行车轨迹估计交通信号灯周期问题

摘要

交通信号灯对于优化交通流量控制和路网规划有着重大作用，探究基于行车轨迹估计交通信号灯周期对于交通管理决策者至关重要。

针对问题一，根据附件 1 提供的数据集及题目要求，对数据进行分析处理与可视化。首先采用欧几里得距离将轨迹数据转变为速度数据，设定阈值判断车辆状态。计算车辆启动偏移时长，采用分段均值差分法进行启动偏移时长的差分计算，得出红绿灯周期。随后对每辆车的停车时间进行 K-means 聚类，根据聚类结果得出红灯时长。最后利用周期时长和红灯时长，计算出绿灯时长。

针对问题二，问题二模型在问题一模型的基础上进行了优化，采用聚类算法对差分的结果进行优化，剔除掉相对偏移时长的差分结果中可能存在的离群点和异常值，可得出更为准确的红绿灯周期时间。样本车辆比例、车流量对模型一的精确度有影响，定位误差对模型的精确度影响较小。

针对问题三，根据附件 3 提供的数据集及题目要求，首先利用相对偏移时间的差分结果获取大致的完整周期间隔，对差分结果进行清洗后，采用 CUSUM 算法检测变化点，随后采用 K-means 聚类算法获取合适的周期数量的聚类结果，根据聚类结果划分不同周期对应的不同车辆，利用停车较长的时间数据计算红灯周期。

针对问题四，根据部分数据可视化可知需要对方向进行划分，对数据划分处理后，利用模型三进行周期计算，对计算结果进行平均运算和异常值处理后得出相应周期。

行车轨迹估计对交通信号灯周期的估计至关重要，它通过分析车辆的运动轨迹，优化信号配时方案，改善交通安全，提高路网效率，减少能源消耗，从而有效改善城市交通的运行状况，提高整体交通系统的效率和安全性。因此，本文建立的模型旨在通过行车轨迹估计，结合信号灯周期的推断，为交通管理者提供决策建议。建议交通管理者根据模型输出的结果，及时调整信号灯周期，以应对不同时间段的交通流量变化，优化路口通行效率，减少拥堵和交通事故发生的可能性。同时，建议在调整信号灯周期时考虑到城市的交通规划和环境保护的因素，以实现交通系统的可持续发展。

关键字：分段均值差分；聚类算法；动态周期变化检测；CUSUM 算法

目录

一、问题重述

1.1 问题背景

交通信号灯是提高道路使用效率，改善交通状况的一种重要工具。为给司机提供更好的导航服务，某电子地图服务商希望能够获取城市路网中所有交通信号灯的红绿周期。但由于许多信号灯未接入网络，周期数据获取难度较大，该公司计划使用大量客户的行车轨迹数据估计交通信号灯的周期。在已知所有信号灯只有红、绿两种状态的背景下解决以下四个问题。

1.2 问题提出

问题一：若信号灯周期固定不变，且已知所有车辆的行车轨迹，建立模型，利用车辆行车轨迹数据估计信号灯的红绿周期。根据附件 1 中 5 个不相关路口各自一个方向连续 1 小时内车辆的轨迹数据，求出这些路口相应方向的信号灯周期。

问题二：实际上，只有部分用户使用该公司的产品，即只能获取部分样本车辆的行车轨迹。同时，受各种因素的影响，轨迹数据存在定位误差，误差大小未知。讨论样本车辆比例、车流量、定位误差等因素对上述模型估计精度的影响。根据附件 2 中另外 5 个不相关路口各自一个方向连续 1 小时内样本车辆的轨迹数据，求出这些路口相应方向的信号灯周期。

问题三：如果信号灯周期有可能发生变化，能否尽快检测出这种变化，以及变化后的新周期。根据附件 3 中另外 6 个不相关路口各自一个方向连续 2 小时内样本车辆的轨迹数据，判断这些路口相应方向的信号灯周期在这段时间内是否有变化，求出周期切换的时刻，以及新旧周期参数，并指明识别出周期变化所需的时间和条件。

问题四：根据附件 4 中某路口连续 2 小时内所有方向样本车辆的轨迹数据，识别出该路口信号灯的周期。

二、模型假设

假设 1：当车辆速度小于 0.1 米/秒时，车辆处于停止状态;

假设 2：原始样本的数据测量结果正确、本文中数据处理步骤正确；

假设 3：在选择影响变量时所使用的方法符合题目要求；

假设 4：在建立预测模型及分类预测模型时，模型的预测结果准确；

假设 5：假设给出的脱敏数据没有造成数据的损坏;

假设 6：假设车辆坐标数据足够精确，可以反映车辆的实际位置和移动方向。

三、符号说明

符号	意义
T_i	偏移时长
S_i	启动时间点
v_i	速度
D_i	差分值
C	红绿灯周期
C_{red}	红灯周期
C_{green}	绿灯周期
E_i	不同周期变化点
$Class_i$	不同周期

四、问题一的模型建立与求解

4.1 问题分析

在题目一中，已知所有车辆的行车轨迹，并且信号灯周期是固定不变的。目标是建立一个模型，利用车辆行车轨迹数据来估计信号灯的红绿周期。首先需要对数据进行探索性数据分析，结合人为判定，附件 1 中数据集无需进行异常值处理，采用分段均值差分算法计算出完整的红绿灯周期，再使用聚类算法筛选出相应的结果，利用红绿灯时间减去红灯时间即为绿灯时间。

4.2 探索性数据分析

4.2.1 数据可视化

经初步观察，附件 1 中数据集具有相同的数据特征，对于数据集 A1 的数据与处理与建模可以应用于其余数据集中，由于数据量相对较大，故选择对数据进行局部可视化，绘图如下：

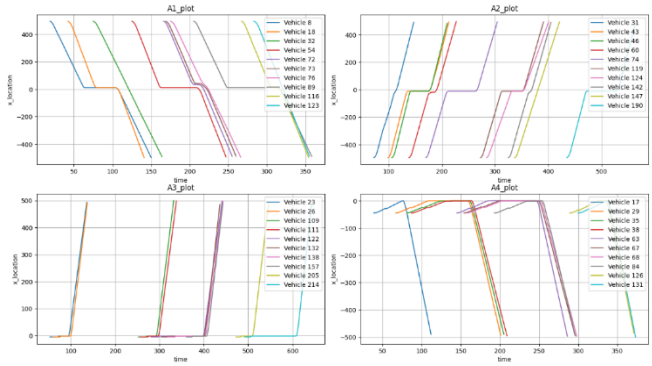


图 1 文件 A1-A4 部分数据可视化

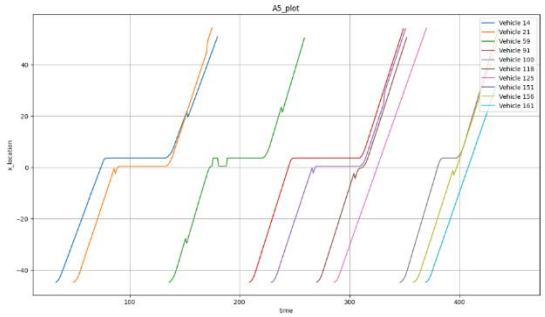


图 2 文件 A4-A5 部分数据可视化

分析图表可知，纵坐标曲线呈水平状态时，表示车辆处于静止状态，即该时间段为等待红灯的时间。为确定是否处于停车等待状态，可以通过设置速度阈值来筛选数据。在问题一中，无需对原始数据进行异常值处理，而是可以直接利用速度阈值来识别停车状态。

4.2.2 速度转换

采用欧几里得距离计算相邻两个时间点的距离变化，公式如下：

$$v_i = \frac{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}}{t_i - t_{i-1}} \tag{1}$$

其中，关于停车时速度的阈值，本研究采用 0.1m/s 为标准，即v_i小于 0.1m/s 时视为开始停车时间。

4.3 算法介绍

4.3.1 分段均值差分算法

分段均值差分算法（Piecewise Mean Difference Algorithm）是一种用于时间序列数据异常检测的方法。它通过将时间序列数据分成多个连续的段，并计算每个段的均值，然后检测每个数据点与其所在段的均值之间的差异是否超过了预先设定的阈值，以确定是否存在异常，本文中的异常即为车辆停车等待时间。

具体步骤如下：

（1）将时间序列数据分成连续的若干段。

（2）对每个段计算均值。

（3）对于每个数据点，计算其与所在段的均值之间的差异。

（4）设定阈值，判断差异是否超过阈值，如果超过阈值则认为该数据点存在异常。

4.3.2 层次聚类算法

层次聚类（Hierarchical Clustering）是一种重要的无监督机器学习聚类方法，它通过构建一棵层次分明的聚类树（Dendrogram），以递归方式将数据点逐步合并或分割，从而揭示数据内在的层次结构和相似性关系。层次聚类的核心定理并不像其他算法那样明确地定义一个定理，它的核心思想在于利用距离或相似度矩阵，通过构建聚类树来表示数据点之间的层次关系。聚类树的构建可以通过两种主要策略：

凝聚式聚类（Agglomerative Clustering）和**分裂式聚类**（Divisive Clustering）。

凝聚式聚类将每个数据点视为一个单独的簇，然后在每一步中，找出距离最近的两个簇进行合并，直到所有数据点都被合并到一个簇为止。常用的凝聚策略有单链接（Single Linkage）、全链接（Complete Linkage）、平均链接（Average Linkage）和 ward 方法等。分裂式聚类则相反，从整体出发，首先将所有数据点视为一个簇，然后在每一步中，将当前簇中最分散的部分拆分成两个子簇，重复此过程直至满足终止条件。

本文采用凝聚式聚类算法获取红绿灯时间，对聚类结果可视化进行了可视化，对最接近完整红灯时间的一类数据进行平均处理，获取最终红灯时间。

4.3.3 K-means 聚类算法

K-means 聚类算法是一种常用的无监督学习算法，用于将数据集中的观测值划分为 k 个不同的组或簇。其目标是通过最小化每个观测值与所属簇的聚类中心之间的平方距离来确定簇的分配，从而使得簇内的观测值尽可能相似，而簇间的观测值尽可能不同。

K-means 算法的主要步骤如下：

（1）初始化：随机选择 k 个点作为初始的聚类中心。

（2）分配：将每个观测值分配到距离其最近的聚类中心所对应的簇中。通常使用欧式距离来度量观测值与聚类中心之间的距离。

（3）更新：重新计算每个簇的聚类中心，通常是该簇中所有观测值的平均值。

（4）重复：重复步骤 2 和 3，直到满足停止条件，例如聚类中心不再发生变化或达到最大迭代次数。

（5）输出：输出最终的 k 个聚类中心以及每个观测值所属的簇。

欧式距离的表达式如下：

$$\mathrm{ED}(p,q)=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2} \tag{2}$$

其中，(p)和(q)是两个点的坐标，(n)是坐标的维度。(p_i)和(q_i) 分别表示两个点在第（i）维上的坐标。

4.4 问题求解

4.4.1 周期时间计算

将整个样本里面第 1 辆车作为起始的观测时间点，将后面所有车辆的启动时间点减去这个起始观测时间的偏移时间，

（1）计算偏移时间

$$T_i = S_i - S_1, (i > 1) \tag{3}$$

对启动偏移时长进行可视化，并按照时间点升序排列，得到如下图:

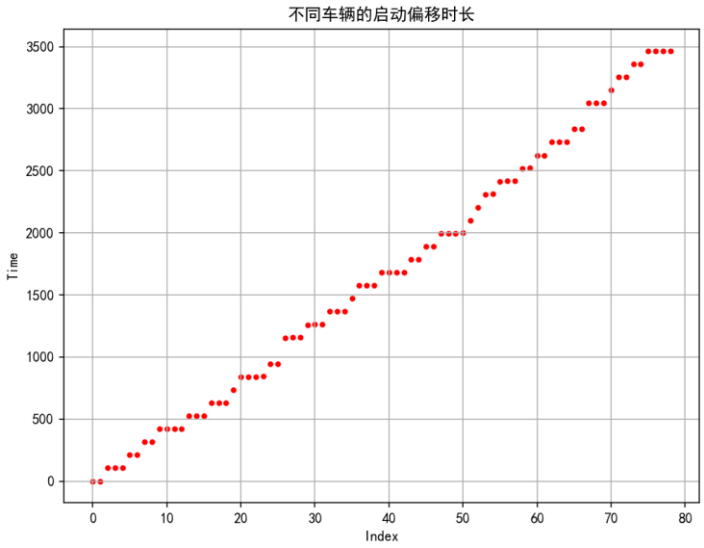


图 3 不同车辆的启动偏移时长

图 3 中横坐标拟代表不同车辆，纵坐标代表相对与第一辆车的偏移时长，可以看出呈现规律性变化。

(2)计算差分：

$$D_i = T_i - T_{i-1}, (i > 2) \tag{4}$$

差分表示当前启动偏移时间值减去前一个启动偏移时间点，将计算的差分结果进行层次聚类并可视化如下：

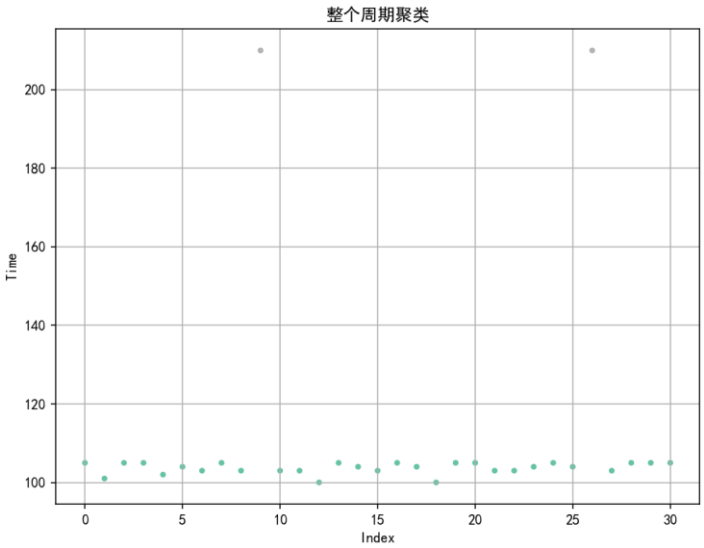


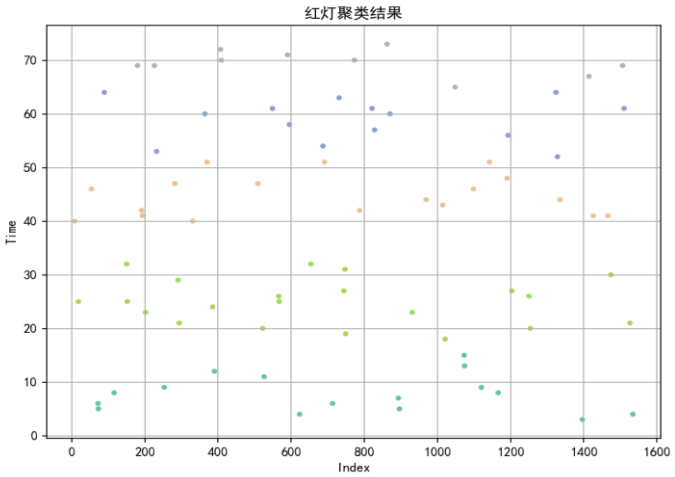
图 4 整个周期聚类

根据上述结果，采用聚类算法去除了异常值（例如图中的灰色点），保留正常值，对这些正常结果平均处理，用来表示整个红绿灯的周期。

$$C = \frac{1}{n}\sum_{i=1}^{n} AgglomerativeClustering\,(D_i) \tag{5}$$

4.4.2 红灯时间计算

利用周期时间计算中获取到的每辆车的停止时间来进行计算，对每辆车的停车时间进行 K-means 聚类并可视化，得到如图所示：



图表 5 红灯聚类结果

根据现实情况，车辆的停车时间与红灯周期之间并没有明显的相关关系或规律。因为并非所有车辆都是在红灯开始时停车的，而后来的车辆在接近红绿灯时，可能已经过了很长时间。在这种情况下，停车时间不能准确地代表红灯的周期。

考虑到这一点，使用 K-means 算法对停车时间进行聚类，识别和找出那些在红灯周期内停车的情况。通过对这些聚类结果求平均值，可以得到一个更为准确的红灯周期长度。这个长度将反映实际交通情况下红灯的平均周期，为交通管理和规划提供更有效的参考。

$$C_{red} = \frac{1}{n}\sum_{i=1}^{n} K - means\,(T_i) \tag{6}$$

4.4.3 计算绿灯时间

之前的问题中已知红绿灯周期与红灯周期，故可用红绿灯周期减去绿灯周期。

$$C_{green} = C - C_{red}$$

4.4.4 模型汇总

$$\begin{cases} T_i = S_i - S_1, & (i > 1) \\ D_i = T_i - T_{i-1}, & (i > 2) \\ C = \frac{1}{n}\sum_{i=1}^{n} AgglomerativeClustering\,(D_i) \\ C_{red} = \frac{1}{n}\sum_{i=1}^{n} K - means\,(T_i) \\ C_{green} = C - C_{red} \end{cases} \tag{7}$$

问题一实验结果如下表所示：

路口	A1	A2	A3	A4	A5
红灯时长（秒）	69	51	74	62	59
绿灯时长（秒）	35	37	31	26	29

表格 1 路口 A1-A5 各自一个方向信号灯周期识别结果

五、问题二的模型建立与求解

5.1 问题分析

在题目二中，由于只有部分用户使用该公司的产品，即只能获取部分样本车辆的行车轨迹。同时，受各种因素的影响，轨迹数据存在定位误差，误差大小未知。需要讨论样本车辆比例、车流量、定位误差等因素对上述模型估计精度的影响。

问题二的模型为问题一模型的优化，相对偏移时间的差分结果可能存在的离群点和异常值，使用聚类算法对差分的结果进行优化，选取最能代表完整周期的时间长度，后续算法同问题一。

5.2 偏移时长聚类优化

偏移时长会出现异常值与离群点，使用 K-means 对差分后的结果进行优化，在问题二中使用$argmax_{x_i} f(C_{red_i})$公式去获取$C_{red}$出现次数最多的公式，如下：

$$\begin{cases} T_i = S_i - S_1, & (i > 1) \\ D_i = T_i - T_{i-1}, & (i > 2) \\ C = \frac{1}{n}\sum_{i=1}^{n} AgglomerativeClustering\,(D_i) \\ C_{red_i} = \frac{1}{n}\sum_{i=1}^{n} K - means\,(T_i) \\ C_{red} = argmax_{x_i} f(C_{red_i}) \\ C_{green} = C - C_{red} \end{cases} \tag{8}$$

5.3 因数影响

(1) 车辆比例影响

在本研究中，采用以前 2500 条为数据起始点，观测模型一与模型二每隔 100 条数据对于红绿灯周期的影响，并进行可视化如下：

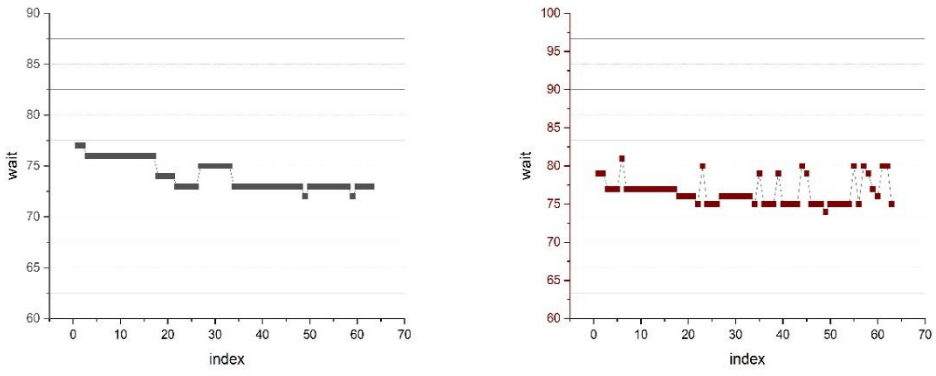


图 6 车辆比例影响模型对比

实验结果表明，对于模型一，随着数据量的增加即车辆比例的增加，会使红绿灯周期收到影响，而对于模型二，随着数据量的增加，数据呈现稳定性浮动，说明对于差分结果的聚类可以使减少车辆比例对于模型的影响。

(2)定位影响

定位误差是指在实际应用中，由于 GPS 或其他定位技术的局限性，记录的车辆位置数据可能存在一定程度的不确定性和偏差，从而影响了数据的精确性和可靠性。这种误差的模拟通常体现在对车辆的 x 和 y 坐标加入随机噪声，以模拟现实世界中定位系统的不确定性。

这种误差可能由多种因素引起，例如信号遮挡、大气条件变化、设备精度等。因此，对车辆位置数据进行处理时，常常会引入一个偏移量，该偏移量通常是基于原始坐标的标准差的一定百分比（例如 3%）来确定的。这样的做法旨在模拟真实世界中定位数据的自然波动，并使分析结果更符合实际情况。

以附件 2 中数据集 B1.csv 进行实验验证，结果如下：

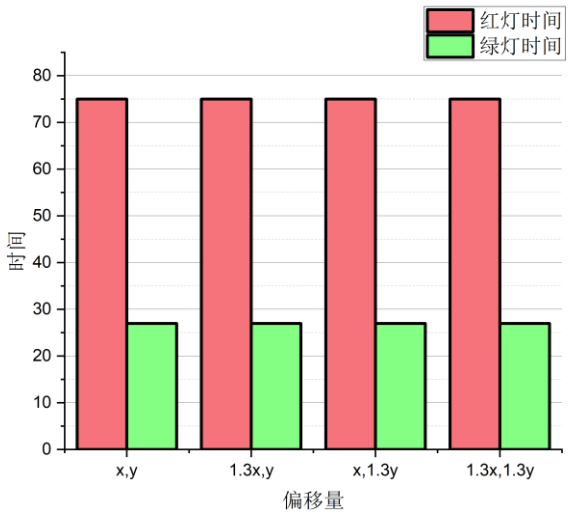


图 7 偏移量对于周期的影响

偏移量	x, y	1.3x, y	x, 1.3y	1.3x, 1.3y
红灯时长（秒）	75	75	75	75
绿灯时长（秒）	27	27	27	27

实验结果表明，加入随机坐标误差后对于红灯时间没有影响，同时也验证了优化后模型的泛化能力。

5.4 问题求解

问题二的模型汇总在偏移时长聚类优化中已提出。

问题二实验结果如下表所示：

路口	A1	A2	A3	A4	A5
红灯时长（秒）	75	75	71	74	88
绿灯时长（秒）	27	40	42	30	27

表格 2 路口 B1-B5 各自一个方向信号灯周期识别结果

六、问题三的模型建立与求解

6.1 问题分析

问题三信号灯周期可能会发生变化，需要根据模型快速检测出这种变化以及变化后的新周期， 首先利用相对偏移时间的差分结果来获取大致的完整周期间隔，由于周期是不断变化的，对差分结果进行清洗之后利用 CUSUM 算法检测变化点，获取每个变化时段里面的起始启动点;接着利用 k-means 聚类的轮廓系数自动确定合适的周期数量，然后利用 k-means 聚类获取聚类结果，对每一簇的结果分别求平均，其平均值可以代表当前周期的周期值;接着，根据聚类的结果将不同周期对应的不同车辆分离出来。统计他们的停车时间，去前 k 个停车较大的时间计算其平均值，作为每一组的红灯周期。

6.2 CUSUM 算法

CUSUM 是一种序列化的检测方法，用于确定一系列数据点中是否存在显著的变化，比如交通信号灯的周期变化。其基本原理是通过计算数据点与过程平均值或目标值的累积偏差来工作。如果过程只受到随机波动的影响，累积偏差应该会保持相对稳定；但如果过程的基本性质发生了变化（比如信号灯的红绿灯周期改变），CUSUM 值将会显著偏离零，表现出持续的上升或下降趋势。

算法如下：

（1）初始化：设置一个目标值和一个阈值。目标值是你期望的过程平均值，阈值则是判断过程是否发生改变的界限。

（2）计算每一个数据点的偏差：这是数据点的值减去目标值。

（3）计算 CUSUM：这是所有偏差的累积和。

（4）检测是否超过阈值：如果 CUSUM 的绝对值超过了阈值，那么就认为过程已经发生了改变。

6.3 问题求解

算法如下：

（1）利用相对偏移时间的差分结果获取大致的完整周期间隔。

（2）对差分结果进行清洗，利用 CUSUM 算法来检测变化点。

（3）使用 k-means 聚类算法来自动确定合适的周期数量。

（4）使用 k-means 聚类算法获取聚类结果。

（5）根据聚类结果，分离出来不同周期对应的不同车辆。

（6）统计停车时间并计算红灯周期。

$$\begin{cases} T_i = S_i - S_1, & i > 1 \\ D_i = T_i - T_{i-1}, & i > 2 \\ E_i = CUSUM(Di) \\ Class_i = K - means(E_i) \\ C = \frac{1}{n}\sum_{i=1}^{n} AgglomerativeClustering\,(Class_i) \\ C_{red_i} = \frac{1}{n}\sum_{i=1}^{n} K - means\,(T_i) \\ C_{red} = argmax_{x_i} f\left(C_{red_i}\right) \\ C_{green} = C - C_{red} \end{cases} \tag{9}$$

绘制不同车辆的启动偏移时长如下图所示：

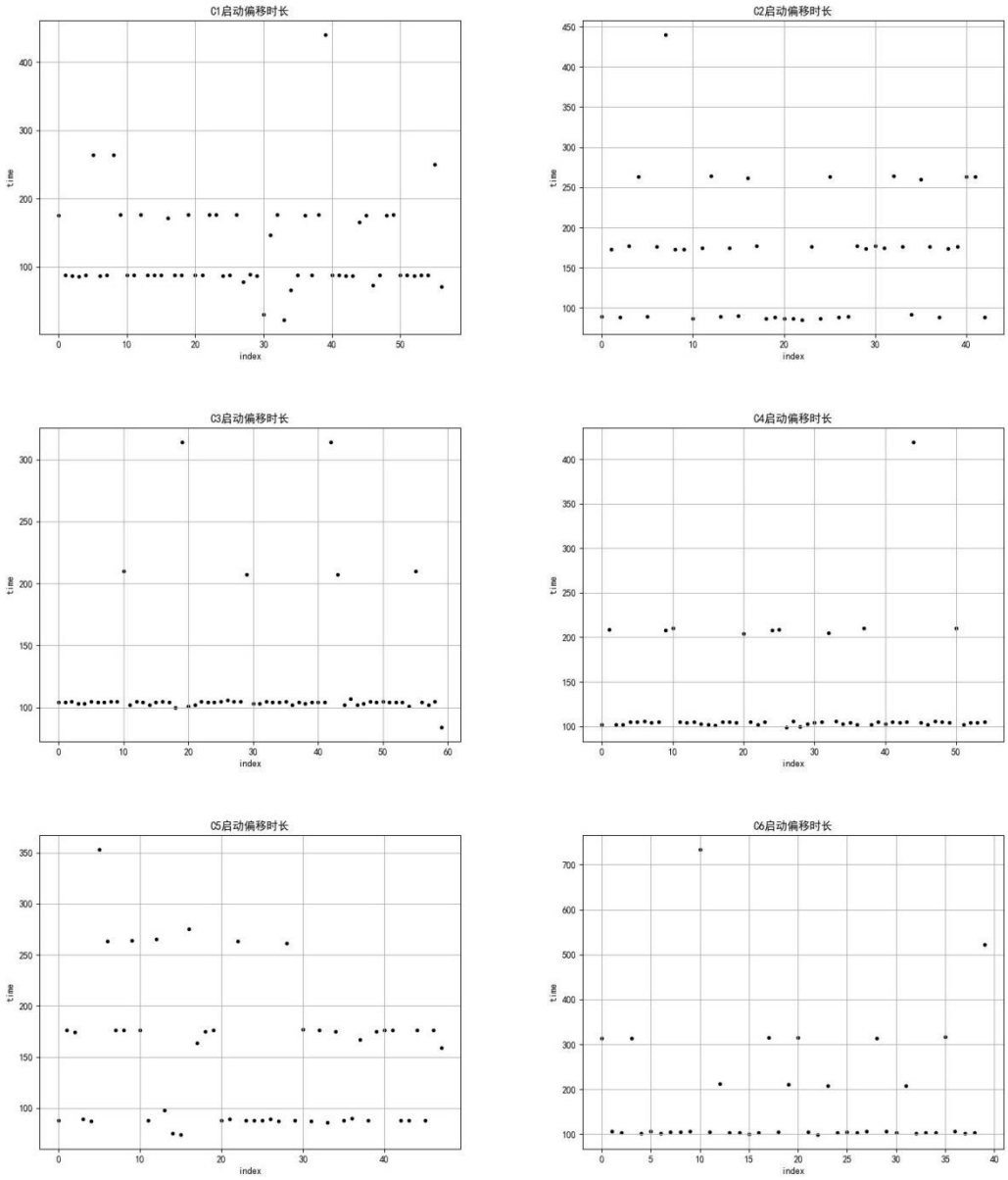

C1启动偏移时长
C2启动偏移时长
C3启动偏移时长
C4启动偏移时长
C5启动偏移时长
C6启动偏移时长
time
index

图 8 启动偏移时长

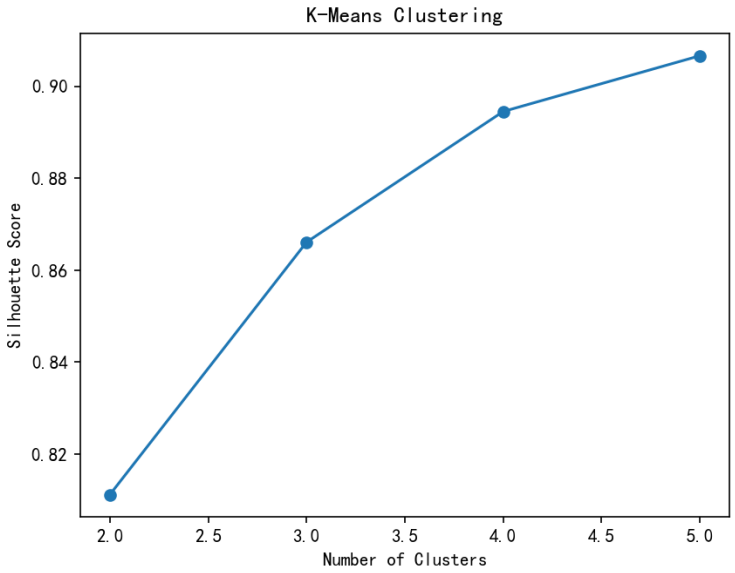
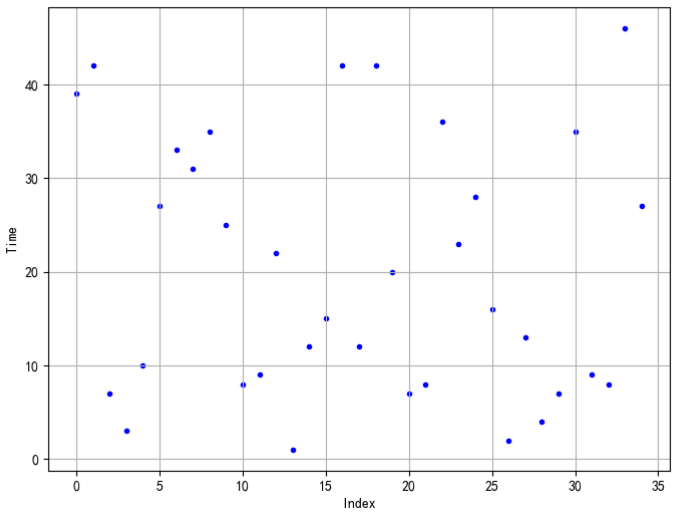


图 9 聚类结果

对时间小于指定阈值的数据做过滤，保留 4 个周期，然后我们分别求解这些周期各自的均值，获得最终每个周期的周期值。



图表 10 红灯时间周期

根据不同的周期的停车时间，进一步利用层级聚类的算法，提取最能代表红灯停车时间的时间作为红灯周期，然后计算平均值即可获得对应不同周期的红灯周期。

实验结果如下表所示：

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	37	58	67	71	46	68
周期 2 绿灯时长（秒）	44	114	171	173	133	182
周期切换时刻	263	243	1186	1186	174	2309
周期 2 红灯时长（秒）	42	61	38	35	41	28
周期 2 绿灯时长（秒）	44	114	171	173	133	182
周期切换时刻	615	686	无	无	无	5254
周期 3 红灯时长（秒）	14	35				38
周期 3 绿灯时长（秒）	245	228				276
周期切换时刻	无	无				无

表格 3 路口 C1-C6 各自一个方向信号灯周期识别结果

七、问题四的模型实践

7.1 问题分析

某路口连续 2 小时内所有方向样本车辆的轨迹数据，根据部分数据可视化需要根据车辆在路口的运动模式或方向划分路段。可以将路口划分为东、东北、北、西北、西、西南、南、东南），对于部分数据可视化如下图。

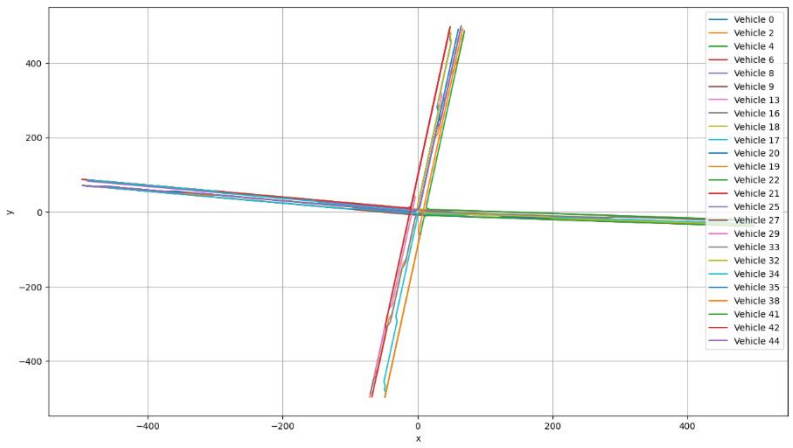


图 11 数据可视化

7.2 模型实践

使用问题三的模型对划分后的数据进行周期计算，由于在实际情况中误差因数较多，可能会出现特殊情况，故对运算的周期进行平均运算与异常值处理。

实验结果如下：

周期	时间(秒)
周期 1	94
周期 2	142

八、模型的评价与推广

8.1 模型的优点

问题一模型融合了分段均值差分算法与聚类算法，模型能更精确地识别并解释数据中的模式，拥有强大的适应性，可以处理静态数据，也能应对动态变化的数据流。此外，模型的结果具有很好的可解释性，使得预测结果易于理解。实现上，由于分段均值差分算法和聚类算法都是已经广泛研究的算法，因此模型的实现相对简单，容易调整和优化。

问题二模型使对模型一的进一步优化，在差分过程中引用聚类算法，能够检测出异常值对于实验结果的影响，在计算红灯周期是由于实验结果因为聚类因数有类别之分，故采取多次实验取众数的方法获取准确的结果。

问题三的模型是对 CUSUM 算法与模型二的结合，结合问题二的聚类思维进行对于不同周期的识别。

8.2 模型缺点及改进

模型的缺点主要集中在对周期性变化较强的时间序列数据的适用性以及参数设定的依赖性上。CUSUM 算法虽然能够捕捉突然变化的异常情况，但对于周期性变化较强的数据，其表现可能不够理想。此外，模型需要事先确定分段的

方式和阈值的大小，这对于不同数据集和场景可能需要不同的调整，增加了使用的复杂性。

针对以上缺点，可以采取一些改进措施：

（1）考虑算法的适用性：针对周期性变化较强的数据，可以考虑采用其他更适合的算法或模型，例如时序分析中的季节性分解、周期性成分分析等方法，以更好地捕捉数据的周期性变化。

（2）参数自适应调整：设计算法时可以考虑引入参数自适应调整的机制，使模型能够根据数据的特征自动调整分段方式和阈值大小，减少人工干预，提高模型的鲁棒性和通用性。

（3）结合其他算法：可以将 CUSUM 算法与其他能够处理周期性变化数据的算法结合起来，形成一个更综合、更有效的异常检测模型。例如，可以将 CUSUM 算法与周期性模型、聚类算法等结合，以充分利用不同算法的优势，提高模型的检测性能和适用范围。

通过以上改进措施，可以进一步提升模型对周期性变化数据的适用性和准确性，减少参数设定的依赖性，从而使模型在实际应用中更加可靠和有效。

8.3 模型的推广

基于行车轨迹预测交通信号灯周期有利于该模型应用于基于行车轨迹的交通信号灯周期预测可以带来多方面的好处和应用前景：

（1）交通流优化： 通过预测交通信号灯的周期，可以更好地调控交通流量，减少拥堵和交通事故发生的可能性，提高道路通行效率。

（2）智能信号控制： 基于行车轨迹的信号灯周期预测可以为智能交通信号系统提供重要依据，实现智能化的信号控制，根据实时交通情况动态调整信号灯周期，优化交通流畅度。

（3）能源节约： 有效的信号灯控制可以减少车辆的停等时间，降低车辆的燃油消耗和尾气排放，从而实现能源节约和环境保护的目标。

（4）数据驱动决策： 通过分析行车轨迹数据，结合预测的信号灯周期信息，可以为交通管理部门和城市规划者提供数据支持，帮助他们制定更科学、更有效的交通管理策略和城市规划方案。

（5）智慧城市建设： 基于行车轨迹的交通信号灯周期预测是智慧城市建设的重要组成部分，能够提升城市交通管理水平，改善居民出行体验，推动城市可持续发展。

通过推广应用该模型，可以实现交通管理的智能化、数据驱动化，为城市交通系统的优化和改进提供重要支持，促进城市交通的可持续发展。

参考文献

[1] 谢鹰. 基于车流量预测的交通信号控制算法[D]. 大连交通大学, 2023.

[2] 秦侨, 杨超, 杨海涛, 等. 结合模糊控制的深度强化学习交通灯控制策略[J]. 计算机应用研究, 2024, 41(01): 165-169.

[3] 李建春, 陶崇瑾. 基于车路协同的智能交通信号灯优化控制策略[J]. 科技创新与应用, 2023, 13(30): 144-147.

[4] 程云欢. 车路协同环境下干线交叉口信号协调控制模型分析[J]. 长江信息通信, 2023, 36(11): 130-133.

[5] 徐明杰, 韩印. 基于粒子群算法下的交叉口信号配时优化[J]. 物流科技, 2020, 43(01): 106-110.

[6] 黄秋实, 王艳阳, 邬昌良, 等. 信号交叉口混行交通协同控制方法[J/OL]. 系统仿真学报: 1-12.

[7] 刘佳佳, 左兴权. 交叉口交通信号灯的模糊控制及优化研究[J]. 系统仿真学报, 2020, 32(12): 2401-2408.

[8] 帅庆珍, 张家铭, 周凤. 基于低频采集数据的城市道路车辆轨迹重构[J/OL]. 交通科学与工程: 1-8.

[9] 邵嘉琪. 基于聚类分析的交通状况研究[D]. 武汉大学, 2022.

[10] 张昀, 李小龙. 基于 DBSCAN 算法的浮动车数据预处理[J]. 江西科学, 2020, 38(03): 293-297+319.

[11] 史帅彬, 张恒, 邓世聪, 等. 基于复合滑动窗的 CUSUM 暂态事件检测算法[J]. 电测与仪表, 2019, 56(17): 13-18.

[12] NCCLight: Neighborhood Cognitive Consistency for Traffic Signal Control[J]. Kong Yan;Cong Shan. Sensors and Materials, 2022.

[13] Multistate-Constrained Multiobjective Differential Evolution Algorithm With Variable Neighborhood Strategy.[J]. Hou Ying;Wu Yilin;Han Honggui.IEEE transactions on cybernetics,2022.

[14] Forecasting of Wind Capacity Ramp Events Using Typical Event Clustering Identification[J]. Li Jiang;Song Tianyu;Liu Bo;Ma Haotian;Chen Jikai;Cheng Yujian.IEEE ACCESS,2020.

[15] Deep reinforcement learning for traffic signal control under disturbances: A case study on Sunway city, Malaysia[J]. Faizan Rasheed;;Kok-Lim Alvin Yau;;Yeh-Ching Low.Future Generation Computer Systems,2020.

[16] A survey on deep reinforcement learning approaches for traffic signal control[J]. Haiyan Zhao,Chengcheng Dong,Jian Cao,Qingkui Chen.Engineering Applications of Artificial Intelligence,2021.

[17] A Hybrid of Deep Reinforcement Learning and Local Search for the Vehicle Routing Problems[J]. Jiuxia Zhao;Minjia Mao;Xi Zhao;Jianhua Zou.IEEE Transactions on Intelligent Transportation Systems,2020.

附录

```
# 问题一代码（数据可视化）

# -*- coding: utf-8 -*-



import matplotlib.pyplot as plt

import pandas as pd
```

```
file_paths = ['../附件/附件一/A1.csv', '../附件/附件一/A2.csv', '../附件/附件一/A3.csv', '../附件/附件一/A4.csv']

fig, axs = plt.subplots(2, 2, figsize=(15, 10))

for i, file_path in enumerate(file_paths):

    df = pd.read_csv(file_path)

    unique_vehicle_ids = df['vehicle_id'].unique()

    first_10_vehicle_ids = unique_vehicle_ids[:10]

    for vehicle_id in first_10_vehicle_ids:

        vehicle_df = df[df['vehicle_id'] == vehicle_id]

        axs[i // 2, i % 2].plot(vehicle_df['time'], vehicle_df['x'], label=f'Vehicle {vehicle_id}')

    axs[i // 2, i % 2].set_title(f'{file_path[2:4]}_plot')

    axs[i // 2, i % 2].set_xlabel('time')

    axs[i // 2, i % 2].set_ylabel('x_location')

    axs[i // 2, i % 2].legend(loc='upper right')  # 放置图例在右上角

    axs[i // 2, i % 2].grid(True)
```

```
plt.tight_layout()

plt.show()

file_path = '../附件/附件一/A5.csv'

df = pd.read_csv(file_path)

unique_vehicle_ids = df['vehicle_id'].unique()

first_10_vehicle_ids = unique_vehicle_ids[:10]

for vehicle_id in first_10_vehicle_ids:

    vehicle_df = df[df['vehicle_id'] == vehicle_id]

    plt.plot(vehicle_df['time'],      vehicle_df['x'],
label=f'Vehicle {vehicle_id}')



plt.title('A5_plot')

plt.xlabel('time')

plt.ylabel('x_location')

plt.legend(loc='upper right')

plt.grid(True)

plt.show()
```

```
# 计算速度
# -*- coding: utf-8 -*-


import pandas as pd
import numpy as np
import warnings


# 忽略特定类型的警告信息
warnings.filterwarnings("ignore")


def calculate_speed(df):
    """
    根据位置和时间计算速度


    :param df: 包含位置和时间信息的 DataFrame
    :return: 速度的 Series
    """
    # 计算速度
```

```
    df["speed"] = np.sqrt((df['x'].diff()) ** 2 +
(df['y'].diff()) ** 2) / (df['time'].diff())


    return df["speed"]


def main():
    # 遍历文件编号
    for num in range(1, 6):
        file_path = f"../附件/附件 1/A{num}.csv"
        df = pd.read_csv(file_path)
        # 按车辆 ID 分组计算速度
        df_speed                                        =
df.groupby("vehicle_id").apply(calculate_speed).reset
_index()
        df_speed = df_speed.sort_values(by="level_1")
        df["speed"]                                     =
df_speed.reset_index(drop=True)["speed"]
        # 将计算后的文件写入相同目录下
```

```
        df.to_csv("../附件/附件 1/" + "speed_" +
str(file_path[7:]), index=False)
        print("已完成", file_path)
if __name__ == '__main__':
    main()


 # 模型一
 # -*- coding: utf-8 -*-


 from sklearn.cluster import DBSCAN,
AgglomerativeClustering
 from matplotlib import pyplot as plt
 from collections import Counter
 from scipy.stats import mode
 import pandas as pd
 import numpy as np
 import warnings
```

```
# 忽略特定类型的警告信息
warnings.filterwarnings("ignore")
# 设置字体
plt.rcParams['font.sans-serif'] = ['SimHei']


def calculate_speed(df):
    """
    根据位置和时间计算速度

    :param df: 包含位置和时间信息的 DataFrame
    :return: 速度的 Series
    """
    # 计算速度
    df["speed"] = np.sqrt((df['x'].diff()) ** 2 + (df['y'].diff()) ** 2) / (df['time'].diff())
    return df["speed"]


def get_start_time(data, threshold, position):
```

```
    """
    获取第一个速度小于阈值的时间点

    :param data: 包含速度信息的 DataFrame
    :param threshold: 速度阈值
    :param position: 位置索引
    :return: 第一个速度小于阈值的时间点
    """
    data = data.sort_values(by='time')
    data = data.dropna(subset=["speed"])
    groups = data[data["speed"] <= threshold].groupby("vehicle_id")
    first_group_key = list(groups.groups.keys())[0]
    first_group = groups.get_group(first_group_key)
    start_time = first_group.iloc[position]["time"]

    return start_time
```

```
 def get_gaode(data, threshold, position=-1):
     """
     使用高德算法获取时间偏移

     :param data: 包含速度信息的 DataFrame
     :param threshold: 速度阈值
     :param position: 位置索引
     :return: 时间偏移列表
     """

     start_time = get_start_time(data, threshold,
position)
     res = []
     for index, group in
enumerate(data.groupby("vehicle_id")):
         vid, g = group
         g = g[g["speed"] <= threshold]
         if len(g) == 0:
```

```
            continue
        time = g.iloc[position]["time"]
        deviation_time = time - start_time
        res.append(deviation_time)
    return res


def scatter(data, x=None, c='b', cmap='Set2',
marker='x', label='DBSCAN Clustering'):
    """
    绘制散点图
    """

    if x is None:
        x = range(0, len(data))
    # 绘制散点图
    plt.figure(figsize=(8, 6))
    plt.scatter(x, data, c=c, cmap=cmap,
marker=marker, label=label)
```

```
    plt.xlabel('Index')
    plt.ylabel('Time')
    plt.title('')
    plt.grid(True)
    plt.show()


def aggregate(data, threshold):
    """
    对结果进行聚合操作，进一步提高精度
    """
    result = []
    sum_val = 0
    count = 0
    for i in range(1, len(data)):
        diff = data[i] - data[i - 1]
        if diff <= threshold:
            sum_val += data[i]
            count += 1
```

```
        else:
            if count > 0:
                avg = sum_val / count
                result.append(avg)
            sum_val = 0
            count = 0

    # 处理最后一段数据
    if count > 0:
        avg = sum_val / count
        result.append(avg)
    return result


def get_stop_time(data, threshold):
    """
    获取停车时间
    """
```

```
    result = data[data["speed"] <= threshold]["vehicle_id"].value_counts().to_dict()
    return result


def get_red_cycle(data, n=5, verbose=True):
    """
    利用聚类计算红灯周期
    """
    x_list = list(data.values())
    agg_clustering = AgglomerativeClustering(n_clusters=n)
    agg_clustering.fit(np.array(x_list).reshape(-1, 1))
    labels = agg_clustering.labels_
    result = {}
    count = Counter(labels)
    for label, x in zip(labels, x_list):
        if result.get(label, None) is None:
```

```
            result[label] = 0
        result[label] += x
    if verbose:
        scatter(x_list, data.keys(), c=labels)
    for k, v in result.items():
        result[k] = v / count.get(k)

    return result[max(result, key=result.get)]
 def main():

    for num in range(1, 6):

        file_path = f"../附件/附件 2/B{num}.csv"
        data = pd.read_csv(file_path)
        df_speed                                        =
data.groupby("vehicle_id").apply(calculate_speed).res
et_index()
        df_speed = df_speed.sort_values(by="level_1")
```

```
        data["speed"] =
df_speed.reset_index(drop=True)["speed"]


        # 利用高德算法获取启动时间点
        res = get_gaode(data, 0.5, -1)


        # 对结果进行聚合，进一步提取周期的差值
        r = aggregate(res, 3)
        diff = np.diff(r)


        scatter(diff)


        stopTime = get_stop_time(data, 0.1)


        scatter(stopTime.keys(), stopTime.values())


        cycle = round(mode(diff)[0])
        red = get_red_cycle(stopTime, 5)
```

```
        rounded_cycle = round(red)


        print(f"周期:{cycle},红灯时间:{rounded_cycle},绿灯时间{cycle - rounded_cycle}", )


 if __name__ == '__main__':
     main()


# 问题二代码


# 模型二
import math
from collections import Counter


import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy.stats import mode
```

```
from sklearn.cluster import DBSCAN, AgglomerativeClustering, KMeans
import warnings
from sklearn.metrics import silhouette_score
warnings.filterwarnings("ignore")
plt.rcParams['font.sans-serif'] = ['SimHei']


def calculate_speed(df):
    """
    根据位置和时间计算速度
    :param df: 包含位置和时间信息的 DataFrame
    :return: 速度的 Series
    """
    # 计算速度
    df["speed"] = np.sqrt((df['x'].diff()) ** 2 + (df['y'].diff()) ** 2) / (df['time'].diff())
    return df["speed"]
```

```
def getStartTime(data, t, pos):
    data = data.sort_values(by='time')
    data = data.dropna(subset=["speed"])
    groups = data[data["speed"] <=
t].groupby("vehicle_id")
    first_group_key = list(groups.groups.keys())[0]
    first_group = groups.get_group(first_group_key)
    start_time = first_group.iloc[pos]["time"]
    return start_time



def getGaoDe(data, t, pos=-1):
    start_time = getStartTime(data, t, pos)
    # start_time+=1
    # print("开始时间", start_time)
    res = []
    for index, group in
enumerate(data.groupby("vehicle_id")):
```

```
        vid, g = group
        g = g[g["speed"] <= t]
        if len(g) == 0:
            continue
        time = g.iloc[pos]["time"]
        deviation_time = time - start_time
        # print(deviation_time)
        res.append(deviation_time)
    return np.array(res)


def scatter(data, x=None, c='b', cmap='Set2',
marker='x', label='DBSCAN Clustering',title=""):
    if x is None:
        x = range(0, len(data))
    # 绘制散点图
    plt.figure(figsize=(8, 6))
    plt.scatter(x, data, c=c, cmap=cmap, marker=marker,
label=label)
```

```
    plt.xlabel('Index')
    plt.ylabel('Time')
    plt.title(title)
    plt.grid(True)
    plt.show()


def getStopTime(data, t):
    r = data[data["speed"] <=
t]["vehicle_id"].value_counts().to_dict()
    return r


# 利用聚类计算红灯周期
def getRedCycle(data, t,n=5, verbose=True):
    stopTime=getStopTime(data,t)
    x_list=np.array(list(stopTime.values()))
    x_var = x_list.var() / len(x_list)
    # 自动确定类别数量
    n = determine_kmeans_clusters(x_list)
```

```
    print("方差:", x_var, "推算类别(未使用):", n)
    print("计算红灯周期的原始数据(停车时间)",x_list)
    res,labels=clusttering(x_list,n=5,type="max")
    if verbose:
        scatter(x_list, stopTime.keys(),
c=labels,title="红灯聚类结果")
    return res


def determine_kmeans_clusters(data,
max_clusters=5,verbose=True):
    silhouette_scores = []
    x_np=data.reshape(-1, 1)
    for n_clusters in range(2, max_clusters + 1):
        kmeans = KMeans(n_clusters=n_clusters)
        cluster_labels = kmeans.fit_predict(x_np)
        silhouette_avg = silhouette_score(x_np,
cluster_labels)
        silhouette_scores.append(silhouette_avg)
```

```
    if verbose:
        plt.plot(range(2, max_clusters + 1), silhouette_scores, marker='o')
        plt.xlabel('Number of Clusters')
        plt.ylabel('Silhouette Score')
        plt.title('Silhouette Score for K-Means Clustering')
        plt.show()
    optimal_clusters = np.argmax(silhouette_scores) + 2
    return optimal_clusters


def clusttering(x_np, n=5,type="max"):
    kmeans = KMeans(n_clusters=n)
    cluster_labels = kmeans.fit_predict(x_np.reshape(-1, 1))
    # 获取每个样本的簇标签
    labels = cluster_labels
    result = {}
```

```
    count = Counter(labels)
    for label, x in zip(labels, x_np.tolist()):
        if result.get(label, None) is None:
            result[label] = 0
        result[label] += x
    # 求平均
    for k, v in result.items():
        result[k] = v / count.get(k)
    if type=="max":
        return   result[max(result,   key=result.get)],
labels
    elif type=="min":
        return   result[min(result,   key=result.get)],
labels
    elif type=="mode":
        index,c=count.most_common()[0]
        # print(index)
        return result[index],labels
```

```
# 根据高德算法的结果计算整个周期
def getCycle(data, t, n,verbose=True):
    gaoRes = getGaoDe(data, t=t, pos=-1)
    x_list=np.diff(gaoRes)
    # x_list=gaoRes
    #过滤一些特别小的值
    x_list=x_list[(x_list>10)]
    x_var=x_list.var()/len(x_list)
    #自动确定类别数量

    n=determine_kmeans_clusters(x_list)
    res, labels = clusttering(x_list, n=n, type="mode")
    print("方差:",x_var,"推算类别:",n)
    print("计算整个周期的原始数据",x_list)
    if verbose:
        scatter(x_list, c=labels,title="整个周期聚类")
    return res
```

```
def main(file_path):
    data = pd.read_csv(file_path)
    df_speed = data.groupby("vehicle_id").apply(calculate_speed).reset_index()
    df_speed = df_speed.sort_values(by="level_1")
    data["speed"] = df_speed.reset_index(drop=True)["speed"]

    cycle = math.ceil(getCycle(data, t=0.5,n=5))

    red = math.ceil(getRedCycle(data,t=0.5, n=5))
    print("-------------" + file_path + "-------------")
    print("周期:", cycle)
    print("红灯周期:", red)
    print("绿灯周期:", cycle - red)
```

```
    print("-----------------------------------------
")

main("附件/附件 2/B1.csv")
```