赛区评阅编号（由赛区组委会填写）：

2024 高教社杯全国大学生数学建模竞赛

承　诺　书

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》（以下简称“竞赛章程和参赛规则”，可从 http://www.mcm.edu.cn 下载）。

我们完全清楚，在竞赛开始后参赛队员不能以任何方式，包括电话、电子邮件、“贴吧”、QQ 群、微信群等，与队外的任何人（包括指导教师）交流、讨论与赛题有关的问题；无论主动参与讨论还是被动接收讨论信息都是严重违反竞赛纪律的行为。

我们完全清楚，在竞赛中必须合法合规地使用文献资料和软件工具，不能有任何侵犯知识产权的行为。否则我们将失去评奖资格，并可能受到严肃处理。

我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。

我们授权全国大学生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号（从 A/B/C/D/E 中选择一项填写）：1

我们的报名参赛队号（12 位数字全国统一编号）：226

参赛学校（完整的学校全称，不含院系名）：南开大学

参赛队员 (打印并签名)：1. 周钰宸

2. 纪潇洋

3. 石若川

指导教师或指导教师组负责人 (打印并签名)：

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期：2023 年 08 月 13 日

(请勿改动此页内容和格式。此承诺书打印签名后作为纸质论文的封面，注意电子版论文中不得出现此页。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。)

赛区评阅编号:
（由赛区填写）_____

全国评阅编号:
（全国组委会填写）_____

2024 高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅记录（可供赛区评阅时使用）:

评阅人						
备注						

送全国评阅统一编号:
（赛区组委会填写）

（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）

基于机器学习预测模型的共享单车数据定位分析

摘要

针对问题一，我们首先对原始数据集进行了包括日期转换、时间筛选、经纬度转换、城市的确定等预处理。在确定了城市为北京后，使用 Python 对数据进行初步筛选，根据题目要求将数据分为早高峰，晚高峰的两类。然后借助 Navicat 工具使用 **SQL 语言**，查询到了获取不同时段的相同地点作为起始和终止点的热力值。最后利用这些数据，使用 **DBSCAN 聚类方法**，将纬度与经度作为 X 和 Y 轴，地点热力值作为 Z 轴，通过不断调整参数，获得了可视化的三维立体聚类结果。在各个聚类结果中依次取出前 10 类，最终得到了早高峰和晚高峰的热门区块，并分别进行了定位。

针对问题二，我们引入了机器学习预测模型，首先将 train 集中所用到的数据进行清洗，以 userid、起始经度、起始纬度、是否是工作日、时段这五个变量作为训练的特征 X，以其终点的经纬度作为训练的标签 Y，应用了 **KNN 算法**与**随机森林算法**，对上述模型进行训练，寻找 X 和 Y 的函数关系，最终预测得到 test 集中所有用户的终点经纬度，再根据问题一中的“热门区块”进行筛选，用聚类的方式描述热门区块的主要目的地，并用 Python matplotlib 库进行可视化，得出结论。

针对问题三，我们首先根据问题一中得到的热门区块，结合问题二的预测模型，预测出 test 数据集中的早高峰和晚高峰的热门区块的主要目的地区块，然后筛选出每个区块在早高峰开始前到晚高峰结束这一周期内的共享单车净增量；以不同区块间的运输成本作为人工调节的主要成本，建立**多目标整数规划模型**：即在满足居民出行需求的前提下尽可能降低运输成本。最终我们通过 **Matlab** 求解模型得出最优人工调节方案：如将区块 3 的 860 辆车转移至区块 8，211 辆车转移向区块 2；将区块 4 的 457 辆车转移至区块 9，351 辆车转移至区块 8 等。

关键字：MySQL　DBSCAN 聚类　KNN 算法　随机森林算法　机器学习　多目标整数规划

一、 问题重述

1.1 问题的背景及意义

共享单车作为城市短途出行的一种便捷方式，已经在全球范围内取得了广泛的普及。作为一种创新型的城市短途出行方式，共享单车具有诸多显著的优势。首先，共享单车为城市居民提供了便捷的最后一公里解决方案，缩短了从交通枢纽到目的地的距离，实现了出行的无缝连接。其次，共享单车具有环保特点，鼓励人们选择绿色出行方式，减少了汽车尾气排放，有助于改善城市空气质量和缓解环境污染。此外，共享单车的灵活性和低成本使得用户能够随时随地租赁和还车，大大提高了出行的自由度和经济性。

但随着共享单车数量的增加，在早晚高峰期间出现了共享单车过度集中于某些热门区块的问题，对市民出行和城市管理造成了不便。因此研究早晚高峰共享单车的分布情况并进行预测，具有以下几方面的意义:

- 交通拥堵缓解：早高峰时段道路交通常会出现拥堵，这不仅会影响个人出行效率，还会造成环境污染和能源浪费。通过研究在这些区块投放共享单车，可以鼓励更多人选择绿色出行方式，减少汽车使用，从而在一定程度上缓解交通拥堵问题。
- 资源利用优化：通过人工调节共享单车数量，可以更精准地配置单车资源，避免过度投放或不足的情况发生。这有助于提高共享单车的使用效率，降低维护成本，为用户提供更好的出行体验。
- 城市规划优化：研究共享单车在早高峰时段的使用情况，以为城市规划者提供有关城市交通流量和人流分布的重要数据。这些数据可以用于优化交通规划，改善道路设计，提升城市整体运行效率。
- 环保减排：汽车尾气排放是城市空气质量恶化的主要原因之一。使用共享单车作为代步工具可以减少尾气排放，改善城市空气质量，有助于应对气候变化和环境问题。

1.2 需要解决的问题

利用提供的共享单车数据，完成以下任务:

1. 根据 train 数据集分别找到早高峰时间段（7 点到 9 点）与晚高峰时间段（17 点到 19 点）的热门区块（前 10）。
2. 根据第一问的结果以及数据，预测 test 数据集中热门区块早高峰时间段内骑行的目的地所在区块。

3. 根据前两问结果以及数据判断是否需要人工调节热门区块单车数量，并考虑在满足人群出行需求的条件下以最低成本进行调节。

二、 问题分析

2.1 问题一分析

针对问题一，我们首先将数据进行了预处理,包括日期转换、时间筛选、经纬度转换以及城市的确定。在确定了题目中所述的城市就为北京市后，我们采取进一步分析。由于最终目标需要分别确定早晚高峰的起始和终止的热力区块，因为我们先使用 python 对数据分为两类：早高峰和晚高峰数据。

在这之后，由于数据量过大的原因，我们使用 txt 格式将文件导入 Navicat 中，借助 SQL 查询语言，分组查询并创建视图，，以由多少条与之相关的记录作为衡量一个地点热力值的指标，得到了早晚高峰的不同时间段的相同地点分别作为起始点和终止点的热力值与一个初步的以经纬度点的热力值作为排行的“热门点”排名。

由于题目中所要求的是热门区块，因为我们所求得的单独用经度和维度确认的“热门点”并不足够准确，为了能够更好找到热门区块，我们选择使用聚类的方法进行处理。在聚类算法的选择方面，我们考虑到热门区块需要考虑两个因素：“热门”即热力值需要相近且较高的数据聚为一类，“区块”代表着邻近的地点要聚为一类并且区块形状可能为任意形状。

同时考虑到上述两个因素后，我们决定使用基于密度的空间聚类算法：DBSCAN 算法进行聚类。其中经度与纬度作为 X 和 Y 轴，地点热力值作为 Z 轴，通过不断调整参数，获得最佳的聚类结果后，再分别挑选出早高峰和晚高峰中最热门的十个聚类，即热门区块。最后使用 Python 的 matplotlib 库进行可视化展示，并对实际的北京城市的对应地点进行定位，至此完成问题一的全部工作。

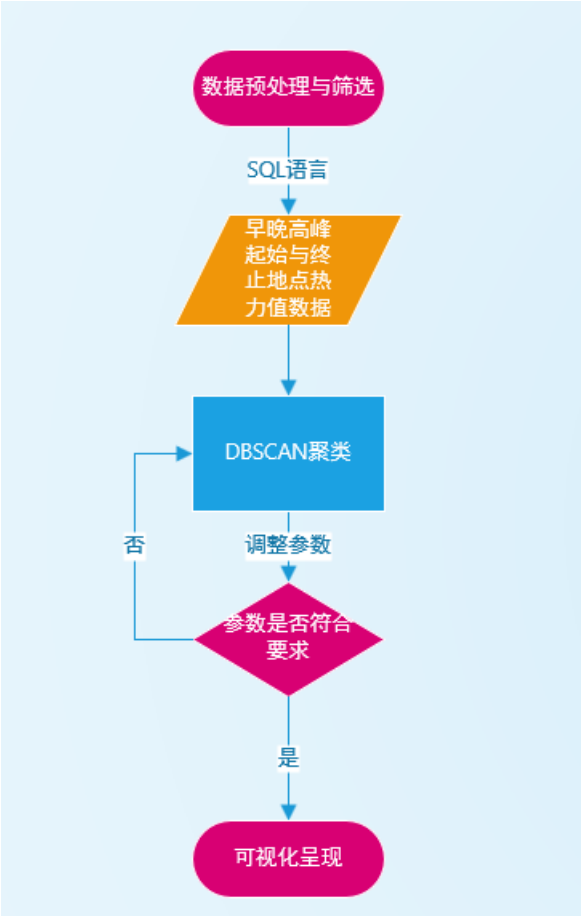


图 1　问题一思路流程图

2.2 问题二分析

针对问题二，我们首先注意到 test 数据集相对于 train 数据集的异同点：test 集没有终止点的经纬度坐标，而其余指标两个数据集均相同。因此考虑到要借助机器学习的方法，以两个数据集相同的指标作为特征，以待预测的指标（终点的经纬度坐标）作为标签，利用相关算法，训练一个预测模型。

在算法的选择方面，我们考虑到随机森林算法能够处理大量的数据，对于共享单车位置预测这种需要考虑多个因素的问题尤为适用；KNN 算法适用于共享单车位置预测这种需要考虑周围环境和近邻特征的问题，能够捕捉到空间相关性。因此我们选择这两种算法分别进行预测，将结果进行对比分析。

在特征指标的筛选方面，我们考虑到了时间和空间两个维度。在时间上，人们在工作日和节假日的早晚高峰时段的出行需求显然不同，因此我们选择将数据集中的时间列进行 0-1 分类，分别代表工作日和节假日，同时在一天之中不同时段的出行需求量也不尽相同，需要对小时的指标进行筛选；在空间上，经度和纬度是位置差异的直接体现，因此选取上述指标，作为自变量进行训练。

预测结束后，我们得到所有点的预测目的地的经纬度，为了准确形象地描述其时空

分布特征，我们采取 K-means 聚类的方法，将问题一中涉及到的热门区块中的点筛选出来，将其去向的经纬度坐标进行聚类，划分出主要的区块，利用 Python 的 matplotlib 库进行可视化展示，至此完成问题二的全部工作。

2.3 问题三分析

针对问题三，我们首先考虑到热门区块的多变性，在不同时段的热门区块可能不同，为了方便分析，我们只取问题一中的早高峰起始地点作为热门区块，并利用问题二的预测模型，预测出 test 数据集中的早高峰和晚高峰的热门区块的主要目的地区块，并将 test 数据集的起始位置和终止位置作为主要分析指标。

接着，我们筛选出问题一中所涉及到的热门区块在 test 集所预测的目的地，并且统计出所有区块间的车辆移动情况，以二维矩阵的形式体现，最后计算出每个区块在早高峰开始前到晚高峰结束这一周期内的共享单车净增量。

考虑到人工调节的成本主要为运输成本，因此我们以两点间的距离为主要衡量标准，计算出是热门区块中心之间两两的直线距离，得到成本矩阵，至此数据准备完成。

最后，我们结合上述多个因素建立多目标规划模型：即在满足居民出行需求的前提下尽可能降低运输成本。可通过 **Matlab** 进行模型的求解。

三、 模型假设

1. 假设热门区块的定义是在早晚高峰时间段内有较多骑行起始点或终点的区块。
2. 假设人工调节热门区块单车数量的目的是保持骑行需求和供应之间的平衡，减少用户等待时间和满足用户出行需求。
3. 假设在一天之中只进行一次共享单车的人工调配，并在早晚高峰之间进行。
4. 假设每次共享单车的人工调配在极短时间内完成。
5. 为简化模型，假设人工调配均在热门区块的中心位置完成。
6. 假设共享单车无损坏和废弃。
7. 假设用户每次完成共享单车骑行后，均完成关锁。
8. 假设共享单车在某一城市中的数量是恒定的，没有外地共享单车进入，也没有本地共享单车离开。
9. 假设区别于传统的公共自行车系统，共享单车投放区域没有停车桩、卡槽等装置，只要用户手机终端的定位在预设区域内则认定还车成功，所以模型中定义的投放区域无容量限制。

四、 符号说明

符号	说明
S	投放区域的集合
$y_i(t)$	t 时段在区域 i 的共享单车初始数量
$z_{ij}(t)$	t 时段从区域 i 到区域 j 人工调配的自行车数量
$q_{ij}(t)$	t 时段区域 i 到 j 的自行车出行量
d_{ij}	i 和 j 两区域中心的距离
c_d	人工调配一辆共享单车的成本
$\alpha_{ij}(t)$	判断 t 时段是否从区域 i 到区域 j 进行人工调配

五、 数据预处理

本题提供的附件数据中，未发现数据缺失和数据异常的情况，因此未进行该方面的数据处理。数据预处理包括了一下几个方面。

5.1 日期转换

在研究哪些因素与用户的共享单车使用情况相关时，我们需要考虑到时间特征，即是否为工作日。我们将 2017 年的所有节假日提取出来放在列表中，然后利用分词提取时间这一列的日期部分，将其转化为”0：工作日“或“1：节假日”，以便后续处理。

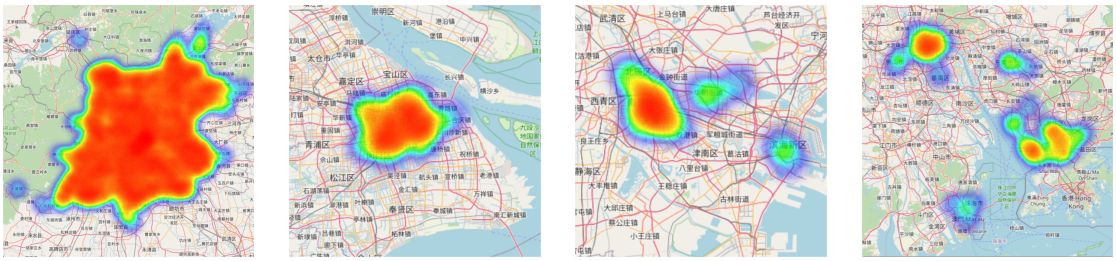
5.2 时间筛选

由于本题主要研究的为早晚高峰时段的共享单车使用情况，所以我们将骑行起始时间在 7:00-9:00 区间的数据作为为早高峰数据，骑行起始时间在 17:00-19:00 区间的数据作为晚高峰数据。

5.3 经纬度转换

附件中经纬度数据经过了 Geohash 加密。Geohash 是一种地理编码系统，用于将地理位置坐标（经度和纬度）转换为一串字符，通过字符串来表示一个特定的地理位置。我们利用 Python 中 geohash2 库对 Geohash 编码进行解码，得到了对应的经纬度数据。

图图 12展示了根据解码后经纬度绘制的部分地区共享单车热力图，可以看出附件中共享单车的使用地点主要集中于中国各主要城市的城区。



(a) 北京市　(b) 上海市　(c) 天津市　(d) 珠三角地区

图 2　部分地区共享单车使用情况热力图

5.4 城市的确定

根据题干所述，某品牌单车在某城市的单车投放量已经超过 40 万。在对附件中车辆 ID 进行统计后发现，只有北京地区（39.4°N-41.6°N，115.7°E-117.4°E）的共享单车数超过 40 万，为 440671 辆。因此可以推断题目所指的城市为北京。

六、 问题一的模型建立与求解

6.1 数据筛选与统计

在对数据进行预处理之后，我们已经得到了北京地区时间和经纬度转化完成的 train 数据集了，在这之后，为了进一步满足题目条件，我们首先需要将早高峰（7 点到 9 点）和晚高峰（17 点到 19 点）的数据区分开来。这部分我们计划使用 python 解决。

除此之外，我们意识到比如某居民区可能在早高峰人们出行上班工作时会成为一个热门的起点，但并不是一个热门的终点；同理，某工作区可能在早高峰人们工作结束返回居民区时成为一个热门的终点，但并不是一个热门的起点。因此我们想到再进行了初步的时间段筛选后，有必要通过每条记录的起始点和终止点再进一步区分。即总体将数据分为四类来处理：早高峰起始地、早高峰终止地、晚高峰起始地、晚高峰终止地。

在这之后，我们还需要有对一个地点是否应该归为热力区块进行一个判断，即选取一个指标作为其衡量指标，我们选取的是骑行记录。在这之后，我们使用 SQL 查询语言进行统计得到目标的四组数据。

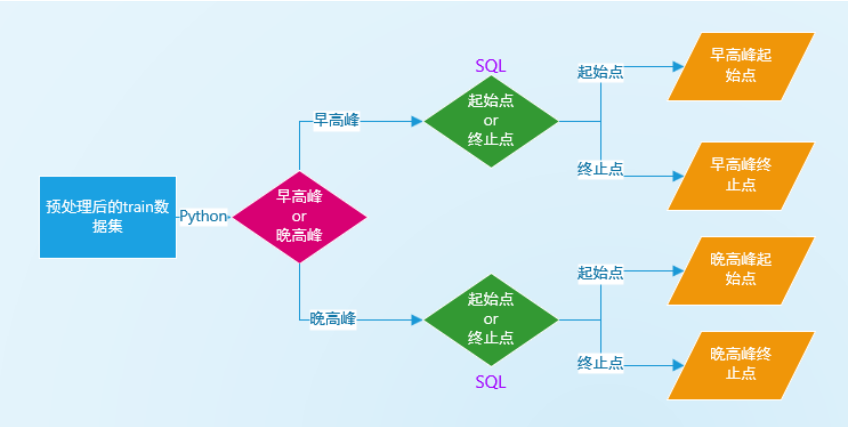


图 3　数据筛选与统计思路

6.1.1 时间段筛选

对于数据时间段的筛选，我们使用 python 的 dataframe 库，将数据通过 todatetime 函数转换为 datetime 类型后，使用 setindex 和 betweentime 函数将早晚高峰的时间分开筛选出来。具体 Python 代码详见附件。

表 1　早晚高峰 train 数据分布情况

	早高峰	晚高峰	总计
记录数	601171	563481	1164652
百分比（%）	51.62%	48.38%	100%

如上图所示，train 数据集经过筛选后，得到的早高峰和晚高峰数据量分布如上表所示，几乎均等，因此筛选较为成功。

6.1.2 热力值统计

在我们分开筛选出了早高峰和晚高峰数据后，我们意识到比如某居民区可能在早高峰人们出行上班工作时会成为一个热门的起点，但并不是一个热门的终点；同理，某工作区可能在早高峰人们工作结束返回居民区时成为一个热门的终点，但并不是一个热门的起点。因此有必要通过每条记录的起始点和终止点再进一步区分。即总体将数据分为四类来处理：早高峰起始地、早高峰终止地、晚高峰起始地、晚高峰终止地。

统计的目的是分别统计出早高峰起始地、早高峰终止地、晚高峰起始地、晚高峰终止地这四组地点的热力值，以为后面聚类获取热门区块打下基础。而热力值的统计我们需要一个指标来衡量，我们选择以该地作为起始或终止地的骑行记录数量来衡量。

在选取了合适的衡量指标后，我们选择使用 SQL 语言进行统计，原因是 SQL 是一种结构化查询语言，它可以用来对数据库中的数据进行高效的查询和统计分析。

- 高效性：SQL 语言可以快速地对大量数据进行统计分析，无需编写复杂的程序代码。
- 灵活性：SQL 语言支持多种统计函数和查询语句，可以满足各种复杂的统计需求。
- 易用性：SQL 语言具有良好的可读性和简洁的语法，易于学习和使用。
- 可扩展性：SQL 语言可以与其他编程语言结合使用，实现更加复杂的统计分析功能。

由于本次我们需要处理并统计的数据量级达到了百万量级，因此我们选择使用 SQL 语言作为统计工具。

我们将 6.1.1 中筛选得到的数据放入 Navicat 中，然后使用 SQL 查询语言，主要使用分组和聚合操作进行统计。以某地为起始或终点进行 group by 分组后，使用 count(*) 进行统计，最后得到如下表所示的四组数据：

	早高峰	晚高峰	总计
起始点	3382	3667	7049
终止点	3367	3633	7000

表 2　早晚高峰热门地点统计结果

除此之外，我们还使用了视图，得到了早晚高峰热门地点的排行榜作为一个初步的参考，具体结果详见支撑材料。

6.2 DBSCAN 聚类

6.2.1 DBSCAN 聚类方法介绍

1. DBSCAN 简介：

   DBSCAN（Density-Based Spatial Clustering of Applications with Noise）是一种基于密度的聚类算法，可以发现任意形状的簇，并且能够在噪声数据的情况下不受干扰地识别出核心对象。其主要适用于如下几种情况：

   - 数据分布具有密度区别：对于密度高的数据点，会被分为一个簇；而对于密度低的数据点，会被认为是噪声点。
   - 数据分布多样化：DBSCAN 能够发现任意形状的簇，不需要预先知道簇的数量和形状。
   - 噪声数据较多：DBSCAN 能够正确区分噪声数据和有效数据，有效避免噪声数据对聚类结果的影响。

- 聚类效果需求较高：相对于传统的 K-Means 聚类算法，DBSCAN 能够处理复杂的非球形簇，并且具有更好的鲁棒性。

因此，如果在聚类任务中，数据分布具有明显的密度区别，且需要发现任意形状和数量的簇，同时希望聚类结果具有更好的鲁棒性，那么就可以考虑使用 DBSCAN 聚类算法。然而，对于高维度和高斯分布的数据，DBSCAN 表现不如传统的聚类算法，并且需要合理设置参数才能得到好的聚类效果。

2. DBSCAN 算法步骤:

- 选择一个未访问的数据点作为起始点，找到该数据点的半径范围内的所有数据点。
- 如果半径范围内的数据点数量大于等于阈值 minsamples，则认为这些数据点属于同一个簇，并进行标记。如果数量小于阈值，则将该数据点标记为噪声点。
- 对于半径范围内的所有数据点，递归地进行相同的操作，直到所有相邻的点都被访问。
- 将未被访问的数据点作为起点，重复执行上述步骤，直到所有数据点都被访问。
- 所有被标记为同一簇的数据点构成一个簇，所有未被标记的数据点构成噪声点。
- 最终得到若干个簇和若干个噪声点，可以通过可视化方法来展示聚类结果。

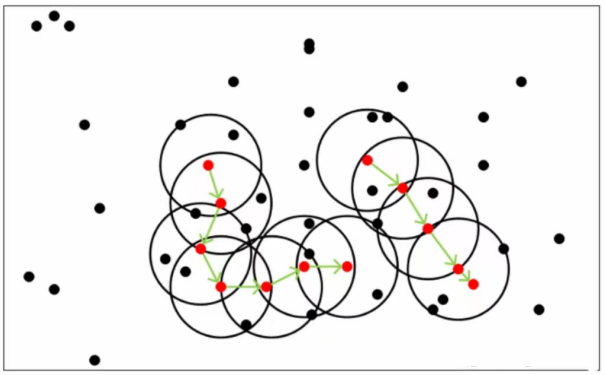


图 4　DBSCAN 算法思想示意图

6.2.2 框架修正与参数调整

由于我们从 6.1 中筛选和统计得到的数据分布具有非常明显的密度区别，有些地点的热力值较高而其它地点较低；同时数据分布多样化，热门区块的数量和形状都不能够实现确定；最后，为了能够使得后两问能够更好的使用第一问通过聚类获取的热门区块，因此我们对聚类效果需求较高，而 DBSCAN 能够处理复杂的非球形簇，并且具有更好的鲁棒性。**综上所述：我们使用 DBSCAN 方法能够较好的达成我们想要通过聚类得到早晚高峰的热门区块的目的，因此我们选择该方法进行聚类。**

1. 框架修正：于是我们使用 6.1 得到的早高峰起始地、早高峰终止地、晚高峰起始地、

晚高峰终止地四组数据，通过使用 DBSCAN 框架，并稍作调整：我们将 X 轴和 Y 轴分别表示纬度和经度，令 Z 轴表示地点的热力值，以三个维度进行聚类。这样做的好处是：我们考虑到热门区块需要考虑两个因素："热门" 即热力值需要相近且较高的数据聚为一类，"区块" 代表着邻近的地点要聚为一类并且区块形状可能为任意形状。因此我们综合上面两点进行聚类，进行框架的修正。

2. 参数调整：使用 sklearn 中的 DBSCAN 模型来实现 DBSCAN 聚类算法，需要调整两个参数：eps 和 minsamples。eps 表示样本点的邻域半径；minsamples 表示样本点在 eps 半径内的最小数量。这两个参数的值至关重要，决定着我们聚类的效果，而具体聚类的效果又需要通过如下三个指标进行衡量：Silhouette score、Calinski-Harabasz score 和 Davies-Bouldin Index。其中:

   - Silhouette score 的取值范围为-1 到 1。当值接近 1 时，表示聚类效果较好，即簇内距离较小，簇间距离较大。当值接近-1 时，表示聚类效果较差，即簇内距离较大，簇间距离较小。当值接近 0 时，表示聚类效果一般。
   - Calinski-Harabasz score 的取值范围为 0 到正无穷。值越大，表示聚类效果越好。
   - Davies-Bouldin Index 的取值范围为 0 到正无穷。值越小，表示聚类效果越好。

   我们对四组数据的 eps 和 minsamples 两个参数进行多次调整，并结合以上三个指标进行评估，最后发现当 eps=5.5, minsamples=2 左右时，聚类效果最好：能得到 50 左右个聚类簇，并有较高的三个指标得分。具体得分和分簇结果详见附件和支撑材料。

6.2.3 可视化结果展示与分析

在使用 DBSCAN 进行了聚类后，我们使用 python 的 matplotlib 库进行可视化的三维展示。同时我们使用平均热力值（即每个聚类簇内所有点热力值加和求平均）作为一个区块（类簇）的总得分，得分越高的类簇排名越高，我们依次取出早高峰起始地、早高峰终止地、晚高峰起始地、晚高峰终止地四组数据中聚类结果的前 10 名类簇作为题目要求的 "早高峰时间段（7 点到 9 点）与晚高峰时间段（17 点到 19 点）的热门区块（前 10）"。表3展示了四个类别下的热门区块具体组成，图6和7展示了四组数据的聚类情况，图8展示了四个类别下的热门区块在地图上的具体位置并标注了每个热门区块的中心。

类别	排序	中心位置	区块组成
早高峰起始地	1	(39.88 116.45)	(39.85 116.39)(39.92 116.52)
	2	(39.85 116.43)	(39.88 116.48)(39.84 116.46)(39.88 116.48)(39.85 116.37)
	3	(39.91 116.41)	(39.89 116.48)(39.94 116.35)
	4	(39.84 116.39)	(39.80 116.42)(39.89 116.36)
	5	(39.93 116.32)	(39.86 116.43)(40.03 116.32)(39.91 116.21)
	6	(39.90 116.44)	(39.94 116.44)(39.84 116.38)(39.92 116.51)
	7	(39.90 116.39)	(39.88 116.37)(39.93 116.33)(39.91 116.49)
	8	(39.92 116.44)	(39.88 116.46)(39.98 116.42)(39.96 116.44)(39.87 116.44)
	9	(39.94 116.33)	(39.97 116.42)(39.91 116.25)
	10	(39.86 116.46)	(39.93 116.32)(39.91 116.53)(39.76 116.55)
早高峰终止地	1	(39.895 116.53)	(39.91 116.58)(39.88 116.48)
	2	(39.90 116.47)	(39.92 116.49)(39.89 116.46)
	3	(39.88 116.49)	(39.85 116.39)(39.91 116.60)
	4	(39.97 116.39)	(40.08 116.36)(39.86 116.42)
	5	(40.00 116.32)	(39.93 116.33)(40.07 116.32)
	6	(39.88 116.37)	(39.93 116.30)(39.86 116.38)(39.86 116.43)
	7	(39.92 116.53)	(39.91 116.47)(39.93 116.6)
	8	(39.91 116.44)	(39.84 116.43)(39.98 116.37)(39.92 116.52)
	9	(39.92 116.40)	(39.92 116.58)(39.93 116.28)(39.91 116.46)(39.89 116.36)(39.90 116.51)(39.97 116.3)(39.96 116.37)
	10	(39.86 116.43)	(39.93 116.31)(39.89 116.44)(39.78 116.56)
晚高峰起始地	1	(39.92 116.48)	(39.92 116.52)(39.92 116.44)
	2	(39.94 116.44)	(39.92 116.56)(39.88 116.46)(40.04 116.31)
	3	(39.88 116.34)	(39.86 116.42)(39.91 116.27)
	4	(39.89 116.42)	(39.85 116.38)(39.93 116.46)
	5	(39.92 116.44)	(39.91 116.46)(39.95 116.37)(39.92 116.49)
	6	(39.92 116.52)	(39.93 116.6)(39.92 116.45)
	7	(39.85 116.44)	(39.87 116.5)(39.84 116.39)
	8	(39.93 116.31)	(39.93 116.44)(39.93 116.18)
	9	(39.94 116.326)	(39.94 116.19)(39.98 116.37)(39.90 116.42)
	10	(39.88 116.37)	(39.91 116.36)(39.86 116.39)
晚高峰终止地	1	(39.85 116.42)	(39.85 116.43)(39.85 116.41)
	2	(39.92 116.40)	(39.94 116.35)(39.91 116.46)
	3	(39.91 116.23)	(39.91 116.19)(39.91 116.27)
	4	(39.95 116.47)	(39.99 116.47)(39.91 116.48)
	5	(39.87 116.32)	(39.90 116.24)(39.84 116.4)
	6	(39.89 116.48)	(39.92 116.45)(39.86 116.51)
	7	(39.87 116.45)	(39.87 116.48)(39.88 116.48)(39.86 116.4)
	8	(39.89 116.48)	(39.92 116.50)(39.87 116.47)
	9	(39.92 116.38)	(39.84 116.29)(40.00 116.47)
	10	(39.92 116.29)	(39.90 116.22)(39.98 116.32)(39.98 116.34)(39.82 116.29)

表 3　各类别下的热门区块

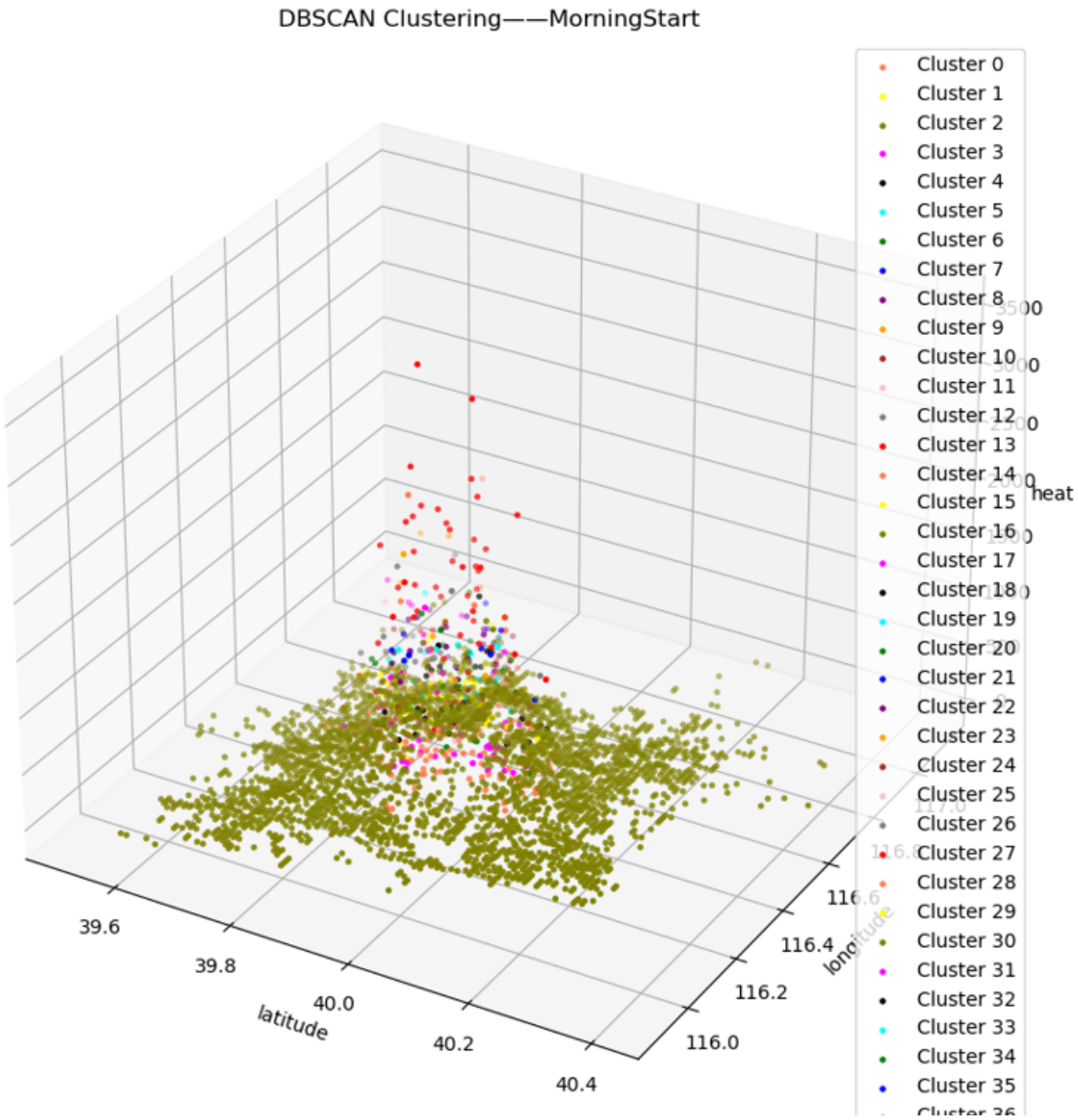
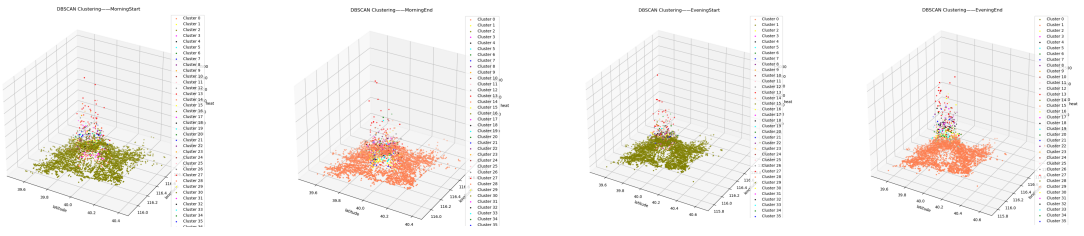
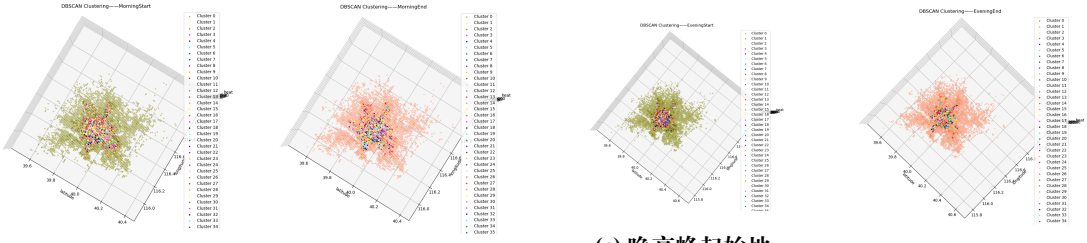


图 5　早高峰起始地三维正视图



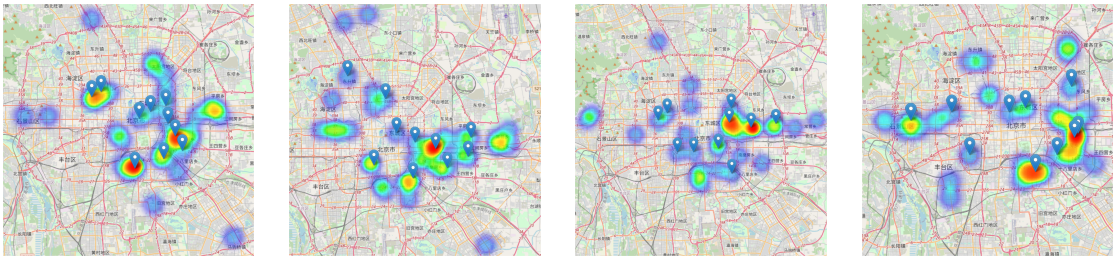
(a) 早高峰起始地　(b) 早高峰终止地　(c) 晚高峰起始地　(d) 晚高峰终止地

图 6　早晚高峰热门区块三维可视化正视图



(a) 早高峰起始地　(b) 早高峰终止地　(c) 晚高峰起始地　(d) 晚高峰终止地

图 7　早晚高峰热门区块三维可视化俯视图



(a) 早高峰起始地　(b) 早高峰终止地　(c) 晚高峰起始地　(d) 晚高峰终止地

图 8　早晚高峰热门区块

七、 问题二的模型建立与求解

7.1 特征指标的筛选

我们知道，影响共享单车的需求量的因素有很多，其需求量与人类活动的规律紧密相关。我们大致可以从时间和空间两个维度去考虑。

首先，从时间维度上讲，在工作日人们的出行需求较大，尤其是中短距离的形成对共享单车提出了较大需求量；而在节假日，居民活动较为分散，因此对于共享单车的需求量不会出现如工作日一样的激增。在工作日中，早高峰与晚高峰早高峰时间段对于共享单车的需求量尤其突出，我们可以假设在早高峰其目的地主要是通勤或上学的地点，例如办公楼、学校、交通枢纽等地点；在晚高峰时间段内的骑行目的地主要是回家或娱乐休闲的地点，例如住宅区、购物中心、公园等地点。可以预见，用户在一天的行程具有弱中心对称性，即早高峰的起始地与晚高峰的目的地大致吻合，具有很强的统计学特征。据此我们可以自然得出要将“是否为工作日”、“是否是早高峰或晚高峰”作为两个重要时间特征进行训练。

其次，在空间维度上来说，经纬度是描述一个物体的位置的最精准的指标，因此我们选取了 train 数据集中每个地点的经度和纬度作为空间特征投入训练。

由于在数据集中，userid 是一个用户的唯一标识，因此我们为了探究每个用户的出行习惯，需要将 userid 作为特征投入训练，以训练出该用户在时间和空间因素的影响下

的目的地坐标。而 orderid 与 biketype 在本次训练中无明显作用，故舍弃。

综上，最终选取 userid, 起始经度、起始纬度、是否工作日、时段这五个特征作为训练指标，终止经度、终止纬度作为训练标签。

7.2 机器学习模型的建立

7.2.1 随机森林预测模型

随机森林算法（Random Forest）是一种基于决策树的集成学习算法，常用于分类和回归问题。其原理是通过构建多个决策树，并综合这些决策树的结果进行预测。

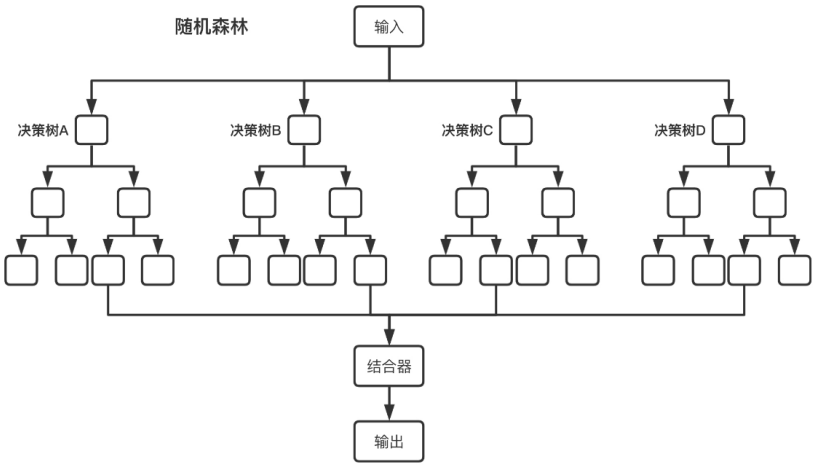


图 9　随机森林算法示意图

其步骤如下：

step1: 随机从训练集中有放回地抽取一定数量的样本，筛选出 train 数据集中早高峰时间段内的数据, 构成一个新的训练子集。

step2: 针对每个子集，随机选择部分特征进行树的构建。

step3: 对于每个树，通过递归将数据集分割成不同的节点，使得每个节点上的样本尽可能相似。

step4: 在每个节点上根据某种准则选择最优的切分特征和切分点，使得切分后的子集的纯度最大化。

step5: 重复步骤 2-4，构建多棵树, 通过平均值确定预测结果。

其优势为：

1. 可以有效处理高维数据和大规模数据集, 对于本题百万量级的数据非常适合。
2. 能够评估特征的重要性，可用于特征选择, 本题涉及多个特征，能够利用随机森林挑选重要的特征，使模型更加鲁棒。

表 4　随机森林算法部分结果

userid	起点纬度	起点经度	终点纬度	终点纬度
1	39.92	116.36	39.937	116.377
2	39.91	116.31	39.943	116.383
5	40	116.48	39.923	116.37
6	39.83	116.62	39.923	116.37
7	39.92	116.42	39.948	116.379
9	39.98	116.42	39.924	116.351
12	39.93	116.47	39.972	116.409
13	39.93	116.17	39.939	116.357
16	39.89	116.27	39.962	116.379
20	39.97	116.29	39.933	116.434

7.2.2 KNN 预测模型

KNN 算法（K-Nearest Neighbors）是一种监督学习算法，常用于分类和回归问题，其原理为通过寻找最近的 K 个邻居来确定新样本的分类或数值。

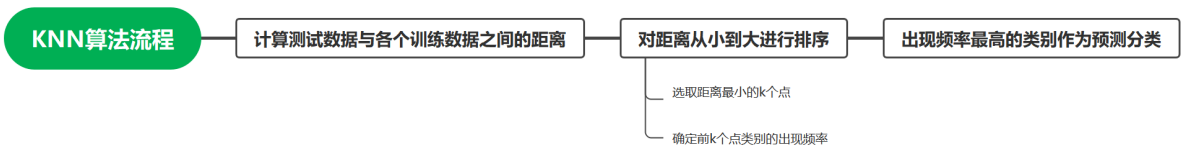


图 10　KNN 算法示意图

其优势如下：

1. 简单易实现，无需训练过程，对于本题数据量较大的数据集来说，节省时间，简单易行。
2. 在训练集中存在噪声的情况下，对于离群点的处理较好。
3. 可以使用不同的距离度量方式，根据具体问题进行选择。

表 5　KNN 算法部分结果

userid	起点纬度	起点经度	终点纬度	终点纬度
1	39.92	116.36	39.922	116.37
2	39.94	116.36	39.922	116.37
5	40	116.48	39.954	116.406
6	39.83	116.62	39.898	116.505
7	39.92	116.42	39.942	116.41
9	39.98	116.42	39.958	116.362
12	39.93	116.47	39.979	116.441
13	39.93	116.17	39.939	116.351
16	39.89	116.27	39.94	116.332
20	39.97	116.29	39.929	116.412

查阅资料，在北纬 40° 附近，两地之间经度差 1 度距离约为 $111 \times \cos 40° = 85$km，纬度差 1 度距离 111km，我们通过随机森林算法预测得到的用户平均移动的经度为 0.012，纬度为 0.007，折合成距离 1.544769 公里；通过 KNN 算法预测得到的用户平均移动的经度为 0.016，纬度为 0.012，折合成距离 2.223899 公里；查阅资料得知共享单车的平均使用距离为 1.69 公里，可见随机森林算法的准确率较高，与其适合处理大规模数据的特性相吻合。

7.3 K-means 聚类算法的应用

7.3.1 K-means 算法介绍

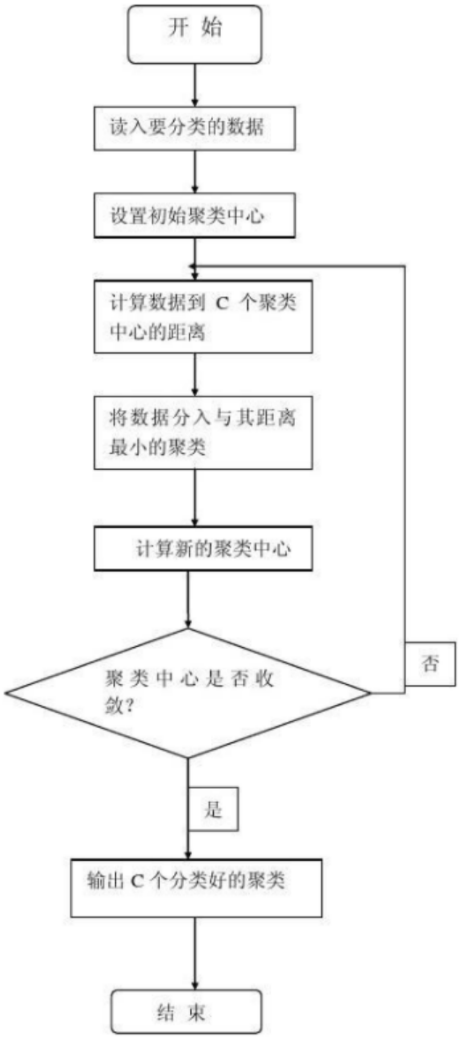


图 11　K-means 算法示意图

K 均值算法（K-means algorithm）是一种常用的聚类算法，用于将给定的数据集划分为 K 个不重叠的簇。其基本原理如下：

1. 初始化：首先需要选择 K 个聚类中心点，可以是随机选取数据集中的 K 个点或者根据领域知识进行选择。
2. 分配：依次计算数据集中每个样本点与当前的聚类中心距离，将每个样本点分配到距离最近的中心点所属的簇。
3. 更新：对于每个簇，重新计算其质心（即计算该簇中所有样本点的平均值），作为新的聚类中心。
4. 重复：重复步骤 2 和步骤 3，直到达到预定的停止条件。常用的停止条件有：簇分配不再发生改变、达到最大迭代次数或者误差的变化小于某个阈值。

5. 输出：最终得到 K 个簇的聚类结果。

K 均值算法的目标是最小化簇内样本点与簇中心的平方误差和，即最小化各个簇的内部差距，同时最大化不同簇之间的差异。其优点包括简单易实现、计算速度较快。然而，由于其使用平方误差作为优化目标，容易受到初始聚类中心的选择和局部最优解的影响。此外，K 值的选择也会对聚类结果产生影响。因此，在应用 K 均值算法时需要谨慎选择初始聚类中心和 K 值，并对结果进行评估和优化。

7.3.2 结果分析

在本问中，我们需要将先前机器学习模型预测的结果进行聚类分析，以显示问题一中的热门区块的共享单车的目的地的集群分布特征，我们将问题一中经过 DBSCAN 方法聚类得到的十个热门区块的中心点以及其包含的所有点的经纬度坐标在预测结果的 csv 文件中找到，并且将其预测的位置的坐标放入该区块的目的地集合中，以此类推，利用 K-means 算法按照距离远近进行聚类，最终得到十个区块的目的地的中心坐标，并且进行了可视化显示，结果如下 (所有点的坐标在附录中体现)：

表 6　预测目的区块中心坐标

区块	目的地中心纬度	目的地中心经度
区块 1	39.885	116.455
区块 2	39.85666667	116.43666667
区块 3	39.915	116.415
区块 4	39.845	116.39
区块 5	39.93333333	116.32
区块 6	39.9	116.443333332
区块 7	39.90666667	116.39666667
区块 8	39.92725	116.44
区块 9	39.94	116.335
区块 10	39.86666667	116.46666667

聚类后的可视化示意图如下：

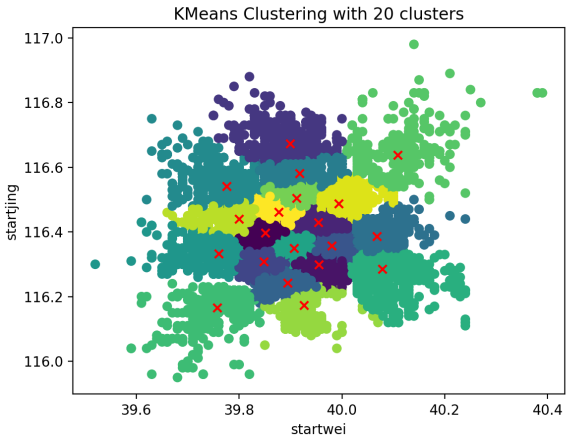


图 12　K-means 算法可视化

八、 问题三的模型建立与求解

8.1 模型的建立

首先考虑到热门区块的多变性，在不同时段的热门区块可能不同，为了方便分析，我们只取问题一中的早高峰起始地点作为热门区块，并利用问题二的预测模型，预测出 test 数据集中的早高峰和晚高峰的热门区块的主要目的地区块，并将 test 数据集的起始位置和终止位置作为主要分析指标。

接着，我们筛选出问题一中所涉及到的热门区块在 test 集所预测的目的地，并且统计出所有区块间的车辆移动情况，以二维矩阵的形式体现，最后计算出每个区块在早高峰开始前到晚高峰结束这一周期内的共享单车净增量。

考虑到人工调节的成本主要为运输成本，因此我们以两点间的距离为主要衡量标准，计算出是热门区块中心之间两两的直线距离，得到成本矩阵，至此数据准备完成。

表 7　各区块共享单车净增量

区块	车辆净增量
区块 1	411
区块 2	-624
区块 3	1084
区块 4	891
区块 5	-565
区块 6	615
区块 7	85
区块 8	-1673
区块 9	-524
区块 10	23

根据上述分析与假设，我们建立如下的数学模型。

$$\min \quad c_d \sum_{i \in S} \sum_{j:j \neq i} z_{ij}(t) + \sum_{i \in S} [y_i(t+1) - y_i(t)] \tag{1}$$

$$\text{s.t.} \quad c_d = \epsilon d_{ij} \tag{2}$$

$$y_i(t+1) = y_i(t) - \sum_{j:j \neq i} (q_{ij}(t) + z_{ij}(t) + \sum_{j:j \neq i} (q_{ji}(t) + z_{ji}(t)) \tag{3}$$

$$y_i(t+1) \geq y_i(t) \tag{4}$$

$$\alpha_{ij}(t) = \begin{cases} 1 & d_{ij} \leq 5km \\ 0 & d_{ij} > 5km \end{cases} \tag{5}$$

$$z_{ij}(t) \leq M \alpha_{ij}(t) \tag{6}$$

$$\sum_{j:j \neq i} z_{ij}(t) \leq y_i(t) \tag{7}$$

$$y_i(t), z_{ij}(t) \in N \tag{8}$$

模型说明：

- 式1为模型目标函数，考虑到居民出行需求和运营成本两个因素，在满足居民出行需

求的前提下尽可能降低运营成本。在模型中，我们将这两个因素的权重设为 1:1。

- 式2定义了运营成本，该模型中认为运营成本与调配距离成正比。为简化计算，设 $\epsilon = 1$。
- 式3描述了自行车一天内的变化情况。一天结束后某区域自行车数量等于一天开始时自行车数量减去当天骑出和人工调出该区域的自行车数量，再加上当天骑入和人工调入该区域的自行车数量。
- 式4描述了自行车数量变化的要求。为满足居民的出行需求，在一天结束后，自行车数量应不少于一天开始前，以满足第二天居民的出行需要。
- 式5和6通过 i 和 j 两地之间的距离，判断是否能在 i 和 j 两地进行人工调配。调配距离选择 5km 的原因是：根据《2017 年中国共享出行行业大数据观察》中的数据，绝大多数用户的共享单车单次骑行距离在 5km 以内。为了节约运营成本，我们选择 5km 作为人工调配的最远距离。
- 式7限制了从某区域人工调出自行车的数量需小于一天开始时该区域的自行车数量。
- 式8说明了变量的值域。

8.2 模型求解结果

我们利用 **Matlab** 的工具包进行模型的求解，结果如下：

表 8　人工调节策略

区块	转出目的地及数量	转入来源及数量	调节后数量
区块 1	–	区块 8 398	13
区块 2	–	区块 3 211；区块 7 371	-79
区块 3	区块 8 860；区块 2 211		13
区块 4	区块 9 457 区块 8 351		83
区块 5	–	区块 6 565	0
区块 6	区块 5 565	–	50
区块 7	区块 2 371		0
区块 8	–	区块 3 860；区块 4 351；区块 1 398	-64
区块 9	–	区块 10 23	-75
区块 10	区块 9 23	–	0

九、 模型评价

9.1 模型的优点

1. 第一问采取 SQL 进行大数据量的统计，方便快捷，无需复杂的程序代码语言。
2. 第二问采用随机森林算法和 KNN 算法，合理而准确地预测，并能够实现对比分析。
3. 针对不同问题，采取了不同的聚类方法：针对热门区域的划分问题，采取 DBSCAN 三维聚类，直观、形象、准确；针对用户骑行目的地定位问题，采用 K-means 聚类，准确反映出目的地的集群分布状态。
4. 第三问对问题考虑周全，将原问题转换为一个多目标整数规划问题求解，通过多种约束条件来进行约束，科学而合理。

9.2 模型的缺点

1. 第三问没有将 biketype 这一指标应用于我们的模型，或许不同的 biketype 存在费用成本上的差异。
2. 为简化模型，将投放地点设置为热门区块中心位置，但是实际投放地点显然将更加分散，且不局限于热门区块。
3. 在运营成本方面仅考虑了运输过程的成本，没有考虑到人力成本、租金成本等其他成本

9.3 模型的改进和推广

1. 可将该模型应用到其他公共设施的配置方面，有利于优化资源配置、节省成本。
2. 随机森林预测模型在使用之前可以进行预训练，提升预测精度。

参考文献

[1] 苏影. 基于数据分析的共享单车动态调配优化研究 [D]. 北京交通大学,2019.

[2] 陈植元, 林泽慧, 金嘉栋等. 基于时空聚类预测的共享单车调度优化研究 [J]. 管理工程学报,2022,36(01):146-158.

[3] 丁月, 郝泽政, 张翼然等. 北京市核心区共享单车骑行时空特征分析 [J]. 北京测绘,2023,37(02):295-301.

附录 A　train 数据处理程序

```
import geohash2
import csv
from datetime import datetime

csv_file_path = 'train.csv'

with open(csv_file_path, 'r', newline='', encoding='utf-8') as csv_file:
    # 创建CSV字典读取器
    csv_reader = csv.DictReader(csv_file)

    # 保存第4、5、6、7列的数据
    selected_columns = []
    for row in csv_reader:
        if len(row) >= 7: # 确保至少有7列数据
            selected_data = [ row['biketype'], row['starttime'], row['geohashed_start_loc'],
                row['geohashed_end_loc'], row['userid'], row['bikeid']]
            selected_columns.append(selected_data)

start = []
end = []
time=[]
bike=[]
user=[]
bikeid=[]

for i in range(len(selected_columns)):
    start_geo = selected_columns[i][2]
    end_geo = selected_columns[i][3]

    time.append(selected_columns[i][1])
    bike.append(selected_columns[i][0])
    user.append(selected_columns[i][4])
    bikeid.append(selected_columns[i][5])

    lat_start, lng_start = geohash2.decode(start_geo)
    lat_end, lng_end = geohash2.decode(end_geo)

    start.append([lat_start,lng_start])
    end.append([lat_end,lng_end])

# 将时间字符串解析为datetime对象，并筛选出7点到9点和17点到19点的数据
morning_data = []
evening_data = []
i = 0
```

```
for entry in time:
    dt_obj = datetime.strptime(entry, "%Y-%m-%d %H:%M:%S")
    start_temp_lat = float(start[i][0])
    start_temp_lng = float(start[i][1])
    end_temp_lat = float(end[i][0])
    end_temp_lng = float(end[i][1])
    bike_temp = bike[i]
    user_temp = user[i]
    bikeid_temp = bikeid[i]
    # 筛选条件: 7点到9点 且经纬度位于北京
    if ((7 <= dt_obj.hour < 10)) and ((115.7<=start_temp_lng<=117.4) and
        (115.7<=end_temp_lng<=117.4) and (39.4<=start_temp_lat<=41.6) and
        (39.4<=end_temp_lat<=41.6)):
        morning_data.append([user_temp,bikeid_temp,
            entry,bike_temp,start_temp_lat,start_temp_lng,end_temp_lat,end_temp_lng])
     # 筛选条件: 7点到9点 且经纬度位于北京
    elif (17 <= dt_obj.hour < 20) and ((115.7<=start_temp_lng<=117.4) and
        (115.7<=end_temp_lng<=117.4) and (39.4<=start_temp_lat<=41.6) and
        (39.4<=end_temp_lat<=41.6)):
        evening_data.append([user_temp,bikeid_temp,
            entry,bike_temp,start_temp_lat,start_temp_lng,end_temp_lat,end_temp_lng])
    i = i + 1

# 指定保存的文件路径
file_path = 'beijing_morning_train.csv'

# 保存为CSV格式文件
with open(file_path, 'w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file)
    for row in morning_data:
        csv_writer.writerow(row)

# 指定保存的文件路径
file_path = 'beijing_evening_train.csv'

# 保存为CSV格式文件
with open(file_path, 'w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file)
    for row in evening_data:
        csv_writer.writerow(row)

# ---处理数模数据, 将早高峰和晚高峰分开---
import pandas as pd

path = 'data/beijing_train.txt'

df = pd.read_csv(path, header=None,
```

```
                names=['userid', 'bikeid', 'starttime', 'biketype', 'start_latitude',
                    'start_longitude', 'end_latitude',
                      'end_longtitude'])
print(df)
df.info()
df['starttime'] = pd.to_datetime(df['starttime'])
df.info()

df = df.set_index('starttime')
# df_morning_part = df.between_time('7:00', '9:00')
# print(df_morning_part)
# df_morning_part.to_csv('data/beijing_train_morning.txt',index=False,sep=' ')
df_evening_part=df.between_time('17:00','19:00')
print(df_evening_part)
df_evening_part.to_csv('data/beijing_train_evening.txt',index=False,sep=' ')
```

附录 B　SQL 查询语言统计处理

```
SELECT start_latitude,start_longitude,COUNT(*) AS heat
FROM beijing_train_morning
GROUP BY start_latitude,start_longitude;

SELECT start_latitude,start_longitude,COUNT(*) AS heat
FROM beijing_train_evening
GROUP BY start_latitude,start_longitude;

SELECT end_latitude,end_longitude,COUNT(*) AS heat
FROM beijing_train_morning
GROUP BY end_latitude,end_longitude;

SELECT end_latitude,end_longitude,COUNT(*) AS heat
FROM beijing_train_evening
GROUP BY end_latitude,end_longitude;

CREATE VIEW top_train_morning_orig_ranking
AS SELECT b1.start_latitude,b1.start_longitude,b1.heat,
   (SELECT COUNT(*) FROM beijing_train_morning_orig b2 WHERE b2.heat>b1.heat)+1 AS ranking
   FROM beijing_train_morning_orig b1
   ORDER BY ranking;
```

附录 C　DBSCAN 聚类程序

```
    # -*- coding = utf-8 -*-

import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
import matplotlib.pyplot as plt

# 修改 numpy 的打印选项
np.set_printoptions(threshold=np.inf)

# path = 'data/beijing_train_morning_orig.txt' # 早高峰起始地
# path = 'data/beijing_train_morning_des.txt' # 早高峰终止地
path = 'data/beijing_train_evening_orig.txt' # 晚高峰起始地
# path = 'data/beijing_train_evening_des.txt' # 晚高峰终止点

df = pd.read_csv(path, header=None, names=['start_latitude', 'start_longitude', 'heat']) #
    起始地
# df = pd.read_csv(path, header=None, names=['end_latitude', 'end_longitude', 'heat']) # 终止地

data = []

for i in range(0, len(df["start_latitude"])): # 起始地
    mid = []
    mid.append(df["start_latitude"][i])
    mid.append(df["start_longitude"][i])
    mid.append(df["heat"][i])
    data.append(mid)
data = np.array(data)

# for i in range(0, len(df["end_latitude"])): # 终止地
#     mid = []
#     mid.append(df["end_latitude"][i])
#     mid.append(df["end_longitude"][i])
#     mid.append(df["heat"][i])
#     data.append(mid)
# data = np.array(data)

# 定义 DBSCAN 模型，设置 eps 和 min_samples 参数
# dbscan_model = DBSCAN(eps=5.5, min_samples=2) # 早高峰起始地
# dbscan_model = DBSCAN(eps=6, min_samples=2) # 早高峰终止地
dbscan_model = DBSCAN(eps=5.5, min_samples=2) # 晚高峰起始地
# dbscan_model = DBSCAN(eps=5.5, min_samples=2) # 晚高峰终止地

# 对数据进行拟合
dbscan_model.fit(data)
```

```
# 输出模型估算的聚类数量
print("Estimated number of clusters:", len(set(dbscan_model.labels_)) - (1 if -1 in
    dbscan_model.labels_ else 0))
# 注意需要排除噪音点

# 输出 Silhouette score
print("Silhouette score:", silhouette_score(data, dbscan_model.labels_))

# 输出 Calinski-Harabasz score
print("Calinski-Harabasz score:", calinski_harabasz_score(data, dbscan_model.labels_))

# 输出 Davies-Bouldin Index
print("Davies-Bouldin Index:", davies_bouldin_score(data, dbscan_model.labels_))

# 将 DBSCAN 结果与原始数据合并
clustered_data_df = pd.DataFrame(data, columns=['Dimension 1', 'Dimension 2', 'Dimension 3'])
clustered_data_df['Cluster'] = dbscan_model.labels_

from mpl_toolkits.mplot3d import Axes3D

# 创建三维坐标轴
fig = plt.figure(figsize=(70, 70))
ax = fig.add_subplot(111, projection='3d')

# 根据聚类标签绘制散点图

# colors = ['coral', 'yellow', 'olive', 'magenta', 'black', 'cyan', 'green', 'blue', 'purple',
    'orange', 'brown', 'pink',
#         'gray', 'red'] # 早高峰颜色集
colors = ['coral', 'olive', 'yellow', 'magenta', 'black', 'cyan', 'green', 'blue', 'purple',
    'orange', 'brown', 'pink',
        'gray', 'red'] # 晚高峰起始颜色集
# colors = ['olive', 'coral', 'yellow', 'magenta', 'black', 'cyan', 'green', 'blue', 'purple',
    'orange', 'brown', 'pink',
#         'gray', 'red'] # 晚高峰终止颜色集

for label in set(dbscan_model.labels_):
    if label == -1:
        cidx = -1
    else:
        cidx = label % len(colors)
    ax.scatter(clustered_data_df.loc[clustered_data_df['Cluster'] == label, 'Dimension 1'],
               clustered_data_df.loc[clustered_data_df['Cluster'] == label, 'Dimension 2'],
               clustered_data_df.loc[clustered_data_df['Cluster'] == label, 'Dimension 3'],
               c=colors[cidx], marker='.', label='Cluster ' + str(label))
```

```
# 聚类标签
labels = dbscan_model.labels_
# 计算每个聚类中心点和离群点
centers = []
outliers = []
clusters = []
means = []
for i in set(labels):
    if i == -1: # 噪声点
        outliers.extend(data[labels == i])
        continue
    cluster = data[labels == i]
    centers.append(np.mean(cluster, axis=0))
    clusters.append(cluster)
    means.append(np.mean(cluster[:,2]))
# 计算每个聚类中心点离所有点的距离
distances = []
for center in centers:
    distance = np.sqrt(np.sum(np.square(data - center), axis=1))
    distances.append(np.min(distance))
# 选取距离最小的点作为最中心的点
index = np.argmin(distances)
result = centers[index]


means_sorted_indices = np.argsort(means)[::-1]


for i in means_sorted_indices:
    rank = np.where(means_sorted_indices == i)[0][0] + 1
    center, cluster = centers[i], clusters[i]
    print(f'Cluster {i}:')
    print(f' Center: {center}')
    print(f' Points: {cluster}')
    print(f' Mean of Dimension 3: {means[i]}')
    print(f' Rank by Mean of Dimension 3: {rank}')

# 添加坐标轴标签和图例
ax.set_xlabel('latitude')
ax.set_ylabel('longitude')
ax.set_zlabel('heat')
ax.set_title('DBSCAN Clustering (TSNE)')

plt.legend()
plt.show()
```

附录 D　test 数据处理程序

```
import geohash2
import csv
from datetime import datetime

csv_file_path = 'test.csv'

with open(csv_file_path, 'r', newline='', encoding='utf-8') as csv_file:
    # 创建CSV字典读取器
    csv_reader = csv.DictReader(csv_file)

    # 保存第4、5、6、7列的数据
    selected_columns = []
    for row in csv_reader:
        if len(row) >= 6: # 确保至少有7列数据
            selected_data = [ row['biketype'], row['starttime'], row['geohashed_start_loc'],
                row['userid'], row['bikeid']]
            selected_columns.append(selected_data)

start = []
time=[]
bike=[]
user=[]
bikeid=[]

for i in range(len(selected_columns)):
    start_geo = selected_columns[i][2]


    time.append(selected_columns[i][1])
    bike.append(selected_columns[i][0])
    user.append(selected_columns[i][3])
    bikeid.append(selected_columns[i][4])

    lat_start, lng_start = geohash2.decode(start_geo)

    start.append([lat_start,lng_start])

# 将时间字符串解析为datetime对象，并筛选出7点到9点和17点到19点的数据
morning_data = []
evening_data = []
i = 0
for entry in time:
    dt_obj = datetime.strptime(entry, "%Y-%m-%d %H:%M:%S.0")
    start_temp_lat = float(start[i][0])
```

```
    start_temp_lng = float(start[i][1])

    bike_temp = bike[i]
    user_temp = user[i]
    bikeid_temp = bikeid[i]
    # 筛选条件: 7点到9点 且经纬度位于北京
    if ((7 <= dt_obj.hour < 10)) and ((115.7<=start_temp_lng<=117.4) and
         (39.4<=start_temp_lat<=41.6)):
        morning_data.append([user_temp,bikeid_temp,
            entry,bike_temp,start_temp_lat,start_temp_lng])
     # 筛选条件: 7点到9点 且经纬度位于北京
    elif (17 <= dt_obj.hour < 20) and ((115.7<=start_temp_lng<=117.4) and
         (39.4<=start_temp_lat<=41.6)):
        evening_data.append([user_temp,bikeid_temp,
            entry,bike_temp,start_temp_lat,start_temp_lng])
    i = i + 1

# 指定保存的文件路径
file_path = 'beijing_morning_test.csv'

# 保存为CSV格式文件
with open(file_path, 'w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file)
    for row in morning_data:
        csv_writer.writerow(row)

# 指定保存的文件路径
file_path = 'beijing_evening_test.csv'

# 保存为CSV格式文件
with open(file_path, 'w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file)
    for row in evening_data:
        csv_writer.writerow(row)
```

附录 E　热力图绘制程序

```
import folium
from folium.plugins import HeatMap
import pandas as pd

# 从CSV文件读取数据
data = pd.read_csv('beijing_train.csv')

# 提取纬度和经度数据
```

```
start_latitudes = data.iloc[:, 4]
start_longitudes = data.iloc[:, 5]

# 创建一个Folium地图对象
m = folium.Map(location=[start_latitudes.mean(), start_longitudes.mean()], zoom_start=10) #
    设置初始显示位置和缩放级别

# 将经纬度数据转换为列表
heatmap_data = [[lat, lon] for lat, lon in zip(start_latitudes, start_longitudes)]

# 创建热力图并添加到地图上
HeatMap(heatmap_data).add_to(m)

# 保存地图为HTML文件
m.save('heatmap_from_start_csv.html')
```

附录 F　预测模型训练程序

```
# -*- coding: utf-8 -*-
import datetime
import os
import time
from collections import Counter
from sklearn.model_selection import train_test_split
import geohash
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.utils import shuffle
from sklearn.neighbors import KNeighborsRegressor
def datetime_to_period(date_str):
    """
    描述：把时间分为24段
    返回：0到23
    """
    # date_str=str(date_str)
    time_part = date_str.split(" ")[1] # 获取时间部分
    hour_part = int(time_part.split(":")[0]) # 获取小时
    return hour_part

def date_to_period(date_str):
    """
    描述：把日期转化为对应的工作日或者节假日
    返回：0:工作日 1：节假日 2:小长假
    """
    holiday_list = ['2017/01/01', '2017/01/25', '2017/01/26', '2017/01/27', '2017/01/28',
```

```
        '2017/01/29',
                '2017/01/30', '2017/01/31', '2017/04/03', '2017/04/04', '2017/04/05',
                    '2017/04/29',
                '2017/05/01', '2017/05/02', '2017/05/29', '2017/05/30', '2017/05/31'] # 小长假
    # switch_workday_list = ['2018-02-11', '2018-02-24', '2018-04-08', '2018-04-28'] # 小长假补班
    workday_list = ['1', '2', '3', '4', '5'] # 周一到周五
    weekday_list = ['0', '6'] # 周六、周日，其中0表示周日
    date = date_str.split(" ")[0] # 获取日期部分
    whatday = datetime.datetime.strptime(date_str, '%Y/%m/%d %H:%M').strftime("%w") #
        把日期转化为星期

    # time_data = '2017/5/16 7:22'
    # time_format = '%Y/%m/%d %H:%M'
    # try:
    #     dt = datetime.datetime.strptime(time_data, time_format)
    #     print(dt)
    # except ValueError as e:
    #     print("解析时间错误:", e)
    #
    #
    if date in holiday_list:
        return 2
    # elif date in switch_workday_list:
    #     return 0
    elif whatday in workday_list:
        return 0
    elif whatday in weekday_list:
        return 1
time_start = time.asctime(time.localtime(time.time())) # 程序开始时间
# # ---删除相关文件---
# path = "C:/Users/28678/Desktop/" # 文件路径
# lst = ['predict', 'predict_result', 'view', 'score', 'train'] # 文件名列表
# for file_name in lst:
#     file_path = path + file_name + '.csv'
#     if os.path.exists(file_path):
#         os.remove(file_path)

train_data_path = "C:/Users/28678/Desktop/beijing_morning_train.csv"
train_data = pd.read_csv(train_data_path, low_memory=False)
test_data_path = "C:/Users/28678/Desktop/beijing_morning_test.csv"
test_data = pd.read_csv(test_data_path, low_memory=False)
# n = 0
# test_out_id = Counter(test_data['userid'])
# for userid in test_out_id.keys():
#     # ----train_new数据补充字段 begin----
for i in range(len(train_data)):
    train_data.iloc[i, 6] = datetime_to_period(train_data.iloc[i, 1]) # 添加时间段
```

```
    train_data.iloc[i, 7] = date_to_period(train_data.iloc[i, 1]) # 添加工作日、休息日编码
    # train = train_data[train_data['userid'] ==userid] # 选择出同一个userid的数据
    # train = shuffle(train) # 打乱顺序
    # # train['start_code'] = None # 开始位置的geohash编码
    # # train['end_code'] = None # 结束位置的geohash编码
    # train['period'] = None # 时间段编码 (0-23)
    # train['week_code'] = None # 工作日和休息日编码
    # for i in range(len(train)):
    #     # train.iloc[i, 8] = geohash.encode(train.iloc[i,3], train.iloc[i,4], 8) #
        开始geohash编码
    #     # train.iloc[i, 9] = geohash.encode(train.iloc[i,5], train.iloc[i,6], 8) #
        结束geohash编码
    #     train.iloc[i, 10] = datetime_to_period(train.iloc[i,1]) # 添加时间段
    #     train.iloc[i, 11] = date_to_period(train.iloc[i,1]) # 添加工作日、休息日编码

    # ----train_new数据补充字段 end----
for i in range(len(test_data)):
    test_data.iloc[i,4] = datetime_to_period(test_data.iloc[i,1]) # 添加时间段
    test_data.iloc[i,5] = date_to_period(test_data.iloc[i,1]) # 添加工作日、休息日编码
    # ----test_new数据补充字段 begin----
    # test = test_data[test_data['userid'] == userid]
    # test = shuffle(test) # 打乱顺序
    # test['period'] = None
    # test['week_code'] = None
    # # test['start_code'] = None
    # # test['predict'] = None
    # for i in range(len(test)):
    #     test.iloc[i, 5] = datetime_to_period(test.iloc[i, 1]) # 添加时间段
    #     test.iloc[i, 6] = date_to_period(test.iloc[i, 1]) # 添加工作日、休息日编码
    #     # test.iloc[i, 7] = geohash.encode(test.iloc[i, 3], test.iloc[i, 4], 8) #
        开始geohash编码
    # # ----test_new数据补充字段 end----
train_features = train_data[['userid', 'period', 'startwei', 'startjing']]
train_labels = train_data[['endwei', 'endjing']]
test_features = test_data[['userid', 'period', 'startwei', 'startjing']]
selected_features = ['userid', 'period', 'startwei', 'startjing']
X_train,y_train=train_features, train_labels
knn = KNeighborsRegressor(n_neighbors=10)
knn.fit(X_train, y_train)
    # y_pred = knn.predict(X_valid)
test_labels = knn.predict(test_features[selected_features])
test_data['endwei'] = test_labels[:, 0]
test_data['endjing'] = test_labels[:, 1]
test_data.to_csv('C:/Users/28678/Desktop/predicted_morning.csv', index=False)
```

附录 G　Kmeans 聚类程序

```
import pandas as pd
import numpy as np
import os
from sklearn.cluster import KMeans
from sklearn import cluster
from sklearn import metrics
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
df = pd.read_csv("C:/Users/28678/Desktop/predicted_test_data.csv")
# df.columns=['userid','time','biketype','startwei','startjing']
print(df.isna().sum())
#
# cluster.DBSCAN().get_params()
#
data = df.loc[1:400000,['endwei','endjing']]
# dbscan = cluster.DBSCAN(eps=0.02, min_samples=10)
# labels = dbscan.fit_predict(data)
# n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
#
# colors = ['red', 'blue', 'green', 'orange', 'purple', 'pink', 'yellow', 'brown', 'gray',
    'cyan']
#
# plt.scatter(data['startwei'], data['startjing'], c=labels,
    cmap=matplotlib.colors.ListedColormap(colors))
# plt.xlabel('Latitude')
# plt.ylabel('Longitude')
# plt.title('DBSCAN Clustering')
# plt.show()

# 返回代码
kmeans = KMeans(n_clusters=20)
kmeans.fit(data)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
plt.scatter(data['endwei'], data['endjing'], c=labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red')
plt.xlabel('startwei')
plt.ylabel('startjing')
plt.title('KMeans Clustering with 20 clusters')
plt.show()
# for i in range(10):
#   centroid = centroids[i]
#   print(f"Cluster {i+1} centroid: ({centroid[0]}, {centroid[1]})")
cluster_centers = df.groupby('cluster')['endwei','endjing'].mean().reset_index()
```

```
cluster_centers_sorted = cluster_centers.sort_values(by='endwei', ascending=False)
top_10_clusters = cluster_centers_sorted.head(10)
print(top_10_clusters[['endwei','endjing']])
```

附录 H　距离计算程序

```
import math
import pandas as pd

# 输入十个点的经纬度坐标
points = [
    (39.885,116.455 ),
    (39.85666667,116.43666667),
    (39.915,116.415),
    (39.845,116.39),
    (39.93333333,116.32),
    (39.9,116.44333333),
    (39.90666667,116.39666667),
    (39.92725,116.44),
    (39.94,116.335),
    (39.86666667,116.46666667)
]

# 创建一个空的二维矩阵来存储距离
dist_matrix = [[0.0] * len(points) for _ in range(len(points))]

# 计算两点之间的距离
for i in range(len(points)):
    for j in range(len(points)):
        if i != j:
            lat1, lon1 = points[i]
            lat2, lon2 = points[j]
            # 使用Haversine公式计算球面两点间的距离
            radius = 6371 # 地球平均半径，单位为公里
            dlat = math.radians(lat2 - lat1)
            dlon = math.radians(lon2 - lon1)
            a = math.sin(dlat / 2) * math.sin(dlat / 2) + math.cos(math.radians(lat1)) *
                math.cos(math.radians(lat2)) * math.sin(
                dlon / 2) * math.sin(dlon / 2)
            c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
            distance = radius * c
            dist_matrix[i][j] = distance

# 创建一个DataFrame用于保存矩阵数据
df = pd.DataFrame(dist_matrix)
```

```
# 保存DataFrame到Excel表格
df.to_excel("distance_matrix.xlsx", index=False)
```

附录 I　统计各地净增量程序

```
import pandas as pd
# 读取Excel文件
df = pd.read_csv('C:/Users/28678/Desktop/predicted_evening.csv')
# 定义十个区块的列表
blocks = [
        [[39.85,116.39],
        [39.92,116.52]],
[[39.84,116.46],
 [39.88,116.48],
 [39.85,116.37]],
[[39.89,116.48],
 [39.94,116.35]],
[[39.80,116.42],
 [39.89,116.36]],
[[39.86,116.43],
 [40.03,116.32],
 [39.91,116.21]],
[[39.94,116.44],
 [39.84,116.38],
 [39.92,116.51]],
[[39.88,116.37],
 [39.93,116.33],
 [39.91,116.49]],
   [[39.88, 116.46],
    [39.98, 116.42],
    [39.96, 116.44],
    [39.87, 116.44]],
[[ 39.97,116.42],
 [ 39.91,116.25]],

[[ 39.93,116.32],
 [ 39.91,116.53],
 [ 39.76,116.55]]
]
# # 遍历十个区块的列表
# for i in range(10):
#    # block_data = [] # 用于存储当前区块的数据
#    # 假设经纬度数据储存在名为 blocks_data 的列表中
#    for smallblock in blocks[i]:
#        for point in smallblock:
```

```
#
#       block_data.append((point['wei'], point['jing'])) # 获取每个点的 (wei, jing) 坐标
#    blocks.append(block_data)
# 定义一个函数用于判断坐标是否在某个区块的数据里[],[]
def is_point_in_block(point, blockbiu):
       # block_data = [] # 用于存储当前区块的数据
       # 假设经纬度数据储存在名为 blocks_data 的列表中
    for biu in blockbiu:
      if point[0]==biu[0] and point[1]==biu[1]:
        return True
    return False
# scores=[2444.5,1966.6,1816.0,1574.5,1528.3,1519.0,1496.0,1415.7,1399.5,1385.0]
# scores=[0,0,0,0,0,0,0,0,0,0]
point=[[],[],[],[],[],[],[],[],[],[]]
# 遍历Excel文件的每一行
for index, row in df.iterrows():
    start_point = (row['startwei'], row['startjing'])
    end_point = [row['endwei'], row['endjing']]
      # 遍历十个区块
    for i in range(10):
        if is_point_in_block(start_point, blocks[i]):
            # 在起始位置坐标的区块得分减1
            # 假设得分储存在名为 scores 的列表中
          point[i].append(end_point)
        #    scores[i] -= 1
        # if is_point_in_block(end_point, blocks[i]):
        #    # 在终止位置坐标的区块得分加1
        #    scores[i] += 1
# 创建一个新的DataFrame保存区块得分
scores_df = pd.DataFrame({'Block': range(1, 11), 'point': point})
# 将DataFrame保存为新的Excel文件
scores_df.to_csv('output3.csv', index=False)
```

附录 J　第三问多目标整数规划求解程序

```
    clc,clear
prob=optimproblem;
zij = optimvar('zij',10,10,'Type','integer','LowerBound',0);
yi_t1 = optimvar('yi_t1',10,1,'Type','integer','LowerBound',0);
yi_t = optimvar('yi_t',10,1,'Type','integer','LowerBound',0);
filename1 = 'C:\Users\zyc13\Desktop\distance_matrix.xlsx'; % 指定要读取的文件名
dij = xlsread(filename1); % 读取文件中的数据
filename2='C:\Users\zyc13\Desktop\reslut(1).xlsx'
qij = xlsread(filename2);
result = zeros(10,10); % 初始化结果变量
```

```
for i = 1:10
for j = 1:10
if i ~= j
result = result + dij(i,j)*zij(i,j);
end
end
end
prob.Objective = result + ones(10) * sum(yi_t1-yi_t); %规划目标函数
aij=zeros(10,10);
for i = 1:10
for j = 1:10
if dij(i,j)<=5
aij(i,j)=1;
else
aij(i,j)=0;
end
end
end
con1=[];con2=[];con3=[];con4=[];
for i = 1:10
for j = 1:10
M = 1e6; % 定义一个足够大的常数
con1 = [con1; zij(i,j) <= M*aij(i,j)]; % 使用常数 M 代替 Inf——约束条件一
end
end
temp_zij=0,temp_zji=0,temp_qij=0,temp_qji=0;

for i = 1:10
con2 = [con2; yi_t1(i,1) - yi_t(i,1) >= 0]; % 逐个添加约束条件——约束条件二
for j = 1:10
if i ~= j
temp_zij = temp_zji + zij(i,j);
temp_zji = temp_zji + zij(j,i);
temp_qij = temp_qij + qij(i,j);
temp_qji = temp_qji + qij(j,i);
end
end
con3 = [con3; yi_t(i,1) >= temp_zij]; % 约束条件三
con4 = [con4; yi_t1(i,1) == yi_t(i,1)-temp_zij-temp_qij+temp_zji+temp_qji]; % 约束条件四
temp_zij=0,temp_zji=0,temp_qij=0,temp_qji=0;
end
prob.Constraints.con1 = con1;
prob.Constraints.con2 = con2;
prob.Constraints.con3 = con3;
prob.Constraints.con4 = con4;
[sol,fval,flag] = solve(prob); % 求解优化问题
optimalZij = sol.zij; % 获取 zij 的最优解
```