j，其余变量保持不变。在第t次迭代中，对第j个回归系数的更新过程如下：

原始子问题为：

$$\min_{\beta_j}\left\{\frac{1}{2n}\sum_{i=1}^{n}\left(y_i-\sum_{k\neq j}x_{ik}\,\beta_k^{(t-1)}-x_{ij}\beta_j\right)^2+\alpha|\beta_j|\right\} \tag{47}$$

定义当前残差：$r_i^{(t-1)} = y_i - \sum_{k\neq j} x_{ik}\,\beta_k^{(t-1)}$，子问题可简化为：

$$\min_{\beta_j}\left\{\frac{1}{2n}\sum_{i=1}^{n}\left(r_i^{(t-1)}-x_{ij}\beta_j\right)^2+\alpha|\beta_j|\right\} \tag{48}$$

对其求导并令导数为零，得到更新公式：

$$\beta_j^{(t)}=\text{sign}(z_j)\max\left\{|z_j|-\frac{\alpha n}{\sum_{i=1}^{n}x_{ij}^2},0\right\} \tag{49}$$

其中，$z_j=\frac{1}{\sum_{i=1}^{n}x_{ij}^2}\sum_{i=1}^{n}x_{ij}\,r_i^{(t-1)}$；$\text{sign}(z_j)$是$z_j$的符号函数。

通过不断迭代各个坐标，直至回归系数在相邻两次迭代间的变化低于预设阈值，最终可得到收敛的回归系数$\hat{\beta}$。

为了确定最优的正则化强度α，我们引入了交叉验证策略对多个候选α进行性能评估。以下是误差随α变化的趋势图，误差最小时对应的α值即为最终确定的最优正则化参数。

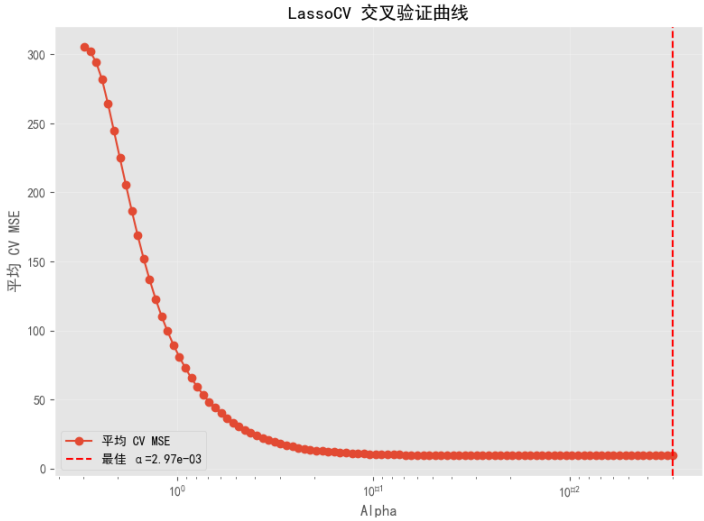

图 13：Lasso CV 交叉验证曲线

5.3.3 随机森林回归模型

随机森林回归模型是一种基于集成学习的方法，它通过构建多个决策树并集成其预测结果以提升模型的稳定性与泛化能力。在训练过程中，每棵树基于自助采样法（Bootstrap）从样本和特征空间中随机抽样生成，最终的预测结果为所有树的平均输出[4]：

$$\hat{y}(x)=\frac{1}{M}\sum_{m=1}^{M}h_m(x) \tag{50}$$

其中，M为决策树的数量；$h_m(x)$为第m棵决策树对样本x的预测值。

理论上，随机森林的泛化误差存在以下上界估计：

$$Error\le\frac{\rho(1-s^2)}{s^2} \tag{51}$$

其中，s为单棵树的强度（可理解为预测准确率）；ρ为任意两棵树预测结果的相关系数。通过提升单棵树的强度s，降低树与树之间的相关性ρ，可有效减少总体误差。

模型中关键超参数包括树的数量、树的最大深度、分裂节点时的最小样本数以及叶子节点的最小样本数。本文采用网格搜索对这些参数进行联合优化，以提高模型的性能。

5.3.4 模型结果与误差分析

为比较以上两种模型在实际数据上的表现，本文引入三种常用评价指标：均方误差（MSE）、平均绝对误差（MAE）和判定系数（R^2）对拟合能力与泛化性能进行定量评估。结果如下所示：

表 6：随机森林与 Lasso 回归模型性能指标对比表

模型	R^2	MSE	MAE
随机森林	0.8655	41.2316	4.9022
Lasso 回归	0.9697	9.2964	2.4556

由表 6 可知，Lasso 回归模型在评价指标上优于随机森林回归模型，对目标变量的解释能力更强，拟合误差更小、预测更为稳定。为直观展现 Lasso 回归模型的表现，本文绘制了其在测试集上的三类可视化图，包括实际值与预测值散点图、残差与预测值图和残差直方图，如下所示：

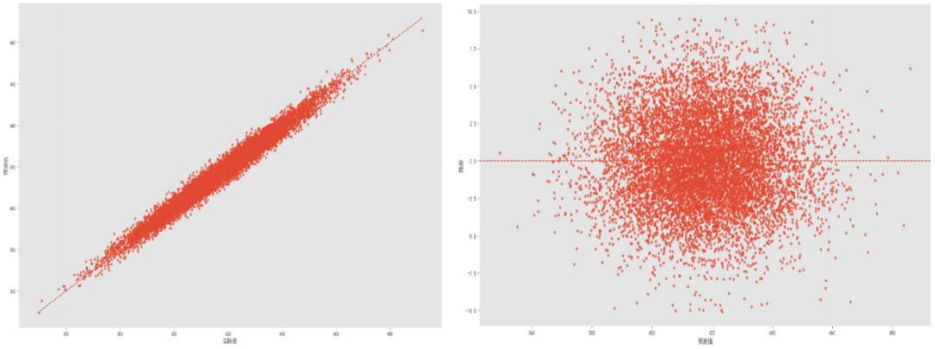

图 14：Lasso 回归模型实际值 vs 预测值与残差 vs 预测值散点图

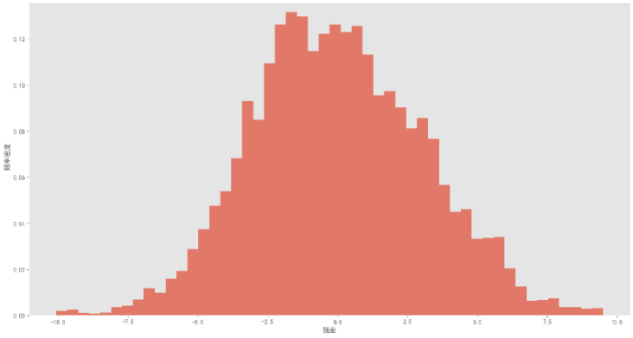

图 15：Lasso 回归模型残差分布直方图

从可视化结果来看，**Lasso 回归模型**表现良好，点云分布紧密，基本沿$y = x$对角线对称排列，在整体趋势拟合上准确性较高。残差与预测值图中，误差围绕零轴随机波动，未出现明显结构性趋势，反映出模型误差主要由随机噪声组成，未发生系统性偏离。残差直方图呈现近似正态分布，中心对称性良好，尾部收敛性强，误差整体波动较小，模型具有较高的稳定性和泛化能力。

总之，Lasso 回归模型凭借其特征筛选与正则化机制，取得了更好的拟合精度与误差稳定性，因此更适合作为本问题的建模方案。

5.4 问题四

5.4.1 数据预处理

为消除不同特征之间的量纲影响、提升模型训练效率，本文对输入变量X进行最小-最大归一化处理，将所有特征缩放至区间[0,1]；随后在标准化后的特征矩阵前添加一列常数项以引入截距，构成设计矩阵X_{design}。最后，将数据按 8：2 比例划分为训练集与测试集，用于模型训练与性能评估。

5.4.2 模型建立与求解

本题的目标是在保证预测精度的前提下，建立一个对参数敏感度适中、拟合优度高且具备良好可解释性的预测模型，以刻画特征变量X与响应变量Y之间的映射关系。鉴于数据中可能存在非线性趋势与噪声扰动，本文选取了**线性核支持向量回归**模型（Linear SVR）进行建模。该模型在处理中小样本、非线性边界与高维特征场景中具有良好的泛化能力。

该模型的核心思想是构造一个回归函数$f(x) = w^{\mathrm{T}}x + b$，其中，$w$为权重向量；$b$为偏置项；$x$为输入特征向量。其优化目标为在控制预测误差的同时最小化模型复杂度，对应数学形式如下：

$$min_{w,b,\{\xi_i,\xi_i^*\}} = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{51}$$

$$st\begin{cases} y_i - f(x) \le \varepsilon + \xi_i, \\ f(x) - y_i \le \varepsilon + \xi_i^*, \\ \xi_i, \xi_i^* \ge 0, \end{cases} \tag{52}$$

其中，C为惩罚系数，平衡模型复杂度与训练误差；ε为不敏感间隔宽度；ξ_i, ξ_i^*为对超出容差范围的样本误差进行补偿的松弛变量。

该模型性能对参数C和ε极为敏感。为确保模型具备最优性能，本文采用**网格搜索**（Grid Search）方法设计参数自适应调整策略，设定以下参数空间：

$$C \in \{10^{-3}, 10^{-2}, \dots, 10^{3}\}, \varepsilon \in \{10^{-3}, 10^{-2}, \dots, 10^{1}\} \tag{53}$$

对每组参数组合，在训练集上训练 SVR 模型，并计算其R^2得分，以选择性能最优的参数。

为求解该凸优化问题，引入拉格朗日乘子$\alpha_i(\alpha_i^* \geqslant 0)$和$\mu_i(\mu_i^* \geqslant 0)$，构造拉格朗日函数：

$$L(w,b,\xi,\xi^*,\alpha,\alpha^*,\mu,\mu^*) = \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + w^T x_i + b) - \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - w^T x_i - b) - \sum_{i=1}^{n}\mu_i\xi_i - \sum_{i=1}^{n}\mu_i^*\xi_i^* \tag{54}$$

对原始变量求偏导并令其为零，得到 KKT 条件：

对ω求偏导可得：

$$\frac{\partial L}{\partial \omega} = \omega - \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)x_i = 0 \Rightarrow \omega = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)x_i \tag{55}$$

对 b 求偏导可得：

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \tag{56}$$

对ξ_i求偏导可得：

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \Rightarrow 0 \leqslant \alpha_i \leqslant C \tag{57}$$

对ξ_i^*求偏导可得：

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \mu_i^* = 0 \tag{58}$$

将上述结果代入拉格朗日函数，即可得到对偶问题：

$$\max_{\alpha,\alpha^*} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)x_i^T x_j - \epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \tag{59}$$

$$st \begin{cases} \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \\ 0 \leqslant \alpha_i, \alpha_i^* \leqslant C, i = 1, \cdots, n \end{cases} \tag{60}$$

通过二次规划求解上述对偶问题，就可得到最优解α_i^*和α_i。最终权重向量由$\omega = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)x_i$给出；对于截距$b$，可以选择一个支持向量（即$0 < \alpha_i < C$或$0 < \alpha_i^* < C$的样本），根据$y_i - w^T x_i - b = \pm\varepsilon$计算得到。

5.4.3 结果分析

模型参数设置如下：正则化参数：α=1×10−4，最大迭代次数 10000。建模前在特征矩阵前添加一列常数项，以支持模型自动拟合偏置项，训练后得到以下数据：

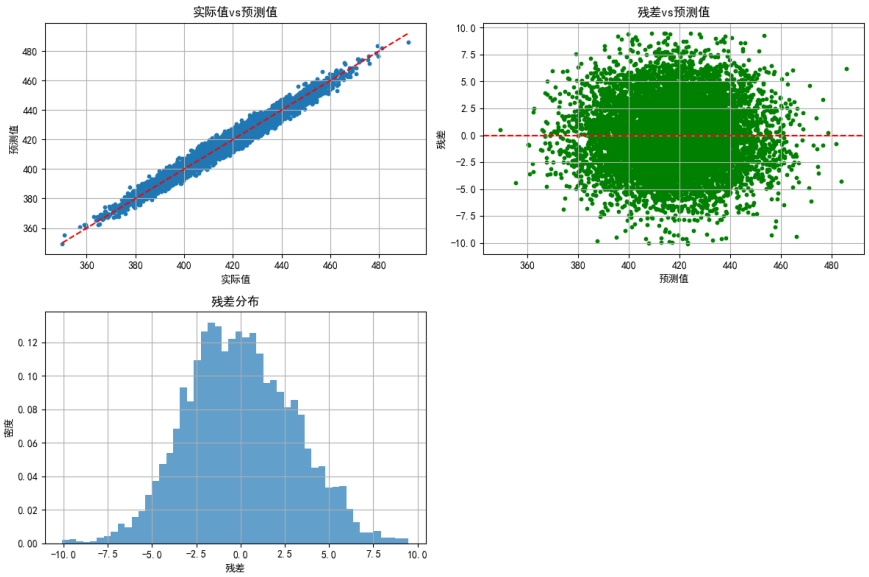

图 16：Linear SVR 模型预测效果图

共 100 组候选参数组合在训练集上并行搜索，最终得到最佳训练集参数设置和拟合结果如下：

表 7：Linear SVR 模型关键参数与性能指标表

指标	数值
C	10.000000
ε	1.291550
$log_{10}C$	1.000000
$log_{10}\varepsilon$	0.111111
R^2	0.941448
训练时间（s）	17.7938

前五个训练集R^2最高的组合依次为：

表 8：Linear SVR 模型在不同参数下性能指标对比表

排名	C	ε	$log_{10}C$	$log_{10}\varepsilon$	R^2	训练时间（s）
1	10.000000	1.29155	1.000000	0.111111	0.941448	17.7938
2	2.154435	1.29155	0.333333	0.111111	0.941447	13.7307
3	46.415888	1.29155	1.666667	0.111111	0.941443	50.6677
4	215.443469	1.29155	2.333333	0.111111	0.941443	181.9100
5	1000.000000	1.29155	3.000000	0.111111	0.941442	551.9609

由上表可知，前 5 组在R^2上几乎没有显著差异，但训练时长差异显著，说明高性能参数组合并不唯一，但在实际应用中应权衡精度与效率，选取兼顾效果与成本的参数配置。

在最优参数配置下，模型在训练集与测试集上的预测精度如下表所示：

表 9：Linear SVR 模型在附件数据集上的预测精度

数据集	R^2	MSE	MAE	RMSE
训练集	0.9414	14.4490	3.0389	3.8012
测试集	0.9384	14.9342	3.0681	3.8645

可以看出，模型在测试集上仍保持了良好的预测能力，平均预测误差约为 3.1 个单位，误差收敛性良好，表明模型具有较强的泛化能力。进一步，通过 5 折交叉验证评估模型稳定性，结果如下表所示：

表 10：Linear SVR 模型性能指标表

平均MSE	标准差
14.8552	0.4153

这表明模型在不同数据子集上的误差波动较小，整体性能稳定。

在**线性核支持向量回归**模型中，只有位于ε-不敏感管道边界上或超出该间隔的样本才会被选作“支持向量”，并参与回归函数的构建。训练集共 8000 条样本，其中 5961 条成为支持向量，占比约 74.5%，支持向量比例较高，这表明：大部分观测值的预测误差已接近或超出了模型设定的容差范围，导致大量样本对模型参数的求解产生实质性影响。虽然高比例的支持向量在一定程度上降低了模型的稀疏性，增大了存储和预测时的计算开销，但结合 5 折交叉验证结果（$R^2 \approx 0.94$），可以看出模型依然保持了良好的泛化能力。该结果说明，在噪声较强的环境中，适度放宽ε可减少支持向量数量，降低模型复杂度，同时略微牺牲精度，以实现更快计算与更强泛化性能。

为进一步验证模型无系统偏差与预测误差分布合理性，本文统计并分析了残差的关键分布特征：

表 11：残差统计量与分位数汇总表

统计量	数值	残差分位数	数值
最大残差	16.222916.2229	5% 分位	−6.1642-6.1642
最小残差	−14.0254-14.0254	50% 分位	−0.0515-0.0515
残差标准差	3.86293.8629	95% 分位	+6.5877+6.5877

残差分布基本对称、中心靠近 0，95%的残差落在$[-6.6, +6.6]$区间，说明模型没有显著偏差，误差波动在可接受范围内，适用于该矿山监测数据的连续预测任务。

5.5 问题五

针对问题五，本文比较主成分分析（PCA）和联合训练的自动编码器（AE）这两种降维方法，并在其基础上进一步评估多种回归模型在不同降维表示下的性能。本文对以下六种回归模型进行了对比分析：线性回归、岭回归、Lasso 回归、支持向量回归、多层感知机回归和随机森林回归。

岭回归是在线性回归的基础上加入了L_2正则化项，以防止过拟合。其损失函数 L 的计算公式为：

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \|\omega\|^2 \tag{61}$$

其中，α为正则化系数。

5.5.1 主成分分析模型建立与求解

主成分分析（PCA）是一种常用的线性降维方法，旨在保留数据中最具代表性的特征信息，同时减少维度，提高处理效率。其核心思想是在保证信息尽量不丢失的前提下，找到一个新的低维空间，使得样本在该空间中的投影具有最大方差[6]。其主要步骤如下所示：

1.确定降维目标

设标准化后的数据矩阵$X_{\text{std}} \in R^{n\times d}$，目标是找到一个投影矩阵$W \in R^{d\times k}$，使得投影后的数据$Z = X_{\text{std}}W \in R^{n\times k}$的方差最大，同时满足$W^{\top}W = I_k$。其中，$n$为样本数量；$d$为原始特征维度；$I_k$为$k \times k$的单位矩阵。

投影后数据Z的协方差矩阵为：

$$\text{Cov}(Z) = \frac{1}{n}Z^{\top}Z = \frac{1}{n}(X_{\text{std}}W)^{T}(X_{\text{std}}W) = \frac{1}{n}W^{\top}X_{\text{std}}^{\top}X_{\text{std}}W \tag{62}$$

降维目标为最大化投影后数据的总方差，即最大化$\text{Var}(Z)$，也就是最大化$\text{tr}(\text{Cov}(Z))$（tr表示矩阵的迹）。令$S = \frac{1}{n}X_{\text{std}}^{\top}X_{\text{std}}$为$X_{\text{std}}$的协方差矩阵，则目标函数可表示为：

$$\max_{W} \text{tr}\,(W^{\top}SW) \quad \text{s.t } W^{\top}W = I_k \tag{63}$$

2.拉格朗日乘子法求解

引入拉格朗日乘子矩阵$\Lambda \in R^{k\times k}$，构建拉格朗日函数：

$$L(W,\Lambda) = \text{tr}(W^{\top}SW) - \text{tr}\big(\Lambda(W^{\top}W - I_k)\big) \tag{64}$$

对W求偏导数并令其为零：

$$\frac{\partial L}{\partial W} = 2SW - 2W\Lambda = 0 \Rightarrow SW = W\Lambda \tag{65}$$

即列向量为协方差矩阵S的特征向量，最大化目标函数时选取对应前k个最大特征值所对应的特征向量。

3.数据重构过程

投影后的数据为$Z = X_{\text{std}}W$；通过投影矩阵的逆变换，可重构标准化后的数据：$\hat{X}_{\text{std}} = ZW^{\top} = X_{\text{std}}WW^{\top}$。最后，结合标准化参数进行反标准化，即可还原出重构后的原始数据矩阵$\hat{X}$。

5.5.2 联合自编码器

联合自编码器（AE）是一种集成重构与预测任务的神经网络结构，在实现特征降维的同时兼顾回归性能，其基本结构和步骤如下所示：

1.自编码器结构

(1)编码器：将输入$x \in R^d$映射到低维表示$h \in R^k$，其计算公式为：

$$h = \sigma(W_{\text{enc}}x + b_{\text{enc}}) \tag{66}$$

其中，$W_{\text{enc}} \in R^{k\times d}$是编码器的权重矩阵，$b_{\text{enc}} \in \mathbb{R}^k$是编码器的偏置向量，$\sigma$是激活函数。

(2)解码器：将低维表示h重构为输入 x 的近似$\hat{x} \in R^d$，其计算公式为：

$$\hat{x} = W_{\text{dec}}h + b_{\text{dec}} \tag{67}$$

其中，$W_{\text{dec}} \in R^{d\times k}$是解码器的权重矩阵，$b_{\text{dec}} \in R^d$是解码器的偏置向量。

(3)回归器：将低维表示h映射到预测值$\hat{y} \in R$，其计算公式为：

$$\hat{y} = w_{\text{reg}}^{\top}h + b_{\text{reg}} \tag{68}$$

其中，$w_{\text{reg}} \in R^k$为回归头的权重向量；$b_{\text{reg}} \in R$为回归头的偏置向量。

2.联合损失函数

联合自编码器的损失函数由重构损失和回归损失两部分构成。重构损失衡量输入x与重构输出$\hat{x}$之间的差异；回归损失衡量目标值y与预测值$\hat{y}$之间的差异。综合联合损失函数定义为：

$$\mathcal{L} = \lambda_1 \frac{1}{n}\sum_{i=1}^{n}\|x_i - \hat{x}_i\|^2 + \lambda_2 \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{69}$$

本文取权重参数$\lambda_1 = \lambda_2 = 0.5$，以平衡重构与预测任务。

3.训练过程

通过最小化联合损失函数 $\mathcal{L}$ 来训练编码器、解码器与回归器的参数，通常使用梯度下降法或其变体，具体更新公式为：

$$\theta \leftarrow \theta - \alpha\frac{\partial \mathcal{L}}{\partial \theta} \tag{70}$$

其中，θ为模型参数；α为学习率。

4.降维与回归评估过程

(1)PCA 阶段

对于每个$k \in \{5,10,\cdots,100\}$，执行 PCA 降维与逆变换重构，计算重构误差 MSE。对重构后的特征，采用五折交叉验证训练六种回归模型，记录R^2、MSE、MAE等指标。

(2)AE 阶段

构建联合自编码器，训练 20 轮；提取编码器输出（低维表示）及重构输出，分别计算其重构误差；同样使用上述六种回归器，在重构特征上进行 5 折交叉验证，评估预测性能。

5.5.3 模型性能分析

为评估两种主流降维方法，主成分分析（PCA）与联合自编码器（AE）在不同回归模型下的表现，本文从重构误差、回归性能和复杂度三个方面进行了分析：

1.重构误差对比

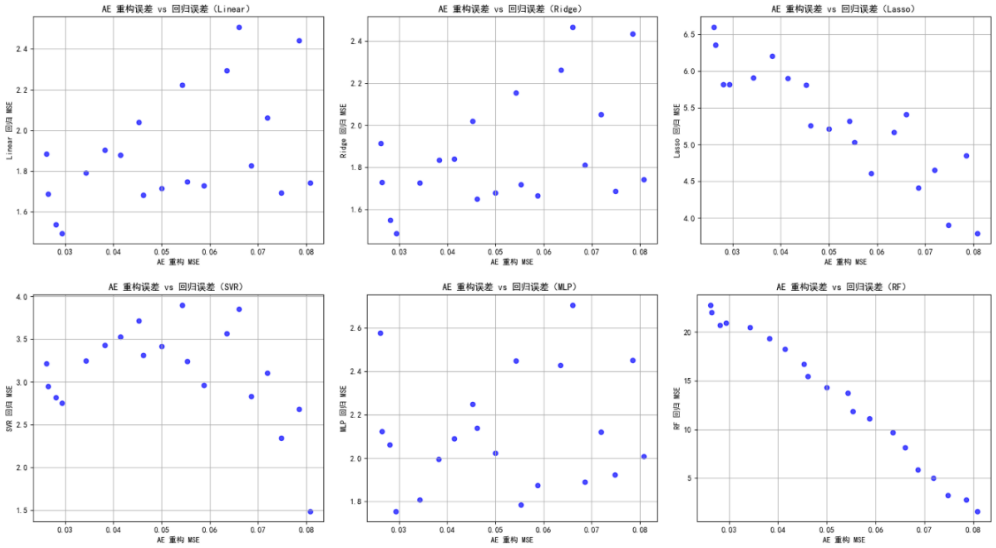

图 17：不同模型下 AE 重构误差与回归误差关系散点图

由上图可知，对于线性回归、岭回归等线性模型，AE 重构 MSE 与预测 MSE 呈**正相关**，即重构误差越小，预测误差越低；而对于随机森林等非线性模型，两者相关性较弱，说明线性模型更依赖降维后的特征线性可分性，而非线性模型对特征空间的不规则分布容忍度更高。

2.回归性能对比

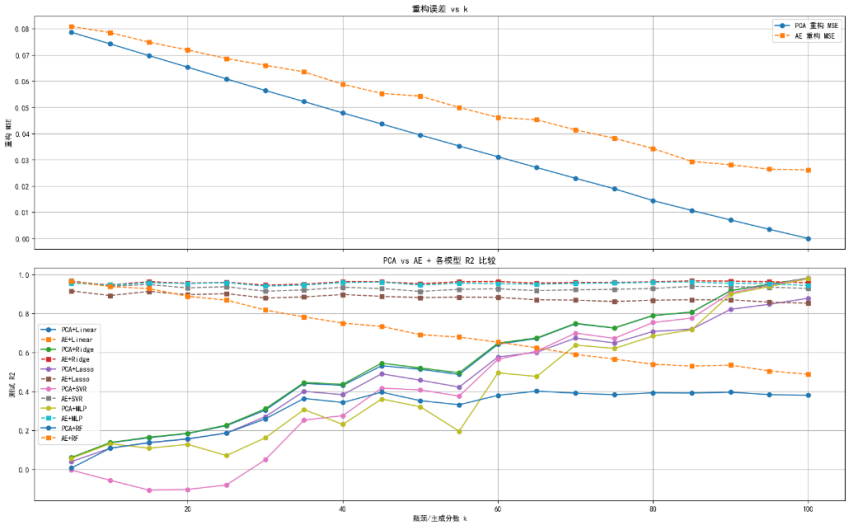

图 18：PCA 与 AE 在六种模型下性能指标对比折线图

表 12：PCA 与 AE 在六种模型在$k = 100$下性能指标对比表

方法	k	R^2	MSE	MAE
PCA + LinearRegression	100	0.9802	0.8805	0.7530
PCA + Ridge	100	0.9802	0.8806	0.7530
PCA + SVR	100	0.9791	0.9301	0.7405
PCA + MLP	100	0.9768	1.0306	0.8047
PCA + RandomForest	100	0.3798	27.5370	4.1800
AE + LinearRegression	100	0.9714	1.2710	0.8995
AE + Ridge	100	0.9706	1.3049	0.9101
AE + SVR	100	0.9513	2.1608	1.1465
AE + MLP	100	0.9605	1.7508	1.0557
AE + RandomForest	100	0.4852	22.8606	3.7976

由图 18、表 12 可知，最优组合为 PCA+LinearRegression，在$k = 100$时达到最优性能（$R^2 = 0.9802, MSE = 0.8805, MAE = 0.7530$），兼顾了模型简单性与预测精度。同时，AE 在搭配非线性模型时相比 PCA 表现更好，但总体回归性能仍不及核/线性模型。

3.复杂度分析

(1)PCA 复杂度：PCA 的理论复杂度为$O(nd^2)$，主要是由于计算协方差矩阵$S = \frac{1}{n}X_{\text{std}}^{\top}X_{\text{std}}$和求解特征值分解的过程。

(2)AE 训练复杂度：AE 训练的理论复杂度为$O(endh)$，其中e为训练轮数，n为样本数量，d为输入维度，h为隐藏层维度，即降维后的维度k。

(3)回归训练复杂度：回归训练的理论复杂度为$O(nk^2)$，其中，n为样本数量；k为降维后的维度。

六、 模型评价与推广

6.1 模型优点

1.模型融合：针对数据压缩与还原问题，提出融合粒子群优化的奇异值分解（PSO-SVD）模型，通过 PSO 自动确定奇异值保留数量，实现压缩比与重构精度之间的动态平衡。相比传统 PCA 方法，该模型在压缩效率与重构精度方面表现更优，具有更强的泛化能力。

2.引入正则化：在特征选择与降维过程中，结合 Lasso 回归的L_1正则化特性，有效剔除冗余特征，降低模型复杂度，同时提升变量选择的可解释性，为建模提供更稳健的特征基础。

3.多模型对比分析：针对不同任务场景，采用多模型对比策略，包括线性与非线性回归模型（如 Lasso 与随机森林）、线性与非线性降维方法（如 PCA 与联合自编码器 AE）。通过评估指标对比分析，为各类任务选择最合适的模型，提升整体建模效果。

4.自动化参数优化：参数调优方面引入网格搜索、粒子群优化等自动化策略，显著降低人工调参成本；且所用算法复杂度可控，适用于中小规模矿山数据的实时建模需求。

6.2 模型缺点

1.高维数据效率瓶颈：PCA 在高维特征场景下计算复杂度为$O(nd^2)$，当数据规模扩大至百万级时，特征分解耗时显著增加；AE 训练需迭代优化参数，训练时间长、对硬件资源要求高，限制了其在超大规模数据场景中的应用效率。

2.参数敏感性：支持向量回归（SVR）、联合自编码器（AE）等模型对关键超参数（如 C 值、隐藏层维度）较为敏感，尽管通过网格搜索可优化，但当数据分布发生剧烈变化时，仍需重新调参，导致模型泛化能力受限。

6.3 模型推广

1.强化非线性建模与效率优化：可引入图神经网络（GNN）或 Transformer 等深度模型，捕捉数据中的复杂时空依赖结构，并嵌入注意力机制以增强关键特征提取能力。同时，采用增量式 PCA（IPCA）、稀疏自编码器（SAE）等方法实现大规模数据的在线降维处理，结合 GPU 并行计算技术显著提升建模效率。

2.跨模态融合与迁移学习：面对多源异构数据（如钻孔监测、无人机影像等），可设计跨模态特征对齐与融合机制，如自编码器配合特征对齐网络，提升整体建模的多维表达能力。此外，引入迁移学习框架，可利用已建模矿区的数据初始化参数，降低新场景的数据标注与模型训练成本，提升模型适应性与可推广性。

七、 参考文献

[1]杨光,李峰,于国强.基于多元线性回归的 EA4T 钢磨削表面残余应力预测[J/OL].机械设计与制造,1-5[2025-05-15].

[2]华有霖,邵亚斌,朱学勤.基于粒球计算的多粒度支持向量回归算法[J/OL].山东大学学报(理学版),1-12[2025-05-15].

[3]高勇,李恒武,王辰阳.基于随机矩阵分解的大数据无向压缩算法设计[J].计算机仿真,2023,40(08):462-466.

[4]董凌,杨鹏年,徐杰,等.基于随机森林法的干旱区冬灌面积反演研究[J/OL].节水灌溉,1-12[2025-05-15].

[5]贺宇轩,王锟,曾进辉,等.基于 KNN-LASSO-PPC 法的改进 BitCN-LSTM 短期光伏功率预测[J/OL].电子测量技术,1-15[2025-05-15].

[6]席磊,李宗泽,刘治洪,等.基于图像编码与多头自注意力卷积神经网络的电网虚假数据注入攻击检测[J/OL].中国电机工程学报,1-13[2025-05-15].

[7]李易霖,王成群.基于自编码器的神经网络聚合传递时间序列压缩算法[J/OL].电子科技,1-9[2025-05-15].

八、 附录

运行环境

操作系统：Windows 11 家庭中文版 24H2
CPU：Intel(R) Core(TM) i7-14650HX 2.20 GHz
Python IDE：PyCharm 2024.2.4 (Community Edition)
Python：Python 3.8

依赖库

matplotlib==3.7.5
numba==0.58.1
numpy==1.24.3
openpyxl==3.1.5
packaging==25.0
pandas==2.0.3
pillow==10.4.0
protobuf==4.25.7
scikit-image==0.21.0
scikit-learn==1.3.2
scipy==1.10.1
seaborn==0.13.2
sympy==1.13.3
tensorboard==2.13.0
tensorboard-data-server==0.7.2
tensorflow==2.13.0
tensorflow-estimator==2.13.0
tensorflow-intel==2.13.0
tensorflow-io-gcs-filesystem==0.31.0
torch==2.4.1
tqdm==4.67.1

问题一

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV, LassoCV, RANSACRegressor
from sklearn.model_selection import cross_validate, train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from statsmodels.api import OLS, add_constant
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from statsmodels.stats.diagnostic import het_breuschpagan

# 1. 读取数据
A = pd.read_excel('A.xlsx', header=None)
B = pd.read_excel('B.xlsx', header=None).values.flatten()
n, p = A.shape
```

```
# 2. 数据分布可视化
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

# 2.1 每列特征直方图（示例绘制前 5 列）
for i in range(min(1, p)):
    plt.figure(figsize=(6,4))
    sns.histplot(A[i].values if i==0 else A.iloc[:, i], bins=30,
kde=True)
    plt.title(f'特征 x{i} 分布')
    plt.xlabel(f'x{i} 值')
    plt.ylabel('频数')
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()

# 2.2 特征箱线图（示例前 5 列）
plt.figure(figsize=(8,4))
sns.boxplot(data=A.iloc[:, :50])
plt.title('前 50 列特征箱线图')
plt.xlabel('特征')
plt.ylabel('值')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# 2.3 特征相关矩阵热力图（降采样显示前 10 特征）
corr = A.iloc[:, :100].corr()
plt.figure(figsize=(6,5))
sns.heatmap(corr, annot=False, fmt='.2f', cmap='coolwarm')
plt.title('前 100 特征皮尔逊相关系数矩阵')
plt.tight_layout()
plt.show()

# 3. 标准化
scaler = StandardScaler()
A_std = scaler.fit_transform(A)

# 4. OLS 回归及诊断
X_const = add_constant(A_std)
model = OLS(B, X_const).fit()
print(model.summary())

# 5. 多重共线性 (VIF)
vif_data = pd.DataFrame({
    'feature': ['const'] + [f'x{i}' for i in range(p)],
    'VIF': [variance_inflation_factor(X_const, i) for i in
range(X_const.shape[1])]
})
print('\nVIF:\n', vif_data)

# 6. 异方差检验
bp_test = het_breuschpagan(model.resid, model.model.exog)
print('\nBreusch-Pagan test:', dict(zip(
    ['LM statistic','LM p-value','F-statistic','F p-value'], bp_test
)))
```

```
# 7. 交叉验证 + 正则化
alphas = np.logspace(-4, 2, 50)
ridge = RidgeCV(alphas=alphas, cv=5).fit(A_std, B)
lasso = LassoCV(alphas=alphas, cv=5, max_iter=5000).fit(A_std, B)
print(f"\nRidge alpha: {ridge.alpha_}")
print(f"Lasso alpha: {lasso.alpha_}")

# 8. 划分并用 LassoCV 模型评估
X_train, X_test, y_train, y_test = train_test_split(A_std, B,
test_size=0.2, random_state=42)
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)
res = y_test - y_pred

# 9. 可视化拟合结果与诊断
# 9.1 实际 vs 预测\plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred, s=20, alpha=0.7)
mn, mx = y_test.min(), y_test.max()
plt.plot([mn, mx], [mn, mx], '--', color='gray')
plt.xlabel('实际')
plt.ylabel('预测')
plt.title('实际 vs 预测 (LassoCV)')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# 9.2 残差 vs 预测\plt.figure(figsize=(6, 4))
plt.scatter(y_pred, res, s=20, alpha=0.7)
plt.axhline(0, linestyle='--', color='gray')
plt.xlabel('预测值')
plt.ylabel('残差')
plt.title('残差 vs 预测值')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# 9.3 残差分布\plt.figure(figsize=(6,4))
plt.hist(res, bins=50, density=True, alpha=0.7)
plt.xlabel('残差')
plt.ylabel('密度')
plt.title('残差分布')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# 10. 离群点检测 (RANSAC)
ransac = RANSACRegressor().fit(A_std, B)
mask = ransac.inlier_mask_
outliers = ~mask
plt.figure(figsize=(6,4))
plt.scatter(ransac.predict(A_std)[mask], B[mask], s=20, alpha=0.7,
label='Inliers')
plt.scatter(ransac.predict(A_std)[outliers], B[outliers], s=20,
alpha=0.7, color='red', label='Outliers')
plt.plot([mn,mx],[mn,mx],'--',color='gray')
plt.xlabel('预测')
plt.ylabel('实际')
```

```
plt.title('RANSAC 检测离群点')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

问题二

```
import numpy as np
import pandas as pd
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import time
import matplotlib.font_manager as fm

# 查找系统中可用的中文字体
def find_chinese_fonts():
    chinese_fonts = []
    for font in fm.findSystemFonts():
        try:
            font_name = fm.FontProperties(fname=font).get_name()
            # 检查字体是否支持中文
            if 'Heiti' in font_name or 'Sim' in font_name or 'Micro Hei'
in font_name:
                chinese_fonts.append(font_name)
        except:
            continue
    return chinese_fonts

# 获取可用的中文字体列表
chinese_fonts = find_chinese_fonts()
print("可用的中文字体:", chinese_fonts)

# 如果找到中文字体，则使用第一个可用的中文字体
if chinese_fonts:
    plt.rcParams["font.family"] = chinese_fonts[0]
else:
    print("警告：未找到中文字体，图表中的中文可能无法正确显示")
    # 使用默认字体，但尝试解决负号显示问题
    plt.rcParams['axes.unicode_minus'] = False

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
"""加载 Excel 数据并返回 numpy 数组"""


def load_data(file_path):
    """加载 Excel 数据并返回 numpy 数组"""
    try:
        df = pd.read_excel(file_path, header=None)
        return df.values
    except Exception as e:
        print(f"加载数据时出错：{e}")
        return None
```

```
"""数据预处理：缺失值填充、异常值处理和标准化/归一化"""


def preprocess_data(data, iqr_threshold=1.5, normalize=True):
    """数据预处理：缺失值填充、异常值处理和标准化/归一化"""
    # 保存原始数据的副本
    original_data = data.copy()

    # 缺失值填充（使用前向填充）
    mask = np.isnan(data)
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            if mask[i, j]:
                if j > 0:
                    data[i, j] = data[i, j - 1]  # 前向填充
                else:
                    data[i, j] = 0  # 如果是第一列，填充 0

    # 四分位距法异常值处理
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - iqr_threshold * IQR
    upper_bound = Q3 + iqr_threshold * IQR

    # 裁剪异常值
    data = np.clip(data, lower_bound, upper_bound)

    return data, original_data


"""执行 SVD 矩阵分解并返回重构矩阵"""


def matrix_decomposition(matrix, n_components):
    """执行 SVD 矩阵分解并返回重构矩阵"""
    svd = TruncatedSVD(n_components=n_components)
    transformed = svd.fit_transform(matrix)
    reconstructed = svd.inverse_transform(transformed)
    return reconstructed, svd


"""计算均方误差"""


def calculate_mse(original, reconstructed):
    """计算均方误差"""
    # 确保两个数组形状相同
    if original.shape != reconstructed.shape:
        min_shape = np.minimum(original.shape, reconstructed.shape)
        original = original[:min_shape[0], :min_shape[1]]
        reconstructed = reconstructed[:min_shape[0], :min_shape[1]]
    return np.mean((original - reconstructed) ** 2)


"""计算平均绝对误差"""
```

```
def calculate_mae(original, reconstructed):
    """计算平均绝对误差"""
    # 确保两个数组形状相同
    if original.shape != reconstructed.shape:
        min_shape = np.minimum(original.shape, reconstructed.shape)
        original = original[:min_shape[0], :min_shape[1]]
        reconstructed = reconstructed[:min_shape[0], :min_shape[1]]
    return np.mean(np.abs(original - reconstructed))


"""计算均方根误差"""


def calculate_rmse(original, reconstructed):
    """计算均方根误差"""
    # 确保两个数组形状相同
    if original.shape != reconstructed.shape:
        min_shape = np.minimum(original.shape, reconstructed.shape)
        original = original[:min_shape[0], :min_shape[1]]
        reconstructed = reconstructed[:min_shape[0], :min_shape[1]]
    return np.sqrt(np.mean((original - reconstructed) ** 2))


"""计算相对误差"""


def calculate_relative_error(original, reconstructed):
    """计算相对误差"""
    # 确保两个数组形状相同
    if original.shape != reconstructed.shape:
        min_shape = np.minimum(original.shape, reconstructed.shape)
        original = original[:min_shape[0], :min_shape[1]]
        reconstructed = reconstructed[:min_shape[0], :min_shape[1]]
    return np.mean(np.abs(original - reconstructed) / (np.abs(original)
+ 1e-10))


"""计算解释方差比例"""


def calculate_explained_variance(original, reconstructed):
    """计算解释方差比例"""
    # 确保两个数组形状相同
    if original.shape != reconstructed.shape:
        min_shape = np.minimum(original.shape, reconstructed.shape)
        original = original[:min_shape[0], :min_shape[1]]
        reconstructed = reconstructed[:min_shape[0], :min_shape[1]]
    return 1 - (np.var(original - reconstructed) / np.var(original))


"""计算压缩比"""


def calculate_compression_ratio(original, decomposed_elements):
    """计算压缩比"""
    # 首先计算原始数据大小
    original_size = np.prod(original.shape)
```

```
    if isinstance(decomposed_elements, tuple):  # 张量分解
        core, factors = decomposed_elements
        compressed_size = core.size
        for factor in factors:
            compressed_size += factor.size
    else:  # 矩阵分解或 PCA
        model = decomposed_elements
        if hasattr(model, 'components_'):
            compressed_size = model.components_.size
            if hasattr(model, 'singular_values_'):
                compressed_size += model.singular_values_.size
            if hasattr(model, 'mean_'):
                compressed_size += model.mean_.size
        else:
            print(f"警告: 无法计算未知类型的压缩比: {type(model)}")
            return 1.0  # 默认返回 1.0

    return original_size / compressed_size


"""计算存储空间节省率"""


def calculate_storage_saving_rate(original, decomposed_elements):
    """计算存储空间节省率"""
    original_size = np.prod(original.shape)
    if isinstance(decomposed_elements, tuple):  # 张量分解
        core, factors = decomposed_elements
        compressed_size = core.size

        # 调整因子矩阵的权重
        factor_weight = 0.5  # 可调整参数
        for factor in factors:
            compressed_size += int(factor.size * factor_weight)
    else:  # 矩阵分解或 PCA
        model = decomposed_elements
        if hasattr(model, 'components_'):
            compressed_size = model.components_.size
            if hasattr(model, 'singular_values_'):
                compressed_size += model.singular_values_.size
            if hasattr(model, 'mean_'):
                compressed_size += model.mean_.size
        else:
            compressed_size = original_size  # 无法计算，默认不节省

    return (1 - compressed_size / original_size) * 100


"""评估降维后数据的聚类质量"""


def evaluate_clustering_quality(transformed_data):
    """评估降维后数据的聚类质量"""
    if transformed_data.shape[1] < 2:  # 至少需要两维数据
        return 0

    try:
```

```
        # 计算轮廓系数，评估聚类质量
        from sklearn.metrics import silhouette_score
        # 由于没有真实标签，我们假设所有样本属于同一类
        score = silhouette_score(transformed_data,
np.zeros(transformed_data.shape[0]))
        return score
    except:
        return 0


"""粒子群算法中的粒子"""


class Particle:
    """粒子群算法中的粒子"""

    def __init__(self, dim, min_rank, max_rank):
        self.position = np.array([
            random.randint(min_rank[0], max_rank[0])
        ], dtype=np.float64)

        self.velocity = np.array([
            random.uniform(-10, 10)
        ])

        self.best_position = self.position.copy()
        self.best_fitness = float('-inf')
        self.fitness_history = []


"""粒子群优化算法寻找最优组件数，平衡 MSE 和压缩比"""


def particle_swarm_optimization(train_matrix, test_matrix,
original_train_matrix, original_test_matrix, max_iter=50,
                                num_particles=20, w=0.7, c1=1.4, c2=1.4,
max_attempts=100):
    """粒子群优化算法寻找最优组件数，平衡 MSE 和压缩比"""
    dims = train_matrix.shape

    # 设置搜索范围
    min_rank = [1]
    max_rank = [min(500, min(dims))]  # 限制最大组件数

    print(f"矩阵维度: {dims}")
    print(f"搜索组件数范围: {max_rank[0]}")

    particles = [Particle(1, min_rank, max_rank) for _ in
range(num_particles)]
    global_best_position = None
    global_best_cr = 0
    global_best_mse = float('inf')
    global_best_test_mse = float('inf')
    global_best_original_mse = float('inf')  # 新增：原始数据 MSE
    global_best_original_test_mse = float('inf')  # 新增：原始测试数据 MSE
    global_svd = None

    # 早停计数器
```

```
    no_improvement_count = 0
    best_original_mse = float('inf')
    best_original_test_mse = float('inf')

    # 尝试次数计数器
    total_attempts = 0

    print("开始粒子群优化搜索最优组件数...")

    for iteration in range(max_iter):
        start_time = time.time()
        iteration_improved = False

        for particle in particles:
            n_components = int(np.round(particle.position[0]))
            n_components = np.clip(n_components, min_rank[0],
max_rank[0])

            total_attempts += 1

            try:
                # 对训练集进行矩阵分解和MSE 计算
                train_reconstructed, svd =
matrix_decomposition(train_matrix, n_components)
                train_mse = calculate_mse(train_matrix,
train_reconstructed)

                # 对测试集进行矩阵分解和MSE 计算
                test_reconstructed, _ = matrix_decomposition(test_matrix,
n_components)
                test_mse = calculate_mse(test_matrix, test_reconstructed)

                cr = calculate_compression_ratio(train_matrix, svd)

                # 计算原始数据的重构和MSE
                original_train_reconstructed =
svd.inverse_transform(svd.transform(original_train_matrix))
                original_mse = calculate_mse(original_train_matrix,
original_train_reconstructed)

                original_test_reconstructed =
svd.inverse_transform(svd.transform(original_test_matrix))
                original_test_mse = calculate_mse(original_test_matrix,
original_test_reconstructed)

                # 只考虑原始训练集和测试集MSE 都小于0.005 的解
                if original_mse < 0.005 and original_test_mse < 0.005:
                    # 检查MSE 是否过低，可能表示过拟合
                    if original_mse < 1e-10 or original_test_mse < 1e-10:
                        print(
                            f"警告：检测到极低MSE （原始训练集:
{original_mse:.10f}, 原始测试集: {original_test_mse:.10f}），可能存在过拟合
")
                        # 计算相对误差作为额外验证
                        original_train_relative_error =
calculate_relative_error(original_train_matrix,

original_train_reconstructed)
```

```
                    original_test_relative_error =
calculate_relative_error(original_test_matrix,

original_test_reconstructed)
                    print(
                        f"原始训练集相对误差:
{original_train_relative_error:.10f}, 原始测试集相对误差:
{original_test_relative_error:.10f}")

                    # 如果相对误差也很小，可能是合理的解
                    if original_train_relative_error < 1e-6 and
original_test_relative_error < 1e-6:
                        print("相对误差也很低，解可能是合理的")
                    else:
                        print("相对误差较高，可能存在数值问题，忽略此解")
                        continue

                if cr > particle.best_fitness:
                    particle.best_fitness = cr
                    particle.best_position = particle.position.copy()

                if cr > global_best_cr:
                    global_best_cr = cr
                    global_best_position = n_components
                    global_best_mse = train_mse
                    global_best_test_mse = test_mse
                    global_best_original_mse = original_mse
                    global_best_original_test_mse = original_test_mse
                    global_svd = svd
                    print(
                        f"迭代 {iteration + 1}/{max_iter}: 新全局最优 -
Components={global_best_position}, 预处理训练集
MSE={global_best_mse:.6f}, 预处理测试集 MSE={global_best_test_mse:.6f}, 原
始训练集 MSE={global_best_original_mse:.6f}, 原始测试集
MSE={global_best_original_test_mse:.6f}, 压缩比={global_best_cr:.2f}x")
                    iteration_improved = True
            else:
                print(
                    f"组件数 {n_components}: 原始训练集
MSE={original_mse:.6f}, 原始测试集 MSE={original_test_mse:.6f}, 未达到要求
（<0.005）")

        except Exception as e:
            print(f"计算组件数 {n_components} 时出错: {e}")

    # 更新粒子速度和位置
    for particle in particles:
        # 修正：分别生成两个随机数
        r1 = random.random()
        r2 = random.random()

        particle.velocity = (w * particle.velocity +
                             c1 * r1 * (particle.best_position -
particle.position) +
                             c2 * r2 * (np.array([global_best_position])
- particle.position))
```

```
            # 限制最大速度，防止粒子跳出搜索空间
            particle.velocity = np.clip(particle.velocity, -50, 50)

            particle.position += particle.velocity
            particle.position = np.clip(particle.position, min_rank,
max_rank)

        iteration_time = time.time() - start_time

        # 更新早停计数器
        if iteration_improved:
            no_improvement_count = 0
        else:
            no_improvement_count += 1

        # 如果找到了符合条件的解，且连续 10 次迭代没有改进，就提前结束
        if global_best_position is not None and no_improvement_count >=
10:
            print(f"早停：连续{no_improvement_count}次迭代没有改进")
            break

        # 如果尝试次数太多，也提前结束
        if total_attempts >= max_attempts:
            print(f"达到最大尝试次数({max_attempts})，提前结束优化")
            break

        print(
            f"迭代 {iteration + 1}/{max_iter} 完成，耗时:
{iteration_time:.2f}秒，当前最优 - 预处理训练集 MSE: {global_best_mse:.6f},
预处理测试集 MSE: {global_best_test_mse:.6f}, 原始训练集 MSE:
{global_best_original_mse:.6f}, 原始测试集 MSE:
{global_best_original_test_mse:.6f}, 压缩比: {global_best_cr:.2f}x")

    # 计算最终结果
    if global_best_position is not None:
        print(
            f"最终结果 - Components={global_best_position}, 预处理训练集
MSE={global_best_mse:.6f}, 预处理测试集 MSE={global_best_test_mse:.6f}, 原
始训练集 MSE={global_best_original_mse:.6f}, 原始测试集
MSE={global_best_original_test_mse:.6f}, 压缩比={global_best_cr:.2f}x")
    else:
        print("警告：未能找到满足原始训练集和测试集 MSE 均小于 0.005 的解")
        # 如果没有找到符合条件的解，返回最佳的可用解
        best_components = max_rank[0]  # 默认使用最大组件数
        print(f"使用最大组件数 {best_components} 作为备选方案")
        train_reconstructed, svd = matrix_decomposition(train_matrix,
best_components)
        train_mse = calculate_mse(train_matrix, train_reconstructed)
        test_reconstructed, _ = matrix_decomposition(test_matrix,
best_components)
        test_mse = calculate_mse(test_matrix, test_reconstructed)
        cr = calculate_compression_ratio(train_matrix, svd)
        original_train_reconstructed =
svd.inverse_transform(svd.transform(original_train_matrix))
        original_mse = calculate_mse(original_train_matrix,
original_train_reconstructed)
        original_test_reconstructed =
```

```
svd.inverse_transform(svd.transform(original_test_matrix))
        original_test_mse = calculate_mse(original_test_matrix,
original_test_reconstructed)

        global_best_position = best_components
        global_best_mse = train_mse
        global_best_test_mse = test_mse
        global_best_original_mse = original_mse
        global_best_original_test_mse = original_test_mse
        global_best_cr = cr
        global_svd = svd

        print(
            f"备选方案结果 - Components={global_best_position}, 预处理训练集
MSE={global_best_mse:.6f}, 预处理测试集 MSE={global_best_test_mse:.6f}, 原
始训练集 MSE={global_best_original_mse:.6f}, 原始测试集
MSE={global_best_original_test_mse:.6f}, 压缩比={global_best_cr:.2f}x")

    return global_best_position, global_best_mse, global_best_test_mse,
global_best_cr, global_best_original_mse,
global_best_original_test_mse, global_svd


"""可视化矩阵分解结果"""


def visualize_results(
        original_matrix, matrix_recon, matrix_components, matrix_mse,
matrix_cr, matrix_time,
        test_matrix, test_matrix_recon, test_matrix_mse,
        original_train_matrix, original_train_recon, original_mse,
        original_test_matrix, original_test_recon, original_test_mse,
        train_matrix_transformed=None, svd=None
):
    """可视化矩阵分解结果"""
    plt.figure(figsize=(15, 15))

    # 预处理训练数据矩阵分解结果
    plt.subplot(3, 2, 1)
    plt.imshow(matrix_recon, cmap='viridis')
    plt.colorbar()
    plt.title(
        f'SVD 矩阵分解重构(预处理训练数据)\n 组件数: {matrix_components}, MSE:
{matrix_mse:.6f}, 压缩比: {matrix_cr:.2f}x')

    # 预处理训练数据矩阵分解误差
    plt.subplot(3, 2, 2)
    matrix_error = original_matrix - matrix_recon
    plt.imshow(matrix_error, cmap='coolwarm')
    plt.colorbar()
    plt.title('SVD 分解误差矩阵(预处理训练数据)')

    # 预处理测试数据矩阵分解结果
    plt.subplot(3, 2, 3)
    plt.imshow(test_matrix_recon, cmap='viridis')
    plt.colorbar()
    plt.title(f'SVD 矩阵分解重构(预处理测试数据)\nMSE:
{test_matrix_mse:.6f}')
```

```
    # 预处理测试数据矩阵分解误差
    plt.subplot(3, 2, 4)
    test_matrix_error = test_matrix - test_matrix_recon
    plt.imshow(test_matrix_error, cmap='coolwarm')
    plt.colorbar()
    plt.title('SVD 分解误差矩阵(预处理测试数据)')

    # 原始训练数据矩阵分解结果
    plt.subplot(3, 2, 5)
    plt.imshow(original_train_recon, cmap='viridis')
    plt.colorbar()
    plt.title(f'SVD 矩阵分解重构(原始训练数据)\nMSE: {original_mse:.6f}')

    # 原始测试数据矩阵分解结果
    plt.subplot(3, 2, 6)
    plt.imshow(original_test_recon, cmap='viridis')
    plt.colorbar()
    plt.title(f'SVD 矩阵分解重构(原始测试数据)\nMSE:
{original_test_mse:.6f}')

    plt.tight_layout()
    plt.savefig('matrix_decomposition_result.png', dpi=300,
bbox_inches='tight')
    plt.close()

    # 性能对比表
    print("\n========== 矩阵分解性能指标 ==========")
    print(
        f"{'数据类型':<15} | {'训练集 MSE':<10} | {'测试集 MSE':<10} | {'压缩
比':<10} | {'存储空间节省率(%)':<15} | {'计算时间(秒)'}")
    print("-" * 80)
    print(
        f"{'预处理数据':<15} | {matrix_mse:<10.6f} |
{test_matrix_mse:<10.6f} | {matrix_cr:<10.2f} |
{calculate_storage_saving_rate(original_matrix, svd):<15.2f} |
{matrix_time:.2f}")
    print(
        f"{'原始数据':<15} | {original_mse:<10.6f} |
{original_test_mse:<10.6f} | {matrix_cr:<10.2f} |
{calculate_storage_saving_rate(original_train_matrix, svd):<15.2f} |
{matrix_time:.2f}")
    print("-" * 80)

    # 计算其他评估指标
    matrix_rmse = calculate_rmse(original_matrix, matrix_recon)
    matrix_mae = calculate_mae(original_matrix, matrix_recon)
    matrix_rel_error = calculate_relative_error(original_matrix,
matrix_recon)
    matrix_exp_var = calculate_explained_variance(original_matrix,
matrix_recon)
    matrix_cluster_score =
evaluate_clustering_quality(train_matrix_transformed)

    # 详细性能对比表
    print("\n========== 矩阵分解详细性能指标 ==========")
    print(
```

```
        f"{'数据类型':<15} | {'训练集 MSE':<10} | {'测试集 MSE':<10} | {'压缩比':<10} | {'存储空间节省率(%)':<15} | {'计算时间(秒)':<12} | {'解释方差(%)':<12} | {'相对误差':<10} | {'聚类质量'}")
    print("-" * 120)
    print(
        f"{'预处理数据':<15} | {matrix_mse:<10.6f} | {test_matrix_mse:<10.6f} | {matrix_cr:<10.2f} | {calculate_storage_saving_rate(original_matrix, svd):<15.2f} | {matrix_time:<12.2f} | {matrix_exp_var * 100:<12.2f} | {matrix_rel_error:<10.6f} | {matrix_cluster_score:.4f}")
    print(
        f"{'原始数据':<15} | {original_mse:<10.6f} | {original_test_mse:<10.6f} | {matrix_cr:<10.2f} | {calculate_storage_saving_rate(original_train_matrix, svd):<15.2f} | {matrix_time:<12.2f} | {calculate_explained_variance(original_train_matrix, original_train_recon) * 100:<12.2f} | {calculate_relative_error(original_train_matrix, original_train_recon):<10.6f} | {matrix_cluster_score:.4f}")
    print("-" * 120)

    # 可视化低维表示
    if train_matrix_transformed is not None:
        plt.figure(figsize=(6, 4))
        plt.scatter(train_matrix_transformed[:, 0], train_matrix_transformed[:, 1], alpha=0.5)
        plt.title('SVD 低维表示')
        plt.xlabel('维度 1')
        plt.ylabel('维度 2')
        plt.tight_layout()
        plt.savefig('matrix_low_dim_representation.png', dpi=300, bbox_inches='tight')
        plt.close()


def main():
    # 1. 加载数据
    print("正在加载数据...")
    file_path = r"D:\新建文件夹 (2)\Data(1).xlsx"
    data = load_data(file_path)
    if data is None:
        print("数据加载失败，程序退出")
        return
    print(f"数据形状: {data.shape}")

    # 2. 数据预处理
    print("正在预处理数据...")
    processed_data, original_data = preprocess_data(data, iqr_threshold=1.5)

    # 3. 划分训练集和测试集
    print("正在划分训练集和测试集...")
    train_idx, test_idx = train_test_split(range(data.shape[0]), test_size=0.2, random_state=42)
    train_data = processed_data[train_idx]
    test_data = processed_data[test_idx]
    original_train_data = original_data[train_idx]
```

```
    original_test_data = original_data[test_idx]

    # 4. 准备矩阵分解数据
    print("正在准备矩阵分解数据...")
    train_matrix = train_data  # 使用原始二维数据
    test_matrix = test_data

    # 5. 矩阵分解（SVD）
    print("\n======= 开始矩阵分解 (SVD) =======")
    start_time = time.time()

    # 使用粒子群优化寻找最优组件数，增加最大尝试次数
    matrix_components, matrix_mse, test_matrix_mse, matrix_cr,
original_mse, original_test_mse, svd = particle_swarm_optimization(
        train_matrix, test_matrix, original_train_data,
original_test_data,
        max_iter=100,  # 增加最大迭代次数
        num_particles=30,  # 增加粒子数量
        max_attempts=500  # 增加最大尝试次数
    )

    matrix_time = time.time() - start_time

    print(f"\n最优组件数: {matrix_components}")
    print(f"预处理训练集 MSE: {matrix_mse:.6f}")
    print(f"预处理测试集 MSE: {test_matrix_mse:.6f}")
    print(f"原始训练集 MSE: {original_mse:.6f}")
    print(f"原始测试集 MSE: {original_test_mse:.6f}")
    print(f"SVD 分解压缩比: {matrix_cr:.2f}x")

    # 执行 SVD 分解，确保所有需要的变量都被正确初始化
    matrix_recon, _ = matrix_decomposition(train_matrix,
matrix_components)
    test_matrix_recon, _ = matrix_decomposition(test_matrix,
matrix_components)  # 修复未解析的引用
    original_train_recon =
svd.inverse_transform(svd.transform(original_train_data))
    original_test_recon =
svd.inverse_transform(svd.transform(original_test_data))

    # 计算 SVD 降维后的低维表示
    train_matrix_transformed = svd.transform(train_matrix)

    # 6. 可视化结果
    print("\n正在生成可视化结果...")
    visualize_results(
        train_matrix, matrix_recon, matrix_components, matrix_mse,
matrix_cr, matrix_time,
        test_matrix, test_matrix_recon, test_matrix_mse,
        original_train_data, original_train_recon, original_mse,
        original_test_data, original_test_recon, original_test_mse,
        train_matrix_transformed, svd
    )

    # 7. 导出重构数据
    print("\n正在导出重构数据...")
```

```
    try:
        # 导出预处理数据的重构结果
        recon_df = pd.DataFrame(matrix_recon)
        recon_df.to_excel('preprocessed_reconstructed_data.xlsx',
index=False, header=False)

        # 导出原始数据的重构结果
        original_recon_df = pd.DataFrame(original_train_recon)
        original_recon_df.to_excel('original_reconstructed_data.xlsx',
index=False, header=False)

        # 导出完整数据集的重构结果
        full_recon =
svd.inverse_transform(svd.transform(processed_data))
        full_recon_df = pd.DataFrame(full_recon)
        full_recon_df.to_excel('combined_preprocessed_recon.xlsx',
index=False, header=False)

        full_original_recon =
svd.inverse_transform(svd.transform(original_data))
        full_original_recon_df = pd.DataFrame(full_original_recon)
        full_original_recon_df.to_excel('combined_original_recon.xlsx',
index=False, header=False)

        print("重构数据已成功导出为 Excel 文件!")
    except Exception as e:
        print(f"导出重构数据时出错: {e}")

    print("分析完成! 结果已保存到 'matrix_decomposition_result.png' 和相关
Excel 文件")


if __name__ == "__main__":
    main()
```

问题三

```
import numpy as np
import pandas as pd
import pywt
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import lasso_path, Lasso, LassoCV
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from mpl_toolkits.mplot3d import Axes3D

# ----------------------------------------------------------------------
---------
# 全局绘图设置
# ----------------------------------------------------------------------
---------
plt.style.use('ggplot')
plt.rcParams['font.sans-serif'] = ['SimHei']        # 中文字体
plt.rcParams['axes.unicode_minus'] = False         # 负号用 ASCII

# ----------------------------------------------------------------------
```

```
---------
# 1. 数据加载
# ------------------------------------------------------------------------
---------
X_raw = pd.read_excel('3-X.xlsx', header=None).values
y     = pd.read_excel('3-Y.xlsx', header=None).values.ravel()

# ------------------------------------------------------------------------
---------
# 2. 归一化 + 标准化
# ------------------------------------------------------------------------
---------
X_norm   = MinMaxScaler().fit_transform(X_raw)     # [0,1]
X_scaled = StandardScaler().fit_transform(X_norm)  # 零均值单位方差

# ------------------------------------------------------------------------
---------
# 3. 小波去噪（在归一化后去噪，再标准化）
# ------------------------------------------------------------------------
---------
def wavelet_denoise(X, wavelet='db4', level=5):
    Xden = np.zeros_like(X)
    for j in range(X.shape[1]):
        coeffs = pywt.wavedec(X[:, j], wavelet, level=level)
        coeffs[0] = np.zeros_like(coeffs[0])
        Xden[:, j] = pywt.waverec(coeffs, wavelet)[:X.shape[0]]
    return Xden

X_den_norm = wavelet_denoise(X_norm)
X_den      = StandardScaler().fit_transform(X_den_norm)

# ------------------------------------------------------------------------
---------
# 4. 相关系数对比（前 20 特征）
# ------------------------------------------------------------------------
---------
def compute_corr(X):
    return [pearsonr(X[:, j], y)[0] for j in range(X.shape[1])]

corr_raw      = compute_corr(X_raw)
corr_norm     = compute_corr(X_norm)
corr_denoised = compute_corr(X_den_norm)

k = 20
inds = np.arange(1, k+1)
w = 0.25

plt.figure(figsize=(12, 6))
plt.bar(inds - w, corr_raw[:k],      width=w, label='原始', alpha=0.8)
plt.bar(inds,     corr_norm[:k],     width=w, label='归一化', alpha=0.8)
plt.bar(inds + w,  corr_denoised[:k], width=w, label='去噪', alpha=0.8)
plt.xlabel('特征索引')
plt.ylabel('Pearson 相关系数')
plt.title('前 20 特征 vs Y 相关系数对比')
plt.xticks(inds)
plt.legend()
plt.tight_layout()
plt.show()
```

```
# ------------------------------------------------------------------------
---------
# 5. 3D 点云可视化（三个示例特征）
# ------------------------------------------------------------------------
---------
def plot_3d(X, title):
    fig = plt.figure(figsize=(8, 6))
    ax  = fig.add_subplot(111, projection='3d')
    sc  = ax.scatter(X[:,0], X[:,1], X[:,2], c=y, cmap='viridis', s=20,
alpha=0.7)
    ax.set_xlabel('特征 1'); ax.set_ylabel('特征 2'); ax.set_zlabel('特征
3')
    ax.set_title(title)
    fig.colorbar(sc, ax=ax, label='Y 值')
    plt.tight_layout()
    plt.show()

plot_3d(X_raw,      '原始 3D 点云')
plot_3d(X_norm,     '归一化 3D 点云')
plot_3d(X_den_norm, '去噪后 3D 点云')

# ------------------------------------------------------------------------
---------
# 6. Lasso 正则化路径（自动 alphas）
# ------------------------------------------------------------------------
---------
alphas_path, coefs, _ = lasso_path(X_den, y)
print(f"自动生成的 alphas_path 范围: {alphas_path.min():.2e} —
{alphas_path.max():.2e}")

plt.figure(figsize=(8, 6))
for coef in coefs:
    plt.plot(alphas_path, coef, alpha=0.5)
plt.xscale('log')
plt.gca().invert_xaxis()
plt.xlabel('Alpha (log scale)')
plt.ylabel('系数值')
plt.title('Lasso 正则化路径（标准化后）')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------------------
---------
# 7. LassoCV 自动选 α & CV 曲线
# ------------------------------------------------------------------------
---------
lasso_cv = LassoCV(
    alphas=alphas_path,
    cv=5,
    max_iter=5000,
    random_state=42
)
lasso_cv.fit(X_den, y)

mp = lasso_cv.mse_path_
print("mse_path_ shape =", mp.shape)
```

```
# 根据 shape 决定沿哪个轴平均
if mp.shape[0] == len(lasso_cv.alphas_):
    mse_mean = mp.mean(axis=1)
else:
    mse_mean = mp.mean(axis=0)

plt.figure(figsize=(8, 6))
plt.plot(lasso_cv.alphas_, mse_mean, 'o-', label='平均 CV MSE')
plt.xscale('log')
plt.gca().invert_xaxis()
plt.axvline(lasso_cv.alpha_, linestyle='--', color='red',
           label=f'最佳 α={lasso_cv.alpha_:.2e}')
plt.xlabel('Alpha')
plt.ylabel('平均 CV MSE')
plt.title('LassoCV 交叉验证曲线')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# ------------------------------------------------------------------------
---------
# 8. 最终 Lasso 拟合与诊断
# ------------------------------------------------------------------------
---------
model = Lasso(alpha=lasso_cv.alpha_, max_iter=5000)
model.fit(X_den, y)
y_pred = model.predict(X_den)
res    = y - y_pred

print("\nLassoCV 最终模型评估：")
print(f" 最佳 α = {lasso_cv.alpha_:.2e}")
print(f" R²  = {r2_score(y, y_pred):.4f}")
print(f" MSE = {mean_squared_error(y, y_pred):.4f}")
print(f" MAE = {mean_absolute_error(y, y_pred):.4f}")

# 实际 vs 预测
plt.figure(figsize=(6, 6))
plt.scatter(y, y_pred, s=20, alpha=0.7)
mn, mx = y.min(), y.max()
plt.plot([mn, mx], [mn, mx], '--', color='gray')
plt.xlabel('实际')
plt.ylabel('预测')
plt.title('实际 vs 预测 (LassoCV)')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# 残差 vs 预测
plt.figure(figsize=(6, 4))
plt.scatter(y_pred, res, s=20, alpha=0.7)
plt.axhline(0, linestyle='--', color='gray')
plt.xlabel('预测值')
plt.ylabel('残差')
plt.title('残差 vs 预测值')
plt.grid(alpha=0.3)
plt.tight_layout()
```

```
plt.show()

# 残差分布
plt.figure(figsize=(6,4))
plt.hist(res, bins=50, density=True, alpha=0.7)
plt.xlabel('残差')
plt.ylabel('密度')
plt.title('残差分布')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

问题四

```
import numpy as np
import pandas as pd
from sklearn.svm import SVR
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from sklearn.model_selection import train_test_split, cross_val_score
from joblib import Parallel, delayed
import matplotlib.pyplot as plt
from time import time

# 数据加载、预处理
X = pd.read_excel('4-X.xlsx', header=None).to_numpy()
y = pd.read_excel('4-Y.xlsx', header=None).squeeze().to_numpy()

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# 手动加一列全 1（相当于截距）
X_design = np.concatenate([np.ones((X_scaled.shape[0],1)), X_scaled],
axis=1)

X_tr, X_te, y_tr, y_te = train_test_split(
    X_design, y, test_size=0.2, random_state=42
)

# 网格搜索
C_list   = np.logspace(-3, 3, num=10)
eps_list = np.logspace(-3, 1, num=10)

def fit_and_score(C_val, eps_val, Xdata, ydata):
    t_start = time()
    svr = SVR(kernel='linear', C=C_val, epsilon=eps_val)
    svr.fit(Xdata, ydata)
    y_hat = svr.predict(Xdata)
    score = r2_score(ydata, y_hat)
    t_used = time() - t_start
    print(f"C={C_val:.4g}, ε={eps_val:.4g} ⇒ R²={score:.4f} （耗时
{t_used:.2f}s）")
    return {
        'C': C_val, 'epsilon': eps_val,
        'logC': np.log10(C_val), 'logEps': np.log10(eps_val),
        'R2': score, 'time': t_used
    }
```

```
all_results = Parallel(n_jobs=-1)(
    delayed(fit_and_score)(C, eps, X_tr, y_tr)
    for C in C_list for eps in eps_list
)
df = pd.DataFrame(all_results)

# 挑最佳 看趋势
best_row = df.loc[df['R2'].idxmax()]
best_C, best_eps = best_row['C'], best_row['epsilon']
print("\n最佳参数：")
print(best_row[['C','epsilon','R2']])

print("\nTop5 R2 排行：")
print(df.nlargest(5, 'R2')[['C','epsilon','R2']])

print("\nlogC/logEps 与 R2 的相关性：")
print(df[['logC','logEps','R2']].corr())

# 用最优参数跑一次，输出训练/测试指标
svr_best = SVR(kernel='linear', C=best_C, epsilon=best_eps)
t0 = time()
svr_best.fit(X_tr, y_tr)
print(f"耗时{time() - t0:.2f}s")

# 训练集评估
y_tr_pred = svr_best.predict(X_tr)
r2_tr = r2_score(y_tr, y_tr_pred)
mse_tr = mean_squared_error(y_tr, y_tr_pred)
mae_tr = mean_absolute_error(y_tr, y_tr_pred)
rmse_tr = np.sqrt(mse_tr)

# 测试集评估
y_te_pred = svr_best.predict(X_te)
r2_te = r2_score(y_te, y_te_pred)
mse_te = mean_squared_error(y_te, y_te_pred)
mae_te = mean_absolute_error(y_te, y_te_pred)
rmse_te = np.sqrt(mse_te)

print("\n模型评估——")
print(f"  · 训练集 R²={r2_tr:.4f}, MSE={mse_tr:.4f}, MAE={mae_tr:.4f},
RMSE={rmse_tr:.4f}")
print(f"  · 测试集 R²={r2_te:.4f}, MSE={mse_te:.4f}, MAE={mae_te:.4f},
RMSE={rmse_te:.4f}")

# 交叉验证看看稳定性
cv_mse = -cross_val_score(
    svr_best, X_tr, y_tr, cv=5, scoring='neg_mean_squared_error'
)
print(f"CV MSE 平均={cv_mse.mean():.4f}, 方差={cv_mse.std():.4f}")

# 支持向量数+残差分布
n_support = len(svr_best.support_)
print(f"\n模型用{n_support}个支持向量，共{X_tr.shape[0]}训练样本")
resid = y_te - y_te_pred
print("测试集残差统计：",
      f"最大 {resid.max():.4f}, 最小 {resid.min():.4f},
```

```
σ={resid.std():.4f}")

plt.figure(figsize=(12,4))

# log10(C) vs R2
plt.subplot(1,3,1)
sorted_by_C = df.sort_values('logC')
plt.plot(sorted_by_C['logC'], sorted_by_C['R2'], marker='o')
plt.xlabel('log10(C)')
plt.ylabel('R2')
plt.title('log10(C) vs R2')

# log10(epsilon) vs R2
plt.subplot(1,3,2)
sorted_by_eps = df.sort_values('logEps')
plt.plot(sorted_by_eps['logEps'], sorted_by_eps['R2'], marker='o')
plt.xlabel('log10(epsilon)')
plt.ylabel('R2')
plt.title('log10(epsilon) vs R2')

# R2 热力图
plt.subplot(1,3,3)
heat_data = df.pivot(index='logEps', columns='logC', values='R2')
plt.imshow(heat_data, origin='lower', aspect='auto')
plt.colorbar(label='R2')
plt.xlabel('log10(C)')
plt.ylabel('log10(epsilon)')
plt.title('R2 Heatmap')

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

问题五

```
# -*- coding: utf-8 -*-
"""
降维方法比较：PCA vs AE（联合训练）
回归模型：Linear, Ridge, Lasso, SVR, MLP, RandomForest
评估：重构误差 + 回归误差 + 散点图分析
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from joblib import Parallel, delayed
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import regularizers
import tensorflow.keras.backend as K
```

```
import time
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s
- %(message)s')

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


def build_joint_autoencoder(input_dim: int, bottleneck_dim: int) ->
Model:
    inp = Input(shape=(input_dim,))
    encoded = Dense(bottleneck_dim, activation='relu',
                    activity_regularizer=regularizers.l1(1e-5))(inp)
    decoded = Dense(input_dim)(encoded)
    pred_y = Dense(1)(encoded)

    model = Model(inputs=inp, outputs=[decoded, pred_y])
    model.compile(optimizer='adam',
                  loss=['mse', 'mse'],
                  loss_weights=[0.5, 0.5])
    return model


def evaluate_k(k: int, X_std: np.ndarray, X_orig: np.ndarray, Y_std:
np.ndarray, scaler_y, regressors: dict) -> dict:
    start = time.time()
    input_dim = X_std.shape[1]

    # ---- PCA 降维与重构 ----
    pca = PCA(n_components=k)
    Z = pca.fit_transform(X_std)
    Xp_std = pca.inverse_transform(Z)
    Xp = scaler.inverse_transform(Xp_std)
    pca_recon_mse = mean_squared_error(X_orig, Xp)

    pca_metrics = {}
    for name, mdl in regressors.items():
        cv = KFold(n_splits=5, shuffle=True, random_state=42)
        scores = {'R2': [], 'MSE': [], 'MAE': []}
        for tr, te in cv.split(Xp):
            mdl.fit(Xp[tr], Y_std[tr])
            pred =
scaler_y.inverse_transform(mdl.predict(Xp[te]).reshape(-1, 1)).ravel()
            y_true = scaler_y.inverse_transform(Y_std[te].reshape(-1,
1)).ravel()
            scores['R2'].append(r2_score(y_true, pred))
            scores['MSE'].append(mean_squared_error(y_true, pred))
            scores['MAE'].append(mean_absolute_error(y_true, pred))
        pca_metrics[name] = {k: np.mean(v) for k, v in scores.items()}

    # ---- AE 联合训练 ----
    ae = build_joint_autoencoder(input_dim, k)
    ae.fit(X_std, [X_std, Y_std],
           epochs=20,
           batch_size=256,
           shuffle=True,
           validation_split=0.1,
```

```
          verbose=0)

    Xa_std, pred_y_std = ae.predict(X_std)
    Xa = scaler.inverse_transform(Xa_std)
    ae_recon_mse = mean_squared_error(X_orig, Xa)

    ae_metrics = {}
    for name, mdl in regressors.items():
        cv = KFold(n_splits=5, shuffle=True, random_state=42)
        scores = {'R2': [], 'MSE': [], 'MAE': []}
        for tr, te in cv.split(Xa):
            mdl.fit(Xa[tr], Y_std[tr])
            pred =
scaler_y.inverse_transform(mdl.predict(Xa[te]).reshape(-1, 1)).ravel()
            y_true = scaler_y.inverse_transform(Y_std[te].reshape(-1,
1)).ravel()
            scores['R2'].append(r2_score(y_true, pred))
            scores['MSE'].append(mean_squared_error(y_true, pred))
            scores['MAE'].append(mean_absolute_error(y_true, pred))
        ae_metrics[name] = {k: np.mean(v) for k, v in scores.items()}

    logging.info(
        f"k={k:3d} 完成 | PCA_MSE={pca_recon_mse:.4f} |
AE_R2_Linear={ae_metrics['Linear']['R2']:.4f} | 用时={time.time() -
start:.2f}s")

    return {
        'k': k,
        'pca_recon_mse': pca_recon_mse,
        'ae_recon_mse': ae_recon_mse,
        'pca_metrics': pca_metrics,
        'ae_metrics': ae_metrics
    }


if __name__ == "__main__":
    # === 数据读取 ===
    X_orig = pd.read_excel('5-X.xlsx', header=None).values
    Y = pd.read_excel('5-Y.xlsx', header=None).values.ravel()

    scaler = StandardScaler()
    scaler_y = StandardScaler()

    X_std = scaler.fit_transform(X_orig)
    Y_std = scaler_y.fit_transform(Y.reshape(-1, 1)).ravel()

    # === 回归模型 ===
    regressors = {
        'Linear': LinearRegression(),
        'Ridge': Ridge(alpha=1.0),
        'Lasso': Lasso(alpha=1e-2, max_iter=5000),
        'SVR': SVR(kernel='rbf', C=10, epsilon=0.1),
        'MLP': MLPRegressor(hidden_layer_sizes=(64, 64), max_iter=1000,
random_state=42),
        'RF': RandomForestRegressor(n_estimators=100, random_state=42)
    }

    # === 多维度 k 并行评估 ===
    k_list = list(range(5, 101, 5))
```

```
    results = Parallel(n_jobs=-1)(
        delayed(evaluate_k)(k, X_std, X_orig, Y_std, scaler_y,
regressors) for k in k_list
    )

    # === 汇总结果 ===
    recon_mse = [r['pca_recon_mse'] for r in results]
    ae_recon_mse = [r['ae_recon_mse'] for r in results]
    pca_r2 = {m: [r['pca_metrics'][m]['R2'] for r in results] for m in
regressors}
    ae_r2 = {m: [r['ae_metrics'][m]['R2'] for r in results] for m in
regressors}

    # === 绘图：重构误差 & R2 对比 ===
    fig, axes = plt.subplots(2, 1, figsize=(10, 12), sharex=True)
    axes[0].plot(k_list, recon_mse, 'o-', label='PCA 重构 MSE')
    axes[0].plot(k_list, ae_recon_mse, 's--', label='AE 重构 MSE')
    axes[0].set_ylabel("重构 MSE")
    axes[0].set_title("重构误差 vs k")
    axes[0].legend()
    axes[0].grid(True)

    for name in regressors:
        axes[1].plot(k_list, pca_r2[name], '-o', label=f'PCA+{name}')
        axes[1].plot(k_list, ae_r2[name], '--s', label=f'AE+{name}')
    axes[1].set_xlabel("瓶颈/主成分数 k")
    axes[1].set_ylabel("测试 R2")
    axes[1].set_title("PCA vs AE + 各模型 R2 比较")
    axes[1].legend()
    axes[1].grid(True)

    plt.tight_layout()
    plt.show()

    # === 绘制误差散点图：AE 重构误差 vs 回归误差 ===
    for name in regressors:
        ae_recons = np.array([r['ae_recon_mse'] for r in results])
        ae_reg_mse = np.array([r['ae_metrics'][name]['MSE'] for r in
results])

        plt.figure(figsize=(6, 5))
        plt.scatter(ae_recons, ae_reg_mse, c='blue', alpha=0.7)
        plt.xlabel("AE 重构 MSE")
        plt.ylabel(f"{name} 回归 MSE")
        plt.title(f"AE 重构误差 vs 回归误差（{name}）")
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    # === 输出最佳结果 ===
    best_idx = np.argmax(pca_r2['Linear'])
    best_k = k_list[best_idx]
    best_res = results[best_idx]

    print(f"\n>>> 最优 k (PCA+Linear) = {best_k}\n")
    print("模型       | 方法 |   R²    |   MSE    |   MAE")
    print("-----------|------|---------|----------|---------")
```

```
    for method, metrics in [('PCA', best_res['pca_metrics']),
                            ('AE', best_res['ae_metrics'])]:
        for name, vals in metrics.items():
            print(f"{name:9s} | {method:3s} | {vals['R2']:.4f} |
{vals['MSE']:.4f} | {vals['MAE']:.4f}")
```