基于 K-means 与 LSTM 对空气质量的分析

摘　要

随着全球工业化和城市化的发展，城市空气质量问题日益严重，影响人类健康和环境可持续发展。为了保持城市的可持续发展，本文对一氧化碳（CO）、非甲基烷烃等五个污染物浓度之间的相关性、未来预测等问题进行了分析。

对于问题一，本文对空气质量监测站的 CO（GT）等数据进行了缺失值处理，对于小面积缺失的数据，我们**采用线性插值**进行填补，对大面积缺失的数据，我们采用**线性回归分析**对其进行填补。其次，针对数据的异常值，进行假设检验，依据 **P 值**的大小选择**正态分布 3σ 原则**和**箱型图**对异常值进行处理。对于 P>0.05，则采用**正态分布 3σ 原则**，反之采用**箱型图**。此外，我们建立了**一元线性回归模型**分析浓度与一氧化碳浓度之间的关系，线性方程为 Y = **-0.333 + 0.046x**。最后通过**方差分析**和**检验**验证模型的准确性。

对于问题二，基于问题一的处理后的数据，针对浓度数据，我们绘制了浓度随时间变化的折线图，寻找浓度的变化规律并确定时间间隔为 **24h**。其次，我们采用 **LSTM 模型**对浓度度其污染物的一个月内变化趋势进行预测。最后，依据其污染物的早中晚、月、季变化规律，发现其与其它污染物的相关性较强，之前建立的数学模型能够应用到一氧化碳等其它几个污染物的变化趋势预测中。

对于问题三，依据**环境空气质量指数（AQI）技术规定**，我们利用法计算各污染物的空气质量分指数 $IAQI_P$ 和空气质量指数 AQI。其次，我们计算 AQI 与温度、绝对湿度和相对湿度之间的 **Spearman 秩相关系数**，分别为 0.358、0.075、-0.31，同样，再次利用 **Spearman 秩相关系数**评价不同浓度对温度变化的敏感度，求得非甲烷总烃对温度变化最敏感。其次，我们通过统计五个污染物指标的频率与**一元线性回归模型**实现了在不确定影响模型结果的前提下减少污染物种类。最后，经 **Kolmogorov-Smirnov 检验**确保了模型的准确性。

对于问题四，基于问题三建立的**对空气污染指标模型**及计算出的环境空气质量指数，我们通过分析数据的早中晚、天、月和季度的变化规律，发现早中晚的时间间隔规律变化明显作为分类时间间隔。将一天分为三个时间段并进行数据采样和平均，再将污染物气体浓度及 AQI 作为特征进行 **K-means 聚类**，将簇类中心 AQI 值作为分类标准，将环境污染程度分为六级，并根据各簇类中心的污染物气体浓度分析各分类的污染物特征，发现总氮氧化物及二氧化氮对污染程度影响较大，各污染物与环境污染程度呈现基本**正相关**的特点，并且早中晚时间段污染程度逐渐加深，依据总结的规律我们发现了污染物浓度变化的规律并给出了政策化建议。

关键词:　**LSTM 一元线性回归 空气污染指标 Spearman K-means**

一　问题的背景和重述

1.1　问题背景

随着全球工业化和城市化的发展，城市空气质量问题日益严重，影响人类健康和环境可持续发展[1]。工业排放、交通尾气、建筑扬尘等导致空气质量下降，增加了呼吸系统和其他疾病的发病率。为了解决这一问题，科学的城市空气质量分析与预测变得尤为重要。通过实时监测和数据分析，可以及时了解空气质量状况，为环境保护提供依据，帮助人们合理安排生活和生产，减少污染物排放。

1.2　问题重述

本文基于意大利某城市的多传感器收集的空气质量数据，旨在通过建立相应模型解决以下四个问题：

1. 对附件中所给的真实小时平均 CO 浓度等数据进行缺失数据的补充，并查找异常值进行替换，进而研究苯浓度与一氧化碳浓度之间是否存在函数关系。

2. 基于苯浓度的数据，选择合适的时间间隔，建立数学模型预测其后一个月的变化趋势。并探讨所建立的数学模型能否应用于其他四个污染物的变化趋势预测。

3. 基于五个污染物的数据，建立城市空气污染指标模型，分析该指标与温度、相对湿度和绝对湿度之间的关系。进一步探讨哪种污染物对温度变化最敏感，并研究在不影响模型及结果的前提下，减少污染物的种类。

4. 基于第三问的数学模型，对所给时间段的城市污染程度按早中晚、天、月或季度等合理时间间隔进行分类或分级，分析不同类别或等级下的污染物浓度特征。在此基础上，寻求污染物浓度变化的原因，并提出减少污染物排放的政策建议。

二　问题分析

2.1　问题一的分析

问题一要求对真实小时平均 CO 浓度等数据进行缺失值补充和异常值替换。首先，我们对缺失值通过线性插值法填补，以确保数据的连续性和完整性。对于异常值的检测，采用正态分布的 3σ 原则和箱型图法。3σ 原则指出，正态分布数据中 99.7% 的数据位于均值的 ± 3 倍标准差范围内，而箱型图通过五个统计量描述数据，用于识别并替换异常值。完成数据预处理后，采用一元线性回归模型分析苯浓度与一氧化碳浓度之间的关系，通过方差分析和 t 检验。

2.2　问题二的分析

问题二要求基于苯浓度的数据，选择合适的时间间隔预测后一个月的变化趋势。首先，我们可以绘制苯浓度随时间变化的折线图，寻找苯浓度的变化规律来确定合适的时间间隔，然后利用 LSTM 模型，选择合适时间序列预测模型对苯浓度的变化趋势进行预测。然后依据其它污染物的早中晚、月、季度变化规律，以及苯与其它污染物的相关性分析，按照污染物的特性选择和建立新的模型来预测变化趋势。

2.3　问题三的分析

问题三中要求基于五个污染物的数据，建立城市空气污染指标模型，并分析该指标与温度、相对湿度和绝对湿度之间的关系。首先，我们参考环境空气质量指数（AQI）技术规定，利用差分法计算各污染物的空气质量分指数 IAQIP，并通过公式计算空气质量指数 AQI。接下来，采用 Spearman 秩相关系

数分析 AQI 与温度、绝对湿度和相对湿度之间的关系，确定这些变量对空气质量的影响。其次，使用 Spearman 秩相关系数评估不同污染物对温度变化的敏感度。最后，我们利用一元线性回归模型及统计五个污染物的 IAQIP 出现的频率，在不显著影响模型结果的前提下减少污染物种类，并验证其可行性。

2.4 问题四的分析

问题四要求基于第三问的数学模型，对数据集中的城市污染程度按合理的时间间隔来进行分类或分级，并提取对应的污染物浓度特征，然后在此基础上分析污染物浓度变化的原因并对减少污染物排放提出建议。首先，我们将污染物分为早中晚三类，对一天中这三个时间区取平均值。然后利用 k-means 算法将污染物指标分成六类，并提取不同簇类污染物浓度的特征。接着我们基于污染物的特征对一天内污染物浓度变化的原因进行分析，并由结论得出对污染治理的建议

三 模型假设

1. 假设所使用的数据是完整的，填补后的数据不会对模型的整体准确性造成影响。
2. 假设所有传感器设备的操作都在最佳状态，以确保测量结果的精确性。任何由于设备故障导致的异常数据都将被识别并处理。
3. 模型假设环境因素在短时间内是相对稳定的，不会有极端的事件影响数据的一般性。

四 符号说明

符号	说明	单位
CO	一氧化碳浓度	$\mathrm{mg/m^3}$
C_6H_6	苯浓度	$\mu\mathrm{g/m^3}$
NO_x	氮氧化物浓度	ppb
NO_2	二氧化氮浓度	$\mu\mathrm{g/m^3}$
O_3	臭氧浓度	$\mu\mathrm{g/m^3}$
$IAQI_P$	污染物项目 P 的空气质量分指数	无
AQI	空气质量指数	无
T	温度	°C
RH	相对湿度	%
AH	绝对湿度	$\mathrm{g/m^3}$
β_0, β_1	线性回归模型参数	无
σ	标准差	无
μ	均值	无
$Q1, Q3$	第一个和第三个四分位数	无
IQR	四分位距	无

五　数据预处理

5.1　缺失值处理

由于数据集中存在部分缺失值，直接使用这些数据进行分析可能会导致结果偏差。为确保数据的完整性和连续性，我们检查了数据集中缺失值的分布情况。通过对缺失值的统计分析，发现部分变量存在少量缺失值，缺失值被标记为-200 值。为了合理填补这些缺失值，我们采用了线性插值处理与线性回归分析。

5.1.1　线性插值

对于小面积缺失的数据，我们采取线性回归分析对数据进行填充。线性插值假设两个已知数据点之间的变化是线性的，从而可以通过这两个点 X_1, X_2 的坐标推导出任意一个中间点的值，公式如下：

$$y = y_0 + \frac{(x - x_0)(y_1 - y_0)}{(x_1 - x_0)} \tag{1}$$

其中，(x_0, y_0) 为 X_1 的坐标，(x_1, y_1) 为 X_2 的坐标，x 为介于 x_0 和 x_1 之间的某一值，对应的 y 值即为通过线性插值计算得到的值。

5.1.2　线性回归分析

对于大面积缺失的数据，我们采取线性回归分析对数据进行填充，根据计算的函数对缺失值进行补充。线性回归分析的具体公式如下：

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2}$$

其中，y 是因变量，x 是自变量，β_0 是截距，β_1 是回归系数，ϵ 是误差项。

以补充大面积缺失的 NMHC(GT) 为例，结果如下图：

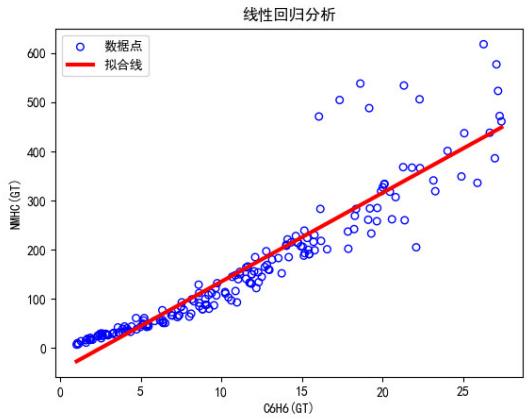

图 1: 对 C6H6 与 NMHC 的线性回归分析

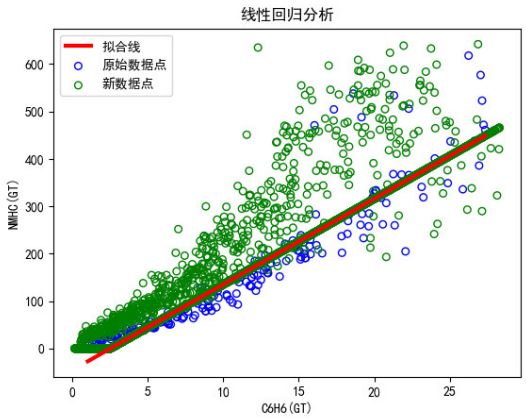

图 2: 新增数据与线性回归分析

5.2 异常值检验与替换

异常值是指在一组数据中显著偏离其他数据点的数值。为了确保数据分析结果的准确性，我们先进行假设检验，依据 p 的大小选择正态分布 3σ 原则或箱型图对异常值进行处理。

- p **值** > 0.05：如果 p 值大于 0.05，表明数据没有显著偏离正态分布，接受原假设（数据符合正态分布）。在这种情况下，使用 3σ 原则处理数据。
- p **值** ≤ 0.05：如果 p 值小于或等于 0.05，表明数据显著偏离正态分布，拒绝原假设（数据符合正态分布）。在这种情况下，使用箱型图来处理数据。

5.2.1 正态分布 3σ 原则

正态分布 3σ 原则是一种非常重要的概率分布，其概率密度函数为：

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3}$$

其中，μ 是均值，σ 是标准差。正态分布的一个重要性质是 68-95-99.7 法则，也称为 3σ 原则。这一原则指出：

$$\begin{cases} 68\% \text{ 的数据位于 } \mu \pm 1\sigma \text{ 之间。} \\ 95\% \text{ 的数据位于 } \mu \pm 2\sigma \text{ 之间。} \\ 99.7\% \text{ 的数据位于 } \mu \pm 3\sigma \text{ 之间。} \end{cases}$$

5.2.2 箱型图

箱线图是利用数据中的五个统计量：最小值、第一四分位数、中位数、第三四分位数与最大值来描述数据的一种方法，公式如下：

$$M = \begin{cases} x_{\frac{n+1}{2}}, & \text{如果} n \text{ 是奇数} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}, & \text{如果} n \text{ 是偶数} \end{cases} \tag{4}$$

其中，$\{x_1, x_2, \ldots, x_n\}$ 为数据集，按升序排列，M 为中位数。

四分位数将数据分成四等份，主要包括第一个四分位数（下四分位数，$Q1$）和第三个四分位数（上四分位数，$Q3$）。

- 第一个四分位数 $Q1$：数据集中第 25 百分位数。
- 第三个四分位数 $Q3$：数据集中第 75 百分位数。

具体表示如下：

$$Q1 = x_{\left(\frac{n+1}{4}\right)} \tag{5}$$

$$Q3 = x_{\left(\frac{3(n+1)}{4}\right)} \tag{6}$$

四分位距是上四分位数与下四分位数之差，其公式为：

$$\mathrm{IQR} = Q3 - Q1 \tag{7}$$

箱线图的示意图如下：

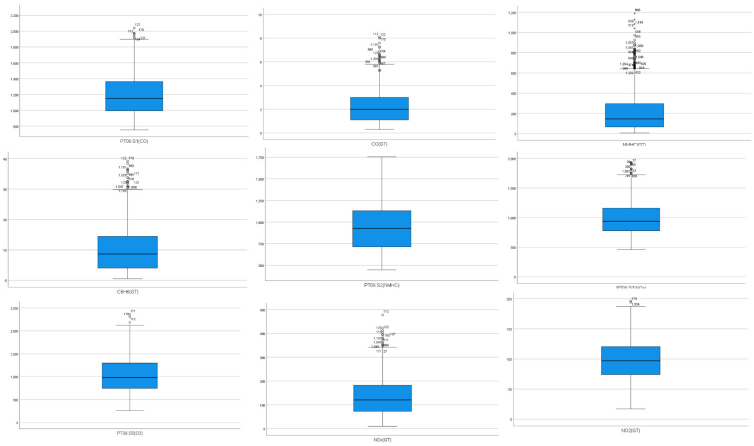

图 3: 数据的箱型图示意

六　模型的建立与求解

6.1　问题一模型的建立与求解

6.1.1　苯浓度与一氧化碳浓度的相关性分析

为了更加直观地分析两个参数的变化关系, 我们画出了苯浓度与一氧化碳浓度的散点图, 结果如下：

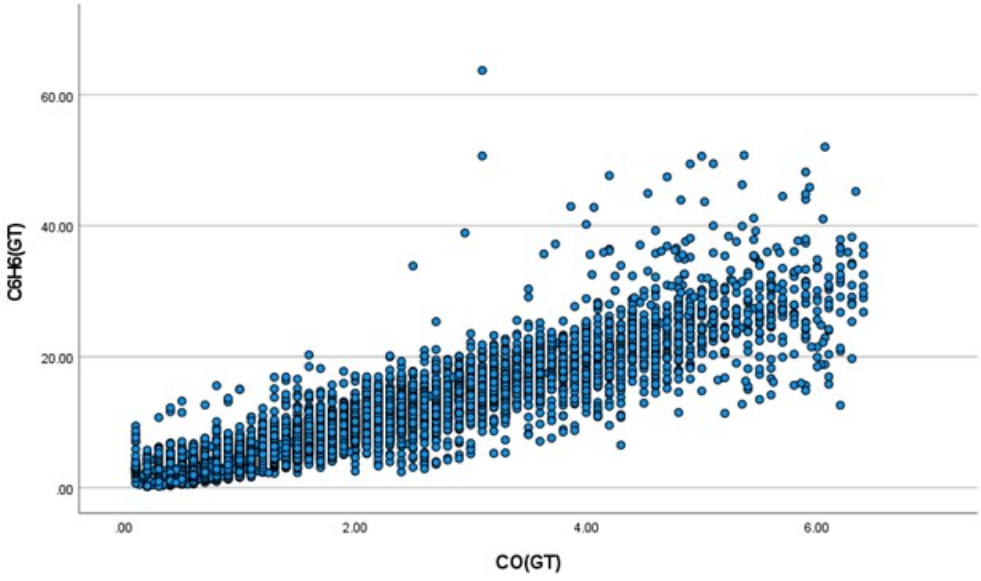

图 4: 苯浓度与一氧化碳浓度的散点图

通过分析苯浓度与一氧化碳浓度的散点图，可判断二者呈线性关系。

6.1.2 一元线性回归模型

针对问题一研究苯浓度与一氧化碳浓度之间是否存在函数关系，我们采用一元线性回归模型。模型只有一个自变量，与单因素分析相对应，可以处理一个自变量与一个因变量之间的线性关系。

1. 一元线性回归模型的建立

苯浓度与一氧化碳浓度存在着如下关系：

$$y = \beta_0 + \beta_1 x \tag{8}$$

其中，y 是因变量（一氧化碳浓度）；x 是自变量（苯浓度）；β_0 是截距项，即当 $x = 0$ 时的 y 值；β_1 是斜率项，表示自变量每增加一个单位，因变量的平均变化量。

2. 模型拟合

为了估计模型参数 β_0 和 β_1，通常使用最小二乘法（Ordinary Least Squares, OLS），其目标是最小化所有样本点的预测值与实际值之间的平方误差和。具体计算公式如下：

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{9}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \tag{10}$$

其中，$\hat{\beta}_0$ 和 $\hat{\beta}_1$ 分别为截距和斜率的估计值；$\bar{x}$ 和 $\bar{y}$ 分别为自变量和因变量的样本均值；n 为样本数量。

6.1.3 模型求解

根据一元线性回归模型求解，我们得出了一条尽可能靠近所有点的直线去探究苯浓度与一氧化碳浓度的关系，如下图所示：

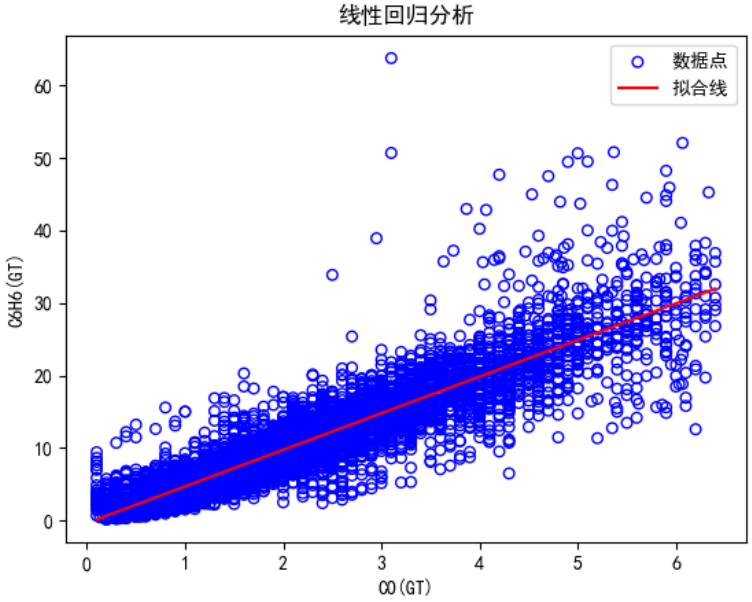

图 5： 苯浓度与一氧化碳浓度的回归直线

一元线性回归模型如下：

$$\hat{y} = -0.333 + 5.046x \tag{11}$$

从图中可以看出，数据点大致分布在回归直线周围，表明苯浓度和一氧化碳浓度之间存在显著的线性关系。

6.1.4 模型检验

为了确保一元线性回归模型的准确性，我们对模型进行检验，步骤如下：

1. 建立假设检验

- 零假设（H0）：$\beta = 0$，即苯浓度和一氧化碳浓度之间无直线关系；
- 备择假设（H1）：$\beta \neq 0$，即苯浓度和一氧化碳浓度之间存在直线关系。

显著性水平（α）设定为 0.05。

2. 方差分析通过 SPSS 计算得到的方差分析结果如下：

$$F = \frac{\mathrm{SS}_{\text{回}}}{\mathrm{V}_{\text{回}}} = \frac{345350.621}{1} = 35971.076 \tag{12}$$

残差方差为：

$$\frac{\mathrm{SS}_{\text{残}}}{\mathrm{V}_{\text{残}}} = \frac{9.601}{7580} \tag{13}$$

方差分析表中的 F 值为 35971.076，对应的 p 值为 $P < 0.001$。

查 F 分布表，取自由度 $v_1 = 1$ 和 $v_2 = 7580$，对应的 P 值小于 0.001。按照显著性水平 $\alpha = 0.05$，拒绝零假设 H0，接受备择假设 H1，因此可以认为苯浓度和一氧化碳浓度之间存在直线关系。

3. t 检验计算的 t 值为：

$$t^2 = F = 35971.076 \tag{14}$$

标准误差为 0.066 和 0.027，拟合效果良好。

查 t 分布表，取自由度 $v = 7580$，对应的 P 值小于 0.001。按照显著性水平 $\alpha = 0.05$，拒绝零假设 H0，接受备择假设 H1，因此可以认为苯浓度和一氧化碳浓度之间存在直线关系。

4. 相关系数与决定系数计算得到的相关系数和决定系数如下：

$$\begin{cases} R = 0.909 \\ R^2 = 0.826 \end{cases}$$

均接近 1，表明拟合效果良好。

5. 线性回归方程最终得到的回归方程为：

$$\hat{y} = -0.333 + 5.046x \tag{15}$$

决定系数为：

$$R^2 = 0.826 \tag{16}$$

综上所述，苯浓度和一氧化碳浓度之间存在显著的直线关系，回归方程和决定系数表明模型的拟合效果良好。

6.2 问题二模型的建立与求解

6.2.1 LSTM

LSTM [2] 通过细胞状态和门控机制来管理控制信息的流动，使其能够有效地捕捉时间序列数据中的长期依赖关系，是一种强大的时间序列预测工具其核心是其特殊的结构单元，包括细胞状态和门控单元。具体结构如下：

1. 细胞状态

细胞状态 C_t ，由线性操作修改，主要功能是长期维持信息的流动。信息在细胞状态中可以被增加或遗忘，这取决于遗忘门和输入门的作用。

2. 遗忘门

遗忘门用于控制上一步的记忆状态，决定哪些信息需要从细胞状态中被遗忘。通过 sigmoid 层输出一个介于 0 和 1 之间的值，该值与权值矩阵相乘，0 表示忘记，1 表示保留。公式如下：

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{17}$$

其中，W_f 和 b_f 分别是遗忘门的权重矩阵和偏置向量，σ 表示 sigmoid 函数，h_{t-1} 是上一个时刻的隐状态，x_t 是当前输入。

3. 输入门

输入门用于控制候选记忆状态，决定当前输入 x_t 中哪些信息是重要的，并准备将其存储在细胞状态中。它包括一个 sigmoid 层和一个 tanh 层，sigmoid 层确定哪些值应更新，tanh 层创建一个新的候选值向量，可以加入到细胞状态中。公式如下：

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{18}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{19}$$

其中，W_i 和 b_i 分别是输入门的权重矩阵和偏置向量，W_C 和 b_C 分别是候选记忆状态的权重矩阵和偏置向量，tanh 表示双曲正切函数。

4. 细胞状态更新

在输入门和遗忘门的作用下，细胞状态被更新。遗忘门输出 f_t 决定丢弃之前细胞状态 C_{t-1} 的哪些部分，输入门输出 i_t 和新的候选值 $\tilde{C}_t$ 决定加入哪些新的信息。公式如下：

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{20}$$

5. 输出门

输出门决定哪些部分的细胞状态将输出到外部。通过一个 sigmoid 函数来决定我们将使用哪部分细胞状态，然后将这部分通过 tanh 函数处理（得到一个介于 -1 和 1 之间的值）和输出门的输出相乘得到最终的输出 h_t。公式如下：

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{21}$$

$$h_t = o_t \cdot \tanh(C_t) \tag{22}$$

其中，W_o 和 b_o 分别是输出门的权重矩阵和偏置向量。

6.2.2 模型求解

针对变化趋势的预测，我们利用 LSTM 根据处理好的前几个月的数据对苯浓度的后一个月的变化趋势预测，模型参数设置如下：

表 1: LSTM 模型参数

单层 LSTM 单元数	全连接层神经元数	迭代次数	损失函数	优化器	预测参考历史数据
10	7	300	均方误差	SGD	24

预测结果如下图所示：

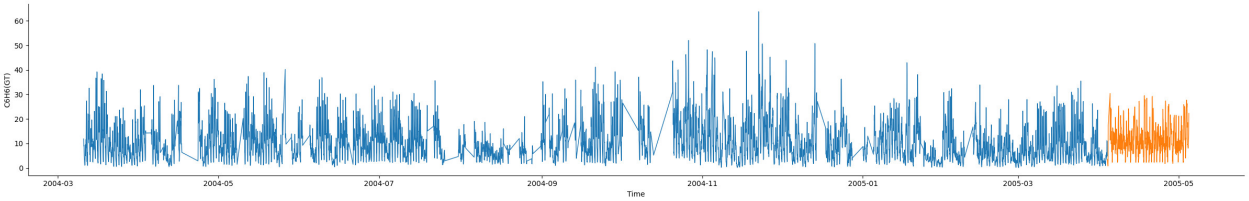

图 6: 苯浓度的后一个月的变化趋势预测

同时，我们对五个污染物之间的相关性以及传感器响应的污染物之间的相关性进行了求解，结果如下：

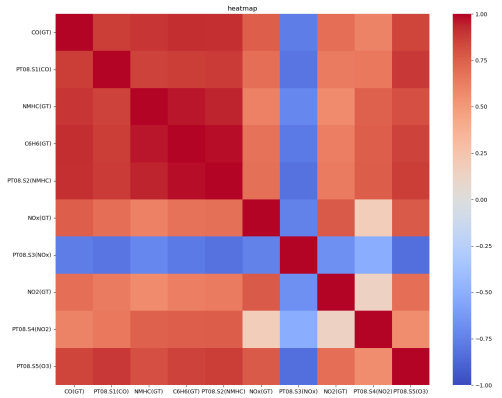

图 7: 污染物及传感器响应的污染物之间的相关性

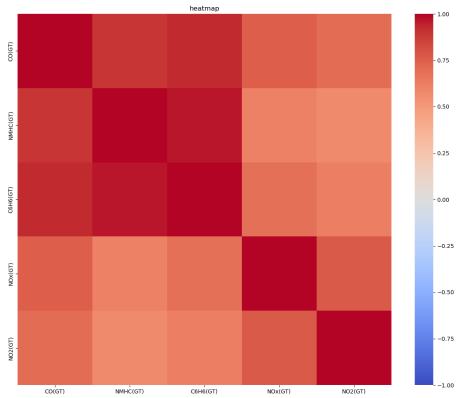

图 8: 污染物之间的相关性

由图可知，五个污染物之间的相关性较强，因此，我们建立的数学模型能应用到其他四个污染物的变化趋势预测中

6.3 问题三模型的建立与求解

6.3.1 城市空气污染指标模型

参考环境空气质量指数（AQI）技术规定 [3]，我们基于一氧化碳 (CO) 等五个污染物数据建立城市空气污染指标模型改模型。空气污染指数为各污染物的空气污染分指数的最大值，能暴露出的是空气质量的最短板。具体模型建立步骤如下:

1. 空气质量分指数计算

空气质量分指数的计算公式如下：

$$IAQI_P = \frac{IAQI_H - IAQI_L}{BP_H - BP_L}(C_P - BP_L) + IAQI_L \tag{23}$$

其中，$IAQI_P$ 是污染物项目 P 的空气质量分指数，C_P 是污染物项目 P 的浓度值，BP_H 是下表中与 C_P 相近的污染物浓度限值的高位值，BP_L 是下表中与 C_P 相近的污染物浓度限值的低位值，$IAQI_H$ 是下表中与 BP_H 对应的空气质量分指数，$IAQI_L$ 是下表中与 BP_L 对应的空气质量分指数。

表 2: 空气质量分指数与污染物浓度对照表

IAQI	**CO** (mg/m^3)	**NMHC** (μg/m^3)	**C$_6$H$_6$** (μg/m^3)	**NOx** (ppb)	**NO$_2$** (μg/m^3)	**O$_3$** (μg/m^3)
0	0	0	0	0	0	0
50	5	250	15	250	100	160
100	10	800	30	500	200	200
150	35	1400	60	1400	700	300
200	60	2000	90	2400	1200	400
300	90	2700	120	4600	2340	800
400	120	3700	150	6000	3090	1000

2. 空气质量指数计算方法空气质量指数按式的计算公式如下：

$$AQI = \max\{IAQI_1, IAQI_2, IAQI_3, \cdots, IAQI_n\} \tag{24}$$

其中，$IAQI$ 为空气质量分指数，n 为污染物项目。

6.3.2 Spearman 秩相关系数

对于研究哪种污染物对温度变化最敏感，我们采取 Spearman 秩相关系数来评估不同污染物对温度变化敏感程度。Spearman 秩相关系数作为一种统计量，用于衡量两个变量之间的相关性，其值在-1 到 1 之间，取值越接近 1，两个变量之间的相关性越高，取值越接近-1，两个变量之间的相关性越低。公式如下：

$$\rho = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(R(x_i) - \overline{R(x)}\right)\left(R(y_i) - \overline{R(y)}\right)}{\sqrt{\left(\frac{1}{n}\sum_{i=1}^{n}\left(R(x_i) - \overline{R(x)}\right)^2\right)\left(\frac{1}{n}\sum_{i=1}^{n}\left(R(y_i) - \overline{R(y)}\right)^2\right)}} \tag{25}$$

其中，ρ 是相关系数，用于衡量变量之间的相关性，n 是样本的数量，$R(x_i)$ 是变量 x 的第 i 个观测值的秩，$R(y_i)$ 是变量 y 的第 i 个观测值的秩，$\overline{R(x)}$ 是变量 x 的秩的平均值，$\overline{R(y)}$ 是变量 y 的秩的平均值。

6.3.3 模型求解

依据空气质量指数计算公式，我们算出了每个时间段对应的空气质量指数，并将其可视化，如下图所示：

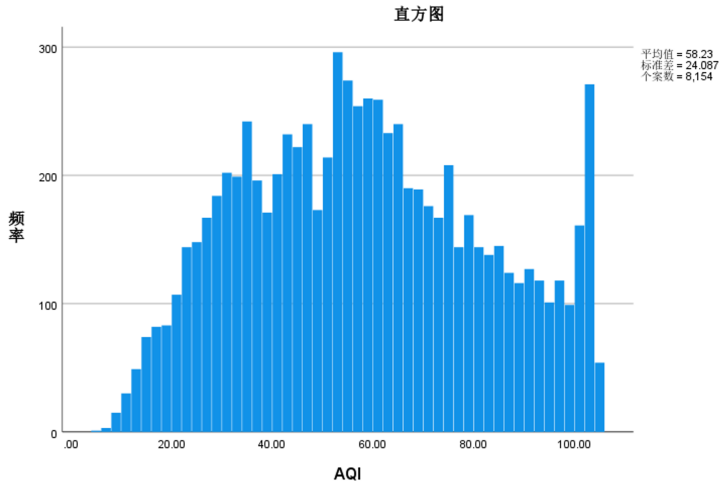

图 9: AQI 的直方图

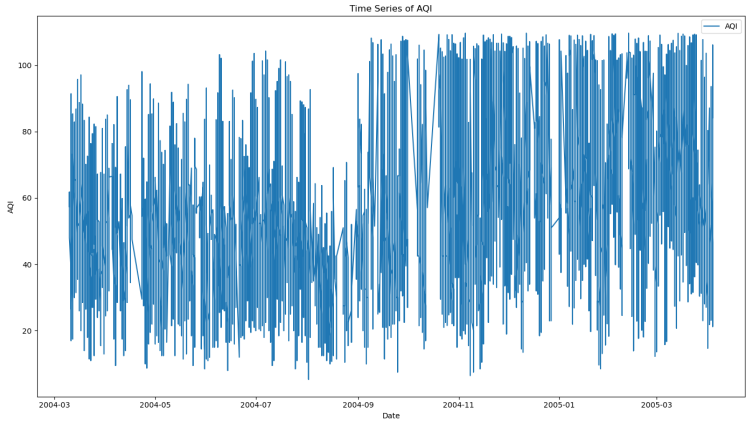

图 10: AQI 时间序列

同时，我们对该指标与温度、绝对湿度与相关湿度之间的年平均斯皮尔曼系数进行了求解，如下表所示：

表 3: 平均相关系数

参数	T AQI	AH AQI	RH AQI
斯皮尔曼系数	0.3580492134511272	0.07559043707700762	-0.31029931447908177

此外，我们还对该指标与温度、绝对湿度与相关湿度之间的日平均斯皮尔曼系数可视化，结果如下：

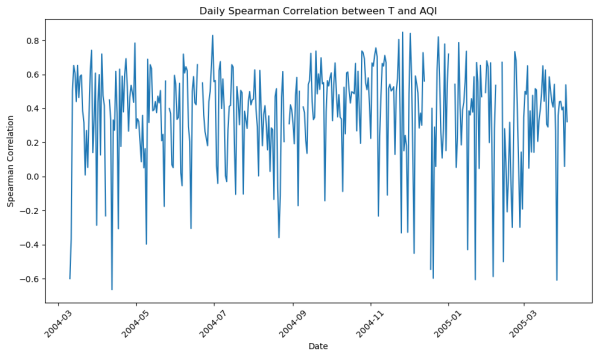

图 11: 温度与 AQI 的日平均 Spearman 系数

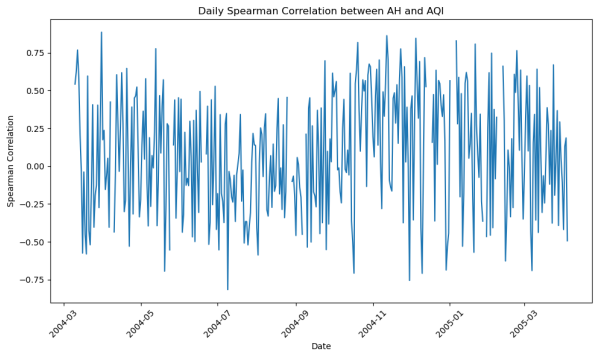

图 12: 绝对湿度与 ADI 的日平均 Spearman 系数

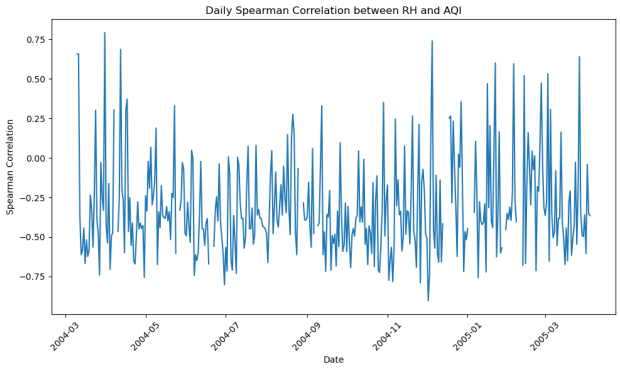

图 13: 相对湿度与 ADI 的日平均 Spearman 系数

根据上述结果，我们可知，温度与 AQI 整体呈中等程度的正相关，但是波动变化大，部分条件下呈负相关，可能与其他环境因素影响了 AQI 有关绝对湿度与 AQI 呈季节性波动变化，总体对空气质量影响较小相对湿度与 AQI 整体呈中等程度的负相关，但是波动变化大，部分呈明显的负相关，可能与其他环境因素影响了 AQI 有关。

对于五个污染物对温度的相关性分析，我们对五个污染物与温度之间的总体斯皮尔曼系数进行求解，如下表所示：

表 4: 五个污染物指标的相关系数

参数	T COGT	T NMHGT	T C6H6GT	T NOxGT	T NO2GT
斯皮尔曼系数	0.082	0.505	0.274	0.253	-0.181

因此，我们可以得知非甲基烃对温度变化最敏感。此外，我们统计了五个污染源都存在时，各污染物对应 IAQI 为 AQI 的频率，结果如下：

经过比较，NOx(GT)_IAQI 和 C6H6(GT)_IAQI 出现的频率较小, 同时我们统计了各 IQAI 出现的总频次及各污染物分指数为污染指数的频率，如下表所示：

表 5: 五个污染物指标的频率

NO2(GT)_IAQI	CO(GT)_IAQI	NMHC(GT)_IAQI	C6H6(GT)_IAQI	NOx(GT)_IAQI
592	139	117	50	3

表 6: 不同气体的频率和总 IAQI 计数

	气体的频率	总 IAQI 计数
NO2(GT)_IAQI	5470	8029
NOx(GT)_IAQI	1571	7862
CO(GT)_IAQI	801	7836
C6H6(GT)_IAQI	243	8995
NMHC(GT)_IAQI	119	915

经分析得知，NO2、NOx 和 NMHC 的分指数对 AQI 的贡献率较高。而 C6H6 虽然缺失率低，但对 IAQI 的贡献率也低，说明苯在该城市的污染程度不高，可以忽略。当 NMHC 缺失时，NOx 的贡献率很高；但 NMHC 不缺失时，NOx 的贡献率几乎为零。这表明 NMHC 可以替代 NOx。对 NMHC 和 NOx 都不缺失的行进行分析。

在分析 NMHC 与 NOx 都不缺失的情况时，计算得出的相关系数为 0.834，表明二者之间的相关度较高。

同时，NO2 与 NMHC 的分指数在大多数情况下都比 NOx 高，对 AQI 的贡献率也大于 NMHC。

根据回归方程：

$$\text{NHMC} = 6.975 + 0.6 \times \text{NOx} \tag{26}$$

可以认为 NHMC 总是大于 NOx，因此可以忽略 NOx。

6.3.4 模型检验

针对上述模型，我们对污染物减少前与减少后的 AQI 进行了 Kolmogorov-Smirnov 检验，其作为一种非参数统计检验，用于比较两个样本的分布或一个样本与参考分布的差异。我们用两样本 KS 检验用比较污染物删减前的 AQI 与污染物删减后的 AQI' 的分布是否显著不同。检验步骤如下：

1. 定义检验统计量

定义检验统计量为

$$D_n = \sup_x |F_n(x) - F(x)|, \tag{27}$$

其中 $F_n(x)$ 为观测序列值，$F(x)$ 为理论序列值或另一观测序列值。

2. 提出假设

提出假设 $H_0 : F_n(x) = F(x)$，即观测序列与理论序列无显著差异。

3. 计算绝对差

计算样本累计频率与理论分布累计概率的绝对差，令最大的绝对差为 D_n，即

$$D_n = \max\{|F_n(x) - F(x)|\}. \tag{28}$$

4. 比较并决策

如果

$$D_n < D_{na}, \tag{29}$$

则认为原假设成立，即观测数据符合理论分布。

5. p 值决策

若 p-value $< \alpha$，则拒绝原假设；若 p-value $\geq \alpha$，则不拒绝原假设。

检验结果如下：

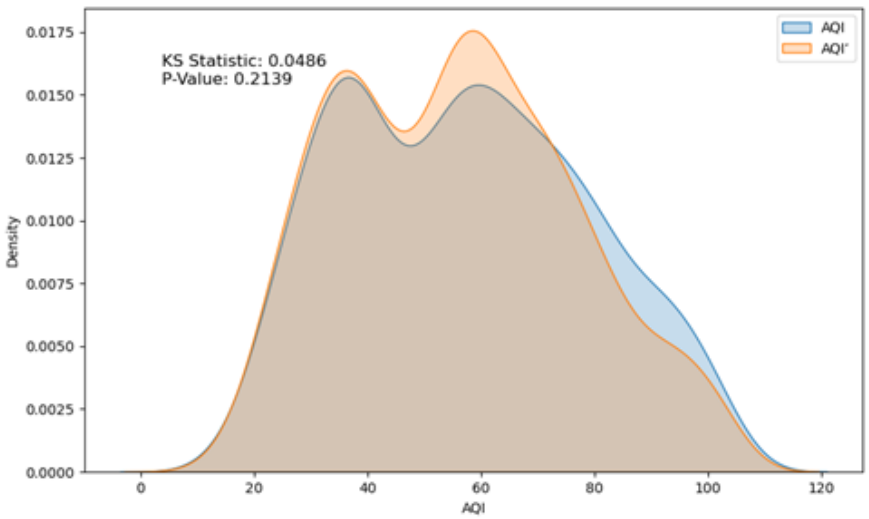

图 14: KS 检验结果

求解得 p=0.2139, 接受假设。

6.4 问题四模型的建立与求解

6.4.1 模型建立

基于第三问的数学模型，我们将将污染物各浓度数据按早中晚进行分类，具体时间划分区间如下:

- 第一区间：0:00-8:00
- 第二区间：8:00-16:00
- 第三区间：16:00-24:00

对这三个时间区间去平均值。其次我们利用 k-means 聚类算法对污染物指标进行分类，将污染物数据分成几个相似的类，通过使用误差平方和（SSE）不断优化结果。

6.4.2 K-means 聚类

为了进一步对污染物数据进行分类，我们采用了 K-means 聚类算法。K-means 聚类算法的步骤如下：

1. 初始化

直接写随机选择 5 个初始聚类中心。

2. 分配：将每个数据点分配给最近的聚类中心

对于数据集中的每个数据点，计算其与每个聚类中心的距离，并将其分配给距离最近的聚类中心。计算公式如下

$$\text{dist}(x, c_i) = \sqrt{\sum_{j=1}^{d}(x_j - c_{ij})^2} \tag{30}$$

其中，x 是数据点，c_i 是第 i 个聚类中心，d 是数据的维度，x_j 和 c_{ij} 分别是 x 和 c_i 在第 j 维上的值。

3. 更新：重新计算每个聚类的中心

重新计算其聚类中心，新的聚类中心是该聚类内所有数据点的均值，计算公式如下：

$$c_i = \frac{1}{|S_i|}\sum_{x \in S_i} x \tag{31}$$

其中，S_i 是第 i 个聚类的数据点集合，$|S_i|$ 是该集合中数据点的数量。

4. 迭代：重复分配和更新步骤，直到满足终止条件

重复步骤 2 和步骤 3，直到聚类中心的位置不再发生显著变化或达到预设的迭代次数为止。

6.4.3 模型求解

我们统计了所有时间段各污染程度所占比重，如下图所示：

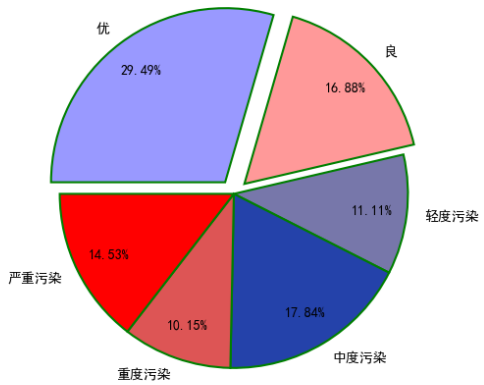

图 15: 所有时间段各污染程度所占比重

同时，将每天的污染数据分为早中晚三个时段，通过 K-means 聚类 [4], 我们求得了不同级别的污染程度，聚类结果如下图所示:

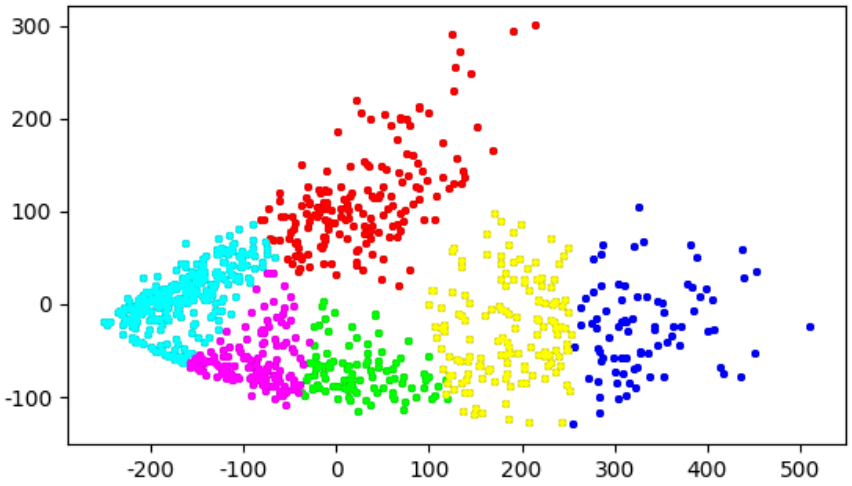

图 16: K-means 聚类结果

按照聚类后的簇 AQI 均值将其环境质量分为优，良，轻度污染，中度污染，重度污染，严重污染。根据不同簇类污染物浓度大小推断出不同污染程度下污染物的特征和主要时间分布。

表 7: 环境质量监测数据

环境质量	AQI	CO(GT)	NMHC(GT)	C6H6(GT)	NOx(GT)	NO2(GT)	时间
优	32.92	0.95	48.05	4.56	89.95	62.13	早
良	49.03	1.71	115.75	8.31	136.17	94.63	早
轻度污染	57.63	1.30	46.78	4.99	210.58	109.65	中
中度污染	67.20	2.92	254.57	15.38	218.07	125.40	中
重度污染	82.03	2.22	124.61	9.40	347.70	154.91	晚
严重污染	92.71	3.48	260.72	16.94	467.69	161.06	晚

据表可知,NOx(GT),NO2(GT) 与污染物程度具有强相关性,对污染程度影响最为严重,NMHC(GT) 与 C6H6(GT) 对污染程度也有影响但影响相对较小，CO(GT) 对污染程度影响较小。

此外，我们统计了各污染物对污染程度的贡献度，线性越强相关性越强。结果如下图所示：

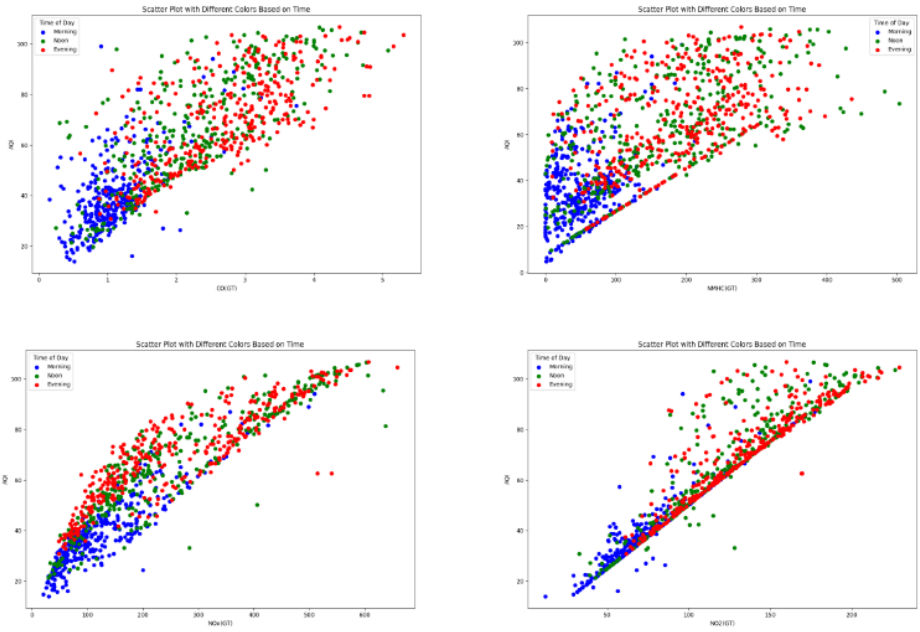

图 17: 各污染物对污染程度的贡献度

6.4.4 污染物浓度变化原因及政策化建议

基于上述模型的求解结果，我们总结了污染物浓度变化的原因以及减少污染物排放的政策化建议。具体原因如下：

1. 早上时段（0 点到 8 点）：此时段污染物浓度最低，环境质量多为优良。主要污染物为一氧化碳（CO）和二氧化氮（NO ）。早上期间人类活动较少，车辆出行和工业活动相对较少，导致污染排放较低。此外，此时大气扩散条件较好，有助于污染物的稀释和扩散。

2. 中午时段（8 点到 16 点）：此时段污染物浓度开始升高，环境质量多为轻度污染和中度污染。主要污染物为二氧化氮（NO ）。随着早高峰的到来，交通活动和工业生产加剧，排放量增加。同时，此时段大气扩散条件较差，导致污染物开始累积。

3. 晚上时段（16 点到 24 点）：此时段污染物浓度达到最高峰，环境质量多为中度污染和严重污染。主要污染物为氮氧化物（NO ，包括 NO 和 NO ）。晚高峰时段交通量巨大，工业活动持续，排放大量污染物。此外，此时段大气稳定，污染物不易扩散，导致浓度累积。

基于上述原因，我们给出了下列政策化建议：

1. 减少私家车使用：推广电动汽车和清洁能源车辆，减少尾气排放。通过政策激励，例如税收优惠和补贴，鼓励市民选择绿色交通工具。

2. 加强工业排放监管：提高对工业企业的排放监管力度，严格执行环保标准，减少工业污染物的排放。鼓励企业采用清洁生产技术，提升环保设备的使用效率。

3. 完善空气质量监测网络：建立并完善覆盖全市的空气质量监测网络，实时监测污染物浓度。利用大数据和物联网技术，提高监测精度和及时性，确保数据的透明和公开。

4. 提高公众环保意识：通过宣传和教育活动，提高市民的环保意识，倡导绿色出行和低碳生活。鼓励社区和学校组织环保活动，培养环保理念，推动社会共同参与环境保护。

通过上述措施的实施，可以有效减少污染物排放，提升城市环境质量，保障居民健康。

七　模型的评价与推广

7.1　模型的评价

7.1.1　问题一的模型的评价

问题一的模型通过线性插值法和正态分布 3 原则、箱型图法进行缺失值补充和异常值替换，确保了数据的完整性和准确性，使用一元线性回归分析苯浓度与一氧化碳浓度的关系，并通过显著性检验验证模型的可靠性。该模型的优点在于处理数据全面且分析方法科学，但其缺点是仅考虑了线性关系，可能忽略了更复杂的非线性关系，且异常值处理方法可能对结果有影响。

7.1.2　问题二的模型的评价

问题二采用了长短期记忆网络（LSTM）模型来预测污染物浓度。LSTM 模型在处理时间序列数据方面表现出色，尤其适用于捕捉污染物浓度变化的复杂模式和长期依赖关系。LSTM 通过引入遗忘门、输入门和输出门的三态门机制，能够有效解决传统循环神经网络（RNN）在处理长时间序列时出现的梯度爆炸和梯度消失问题，这使得 LSTM 可以更好地学习和预测时间序列中的长期依赖关系和模式。相比于传统的自回归综合移动平均模型（ARIMA）和普通的 RNN 模型，LSTM 在处理具有复杂模式的时间序列数据时具有更高的准确性。它可以更好地捕捉污染物浓度随时间变化的非线性关系，从而提高预测的精度和可靠性。总体来说，LSTM 模型在预测污染物浓度方面展示了其强大的性能，能够提供高精度的预测结果，为环境保护和污染控制提供重要的参考依据。

7.1.3　问题三的模型的评价

问题三提出的空气污染浓度指标模型能够有效反映主要污染物的污染程度，提供明确的社会警示作用。该模型可以识别空气污染的最短板，只要有一种污染物的浓度很高，就能通过该指标反映出空气污染的严重性，不易受到低污染物浓度的干扰。同时，该模型综合考虑了多种主要污染物，能够全面反映当地的污染情况。通过集成多种污染物的浓度数据，该模型能够提供更全面和准确的污染评价，为制定环保政策和措施提供科学依据，帮助社会及时应对和管理空气污染问题。

7.1.4　问题四的模型的评价

问题四利用 K-means 聚类算法对污染物数据分类，通过合理的分类和分级，分析不同时间段的污染特征，并提出政策建议。该模型优点在于分类方法有效，结果具有实践指导意义，但聚类结果依赖于初始参数选择，结果可能具有不稳定性。

7.2　模型的推广

7.2.1　问题一的模型的推广

问题一中使用了一元线性回归模型来分析苯浓度与一氧化碳浓度之间的关系。该模型可以推广到其他环境污染物的研究中。具体来说，可以将该模型应用于分析其他气态污染物之间的关系，如臭氧（O3）和氮氧化物（NOx）、颗粒物（PM2.5）和二氧化硫（SO2）等。通过数据预处理、缺失值补充和异常值替换等步骤，确保数据的完整性和准确性，然后使用线性回归模型进行分析。此外，该模型也可以用于其他领域的数据分析，如金融市场中不同股票之间的相关性分析、医学研究中不同生物指标之间的关系研究等。

7.2.2 问题二的模型的推广

问题二中采用了长短期记忆网络（LSTM）模型来预测苯浓度的变化趋势。LSTM 模型在处理时间序列数据方面表现优异，具有广泛的应用前景。除了空气污染物的预测，该模型还可以应用于其他时间序列数据的预测，如股票价格、天气预报、经济指标等。通过适当调整模型参数和训练数据，可以将 LSTM 模型应用于不同的领域，提升预测的准确性和可靠性。

7.2.3 问题三的模型的推广

问题三中建立了基于多种污染物的城市空气污染指标模型，并分析了其与温度、相对湿度和绝对湿度之间的关系。该模型可以推广到其他城市或区域的空气质量分析中。通过收集不同城市的污染物数据，计算空气质量指数（AQI），并分析其与气象因素之间的关系，可以为不同地区的空气质量管理提供科学依据。此外，该模型也可以应用于水质监测、土壤污染分析等环境科学领域，通过建立综合指标模型，全面评估环境质量。

7.2.4 问题四的模型的推广

问题四利用 K-means 聚类算法对污染物数据进行了分类和分级，分析了不同时间段的污染特征，并提出了政策建议。该模型可以推广到其他类型的数据分类和聚类分析中。例如，在客户关系管理中，可以利用 K-means 算法对客户数据进行分类，识别出不同类型的客户，从而制定针对性的营销策略；在医学研究中，可以对患者数据进行聚类，识别出不同疾病类型和特征，从而制定个性化的治疗方案。

参考文献

[1] 吴向莉, 李一格, 靳研, 吴继垣. 基于数据分析的 2019 2020 北京市空气质量影响因素分析 [J]. 统计学与应用, 2022, 11(3): 653-659.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[3] 中华人民共和国环境保护部. (2012-02-29 发布, 2016-01-01 实施). 环境空气质量指数（AQI）技术规定（试行）.

[4] 常彤. (2017). K-means 算法及其改进研究现状. 通讯世界, 19(2).

附录 1 问题一代码

```
#数据预处理
import pandas as pd
from scipy.stats import kstest
import numpy as np

def KsNormDetect(df, column):
    # 计算均值
    u = df[column].mean()
    # 计算标准差
    std = df[column].std()
    # 计算P值
    res = kstest(df[column], 'norm', (u, std))[1]
    # 判断p值是否服从正态分布，p<=0.05 则服从正态分布，否则不服从。
    if res <= 0.05:
        print('该列数据服从正态分布------------')
        print('均值为：%.3f，标准差为：%.3f' % (u, std))
        print('------------------------------')
        return 1
    else:
        return 0


def OutlierDetection(df, column, ks_res):
    # 计算均值
    u = df[column].mean()
    # 计算标准差
    std = df[column].std()

    if ks_res == 1:
        # 定义3 法则识别异常值
        lower_bound = u - 3 * std
        upper_bound = u + 3 * std
        # 将异常值替换为np.nan
        df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
        return df
    elif ks_res == 0:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        # 将异常值替换为np.nan
        df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
        return df
```

```
        else:
            # 如果ks_res既不是0也不是1，直接返回原始数据框
            return df


    def handle_consecutive_minus(series, column_list):
        for column in column_list:
            count = 0 # 用来记录连续-200的个数
            flag = None
            print(len(series[column]))
            for i in range(len(series[column])):
                if series.loc[i, column] == -200:
                    count += 1
                    if count == 1:
                        flag = i
                else:
                    if count >= 3:
                        series.loc[flag:i-1, column] = np.nan
                    elif count != 0:
                        series.loc[flag:i-1, column] = np.nan
                    count = 0
                    flag = None
            df.reset_index(drop=True, inplace=True)
            print(series)

        return series


    def handle_minus(series, column_list):
        for column in column_list:
            count = 0 # 用来记录连续-200的个数
            flag = None
            print(len(series[column]))
            for i in range(len(series[column])):
                if series.loc[i, column] == -200:
                    count += 1
                    if count == 1:
                        flag = i
                else:
                    # if count >= 3:
                        # series.drop(index=range(flag, i), inplace=True) # 删除索引范围内的行
                    if count != 0 and count < 3:
                        series.loc[flag:i-1, column] = np.nan
                    count = 0
                    flag = None
            # 对NaN值进行线性插值
```

```
            series[column] = series[column].interpolate(method='linear')
            df.reset_index(drop=True, inplace=True)
            print(series)


        return series


    def add_data(series):
        count = 0
        flag = 0
        for column in list(series)[2:]:

            for i in range(len(series[column])):
                if pd.isna(series.loc[i, column]):
                    count += 1
                    if count == 1:
                        flag = i
                else:
                    if count != 0 and count <= 6:
                        series.loc[flag-1:i, column] = series.loc[flag-1:i, column].interpolate(method='linear'
                            )
                    count = 0
                    flag = 0
            # 对NaN值进行线性插值
            print(series)
        return series


    # 示例使用
    df = pd.read_excel('./AirQualityUCI.xlsx')
    print(df)
    df = handle_minus(df, list(df)[2:])
    df = handle_consecutive_minus(df, list(df)[2:])
    for column in list(df)[2:]:
        ks_res = KsNormDetect(df, column)
        df = OutlierDetection(df, column, ks_res)
    print(df)
    df = add_data(df)
    df.to_excel('handledData3.xlsx', index=False)



#线性回归填充数据
import pandas as pd
```

```
from scipy.stats import kstest
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import matplotlib.dates as mdates # 导入dates模块

plt.rcParams['font.sans-serif'] = ['SimHei']
df = pd.read_excel('./handledData3.xlsx')
df = df[0:184]
X = df[['C6H6(GT)']] # 自变量，用列名替换'feature'
y = df[['NMHC(GT)']]  # 因变量，用列名替换'target'

# 将X转换为二维数组，因为scikit-learn需要这样的格式
X = X.values.reshape(-1, 1)

# 创建线性回归模型实例
model = LinearRegression()

# 拟合模型
model.fit(X, y)

# 获取模型参数，即斜率和截距
slope = model.coef_[0]
intercept = model.intercept_

# 打印模型参数
print("斜率:", slope)
print("截距:", intercept)

# 绘制数据点
plt.scatter(X, y, s=30, marker='o', facecolors='none', edgecolors='blue', label='数据点')
# 使用模型参数绘制拟合线
X_line = np.array([np.min(X), np.max(X)]).reshape(-1, 1) # 生成拟合线需要的X值范围
y_line = intercept + slope * X_line # 计算对应的y值
plt.plot(X_line, y_line, color='red', label='拟合线', linewidth=3)
#
# 添加图例
plt.legend()

# 设置图表标题和轴标签
plt.title('线性回归分析')
plt.xlabel('C6H6(GT)')
plt.ylabel('NMHC(GT)')

# 显示图表
plt.show()
```

```
182
183 df = pd.read_excel('./handledData3.xlsx')
184 for i in range(185, 9357):
185     if pd.isna(df['NMHC(GT)'][i]):
186         new = df['C6H6(GT)'][i] * slope + intercept;
187         if new < 0: new = 0
188         df['NMHC(GT)'][i] = new
189
190 df.to_excel('handledData4.xlsx', index=False)
191 # ... 您之前的代码 ...
192
193 # 绘制数据点
194 plt.scatter(X, y, s=30, marker='o', facecolors='none', edgecolors='blue', label='原始数据点')
195
196 # 假设您有另一组数据，我们将其命名为X_new和y_new
197 # 您可以这样添加新的散点
198 X_new = df[['C6H6(GT)']][185:9357].values.reshape(-1, 1) # 假设这是新数据的X值
199 y_new = df['NMHC(GT)'][185:9357] # 这是新数据的y值
200
201 # 过滤掉NaN值
202 mask = ~np.isnan(y_new)
203 X_new = X_new[mask]
204 y_new = y_new[mask]
205
206 # 计算新数据的预测值
207 y_pred = intercept + slope * X_new
208
209 # 绘制新数据的散点，使用不同的颜色
210 plt.scatter(X_new, y_new, s=30, marker='o', facecolors='none', edgecolors='green', label='新数据点')
211
212 # 绘制新数据的预测线
213 X_line_new = np.array([np.min(X_new), np.max(X_new)]).reshape(-1, 1)
214 y_line_new = intercept + slope * X_line_new
215
216 # 添加图例
217 plt.legend()
218
219 # 设置图表标题和轴标签
220 plt.title('线性回归分析')
221 plt.xlabel('C6H6(GT)')
222 plt.ylabel('NMHC(GT)')
223
224 # 显示图表
225 plt.show()
226
227
```

```
228
229
230 #寻找线性回归函数
231 import pandas as pd
232 from scipy.stats import kstest
233 import numpy as np
234 from sklearn.linear_model import LinearRegression
235 import matplotlib.pyplot as plt
236 import matplotlib.dates as mdates # 导入dates模块
237 plt.rcParams['font.sans-serif'] = ['SimHei']
238 def KsNormDetect(df, column):
239     # 计算均值
240     u = df[column].mean()
241     # 计算标准差
242     std = df[column].std()
243     # 计算P值
244     res = kstest(df[column], 'norm', (u, std))[1]
245     # 判断p值是否服从正态分布，p<=0.05 则服从正态分布，否则不服从。
246     if res <= 0.05:
247         print('该列数据服从正态分布------------')
248         print('均值为: %.3f, 标准差为: %.3f' % (u, std))
249         print('------------------------------')
250         return 1
251     else:
252         return 0
253
254
255 def OutlierDetection(df, column, ks_res):
256     # 计算均值
257     u = df[column].mean()
258     # 计算标准差
259     std = df[column].std()
260
261     if ks_res == 1:
262         # 定义3 法则识别异常值
263         lower_bound = u - 3 * std
264         upper_bound = u + 3 * std
265         # 将异常值替换为np.nan
266         df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
267         return df
268     elif ks_res == 0:
269         Q1 = df[column].quantile(0.25)
270         Q3 = df[column].quantile(0.75)
271         IQR = Q3 - Q1
272         lower_bound = Q1 - 1.5 * IQR
273         upper_bound = Q3 + 1.5 * IQR
```

```
        # 将异常值替换为np.nan
        df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
        return df
    else:
        # 如果ks_res既不是0也不是1，直接返回原始数据框
        return df


def handle_consecutive_minus_200(series):
    count = 0 # 用来记录连续-200的个数
    flag = 0
    mount = 0
    print(len(series))
    for i in range(len(series)):
        mount += 1
        print(mount)
        print(i)
        if series.loc[i, "CO(GT)"] == -200:
            count += 1
            if count == 1:
                flag = i
        else:
            if count >= 3:
                series.drop(index=range(flag, i), inplace=True) # 删除索引范围内的行
            elif count != 0:
                series.loc[flag:i-1, "CO(GT)"] = np.nan
            count = 0 # 重置计数器
    print("\n\n\n\n")
    count = 0
    for i in range(len(series)):
        mount += 1
        print(mount)
        if i not in series.index:
            continue
        if series.loc[i, "C6H6(GT)"] == -200:
            count += 1
            if count == 1:
                flag = i
        else:
            if count >= 3:
                series.drop(index=range(flag, i), inplace=True) # 删除索引范围内的行
            elif count != 0:
                series.loc[flag:i-1, "C6H6(GT)"] = np.nan
            count = 0 # 重置计数器
    # 对NaN值进行线性插值
    series['CO(GT)'] = series['CO(GT)'].interpolate(method='linear')
```

```
    series['C6H6(GT)'] = series['C6H6(GT)'].interpolate(method='linear')
    return series


df = pd.read_excel('./AirQualityUCI.xlsx')
print(df)
df = handle_consecutive_minus_200(df)
print(df)
for column in list(df)[2:]:
    ks_res = KsNormDetect(df, column)
    df = OutlierDetection(df, column, ks_res)
df['CO(GT)'] = df['CO(GT)'].interpolate(method='linear')
df['C6H6(GT)'] = df['C6H6(GT)'].interpolate(method='linear')
X = df[['CO(GT)']] # 自变量，用列名替换'feature'
y = df[['C6H6(GT)']]  # 因变量，用列名替换'target'
X = X.values.reshape(-1, 1)

# 创建线性回归模型
model = LinearRegression()

# 拟合模型
model.fit(X, y)

# 获取模型参数，即斜率和截距
slope = model.coef_[0]
intercept = model.intercept_

# 打印模型参数
print("斜率:", slope)
print("截距:", intercept)

# 绘制数据点
plt.scatter(X, y, s=30, marker='o', facecolors='none', edgecolors='blue', label='数据点')
# 使用模型参数绘制拟合线
X_line = np.array([np.min(X), np.max(X)]).reshape(-1, 1) # 生成拟合线需要的X值范围
y_line = intercept + slope * X_line # 计算对应的y值
plt.plot(X_line, y_line, color='red', label='拟合线')
# 添加图例
plt.legend()
# 设置图表标题和轴标签
plt.title('线性回归分析')
plt.xlabel('CO(GT)')
plt.ylabel('C6H6(GT)')
# 显示图表
plt.show()
df.to_excel('handledData2.xlsx', index=False)
```

附录 2 问题二代码

```
#LSTM模型与预测可视化
  import pandas as pd
  import numpy as np
  from keras.models import Sequential
  from keras.layers import LSTM, Dense
  from sklearn.preprocessing import MinMaxScaler
  from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
  from math import sqrt
  import matplotlib.pyplot as plt
  from datetime import datetime, timedelta
  import csv
  from scipy.stats import kstest
  import seaborn as sns
  from scipy import stats as scs
  plt.rcParams['font.sans-serif'] = ['SimHei']
  plt.rcParams['axes.unicode_minus'] = False


  # 创建数据集
  def create_dataset(dataset, look_back=1):
      X, Y = [], []
      for i in range(len(dataset) - look_back - 1):
          a = dataset.iloc[i:(i + look_back)][['Time', 'NMHC(GT)']].values
          X.append(a)
          Y.append(dataset.iloc[i + look_back]['NMHC(GT)'])
      print(len(X))
      print(len(Y))
      return np.array(X), np.array(Y)


  # 构建并训练模型
  def build_and_train_model(data_center, look_back=168):
      X, Y = create_dataset(data_center, look_back)
      model = Sequential()
      model.add(LSTM(40, input_shape=(look_back, 2), dropout=0.2))
      model.add(Dense(10))
      model.compile(optimizer='Adam', loss='mean_squared_error')
      model.fit(X, Y, epochs=400, batch_size=1, verbose=2)
      return model


  # 进行未来30天每小时预测
  def predict_next_30_days(model, last_24_hours, scaler, look_back=24):
      predictions = []
```

```
        current_batch = last_24_hours
        for i in range(30 * 24): # 30天每天24小时
            current_pred = model.predict(current_batch)[0][0] # 获取模型预测的单个值
            predictions.append(current_pred)
            next_hour = (current_batch[0, -1, 0] + 1) % 24 # 更新小时
            # 确保新的行是正确类型
            new_row = np.array([[next_hour, current_pred]], dtype=np.float32).reshape(1, 2)
            # 更新批次数据
            current_batch = np.append(current_batch[:, 1:, :], new_row.reshape(1, 1, 2), axis=1)
        # 反归一化预测结果
        inversed_predictions = scaler.inverse_transform(np.array(predictions).reshape(-1, 1))
        # 将预测<0的数据置为0
        return np.maximum(inversed_predictions, 0)

    # 主逻辑
    results_csv_path = '结果表.csv'
    with open(results_csv_path, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(['Date', 'Time', 'NMHC(GT)'])
        df = pd.read_excel('./handledData4.xlsx')
        print(df)
        df = df[0:9355]
        df['Time'] = df['Time'].str.slice(start=0, stop=2).astype(int)
        print(df)
        df['日期时间'] = df['Date'].astype(str) + '-' + df['Time'].astype(str)
        df['日期时间'] = pd.to_datetime(df['日期时间'], errors='coerce')
        df.rename(columns={'日期时间': '日期小时'}, inplace=True)
        df.set_index('日期小时', inplace=True)
        scaler = MinMaxScaler(feature_range=(0, 1))
        df['NMHC(GT)'] = scaler.fit_transform(df[['NMHC(GT)']])
        print(f"Processing ...")
        data_center = df[['Date', 'Time', 'NMHC(GT)']]
        print(data_center)
        model = build_and_train_model(data_center, look_back=168)
        last_24_hours = data_center.tail(24)[['Time', 'NMHC(GT)']].values.reshape(1, 24, 2)
        predictions = predict_next_30_days(model, last_24_hours, scaler)
        start_date = data_center['Date'].max() + timedelta(days=1)
        for i in range(30 * 24):
            predict_date = start_date + timedelta(hours=i)
            writer.writerow([predict_date.strftime('%Y-%m-%d'), predict_date.hour, predictions[i][0]])
    print("预测结果已保存到", results_csv_path)

    df = pd.read_excel('./handledData4.xlsx')
    df = df[0:9355]
    df['Time'] = df['Time'].str.slice(start=0, stop=2).astype(int)
    df['Date'] = pd.to_datetime(df['Date'])
```

```
data2_path = "结果表.csv"
data2 = pd.read_csv(data2_path, encoding='GB2312')
data2['Date'] = data2['Date'].astype(str)
data2['Time'] = data2['Time'].astype(int)
data2['Date'] = pd.to_datetime(data2['Date'], errors='coerce')
# 合并原始及预测数据
df = pd.concat([df, data2], ignore_index=True)
df.sort_values(['Date', 'Time'], inplace=True)
df['Date'] = pd.to_datetime(df['Date'])
df = df.sort_values(by=['Date', 'Time'])

df['Time'] = df['Time'].astype(str).str.pad(side='left', fillchar='0', width=2)

# 合并日期和小时
df['日期时间'] = df['Date'].astype(str) + ' ' + df['Time']
df['日期时间'] = pd.to_datetime(df['日期时间'])
df['日期时间'] = pd.to_datetime(df['日期时间'], errors='coerce')
df.drop(['Date', 'Time'], axis=1, inplace=True)
df.reset_index(drop=True, inplace=True)
df.rename(columns={'日期时间': '日期小时'}, inplace=True)
plt.figure(figsize=(30, 20))
tem = pd.DataFrame()
# 确保'日期时间'列是日期时间类型
df['日期小时'] = pd.to_datetime(df['日期小时'])
# 筛选出12月之前的数据
pre_december_data = df[df['日期小时'] < datetime(2005, 4, 4)].copy()
pre_december_data['颜色'] = 'red' # 为12月前的数据指定颜色
# 筛选出12月之后的数据
post_december_data = df[df['日期小时'] >= datetime(2005, 4, 4)].copy()
post_december_data['颜色'] = 'blue' # 为12月后的数据指定颜色
# 将两部分数据合并，并添加到tem中
tem = pd.concat([tem, pre_december_data, post_december_data], ignore_index=True)
# 创建 FacetGrid 对象
g = sns.FacetGrid(data=tem, hue="颜色", height=4, aspect=6, sharex=False, sharey=False)
# 绘制线图
g = g.map(sns.lineplot, "日期小时", "NMHC(GT)", palette="Set1", lw=1)
# sns.lineplot(data=tem, x='日期小时', y='NMHC(GT)', palette="Set1", lw=2.5)
# 设置x轴标签
plt.xlabel('Time')
# 显示图形
plt.show()
```

附录 3 问题三代码

```
#计算AQI
import pandas as pd
```

```
import numpy as np

# 读取Excel文件
file_path = './handledData4.xlsx'
df = pd.read_excel(file_path)

# 删除不需要分析的列
columns_to_drop = ['Unnamed: 15', 'Unnamed: 16']
df.drop(columns=columns_to_drop, errors='ignore', inplace=True)

# 将剩余的列转换为数值类型
for col in df.columns:
    if col == 'Date' or col == 'Time':
        continue
    df[col] = pd.to_numeric(df[col], errors='coerce')

# 计算分指数（IAQI）
def calculate_IAQI(value, pollutant):
    if pollutant == 'CO(GT)':
        if 0 <= value < 2:
            return ((50 - 0) / (2 - 0)) * (value - 0) + 0
        elif 2 <= value < 10:
            return ((100 - 51) / (10 - 2)) * (value - 2) + 51
        elif 10 <= value < 35:
            return ((150 - 101) / (35 - 10)) * (value - 10) + 101
        elif 35 <= value < 60:
            return ((200 - 151) / (60 - 35)) * (value - 35) + 151
        elif 60 <= value < 90:
            return ((300 - 201) / (90 - 60)) * (value - 60) + 201
        elif 90 <= value < 120:
            return ((400 - 301) / (120 - 90)) * (value - 90) + 301
        else:
            return 0

    elif pollutant == 'C6H6(GT)':
        if 0 <= value < 15:
            return ((50 - 0) / (15 - 0)) * (value - 0) + 0
        elif 15 <= value < 30:
            return ((100 - 51) / (30 - 15)) * (value - 15) + 51
        elif 30 <= value < 60:
            return ((150 - 101) / (60 - 30)) * (value - 30) + 101
        elif 60 <= value < 90:
            return ((200 - 151) / (90 - 60)) * (value - 60) + 151
        elif 90 <= value < 120:
            return ((300 - 201) / (120 - 90)) * (value - 90) + 201
        elif 120 <= value < 150:
```

```
            return ((400 - 301) / (150 - 120)) * (value - 120) + 301
        else:
            return 0

    elif pollutant == 'NOx(GT)':
        if 0 <= value < 250:
            return ((50 - 0) / (250 - 0)) * (value - 0) + 0
        elif 250 <= value < 500:
            return ((100 - 51) / (500 - 250)) * (value - 250) + 51
        elif 500 <= value < 1400:
            return ((150 - 101) / (1400 - 500)) * (value - 500) + 101
        elif 1400 <= value < 2400:
            return ((200 - 151) / (2400 - 1400)) * (value - 1400) + 151
        elif 2400 <= value < 4600:
            return ((300 - 201) / (4600 - 2400)) * (value - 2400) + 201
        elif 4600 <= value < 6000:
            return ((400 - 301) / (6000 - 4600)) * (value - 4600) + 301
        else:
            return 0

    elif pollutant == 'NO2(GT)':
        if 0 <= value < 100:
            return ((50 - 0) / (100 - 0)) * (value - 0) + 0
        elif 100 <= value < 200:
            return ((100 - 51) / (200 - 100)) * (value - 100) + 51
        elif 200 <= value < 700:
            return ((150 - 101) / (700 - 200)) * (value - 200) + 101
        elif 700 <= value < 1200:
            return ((200 - 151) / (1200 - 700)) * (value - 700) + 151
        elif 1200 <= value < 2340:
            return ((300 - 201) / (2340 - 1200)) * (value - 1200) + 201
        elif 2340 <= value < 3090:
            return ((400 - 301) / (3090 - 2340)) * (value - 2340) + 301
        else:
            return 0

    elif pollutant == 'NMHC(GT)':
        if 0 <= value < 250:
            return ((50 - 0) / (250 - 0)) * (value - 0) + 0
        elif 250 <= value < 800:
            return ((100 - 51) / (800 - 250)) * (value - 250) + 51
        elif 800 <= value < 1400:
            return ((150 - 101) / (1400 - 800)) * (value - 800) + 101
        elif 1400 <= value < 2000:
            return ((200 - 151) / (2000 - 1400)) * (value - 1400) + 151
        elif 2000 <= value < 2700:
```

```
            return ((300 - 201) / (2700 - 2000)) * (value - 2000) + 201
        elif 2700 <= value < 3700:
            return ((400 - 301) / (3700 - 2700)) * (value - 2700) + 301
        else:
            return 0

    else:
        return np.nan

# 逐行计算每个污染物的IAQI值
for pollutant in ['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)', 'NMHC(GT)']:
    iaqi_col = pollutant + '_IAQI'
    df[iaqi_col] = df[pollutant].apply(lambda x: calculate_IAQI(x, pollutant) if pd.notna(x) else np.nan)

# 计算AQI
def calculate_AQI(row):
    # 获取所有IAQI值
    iaqis = row[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI', 'NO2(GT)_IAQI', 'NMHC(GT)_IAQI']]
    # 计算有效IAQI的数量
    valid_iaqis = iaqis.dropna()
    if len(valid_iaqis) >= 2:
        return valid_iaqis.max()
    else:
        return np.nan

df['AQI'] = df.apply(calculate_AQI, axis=1)
# 统计每个污染物的分指数在AQI中成为最大值的频率，忽略缺失值的行
contribution = {
    'CO(GT)_IAQI': ((df['CO(GT)_IAQI'] == df['AQI']) & (~df[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI', '
        NO2(GT)_IAQI', 'NMHC(GT)_IAQI']].isnull().any(axis=1))).sum(),
    'C6H6(GT)_IAQI': ((df['C6H6(GT)_IAQI'] == df['AQI']) & (~df[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI
        ', 'NO2(GT)_IAQI', 'NMHC(GT)_IAQI']].isnull().any(axis=1))).sum(),
    'NOx(GT)_IAQI': ((df['NOx(GT)_IAQI'] == df['AQI']) & (~df[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI',
         'NO2(GT)_IAQI', 'NMHC(GT)_IAQI']].isnull().any(axis=1))).sum(),
    'NO2(GT)_IAQI': ((df['NO2(GT)_IAQI'] == df['AQI']) & (~df[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI',
         'NO2(GT)_IAQI', 'NMHC(GT)_IAQI']].isnull().any(axis=1))).sum(),
    'NMHC(GT)_IAQI': ((df['NMHC(GT)_IAQI'] == df['AQI']) & (~df[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI
        ', 'NO2(GT)_IAQI', 'NMHC(GT)_IAQI']].isnull().any(axis=1))).sum(),
}

# 转换为DataFrame并排序
contribution_df = pd.DataFrame.from_dict(contribution, orient='index', columns=['Frequency'])
contribution_df = contribution_df.sort_values(by='Frequency', ascending=False)

# 打印贡献度排序
print(contribution_df)
```

```
136
137 # 保存结果到Excel文件
138 df.to_excel('./handledData.xlsx', index=False)
139
140 #污染物与AQI相关图绘制
141 import numpy as np
142 import pandas as pd
143 from matplotlib import pyplot as plt
144
145 df = pd.read_excel('./handledData.xlsx')
146 df.drop(columns=['PT08.S1(CO)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)',
147                  'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH',
148                  'CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI',
149                  'NO2(GT)_IAQI', 'NMHC(GT)_IAQI'],inplace=True)
150
151 df['Time'] = df['Time'].str.slice(start=0, stop=2).astype(int)
152 df = df[7:]
153 df.reset_index(drop=True, inplace=True)
154 print(df)
155
156 columns = ['Time', 'CO(GT)', 'NMHC(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)', 'AQI']
157 df2 = pd.DataFrame(columns=columns)
158 flag = 0
159 count = 0
160 for index, row in df.iterrows():
161     if row['Time'] % 8 == 0:
162         average = df.iloc[index-7:index].mean(skipna=True)
163         df2.loc[len(df2.index)] = average
164 print(df2)
165 df2.to_excel('./question4.xlsx')
166 plt.figure(figsize=(12, 8))
167 color_map = {
168     4: 'blue',  # 假设4代表早上，用蓝色表示
169     12: 'green', # 假设12代表中午，用绿色表示
170     20: 'red'   # 假设20代表晚上，用红色表示
171 }
172
173 # 应用颜色映射到DataFrame
174 df2['Color'] = df2['Time'].map(color_map)
175 count = 0
176 # 根据颜色分组数据，并为每组数据绘制散点图，同时指定标签
177 for time, group in df2.groupby('Time'):
178     print(time)
179     if count == 0:
180         plt.scatter(group['NO2(GT)'], group['AQI'], c=color_map[time], label='Morning')
181         count += 1
```

```
    elif count == 1:
        plt.scatter(group['NO2(GT)'], group['AQI'], c=color_map[time], label='Noon')
        count += 1
    elif count == 2:
        plt.scatter(group['NO2(GT)'], group['AQI'], c=color_map[time], label='Evening')
        count += 1
    # 注意：这里我们使用f'{time} ({time//4})' 作为标签，其中time//4是将时间转换为小时

# 添加图例，设置标题
plt.legend(title='Time of Day')

# 添加标题和标签
plt.title('Scatter Plot with Different Colors Based on Time')
plt.xlabel('NO2(GT)')
plt.ylabel('AQI')

# 显示图形
plt.show()
```

附录 4 问题四代码

```
    #聚类及其可视化
    import time

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from numpy import nonzero, array
from sklearn.cluster import KMeans
from sklearn.metrics import f1_score, accuracy_score, normalized_mutual_info_score, rand_score,
    adjusted_rand_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA

df = pd.read_excel('./question4.xlsx') # 设置要读取的数据集
df.dropna(inplace=True)
# print(df)

columns = list(df.columns) # 获取数据集的第一行，第一行通常为特征名，所以先取出
features = columns[:len(columns)][1:] # 数据集的特征名（去除了最后一列，因为最后一列存放的是标签，不是数据）
print(features)
dataset = df[features] # 预处理之后的数据，去除掉了第一行的数据（因为其为特征名，如果数据第一行不是特征名，可跳过
    这一步）
# scaler = StandardScaler()
# dataset = scaler.fit_transform(dataset)
print(dataset)
```

```
attributes = len(df.columns) - 1 # 属性数量（数据集维度）
original_labels = list(df[columns[-1]]) # 原始标签



def initialize_centroids(data, k):
    # 从数据集中随机选择k个点作为初始质心
    centers = data[np.random.choice(data.shape[0], k, replace=False)]
    return centers


def get_clusters(data, centroids):
    # 计算数据点与质心之间的距离，并将数据点分配给最近的质心
    distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
    cluster_labels = np.argmin(distances, axis=1)
    print(cluster_labels)
    return cluster_labels


def update_centroids(data, cluster_labels, k):
    # 计算每个簇的新质心，即簇内数据点的均值
    new_centroids = np.array([data[cluster_labels == i].mean(axis=0) for i in range(k)])
    return new_centroids


def k_means(data, k, T, epsilon):
    start = time.time() # 开始时间，计时
    # 初始化质心
    centroids = initialize_centroids(data, k)
    t = 0
    list = [[], [], [], [], [], []]
    while t <= T:
        # 分配簇
        cluster_labels = get_clusters(data, centroids)

        # 更新质心
        new_centroids = update_centroids(data, cluster_labels, k)

        # 检查收敛条件
        if np.linalg.norm(new_centroids - centroids) < epsilon:
            for i in range(len(cluster_labels)):
                if cluster_labels[i] == 0:
                    list[0].append(i)
                elif cluster_labels[i] == 1:
                    list[1].append(i)
                elif cluster_labels[i] == 2:
```

```
                list[2].append(i)
            elif cluster_labels[i] == 3:
                list[3].append(i)
            elif cluster_labels[i] == 4:
                list[4].append(i)
            elif cluster_labels[i] == 5:
                list[5].append(i)
        # centroids = scalar.inverse_transform(centroids)
        print(centroids)
        break
    centroids = new_centroids
    # centroids = scalar.inverse_transform(centroids)
    print("第", t, "次迭代")
    t += 1
print("用时: {0}".format(time.time() - start))
return cluster_labels, centroids, list


# 计算聚类指标
def clustering_indicators(labels_true, labels_pred):
    if type(labels_true[0]) != int:
        labels_true = LabelEncoder().fit_transform(df[columns[len(columns) - 1]]) # 如果数据集的标签为文本类型, 把文本标签转换为数字标签
    f_measure = f1_score(labels_true, labels_pred, average='macro') # F值
    accuracy = accuracy_score(labels_true, labels_pred) # ACC
    normalized_mutual_information = normalized_mutual_info_score(labels_true, labels_pred) # NMI
    rand_index = rand_score(labels_true, labels_pred) # RI
    ARI = adjusted_rand_score(labels_true, labels_pred)
    return f_measure, accuracy, normalized_mutual_information, rand_index, ARI


# 绘制聚类结果散点图
def draw_cluster(dataset, centers, labels):
    center_array = array(centers)
    if attributes > 2:
        dataset = PCA(n_components=2).fit_transform(dataset) # 如果属性数量大于2, 降维
        center_array = PCA(n_components=2).fit_transform(center_array) # 如果属性数量大于2, 降维
    else:
        dataset = array(dataset)
    # 做散点图
    label = array(labels)
    plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c='black', s=7) # 原图
    # plt.show()
    colors = np.array(
        ["#FF0000", "#0000FF", "#00FF00", "#FFFF00", "#00FFFF", "#FF00FF", "#800000", "#008000", "#000080",
            "#808000",
```

```
        "#800080", "#008080", "#444444", "#FFD700", "#008080"])
    # 循换打印k个簇，每个簇使用不同的颜色
    for i in range(k):
        plt.scatter(dataset[nonzero(label == i), 0], dataset[nonzero(label == i), 1], c=colors[i], s=7, marker=
            'o')
    # plt.scatter(center_array[:, 0], center_array[:, 1], marker='x', color='m', s=30) # 聚类中心
    plt.show()

if __name__ == "__main__":
    k = 6 # 聚类簇数
    T = 1000 # 最大迭代数
    n = len(dataset) # 样本数
    epsilon = 1e-5
    # 预测全部数据
    labels, centers, list = k_means(np.array(dataset), k, T, epsilon)
    # print(labels)
    # F_measure, ACC, NMI, RI, ARI = clustering_indicators(original_labels, labels) # 计算聚类指标
    # print("F_measure:", F_measure, "ACC:", ACC, "NMI", NMI, "RI", RI, "ARI", ARI)
    # print(membership)
    print(centers)
    print(list[0])
    print(list[1])
    print(list[2])
    print(list[3])
    print(list[4])
    print(list[5])
    plt.rcParams['axes.unicode_minus'] = False # 坐标轴负号的处理
    plt.axes(aspect='equal') # 将横、纵坐标轴标准化处理，确保饼图是一个正圆，否则为椭圆

    print(dataset)
    draw_cluster(dataset, centers, labels=labels)

#污染物程度占比绘制
import time

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from numpy import nonzero, array
from sklearn.cluster import KMeans
from sklearn.metrics import f1_score, accuracy_score, normalized_mutual_info_score, rand_score,
    adjusted_rand_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA

df = pd.read_excel('./question4.xlsx') # 设置要读取的数据集
```

```
158 df.dropna(inplace=True)
159 # print(df)
160
161 columns = list(df.columns) # 获取数据集的第一行，第一行通常为特征名，所以先取出
162 features = columns[:len(columns)][1:] # 数据集的特征名（去除了最后一列，因为最后一列存放的是标签，不是数据）
163 print(features)
164 dataset = df[features] # 预处理之后的数据，去除掉了第一行的数据（因为其为特征名，如果数据第一行不是特征名，可跳过
        这一步）
165 # scaler = StandardScaler()
166 # dataset = scaler.fit_transform(dataset)
167 print(dataset)
168 attributes = len(df.columns) - 1 # 属性数量（数据集维度）
169 original_labels = list(df[columns[-1]]) # 原始标签
170
171
172
173 def initialize_centroids(data, k):
174     # 从数据集中随机选择k个点作为初始质心
175     centers = data[np.random.choice(data.shape[0], k, replace=False)]
176     return centers
177
178
179 def get_clusters(data, centroids):
180     # 计算数据点与质心之间的距离，并将数据点分配给最近的质心
181     distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
182     cluster_labels = np.argmin(distances, axis=1)
183     print(cluster_labels)
184     return cluster_labels
185
186
187 def update_centroids(data, cluster_labels, k):
188     # 计算每个簇的新质心，即簇内数据点的均值
189     new_centroids = np.array([data[cluster_labels == i].mean(axis=0) for i in range(k)])
190     return new_centroids
191
192
193 def k_means(data, k, T, epsilon):
194     start = time.time() # 开始时间，计时
195     # 初始化质心
196     centroids = initialize_centroids(data, k)
197     t = 0
198     list = [[], [], [], [], [], []]
199     while t <= T:
200         # 分配簇
201         cluster_labels = get_clusters(data, centroids)
202
```

```
        # 更新质心
        new_centroids = update_centroids(data, cluster_labels, k)

        # 检查收敛条件
        if np.linalg.norm(new_centroids - centroids) < epsilon:
            for i in range(len(cluster_labels)):
                if cluster_labels[i] == 0:
                    list[0].append(i)
                elif cluster_labels[i] == 1:
                    list[1].append(i)
                elif cluster_labels[i] == 2:
                    list[2].append(i)
                elif cluster_labels[i] == 3:
                    list[3].append(i)
                elif cluster_labels[i] == 4:
                    list[4].append(i)
                elif cluster_labels[i] == 5:
                    list[5].append(i)
            # centroids = scalar.inverse_transform(centroids)
            print(centroids)
            break
        centroids = new_centroids
        # centroids = scalar.inverse_transform(centroids)
        print("第", t, "次迭代")
        t += 1
    print("用时: {0}".format(time.time() - start))
    return cluster_labels, centroids, list


# 计算聚类指标
def clustering_indicators(labels_true, labels_pred):
    if type(labels_true[0]) != int:
        labels_true = LabelEncoder().fit_transform(df[columns[len(columns) - 1]]) # 如果数据集的标签为文本类型，
            把文本标签转换为数字标签
    f_measure = f1_score(labels_true, labels_pred, average='macro') # F值
    accuracy = accuracy_score(labels_true, labels_pred) # ACC
    normalized_mutual_information = normalized_mutual_info_score(labels_true, labels_pred) # NMI
    rand_index = rand_score(labels_true, labels_pred) # RI
    ARI = adjusted_rand_score(labels_true, labels_pred)
    return f_measure, accuracy, normalized_mutual_information, rand_index, ARI


# 绘制聚类结果散点图
def draw_cluster(dataset, centers, labels):
    center_array = array(centers)
    if attributes > 2:
```

```
248         dataset = PCA(n_components=2).fit_transform(dataset) # 如果属性数量大于2，降维
249         center_array = PCA(n_components=2).fit_transform(center_array) # 如果属性数量大于2，降维
250     else:
251         dataset = array(dataset)
252     # 做散点图
253     label = array(labels)
254     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c='black', s=7) # 原图
255     # plt.show()
256     colors = np.array(
257         ["#FF0000", "#0000FF", "#00FF00", "#FFFF00", "#00FFFF", "#FF00FF", "#800000", "#008000", "#000080",
                "#808000",
258          "#800080", "#008080", "#444444", "#FFD700", "#008080"])
259     # 循换打印k个簇，每个簇使用不同的颜色
260     for i in range(k):
261         plt.scatter(dataset[nonzero(label == i), 0], dataset[nonzero(label == i), 1], c=colors[i], s=7, marker=
                'o')
262     # plt.scatter(center_array[:, 0], center_array[:, 1], marker='x', color='m', s=30) # 聚类中心
263     plt.show()
264
265 if __name__ == "__main__":
266     k = 6 # 聚类簇数
267     T = 1000 # 最大迭代数
268     n = len(dataset) # 样本数
269     epsilon = 1e-5
270     # 预测全部数据
271     labels, centers, list = k_means(np.array(dataset), k, T, epsilon)
272     # print(labels)
273     # F_measure, ACC, NMI, RI, ARI = clustering_indicators(original_labels, labels) # 计算聚类指标
274     # print("F_measure:", F_measure, "ACC:", ACC, "NMI", NMI, "RI", RI, "ARI", ARI)
275     # print(membership)
276     print(centers)
277     print(list[0])
278     print(list[1])
279     print(list[2])
280     print(list[3])
281     print(list[4])
282     print(list[5])
283     plt.rcParams['axes.unicode_minus'] = False # 坐标轴负号的处理
284     plt.axes(aspect='equal') # 将横、纵坐标轴标准化处理，确保饼图是一个正圆，否则为椭圆
285
286     # length = len(dataset)
287     # print(length)
288     # edu = [len(list[0]) / length, len(list[1]) / length, len(list[2]) / length, len(list[3]) / length, len(
            list[4]) / length]
289     # labels = ['batter', 'great', 'middle', 'bad', 'worse']
290     # explode = [0, 0.1, 0, 0, 0] # 生成数据，用于凸显大专学历人群
```

```
    # colors = ['#9999ff', '#ff9999', '#7777aa', '#2442aa', '#dd5555'] # 自定义颜色
    #
    # plt.pie(x=edu, # 绘图数据
    #         explode=explode, # 指定饼图某些部分的突出显示，即呈现爆炸式
    #         labels=labels, # 添加教育水平标签
    #         colors=colors,
    #         autopct='%.2f%%', # 设置百分比的格式，这里保留两位小数
    #         pctdistance=0.8, # 设置百分比标签与圆心的距离
    #         labeldistance=1.1, # 设置教育水平标签与圆心的距离
    #         startangle=180, # 设置饼图的初始角度
    #         radius=1.2, # 设置饼图的半径
    #         counterclock=False, # 是否逆时针，这里设置为顺时针方向
    #         wedgeprops={'linewidth': 1.5, 'edgecolor': 'green'}, # 设置饼图内外边界的属性值
    #         textprops={'fontsize': 10, 'color': 'black'}, # 设置文本标签的属性值
    #         )
    #
    # # 添加图标题
    # # 显示图形
    # plt.show()
    print(dataset)
    draw_cluster(dataset, centers, labels=labels)
```