

# 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,本文利用一元线性回归、LSTM、Spearman、以及 K-means 等模型对一氧化碳(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  $3\sigma$  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  $3\sigma$  原则,反之采用箱型图,我们共计处理了异常数据 18418 例。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,我们利用线性插值方程求解各污染物的空气质量分指数  $IAQI_P$ , 污染物指数  $AQI$  为各分指数的最大值。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,我们使用过滤法对模型的特征进行分析,在删去  $C_6H_6O_2$  与 NHMC 这两种污染物后不影响空气污染指标模型。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,发现凌晨、上午、下午以及晚上的时间间隔规律变化明显作为分类时间间隔。将一天分为四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四级,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,并且凌晨、上午、下午以及晚上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  $3\sigma$  原则和箱型图法。 $3\sigma$  原则指出，正态分布数据中 99.7% 的数据位于均值的  $\pm 3$  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，我们将污染物分为凌晨、上午、下午以及晚上四类，对一天中这四个时间区取平均值。然后利用 k-means 算法将污染物指标分成四类，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	$mg/m^3$
$C_6H_6$	苯浓度	$\mu g/m^3$
$NO_x$	氮氧化物浓度	ppb
$NO_2$	二氧化氮浓度	$\mu g/m^3$
$O_3$	臭氧浓度	$\mu g/m^3$
$IAQIP$	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	$^{\circ}C$
$RH$	相对湿度	%
$AH$	绝对湿度	$g/m^3$
$\beta_0, \beta_1$	线性回归模型参数	无
$\sigma$	标准差	无
$\mu$	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：

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

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：

$$y = \beta_0 + \beta_1 x + \epsilon \quad (2)$$

其中， $y$  是因变量， $x$  是自变量， $\beta_0$  是截距， $\beta_1$  是回归系数， $\epsilon$  是误差项。

以补充大面积缺失的 NMHC(GT) 为例，结果如下图：

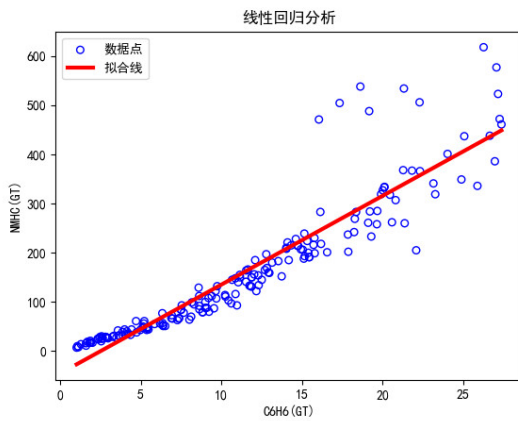


图 1：对 C6H6 与 NMHC 的线性回归分析

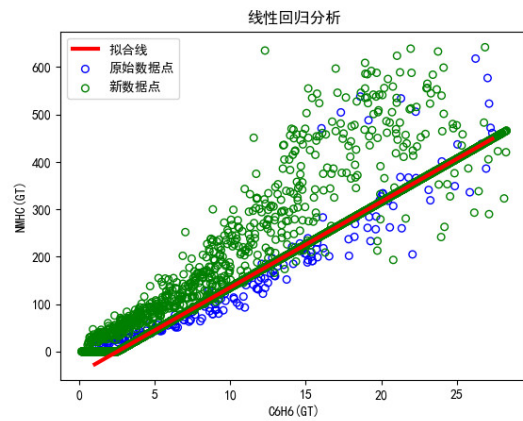


图 2：新增数据与线性回归分析

## 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  $3\sigma$  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  $3\sigma$  原则处理数据。
- $p$  值  $\leq 0.05$ : 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 $3\sigma$ 原则

正态分布  $3\sigma$  原则是一种非常重要的概率分布，其中的一个重要性质是 68-95-99.7 法则，也称为  $3\sigma$  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：

$$M = \begin{cases} x_{\frac{n+1}{2}}, & \text{如果 } n \text{ 是奇数} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}, & \text{如果 } n \text{ 是偶数} \end{cases} \quad (3)$$

其中， $\{x_1, x_2, \dots, x_n\}$  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：

$$Q1 = x_{(\frac{n+1}{4})} \quad (4)$$

$$Q3 = x_{(\frac{3(n+1)}{4})} \quad (5)$$

四分位距是上四分位数与下四分位数之差，其公式为：

$$IQR = Q3 - Q1 \quad (6)$$

箱线图的示意图如下：

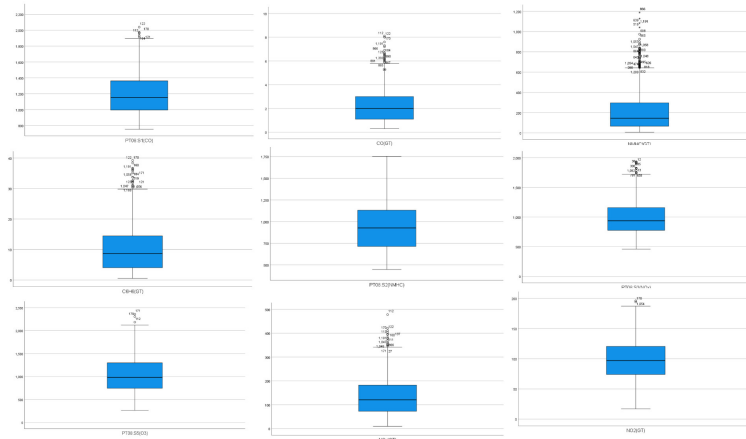


图 3: 数据的箱型图示意

### 5.3 处理结果

对于数据集中每个指标的 9357 例数据，我们对其中的缺失值、异常值进行处理，共计处理了异常数据 18418 例。结果如下图所示：

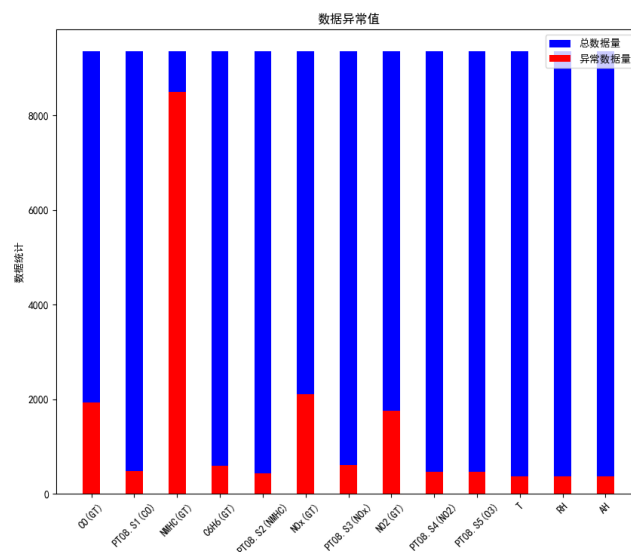


图 4: 数据异常值处理

## 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：

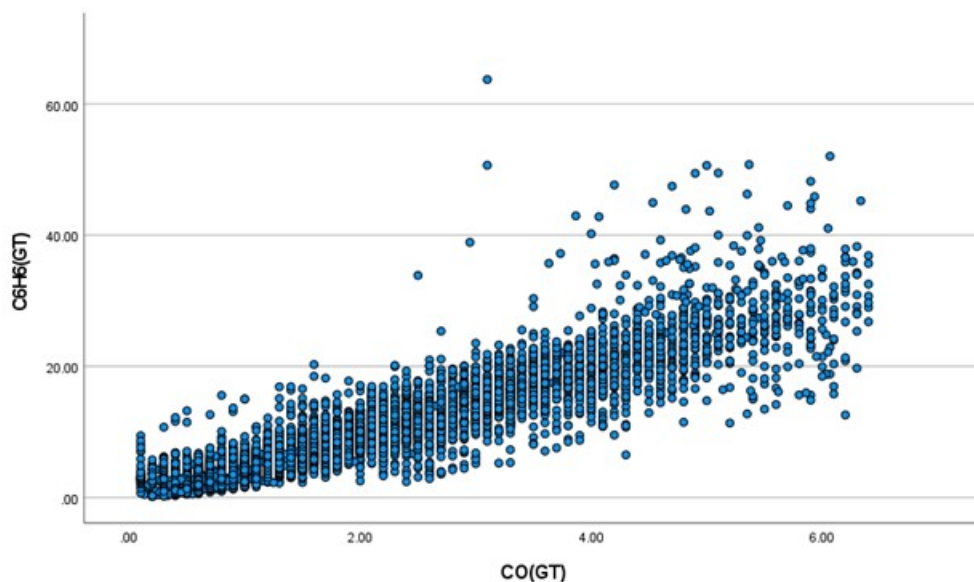


图 5: 苯浓度与一氧化碳浓度的散点图

通过分析苯浓度与一氧化碳浓度的散点图，可判断二者大致呈线性关系。

### 6.1.2 一元回归模型

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，我们采用多个一元回归模型拟合，并选出拟合效果最好的模型。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关系。

#### 1. 一元回归模型的建立：

我们分别采用了一元线性回归模型，一元二次回归模型和一元三次回归模型，对因变量  $y$ （一氧化碳浓度）自变量  $x$ （苯浓度）进行拟合。为了估计模型参数，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。拟合的结果如下：

表 1: 拟合效果比较

模型类型	$R$ 值	$R^2$ 值	$F$ 值	残差
线性回归	0.925	0.856	44652	44766
二元回归	0.925	0.856	22350	45717
三元回归	0.919	0.844	15380	45457

#### 2. 一元回归模型的分析：

经分析，线性回归与二次回归的  $R$  值更高，拟合效果更好，但一元线性回归的  $F$  值更大，表明表示模型的显著性更高。一元线性回归的残差值最小，表明预测的精度稍高一些。综上得一元线性回归模型的拟合度最好。

### 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：

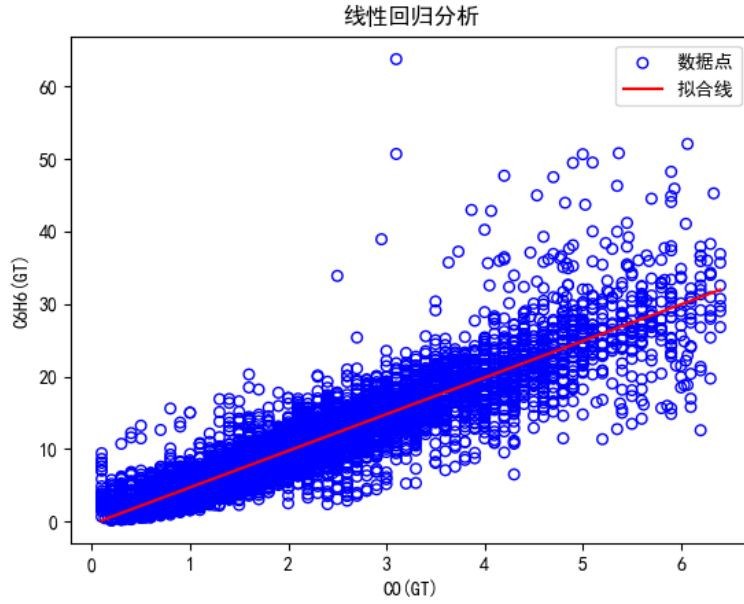


图 6: 苯浓度与一氧化碳浓度的回归直线

一元线性回归模型如下:

$$\hat{y} = -0.333 + 5.046x \quad (7)$$

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 ( $H_0$ ):  $\beta = 0$ , 即苯浓度和一氧化碳浓度之间无直线关系;
- 备择假设 ( $H_1$ ):  $\beta \neq 0$ , 即苯浓度和一氧化碳浓度之间存在直线关系。

显著性水平 ( $\alpha$ ) 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:

$$F = \frac{SS_{\text{回}}}{V_{\text{回}}} = \frac{345350.621}{1} = 44652.736 \quad (8)$$

残差方差为:

$$\frac{SS_{\text{残}}}{V_{\text{残}}} = \frac{9.601}{7580} \quad (9)$$

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  $H_0$ , 接受备择假设  $H_1$ , 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:

$$t^2 = F = 44652.736 \quad (10)$$

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  $H_0$ , 接受备择假设  $H_1$ , 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:

$$\begin{cases} R = 0.925 \\ R^2 = 0.856 \end{cases}$$

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:

$$\hat{y} = -0.333 + 5.046x \quad (11)$$

决定系数为:

$$R^2 = 0.856 \quad (12)$$

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:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (13)$$

其中,  $W_f$  和  $b_f$  分别是遗忘门的权重矩阵和偏置向量,  $\sigma$  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:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (14)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (15)$$

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：

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (16)$$

#### 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：

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (17)$$

$$h_t = o_t \cdot \tanh(C_t) \quad (18)$$

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 2: 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：

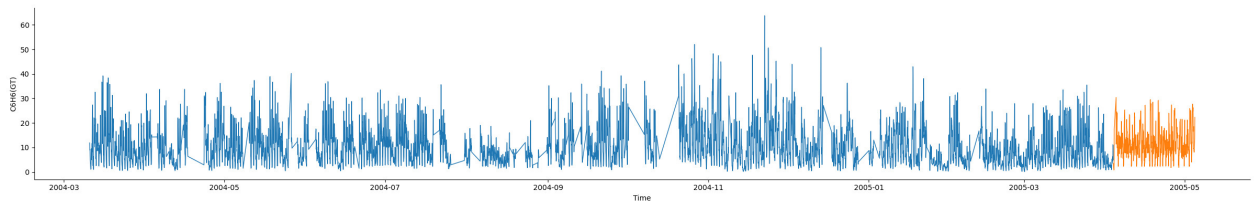


图 7: 苯浓度的后一个月的变化趋势预测

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：

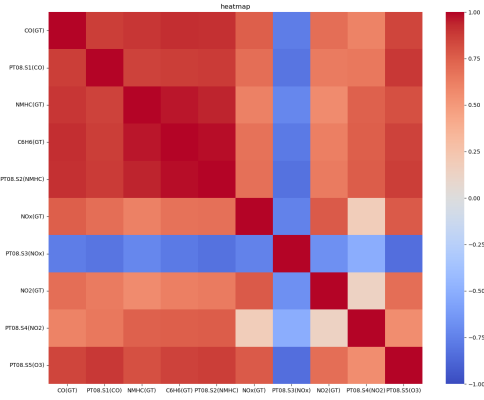


图 8: 污染物及传感器响应的污染物之间的相关性

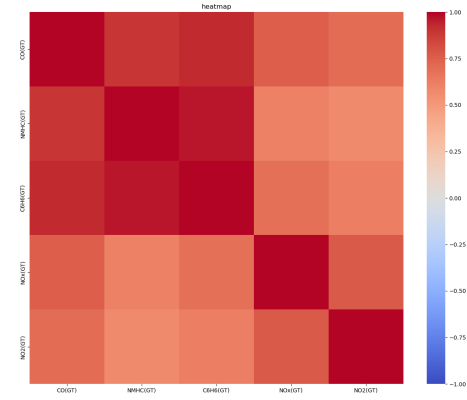


图 9: 污染物之间的相关性

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。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：

$$IAQI_P = \frac{IAQI_H - IAQI_L}{BP_H - BP_L}(C_P - BP_L) + IAQI_L \quad (19)$$

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 3: 空气质量分指数与污染物浓度对照表

IAQI	CO ( $\text{mg}/\text{m}^3$ )	NMHC ( $\mu\text{g}/\text{m}^3$ )	C <sub>6</sub> H <sub>6</sub> ( $\mu\text{g}/\text{m}^3$ )	NO <sub>x</sub> (ppb)	NO <sub>2</sub> ( $\mu\text{g}/\text{m}^3$ )	O <sub>3</sub> ( $\mu\text{g}/\text{m}^3$ )
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：

$$AQI = \max\{IAQI_1, IAQI_2, IAQI_3, \dots, IAQI_n\} \quad (20)$$

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：

$$\rho = \frac{\frac{1}{n} \sum_{i=1}^n (R(x_i) - \overline{R(x)}) (R(y_i) - \overline{R(y)})}{\sqrt{\left(\frac{1}{n} \sum_{i=1}^n (R(x_i) - \overline{R(x)})^2\right) \left(\frac{1}{n} \sum_{i=1}^n (R(y_i) - \overline{R(y)})^2\right)}} \quad (21)$$

其中， $\rho$  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：

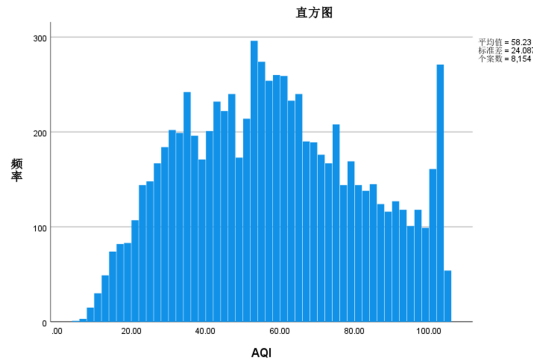


图 10: AQI 的直方图

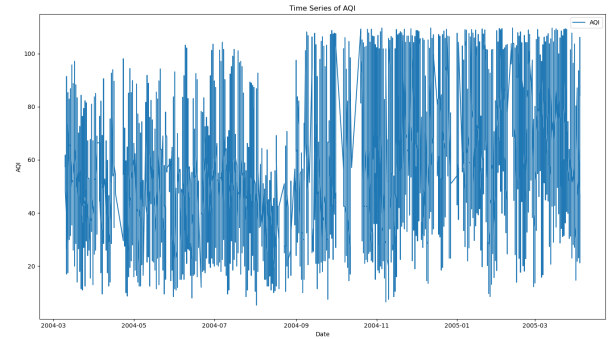


图 11: AQI 时间序列

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 4: 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：

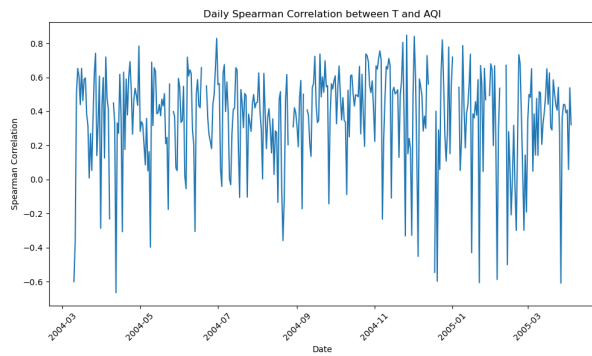


图 12: 温度与 AQI 的日平均 Spearman 系数

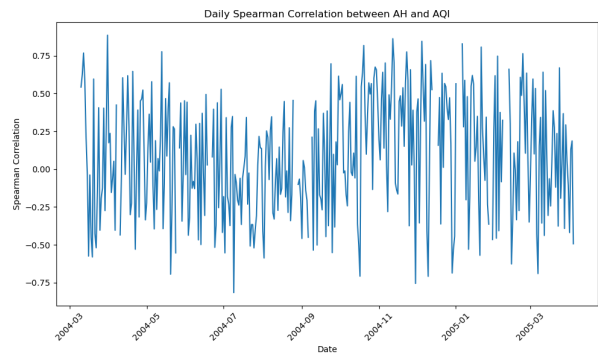


图 13: 绝对湿度与 ADI 的日平均 Spearman 系数

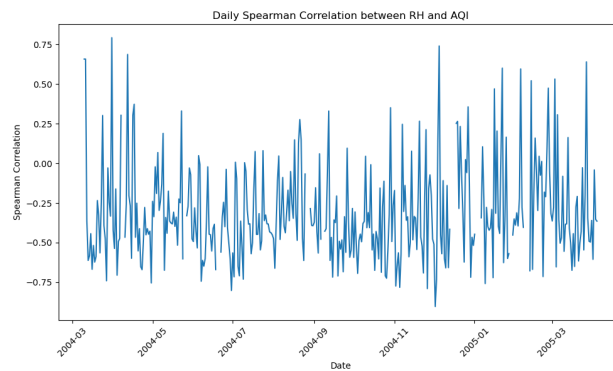


图 14: 相对湿度与 ADI 的日平均 Spearman 系数

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 5: 五个污染物指标的相关系数

参数	T COGT	T NMHGT	T C6H6O6GT	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。

接着我们采用过滤法对模型的特征进行选择。使用过滤法中的相关系数法和 F 统计法 (F-statistic) 来减少空气污染指数 (AQI) 模型的参数量，实质上是通过统计方法来评估每个特征与 AQI 之间的相关性强弱，从而选择对 AQI 预测最有贡献的特征，减少不重要的特征，以达到简化模型的目的。

对于各污染物与空气污染指数相关性的分析，我们对五个污染物与空气污染指数之间的相关系数及 F 统计量进行求解，如下表所示：

表 6: 各污染物与 AQI 的相关系数与 F 统计量

污染物	相关系数	F 统计量	p 值
CO	0.777	14300	0.0
C6H6O6	0.692	8604	0.0
NOx	0.853	25016	0.0
NO2	0.930	60217	0.0
NMHC	0.652	6924	0.0

其中，C6H6O6 和 NMHC 与空气污染指数 (AQI) 的相关系数较低，说明对应的空气污染物分指标与 AQI 指标之间存在较弱的相关性，对 AQI 指标影响较小。C6H6O6 和 NMHC 与空气污染指数 (AQI) 的 F 统计量较低，说明对应的空气污染物分指标与 AQI 指标之间的相关性较弱，对 AQI 指标影响较小。所有污染物浓度的 P 值都非常接近于 0，这表明每个特征对 AQI 的影响都是显著的，即这些特征在统计上对 AQI 的预测具有显著性。

综合上面的过滤法对空气污染指标模型的特征分析，可以认为将 C6H6O6 和 NMHC 两个污染物删去后不影响模型及结果。

#### 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

$$D_n = \sup_x |F_n(x) - F(x)|, \quad (22)$$

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

$$D_n = \max\{|F_n(x) - F(x)|\}. \quad (23)$$

## 4. 比较并决策

如果

$$D_n < D_{na}, \quad (24)$$

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

检验结果如下:

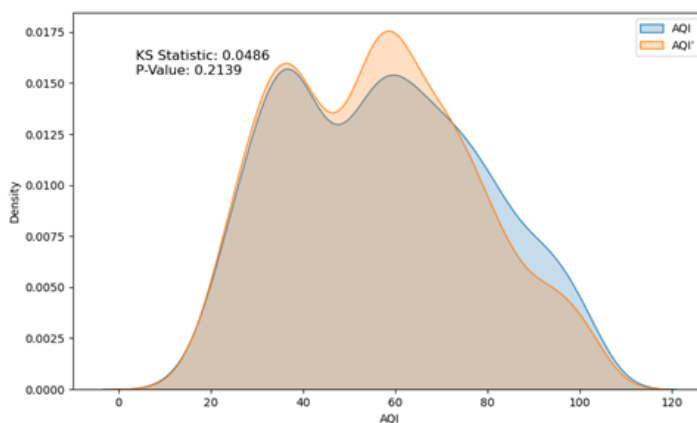


图 15: KS 检验结果

求解得  $p=0.2139$ , 接受假设。因此可以认为删去  $C_6H_6O_2$  与  $NHMC$  这两种污染物后不影响空气污染指标模型的结果。

## 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凌晨、上午、下午以及晚上进行分类, 具体时间划分区间如下:

- 第一区间: 00:00-06:00
- 第二区间: 06:00-12:00
- 第三区间: 12:00-18:00

- 第四区间：18:00-24:00

对这四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

随机选择 4 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

$$\text{dist}(x, c_i) = \sqrt{\sum_{j=1}^d (x_j - c_{ij})^2} \quad (25)$$

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：

$$c_i = \frac{1}{|S_i|} \sum_{x \in S_i} x \quad (26)$$

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：



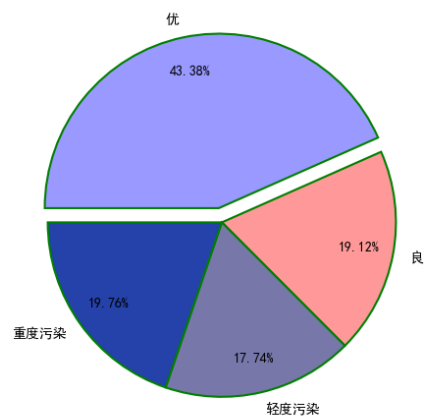


图 16: 所有时间段各污染程度所占比重

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：

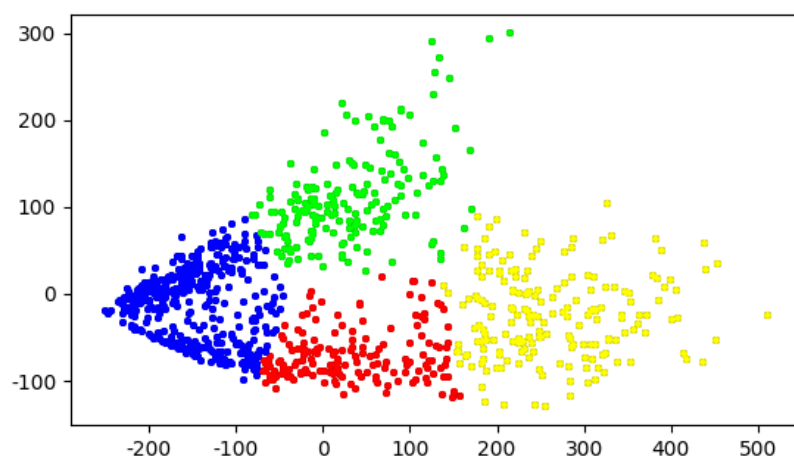


图 17: K-means 聚类结果

按照聚类后的簇 AQI 均值将其环境质量分为优，良，轻度污染，重度污染。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: 环境监测数据

环境质量	AQI	CO(GT)	NMHC(GT)	C6H6(GT)	NO <sub>x</sub> (GT)	NO <sub>2</sub> (GT)	时间
优	1.10	57.63	5.21	100.25	72.48	37.72	凌晨
良	2.78	244.97	14.55	184.31	123.17	64.08	上午
轻度污染	1.82	89.89	7.47	274.85	127.52	67.92	下午
重度污染	3.28	242.16	15.92	451.38	156.22	90.07	晚上

据表可知，NO<sub>x</sub>(GT), NO<sub>2</sub>(GT) 与污染物程度具有强相关性，氮氧化物污染物浓度越高，空气污染指数越高，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，随污染程度加深有线性增强的趋势，但不完全正相关；CO(GT) 对污染程度影响较小，随污染程度加深浓度有变化但变化较小。污染程度随时间递增，时间越晚污染越严重，凌晨污染程度减少，在凌晨污染程度降到最低。

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：

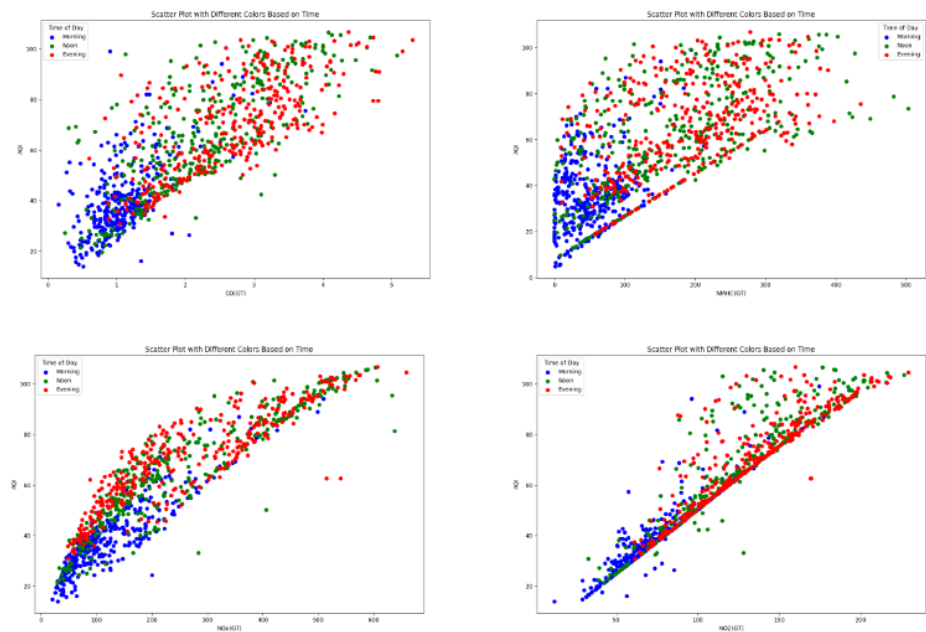


图 18: 各污染物对污染程度的贡献度

由各污染物对污染程度贡献度散点图可以得出以下结论：

从时间上看，晚上的线性程度较强，在晚上各污染物的浓度与污染程度之间呈现较为明显的线性关系。表明在晚间时段，污染物的浓度变化能够较为准确地预测空气质量的变化。中午次之，中午污染物的浓度变化与污染程度之间也具有一定的线性关系，尽管比晚上稍弱，但仍然表现出较高的相关性。而早上的离散程度最高，相比晚上和中午，早上污染物的浓度变化与污染程度之间的关系更为离散。意味着早晨时各污染物的污染程度较低。

从各污染物角度来看，NO<sub>2</sub> 和 NO<sub>x</sub> 的线性程度最强，表明 NO<sub>2</sub> 和 NO<sub>x</sub> 可能是影响空气质量的主要污染物之一，其浓度变化能够较为准确地反映空气质量的变化情况。C<sub>6</sub>H<sub>6</sub>O<sub>6</sub> 次之，与污染程度之间表现出一定的线性关系，虽然比 NO<sub>2</sub> 和 NO<sub>x</sub> 稍弱，但仍具有显著的相关性。NMHC 和 CO 的离散程度最高，意味着在研究和预测空气质量时，需要更多的考虑和调整，以准确反映它们对空气质量的影响。

#### 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<sub>2</sub>）。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<sub>2</sub>）。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<sub>x</sub>，包括 NO 和 NO<sub>2</sub>）。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 模型的优点

1. 数据处理全面：

采用线性插值法、正态分布 3 原则和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。

#### 2. 时间序列处理能力强：

LSTM 模型在捕捉污染物浓度变化的复杂模式和长期依赖关系方面表现出色，解决了传统 RNN 的梯度爆炸和梯度消失问题。

#### 3. 综合反映污染程度：

空气污染浓度指标模型有效反映主要污染物的污染程度，并能识别空气污染的最短板，为制定环保政策提供科学依据。

#### 4. 分类方法有效：

K-means 聚类算法通过合理分类和分级，分析不同时间段的污染特征，结果具有实践指导意义。

## 7.2 模型的缺点

#### 1. 忽略复杂关系：

某些模型仅考虑线性关系，可能忽略更复杂的非线性关系。

#### 2. 高计算复杂度：

LSTM 和 K-means 算法在处理大规模数据时计算复杂度较高，影响处理效率。

#### 3. 数据依赖性强：

模型的准确性和可靠性高度依赖于输入数据的质量和全面性。

#### 4. 结果不稳定性：

K-means 聚类结果依赖于初始参数选择，可能具有不稳定性。

## 7.3 模型的推广

1. 在环境污染物研究中，使用的一元线性回归模型可应用于分析其他气态污染物关系，如臭氧与氮氧化物、颗粒物与二氧化硫等。通过数据预处理后，利用线性回归模型进行分析，也适用于金融市场中的股票相关性分析和医学研究中的生物指标关系研究。

2. 长短期记忆网络（LSTM）模型在时间序列数据处理方面表现出色，可用于预测空气污染物、股票价格、天气和经济指标。调整模型参数和训练数据后，LSTM 模型能在不同领域中提高预测准确性。

3. 多种污染物的城市空气污染指标模型可以推广到其他城市或区域的空气质量分析中。通过收集数据、计算空气质量指数（AQI）并分析与气象因素的关系，该模型也适用于水质监测和土壤污染分析，全面评估环境质量。

4.K-means 聚类算法用于分类和分级污染物数据，分析不同时间段的污染特征并提出政策建议。这种方法也适用于客户关系管理中的客户分类和医学研究中的患者数据聚类，帮助制定针对性的营销策略和个性化治疗方案。

## 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.
- [3] 中华人民共和国环境保护部. 环境空气质量指数 (AQI) 技术规定 (试行) [S]. 2012-02-29 发布, 2016-01-01 实施.
- [4] E. Hot and V. Popović-Bugarin, "Soil data clustering by using K-means and fuzzy K-means algorithm," 2015 23rd Telecommunications Forum Telfor (TELFOR), Belgrade, Serbia, 2015, pp. 890-893.

## 附录 1 问题一代码

```
1 #数据预处理
2 import pandas as pd
3 from scipy.stats import kstest
4 import numpy as np
5
6 def KsNormDetect(df, column):
7     # 计算均值
8     u = df[column].mean()
9     # 计算标准差
10    std = df[column].std()
11    # 计算P值
12    res = kstest(df[column], 'norm', (u, std))[1]
13    # 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。
14    if res <= 0.05:
15        print('该列数据服从正态分布-----')
16        print('均值为: %.3f, 标准差为: %.3f' % (u, std))
17        print('-----')
18        return 1
19    else:
20        return 0
21
22
23 def OutlierDetection(df, column, ks_res):
24     # 计算均值
25     u = df[column].mean()
26     # 计算标准差
27     std = df[column].std()
28
29     if ks_res == 1:
30         # 定义3 法则识别异常值
31         lower_bound = u - 3 * std
32         upper_bound = u + 3 * std
33         # 将异常值替换为np.nan
34         df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
35         return df
36     elif ks_res == 0:
37         Q1 = df[column].quantile(0.25)
38         Q3 = df[column].quantile(0.75)
39         IQR = Q3 - Q1
40         lower_bound = Q1 - 1.5 * IQR
41         upper_bound = Q3 + 1.5 * IQR
42         # 将异常值替换为np.nan
43         df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
44         return df
```

```

45     else:
46         # 如果ks_res既不是0也不是1, 直接返回原始数据框
47         return df
48
49
50 def handle_consecutive_minus(series, column_list):
51     for column in column_list:
52         count = 0 # 用来记录连续-200的个数
53         flag = None
54         print(len(series[column]))
55         for i in range(len(series[column])):
56             if series.loc[i, column] == -200:
57                 count += 1
58                 if count == 1:
59                     flag = i
60             else:
61                 if count >= 3:
62                     series.loc[flag:i-1, column] = np.nan
63                 elif count != 0:
64                     series.loc[flag:i-1, column] = np.nan
65                 count = 0
66                 flag = None
67         df.reset_index(drop=True, inplace=True)
68         print(series)
69
70     return series
71
72
73 def handle_minus(series, column_list):
74     for column in column_list:
75         count = 0 # 用来记录连续-200的个数
76         flag = None
77         print(len(series[column]))
78         for i in range(len(series[column])):
79             if series.loc[i, column] == -200:
80                 count += 1
81                 if count == 1:
82                     flag = i
83             else:
84                 # if count >= 3:
85                 #     series.drop(index=range(flag, i), inplace=True) # 删除索引范围内的行
86                 if count != 0 and count < 3:
87                     series.loc[flag:i-1, column] = np.nan
88                 count = 0
89                 flag = None
90     # 对NaN值进行线性插值

```

```

91         series[column] = series[column].interpolate(method='linear')
92         df.reset_index(drop=True, inplace=True)
93         print(series)
94
95
96         return series
97
98
99     def add_data(series):
100         count = 0
101         flag = 0
102         for column in list(series)[2:]:
103
104             for i in range(len(series[column])):
105                 if pd.isna(series.loc[i, column]):
106                     count += 1
107                     if count == 1:
108                         flag = i
109                     else:
110                         if count != 0 and count <= 6:
111                             series.loc[flag-1:i, column] = series.loc[flag-1:i, column].interpolate(method='linear'
112                                                                                                     )
113                             count = 0
114                             flag = 0
115                         # 对NaN值进行线性插值
116                     print(series)
117                 return series
118
119 # 示例使用
120 df = pd.read_excel('./AirQualityUCI.xlsx')
121 print(df)
122 df = handle_minus(df, list(df)[2:])
123 df = handle_consecutive_minus(df, list(df)[2:])
124 for column in list(df)[2:]:
125     ks_res = KsNormDetect(df, column)
126     df = OutlierDetection(df, column, ks_res)
127 print(df)
128 df = add_data(df)
129 df.to_excel('handledData3.xlsx', index=False)
130
131
132
133
134 #线性回归填充数据
135 import pandas as pd

```



```

136 from scipy.stats import kstest
137 import numpy as np
138 from sklearn.linear_model import LinearRegression
139 import matplotlib.pyplot as plt
140 import matplotlib.dates as mdates # 导入dates模块
141
142 plt.rcParams['font.sans-serif'] = ['SimHei']
143 df = pd.read_excel('./handledData3.xlsx')
144 df = df[0:184]
145 X = df[['C6H6(GT)']] # 自变量, 用列名替换'feature'
146 y = df[['NMHC(GT)']] # 因变量, 用列名替换'target'
147
148 # 将X转换为二维数组, 因为scikit-learn需要这样的格式
149 X = X.values.reshape(-1, 1)
150
151 # 创建线性回归模型实例
152 model = LinearRegression()
153
154 # 拟合模型
155 model.fit(X, y)
156
157 # 获取模型参数, 即斜率和截距
158 slope = model.coef_[0]
159 intercept = model.intercept_
160
161 # 打印模型参数
162 print("斜率:", slope)
163 print("截距:", intercept)
164
165 # 绘制数据点
166 plt.scatter(X, y, s=30, marker='o', facecolors='none', edgecolors='blue', label='数据点')
167 # 使用模型参数绘制拟合线
168 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
169 y_line = intercept + slope * X_line # 计算对应的y值
170 plt.plot(X_line, y_line, color='red', label='拟合线', linewidth=3)
171 #
172 # 添加图例
173 plt.legend()
174
175 # 设置图表标题和轴标签
176 plt.title('线性回归分析')
177 plt.xlabel('C6H6(GT)')
178 plt.ylabel('NMHC(GT)')
179
180 # 显示图表
181 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

```

```

274     # 将异常值替换为np.nan
275     df.loc[(df[column] < lower_bound) | (df[column] > upper_bound), column] = np.nan
276     return df
277 else:
278     # 如果ks_res既不是0也不是1, 直接返回原始数据框
279     return df
280
281
282 def handle_consecutive_minus_200(series):
283     count = 0 # 用来记录连续-200的个数
284     flag = 0
285     mount = 0
286     print(len(series))
287     for i in range(len(series)):
288         mount += 1
289         print(mount)
290         print(i)
291         if series.loc[i, "CO(GT)"] == -200:
292             count += 1
293             if count == 1:
294                 flag = i
295             else:
296                 if count >= 3:
297                     series.drop(index=range(flag, i), inplace=True) # 删除索引范围内的行
298                 elif count != 0:
299                     series.loc[flag:i-1, "CO(GT)"] = np.nan
300                 count = 0 # 重置计数器
301     print("\n\n\n")
302     count = 0
303     for i in range(len(series)):
304         mount += 1
305         print(mount)
306         if i not in series.index:
307             continue
308         if series.loc[i, "C6H6(GT)"] == -200:
309             count += 1
310             if count == 1:
311                 flag = i
312             else:
313                 if count >= 3:
314                     series.drop(index=range(flag, i), inplace=True) # 删除索引范围内的行
315                 elif count != 0:
316                     series.loc[flag:i-1, "C6H6(GT)"] = np.nan
317                 count = 0 # 重置计数器
318     # 对NaN值进行线性插值
319     series['CO(GT)'] = series['CO(GT)'].interpolate(method='linear')

```

```

320     series['C6H6(GT)'] = series['C6H6(GT)'].interpolate(method='linear')
321     return series
322
323
324 df = pd.read_excel('./AirQualityUCI.xlsx')
325 print(df)
326 df = handle_consecutive_minus_200(df)
327 print(df)
328 for column in list(df)[2:]:
329     ks_res = KsNormDetect(df, column)
330     df = OutlierDetection(df, column, ks_res)
331 df['CO(GT)'] = df['CO(GT)'].interpolate(method='linear')
332 df['C6H6(GT)'] = df['C6H6(GT)'].interpolate(method='linear')
333 X = df[['CO(GT)']] # 自变量, 用列名替换'feature'
334 y = df[['C6H6(GT)']] # 因变量, 用列名替换'target'
335 X = X.values.reshape(-1, 1)
336
337 # 创建线性回归模型
338 model = LinearRegression()
339
340 # 拟合模型
341 model.fit(X, y)
342
343 # 获取模型参数, 即斜率和截距
344 slope = model.coef_[0]
345 intercept = model.intercept_
346
347 # 打印模型参数
348 print("斜率:", slope)
349 print("截距:", intercept)
350
351 # 绘制数据点
352 plt.scatter(X, y, s=30, marker='o', facecolors='none', edgecolors='blue', label='数据点')
353 # 使用模型参数绘制拟合线
354 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
355 y_line = intercept + slope * X_line # 计算对应的y值
356 plt.plot(X_line, y_line, color='red', label='拟合线')
357 # 添加图例
358 plt.legend()
359 # 设置图表标题和轴标签
360 plt.title('线性回归分析')
361 plt.xlabel('CO(GT)')
362 plt.ylabel('C6H6(GT)')
363 # 显示图表
364 plt.show()
365 df.to_excel('handledData2.xlsx', index=False)

```



## 附录 2 问题二代码

```
1 #LSTM模型与预测可视化
2 import pandas as pd
3 import numpy as np
4 from keras.models import Sequential
5 from keras.layers import LSTM, Dense
6 from sklearn.preprocessing import MinMaxScaler
7 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
8 from math import sqrt
9 import matplotlib.pyplot as plt
10 from datetime import datetime, timedelta
11 import csv
12 from scipy.stats import kstest
13 import seaborn as sns
14 from scipy import stats as scs
15 plt.rcParams['font.sans-serif'] = ['SimHei']
16 plt.rcParams['axes.unicode_minus'] = False
17
18
19 # 创建数据集
20 def create_dataset(dataset, look_back=1):
21     X, Y = [], []
22     for i in range(len(dataset) - look_back - 1):
23         a = dataset.iloc[i:(i + look_back)][['Time', 'NMHC(GT)']].values
24         X.append(a)
25         Y.append(dataset.iloc[i + look_back]['NMHC(GT)'])
26     print(len(X))
27     print(len(Y))
28     return np.array(X), np.array(Y)
29
30
31 # 构建并训练模型
32 def build_and_train_model(data_center, look_back=168):
33     X, Y = create_dataset(data_center, look_back)
34     model = Sequential()
35     model.add(LSTM(40, input_shape=(look_back, 2), dropout=0.2))
36     model.add(Dense(10))
37     model.compile(optimizer='Adam', loss='mean_squared_error')
38     model.fit(X, Y, epochs=400, batch_size=1, verbose=2)
39     return model
40
41
42 # 进行未来30天每小时预测
43 def predict_next_30_days(model, last_24_hours, scaler, look_back=24):
44     predictions = []
```

```

45     current_batch = last_24_hours
46     for i in range(30 * 24): # 30天每天24小时
47         current_pred = model.predict(current_batch)[0][0] # 获取模型预测的单个值
48         predictions.append(current_pred)
49         next_hour = (current_batch[0, -1, 0] + 1) % 24 # 更新小时
50         # 确保新的行是正确类型
51         new_row = np.array([[next_hour, current_pred]], dtype=np.float32).reshape(1, 2)
52         # 更新批次数据
53         current_batch = np.append(current_batch[:, 1:, :], new_row.reshape(1, 1, 2), axis=1)
54     # 反归一化预测结果
55     inversed_predictions = scaler.inverse_transform(np.array(predictions).reshape(-1, 1))
56     # 将预测<0的数据置为0
57     return np.maximum(inversed_predictions, 0)
58
59 # 主逻辑
60 results_csv_path = '结果表.csv'
61 with open(results_csv_path, 'w', newline='') as csvfile:
62     writer = csv.writer(csvfile)
63     writer.writerow(['Date', 'Time', 'NMHC(GT)'])
64     df = pd.read_excel('./handledData4.xlsx')
65     print(df)
66     df = df[0:9355]
67     df['Time'] = df['Time'].str.slice(start=0, stop=2).astype(int)
68     print(df)
69     df['日期时间'] = df['Date'].astype(str) + '-' + df['Time'].astype(str)
70     df['日期时间'] = pd.to_datetime(df['日期时间'], errors='coerce')
71     df.rename(columns={'日期时间': '日期小时'}, inplace=True)
72     df.set_index('日期小时', inplace=True)
73     scaler = MinMaxScaler(feature_range=(0, 1))
74     df['NMHC(GT)'] = scaler.fit_transform(df[['NMHC(GT)']])
75     print(f"Processing ...")
76     data_center = df[['Date', 'Time', 'NMHC(GT)']]
77     print(data_center)
78     model = build_and_train_model(data_center, look_back=168)
79     last_24_hours = data_center.tail(24)[['Time', 'NMHC(GT)']].values.reshape(1, 24, 2)
80     predictions = predict_next_30_days(model, last_24_hours, scaler)
81     start_date = data_center['Date'].max() + timedelta(days=1)
82     for i in range(30 * 24):
83         predict_date = start_date + timedelta(hours=i)
84         writer.writerow([predict_date.strftime('%Y-%m-%d'), predict_date.hour, predictions[i][0]])
85 print("预测结果已保存到", results_csv_path)
86
87 df = pd.read_excel('./handledData4.xlsx')
88 df = df[0:9355]
89 df['Time'] = df['Time'].str.slice(start=0, stop=2).astype(int)
90 df['Date'] = pd.to_datetime(df['Date'])

```



```

91 data2_path = "结果表.csv"
92 data2 = pd.read_csv(data2_path, encoding='GB2312')
93 data2['Date'] = data2['Date'].astype(str)
94 data2['Time'] = data2['Time'].astype(int)
95 data2['Date'] = pd.to_datetime(data2['Date'], errors='coerce')
96 # 合并原始及预测数据
97 df = pd.concat([df, data2], ignore_index=True)
98 df.sort_values(['Date', 'Time'], inplace=True)
99 df['Date'] = pd.to_datetime(df['Date'])
100 df = df.sort_values(by=['Date', 'Time'])
101
102 df['Time'] = df['Time'].astype(str).str.pad(side='left', fillchar='0', width=2)
103
104 # 合并日期和小时
105 df['日期时间'] = df['Date'].astype(str) + ' ' + df['Time']
106 df['日期时间'] = pd.to_datetime(df['日期时间'])
107 df['日期时间'] = pd.to_datetime(df['日期时间'], errors='coerce')
108 df.drop(['Date', 'Time'], axis=1, inplace=True)
109 df.reset_index(drop=True, inplace=True)
110 df.rename(columns={'日期时间': '日期小时'}, inplace=True)
111 plt.figure(figsize=(30, 20))
112 tem = pd.DataFrame()
113 # 确保'日期时间'列是日期时间类型
114 df['日期小时'] = pd.to_datetime(df['日期小时'])
115 # 筛选出12月之前的数据
116 pre_december_data = df[df['日期小时'] < datetime(2005, 4, 4)].copy()
117 pre_december_data['颜色'] = 'red' # 为12月前的数据指定颜色
118 # 筛选出12月之后的数据
119 post_december_data = df[df['日期小时'] >= datetime(2005, 4, 4)].copy()
120 post_december_data['颜色'] = 'blue' # 为12月后的数据指定颜色
121 # 将两部分数据合并，并添加到tem中
122 tem = pd.concat([tem, pre_december_data, post_december_data], ignore_index=True)
123 # 创建 FacetGrid 对象
124 g = sns.FacetGrid(data=tem, hue="颜色", height=4, aspect=6, sharex=False, sharey=False)
125 # 绘制线图
126 g = g.map(sns.lineplot, "日期小时", "NMHC(GT)", palette="Set1", lw=1)
127 # sns.lineplot(data=tem, x='日期小时', y='NMHC(GT)', palette="Set1", lw=2.5)
128 # 设置x轴标签
129 plt.xlabel('Time')
130 # 显示图形
131 plt.show()

```

## 附录 3 问题三代码

```

1 #计算AQI
2 import pandas as pd

```

```

3 import numpy as np
4
5 # 读取Excel文件
6 file_path = './handledData4.xlsx'
7 df = pd.read_excel(file_path)
8
9 # 删除不需要分析的列
10 columns_to_drop = ['Unnamed: 15', 'Unnamed: 16']
11 df.drop(columns=columns_to_drop, errors='ignore', inplace=True)
12
13 # 将剩余的列转换为数值类型
14 for col in df.columns:
15     if col == 'Date' or col == 'Time':
16         continue
17     df[col] = pd.to_numeric(df[col], errors='coerce')
18
19 # 计算分指数 (IAQI)
20 def calculate_IAQI(value, pollutant):
21     if pollutant == 'CO(GT)':
22         if 0 <= value < 2:
23             return ((50 - 0) / (2 - 0)) * (value - 0) + 0
24         elif 2 <= value < 10:
25             return ((100 - 51) / (10 - 2)) * (value - 2) + 51
26         elif 10 <= value < 35:
27             return ((150 - 101) / (35 - 10)) * (value - 10) + 101
28         elif 35 <= value < 60:
29             return ((200 - 151) / (60 - 35)) * (value - 35) + 151
30         elif 60 <= value < 90:
31             return ((300 - 201) / (90 - 60)) * (value - 60) + 201
32         elif 90 <= value < 120:
33             return ((400 - 301) / (120 - 90)) * (value - 90) + 301
34         else:
35             return 0
36
37     elif pollutant == 'C6H6(GT)':
38         if 0 <= value < 15:
39             return ((50 - 0) / (15 - 0)) * (value - 0) + 0
40         elif 15 <= value < 30:
41             return ((100 - 51) / (30 - 15)) * (value - 15) + 51
42         elif 30 <= value < 60:
43             return ((150 - 101) / (60 - 30)) * (value - 30) + 101
44         elif 60 <= value < 90:
45             return ((200 - 151) / (90 - 60)) * (value - 60) + 151
46         elif 90 <= value < 120:
47             return ((300 - 201) / (120 - 90)) * (value - 90) + 201
48         elif 120 <= value < 150:

```

```

49         return ((400 - 301) / (150 - 120)) * (value - 120) + 301
50     else:
51         return 0
52
53 elif pollutant == 'NOx(GT)':
54     if 0 <= value < 250:
55         return ((50 - 0) / (250 - 0)) * (value - 0) + 0
56     elif 250 <= value < 500:
57         return ((100 - 51) / (500 - 250)) * (value - 250) + 51
58     elif 500 <= value < 1400:
59         return ((150 - 101) / (1400 - 500)) * (value - 500) + 101
60     elif 1400 <= value < 2400:
61         return ((200 - 151) / (2400 - 1400)) * (value - 1400) + 151
62     elif 2400 <= value < 4600:
63         return ((300 - 201) / (4600 - 2400)) * (value - 2400) + 201
64     elif 4600 <= value < 6000:
65         return ((400 - 301) / (6000 - 4600)) * (value - 4600) + 301
66     else:
67         return 0
68
69 elif pollutant == 'NO2(GT)':
70     if 0 <= value < 100:
71         return ((50 - 0) / (100 - 0)) * (value - 0) + 0
72     elif 100 <= value < 200:
73         return ((100 - 51) / (200 - 100)) * (value - 100) + 51
74     elif 200 <= value < 700:
75         return ((150 - 101) / (700 - 200)) * (value - 200) + 101
76     elif 700 <= value < 1200:
77         return ((200 - 151) / (1200 - 700)) * (value - 700) + 151
78     elif 1200 <= value < 2340:
79         return ((300 - 201) / (2340 - 1200)) * (value - 1200) + 201
80     elif 2340 <= value < 3090:
81         return ((400 - 301) / (3090 - 2340)) * (value - 2340) + 301
82     else:
83         return 0
84
85 elif pollutant == 'NMHC(GT)':
86     if 0 <= value < 250:
87         return ((50 - 0) / (250 - 0)) * (value - 0) + 0
88     elif 250 <= value < 800:
89         return ((100 - 51) / (800 - 250)) * (value - 250) + 51
90     elif 800 <= value < 1400:
91         return ((150 - 101) / (1400 - 800)) * (value - 800) + 101
92     elif 1400 <= value < 2000:
93         return ((200 - 151) / (2000 - 1400)) * (value - 1400) + 151
94     elif 2000 <= value < 2700:

```

```

95         return ((300 - 201) / (2700 - 2000)) * (value - 2000) + 201
96     elif 2700 <= value < 3700:
97         return ((400 - 301) / (3700 - 2700)) * (value - 2700) + 301
98     else:
99         return 0
100
101     else:
102         return np.nan
103
104 # 逐行计算每个污染物的IAQI值
105 for pollutant in ['CO(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)', 'NMHC(GT)']:
106     iaqi_col = pollutant + '_IAQI'
107     df[iaqi_col] = df[pollutant].apply(lambda x: calculate_IAQI(x, pollutant) if pd.notna(x) else np.nan)
108
109 # 计算AQI
110 def calculate_AQI(row):
111     # 获取所有IAQI值
112     iaqis = row[['CO(GT)_IAQI', 'C6H6(GT)_IAQI', 'NOx(GT)_IAQI', 'NO2(GT)_IAQI', 'NMHC(GT)_IAQI']]
113     # 计算有效IAQI的数量
114     valid_iaqis = iaqis.dropna()
115     if len(valid_iaqis) >= 2:
116         return valid_iaqis.max()
117     else:
118         return np.nan
119
120 df['AQI'] = df.apply(calculate_AQI, axis=1)
121 # 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
122 contribution = {
123     '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(),
124     '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(),
125     '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(),
126     '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(),
127     '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(),
128 }
129
130 # 转换为DataFrame并排序
131 contribution_df = pd.DataFrame.from_dict(contribution, orient='index', columns=['Frequency'])
132 contribution_df = contribution_df.sort_values(by='Frequency', ascending=False)
133
134 # 打印贡献度排序
135 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

```

```

182     elif count == 1:
183         plt.scatter(group['NO2(GT)'], group['AQI'], c=color_map[time], label='Noon')
184         count += 1
185     elif count == 2:
186         plt.scatter(group['NO2(GT)'], group['AQI'], c=color_map[time], label='Evening')
187         count += 1
188     # 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
189
190 # 添加图例，设置标题
191 plt.legend(title='Time of Day')
192
193 # 添加标题和标签
194 plt.title('Scatter Plot with Different Colors Based on Time')
195 plt.xlabel('NO2(GT)')
196 plt.ylabel('AQI')
197
198 # 显示图形
199 plt.show()

```

## 附录 4 问题四代码

```

1     #聚类及其可视化
2     import time
3
4     import pandas as pd
5     import matplotlib.pyplot as plt
6     import numpy as np
7     from numpy import nonzero, array
8     from sklearn.cluster import KMeans
9     from sklearn.metrics import f1_score, accuracy_score, normalized_mutual_info_score, rand_score,
        adjusted_rand_score
10    from sklearn.preprocessing import LabelEncoder, StandardScaler
11    from sklearn.decomposition import PCA
12
13    df = pd.read_excel('./question4.xlsx') # 设置要读取的数据集
14    df.dropna(inplace=True)
15    # print(df)
16
17    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
18    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）
19    print(features)
20    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）
21    # scaler = StandardScaler()
22    # dataset = scaler.fit_transform(dataset)
23    print(dataset)

```

```

24 attributes = len(df.columns) - 1 # 属性数量（数据集维度）
25 original_labels = list(df[columns[-1]]) # 原始标签
26
27
28
29 def initialize_centroids(data, k):
30     # 从数据集中随机选择k个点作为初始质心
31     centers = data[np.random.choice(data.shape[0], k, replace=False)]
32     return centers
33
34
35 def get_clusters(data, centroids):
36     # 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
37     distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
38     cluster_labels = np.argmin(distances, axis=1)
39     print(cluster_labels)
40     return cluster_labels
41
42
43 def update_centroids(data, cluster_labels, k):
44     # 计算每个簇的新质心，即簇内数据点的均值
45     new_centroids = np.array([data[cluster_labels == i].mean(axis=0) for i in range(k)])
46     return new_centroids
47
48
49 def k_means(data, k, T, epsilon):
50     start = time.time() # 开始时间，计时
51     # 初始化质心
52     centroids = initialize_centroids(data, k)
53     t = 0
54     list = [[], [], [], [], [], []]
55     while t <= T:
56         # 分配簇
57         cluster_labels = get_clusters(data, centroids)
58
59         # 更新质心
60         new_centroids = update_centroids(data, cluster_labels, k)
61
62         # 检查收敛条件
63         if np.linalg.norm(new_centroids - centroids) < epsilon:
64             for i in range(len(cluster_labels)):
65                 if cluster_labels[i] == 0:
66                     list[0].append(i)
67                 elif cluster_labels[i] == 1:
68                     list[1].append(i)
69                 elif cluster_labels[i] == 2:

```

```

70         list[2].append(i)
71     elif cluster_labels[i] == 3:
72         list[3].append(i)
73     elif cluster_labels[i] == 4:
74         list[4].append(i)
75     elif cluster_labels[i] == 5:
76         list[5].append(i)
77     # centroids = scalar.inverse_transform(centroids)
78     print(centroids)
79     break
80     centroids = new_centroids
81     # centroids = scalar.inverse_transform(centroids)
82     print("第", t, "次迭代")
83     t += 1
84     print("用时: {0}".format(time.time() - start))
85     return cluster_labels, centroids, list
86
87
88 # 计算聚类指标
89 def clustering_indicators(labels_true, labels_pred):
90     if type(labels_true[0]) != int:
91         labels_true = LabelEncoder().fit_transform(df[columns[len(columns) - 1]]) # 如果数据集的标签为文本类型,
92         把文本标签转换为数字标签
93     f_measure = f1_score(labels_true, labels_pred, average='macro') # F值
94     accuracy = accuracy_score(labels_true, labels_pred) # ACC
95     normalized_mutual_information = normalized_mutual_info_score(labels_true, labels_pred) # NMI
96     rand_index = rand_score(labels_true, labels_pred) # RI
97     ARI = adjusted_rand_score(labels_true, labels_pred)
98     return f_measure, accuracy, normalized_mutual_information, rand_index, ARI
99
100 # 绘制聚类结果散点图
101 def draw_cluster(dataset, centers, labels):
102     center_array = array(centers)
103     if attributes > 2:
104         dataset = PCA(n_components=2).fit_transform(dataset) # 如果属性数量大于2, 降维
105         center_array = PCA(n_components=2).fit_transform(center_array) # 如果属性数量大于2, 降维
106     else:
107         dataset = array(dataset)
108     # 做散点图
109     label = array(labels)
110     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c='black', s=7) # 原图
111     # plt.show()
112     colors = np.array(
113         ["#FF0000", "#0000FF", "#00FF00", "#FFFF00", "#00FFFF", "#FF00FF", "#800000", "#008000", "#000080",
114          "#808000",

```



```

114         "#800080", "#008080", "#444444", "#FFD700", "#008080"]])
115 # 循环打印k个簇，每个簇使用不同的颜色
116 for i in range(k):
117     plt.scatter(dataset[nonzero(label == i), 0], dataset[nonzero(label == i), 1], c=colors[i], s=7, marker=
        'o')
118 # plt.scatter(center_array[:, 0], center_array[:, 1], marker='x', color='m', s=30) # 聚类中心
119 plt.show()
120
121 if __name__ == "__main__":
122     k = 6 # 聚类簇数
123     T = 1000 # 最大迭代数
124     n = len(dataset) # 样本数
125     epsilon = 1e-5
126     # 预测全部数据
127     labels, centers, list = k_means(np.array(dataset), k, T, epsilon)
128     # print(labels)
129     # F_measure, ACC, NMI, RI, ARI = clustering_indicators(original_labels, labels) # 计算聚类指标
130     # print("F_measure:", F_measure, "ACC:", ACC, "NMI", NMI, "RI", RI, "ARI", ARI)
131     # print(membership)
132     print(centers)
133     print(list[0])
134     print(list[1])
135     print(list[2])
136     print(list[3])
137     print(list[4])
138     print(list[5])
139     plt.rcParams['axes.unicode_minus'] = False # 坐标轴负号的处理
140     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
141
142     print(dataset)
143     draw_cluster(dataset, centers, labels=labels)
144
145 #污染物程度占比绘制
146 import time
147
148 import pandas as pd
149 import matplotlib.pyplot as plt
150 import numpy as np
151 from numpy import nonzero, array
152 from sklearn.cluster import KMeans
153 from sklearn.metrics import f1_score, accuracy_score, normalized_mutual_info_score, rand_score,
    adjusted_rand_score
154 from sklearn.preprocessing import LabelEncoder, StandardScaler
155 from sklearn.decomposition import PCA
156
157 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

```

```

203     # 更新质心
204     new_centroids = update_centroids(data, cluster_labels, k)
205
206     # 检查收敛条件
207     if np.linalg.norm(new_centroids - centroids) < epsilon:
208         for i in range(len(cluster_labels)):
209             if cluster_labels[i] == 0:
210                 list[0].append(i)
211             elif cluster_labels[i] == 1:
212                 list[1].append(i)
213             elif cluster_labels[i] == 2:
214                 list[2].append(i)
215             elif cluster_labels[i] == 3:
216                 list[3].append(i)
217             elif cluster_labels[i] == 4:
218                 list[4].append(i)
219             elif cluster_labels[i] == 5:
220                 list[5].append(i)
221         # centroids = scalar.inverse_transform(centroids)
222         print(centroids)
223         break
224     centroids = new_centroids
225     # centroids = scalar.inverse_transform(centroids)
226     print("第", t, "次迭代")
227     t += 1
228     print("用时: {}".format(time.time() - start))
229     return cluster_labels, centroids, list
230
231
232 # 计算聚类指标
233 def clustering_indicators(labels_true, labels_pred):
234     if type(labels_true[0]) != int:
235         labels_true = LabelEncoder().fit_transform(df[columns[len(columns) - 1]]) # 如果数据集的标签为文本类型,
236         把文本标签转换为数字标签
237     f_measure = f1_score(labels_true, labels_pred, average='macro') # F值
238     accuracy = accuracy_score(labels_true, labels_pred) # ACC
239     normalized_mutual_information = normalized_mutual_info_score(labels_true, labels_pred) # NMI
240     rand_index = rand_score(labels_true, labels_pred) # RI
241     ARI = adjusted_rand_score(labels_true, labels_pred)
242     return f_measure, accuracy, normalized_mutual_information, rand_index, ARI
243
244 # 绘制聚类结果散点图
245 def draw_cluster(dataset, centers, labels):
246     center_array = array(centers)
247     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",
258     "#808000",
259     "#800080", "#008080", "#444444", "#FFD700", "#008080"])
260 # 循环打印k个簇, 每个簇使用不同的颜色
261 for i in range(k):
262     plt.scatter(dataset[nonzero(label == i), 0], dataset[nonzero(label == i), 1], c=colors[i], s=7, marker=
263     'o')
264 # plt.scatter(center_array[:, 0], center_array[:, 1], marker='x', color='m', s=30) # 聚类中心
265 plt.show()
266
267 if __name__ == "__main__":
268     k = 6 # 聚类簇数
269     T = 1000 # 最大迭代数
270     n = len(dataset) # 样本数
271     epsilon = 1e-5
272     # 预测全部数据
273     labels, centers, list = k_means(np.array(dataset), k, T, epsilon)
274     # print(labels)
275     # F_measure, ACC, NMI, RI, ARI = clustering_indicators(original_labels, labels) # 计算聚类指标
276     # print("F_measure:", F_measure, "ACC:", ACC, "NMI", NMI, "RI", RI, "ARI", ARI)
277     # print(membership)
278     print(centers)
279     print(list[0])
280     print(list[1])
281     print(list[2])
282     print(list[3])
283     print(list[4])
284     print(list[5])
285     plt.rcParams['axes.unicode_minus'] = False # 坐标轴负号的处理
286     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
287
288     # length = len(dataset)
289     # print(length)
290     # edu = [len(list[0]) / length, len(list[1]) / length, len(list[2]) / length, len(list[3]) / length, len(
291     list[4]) / length]
292     # labels = ['batter', 'great', 'middle', 'bad', 'worse']
293     # explode = [0, 0.1, 0, 0, 0] # 生成数据, 用于凸显大专学历人群

```

```

291 # colors = ['#9999ff', '#ff9999', '#7777aa', '#2442aa', '#dd5555'] # 自定义颜色
292 #
293 # plt.pie(x=edu, # 绘图数据
294 #         explode=explode, # 指定饼图某些部分的突出显示, 即呈现爆炸式
295 #         labels=labels, # 添加教育水平标签
296 #         colors=colors,
297 #         autopct='%.2f%%', # 设置百分比的格式, 这里保留两位小数
298 #         pctdistance=0.8, # 设置百分比标签与圆心的距离
299 #         labeldistance=1.1, # 设置教育水平标签与圆心的距离
300 #         startangle=180, # 设置饼图的初始角度
301 #         radius=1.2, # 设置饼图的半径
302 #         counterclock=False, # 是否逆时针, 这里设置为顺时针方向
303 #         wedgeprops={'linewidth': 1.5, 'edgecolor': 'green'}, # 设置饼图内外边界的属性值
304 #         textprops={'fontsize': 10, 'color': 'black'}, # 设置文本标签的属性值
305 #         )
306 #
307 # # 添加图标题
308 # # 显示图形
309 # plt.show()
310 print(dataset)
311 draw_cluster(dataset, centers, labels=labels)

```