参赛编号: CS-GXDC-0288

赛题类型:算法分析题

团队人数:5

# 早高峰共享单车潮汐点的群智优化

## 一. 摘要

本项目针对厦门市早高峰时段(7:00-9:00)存在的共享单车潮汐现象-即共享单车”还不进”问题,提出了根据”车多”,”车挤”,”又多又挤”等三种识别潮汐区域的方法.

在区域分割的方法上, 我们发现传统的街道名划分方式与网格划分方式均存在重大缺陷, 而选取数据科学方式的思路如下:



图1. 模型选择逻辑

在后续优化建议中, 使用图方法中的单源最短路算法方法将用户从潮汐点引导至邻近围栏,进行”削峰填谷”, 并给出了选择目标围栏的标准.在数据挖掘过程中, 还提出了共享单车异常状态以及”乱停车”现象的数据挖掘方法和改进建议.

### 作品主要产出:

1. 区域划分方法 (4.1) 及三种潮汐区域识别方法 (5)

2. “削峰填谷” 引导措施 (8.1)

3. 基于数据的共享单车锁具, 车体物理状态异常监测 (3.2.2.1)

4. “乱停车” 现象集中区域及治理建议 (8.2)

### 主要使用技术:

HDBSCAN(Hierarchical Density-Based Spatial Clustering), Geohash, 数据挖掘, 单源最短路(Dijkstra)算法

### 开发工具

Python3.7, Microsoft SQL Server

## 二. 问题需求

## 2.1 项目的商业运用

在早高峰(7:00-9:00), 部分繁忙地段时常发生因共享单车围栏爆满或缺乏导致的共享单车占道问题. 通过对哈啰单车调度人员的调查发现,目前较为常用的,解决”借不到”, ”还不进”问题的方案是通过调度车全天将单车淤积严重地区的共享单车运送至临近非淤积点, 但这无疑是极其耗费人力物力的.

就此问题,我们需要对共享单车订单数据及共享单车围栏坐标进行探索,得出潮汐现象最严重的地区并提供用户引导建议,以减少调度车使用次数,节约成本.同时, 也需要对”乱停车”现象进行一定的识别, 并提供备选解决方案.

## 2.2 城市地理信息的特征及因公开数据源缺失导致的问题

由于障碍物(楼房,宽马路等)和交通规则的存在, 城市中的地理信息与传统数据科学中的空间不同. 城市抽象为数学概念后是一个二维的, 有向的, 有权的, 存在环的, 局部非处处联通的图. 所以不能简单地使用点与点之间的地理直线距离进行计算.

在本项目中, 由于缺乏自行车可行驶道路(与车辆道路,行人道路均不同)的图数据, 所以只能简单地采用直线距离判断当前开/关锁的共享单车应该归属的停车围栏. 但在停车围栏聚类和后续的调度优化过程中,我们均考虑到了城市地理信息特征的问题.

## 三. 数据应用

### 3.1 数据清单

##### 3.1.1 原始文件使用概况

|  |  |  |
| --- | --- | --- |
| 文件名 | 文件说明 | 使用字段及应用说明 |
| gxdc\_dd.csv | 共享单车订单数据 | BICYCLE\_ID: 进行数据挖掘, 以此识别行为异常单车并清洗  LATITUDE, LONGITUDE: 自行车坐标. 进行归属共享单车围栏分类及”乱停车”现象识别  UPDATE\_TIME: 用于配合BICYCLE\_ID进行数据清洗, 在可视化中也进行了使用 |
| gxdc\_tcd.csv | 共享单车围栏数据 | 围栏定位,用于自行车归属共享单车围栏分类和围栏聚类 |

### 3.2. 数据分析细节

#### 3.2.1 数据完善

##### 3.2.1.1 面积计算

使用给定的围栏多边形坐标点计算围栏面积(单位:平方米).命名为AREA字段

##### 3.2.1.2 geohash(./tools/hashing.py)

geohash的原理是将经纬度编码至字符串, 以等长宽方格进行划分, 可大幅缩小距离查找的计算量. 字符串长度越长, 划分精度越高, 优化也越明显,但对于本次竞赛数据,过长的geohash直接加入字典进行查找会导致较大的共享单车围栏无法被全覆盖以至于在其中部分区域的车辆无法被计入.

尽管使用字典树数据结构可以解决这个问题,但在本项目中没有使用. 7位geohash无法覆盖到的自行车数据仅有4894个 (仅占总数据量的1%, 且可直接归为” 乱停车”), 所以对于所有数据使用字典树查找反而会增加计算时间

在上一小节的面积计算中我们可以得知长度最大的停车围栏(展鸿路\_L\_B10002)约为84米. 使用7位即153\*153方格的geohash编码围栏四角可将整个围栏完全覆盖在内,不会出现因围栏中间不被任何一个顶点编码覆盖而无法将此范围内自行车计入点位的状况.

因仅使用单个编码格时容易出现如图3.2.1.2所示的边界问题

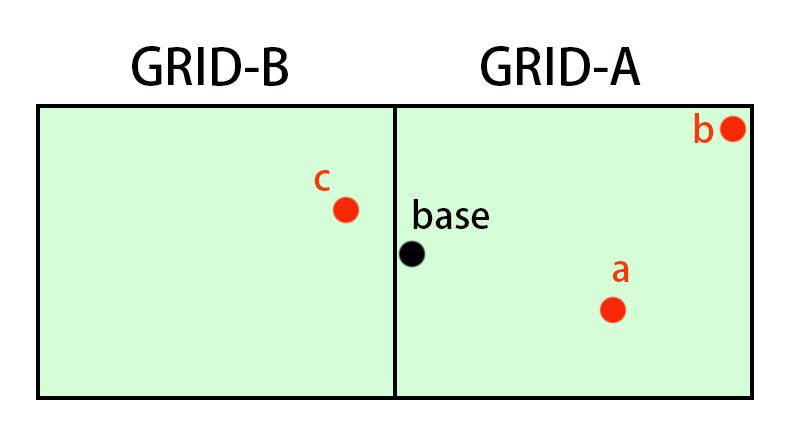


图2.3.2 错误最近点现象: 因geohash划分, 错误地将a识别为离base最近的点,而非c点

所以对于每一个围栏四边形顶点都将其本身编码和相邻的8个编码加入计算过程.

在经过./tools/hashing.py的计算后,我们可以得到两个临时结果:

1. 每一个自行车订单对应的geohash编码

2. 每一个围栏对应的36个可能重复的geohash值(每一个顶点都有9个geohash值,但如果围栏四个顶点在在同一个geohash编码格中,那么这36个geohash就是9个geohash重复四次), 将其反向映射,得到每个geohash编码关联的围栏FID(FID是根据index加的,和FENCE\_ID作用一致但是更加简短),使用{编码:[FID, FID…]}格式的字典(在其后表达为grid\_fence,在find\_nearest.py中有体现)进行存储.

单个停车围栏相关的最近点生成代码如下(使用python-geohash库)

1. **def** point\_grids(point):
2. """
3. :type point: Tuple or List, format: (Latitude, Longitude)
4. :rtype: List, related geohash grids to current point
5. """
6. center = geohash.encode(point[0], point[1], 7)
7. # 将center扩展为九宫格,避免"错误最近点"现象出现
8. nears = geohash.expand(center)
9. **return** [center] + nears
11. **def** fence\_grids(points):
12. """
13. :type points: List[(Float, Float)], with 4 tuples: p1, p2, p3, p4: position or 4 corner points of a polygon(fence)
14. :rtype: List, related geohash grids to current bike fence.
15. """
16. p1, p2, p3, p4 = points
17. p1\_g, p2\_g, p3\_g, p4\_g = point\_grids(p1), point\_grids(p2), point\_grids(p3), point\_grids(p4)
18. **return** list(set(p1\_g + p2\_g + p3\_g + p4\_g))

##### 3.2.1.3 中心点, 面积, 长宽计算

根据顶点坐标计算围栏中心点, 面积, 长宽

##### 3.2.1.4. 最近共享单车围栏的查找(./tools/find\_nearest.py)

将每一个自行车点位对应的geohash作为key,在字典中查询对应的围栏FID,并计算该点位与这些围栏的中心点距离,找出最小值所对应的FID.使用该FID的顶点坐标,计算点位与围栏边界的地理直线距离(若在围栏内则为0), 记为DISTANCE.

\* 此处因缺少自行车通行道路的有向图数据, 故只能采用地理直线距离计算.

##### 3.2.1.5 加入日期字段

原始的gxdc\_dd.csv包括12.21-12. 25五天的数据.我们注意到12.25是西方圣诞节, 在这一天的共享单车流向或许会与另外几天不同, 所以使用SQL在自行车数据中单独加入”日”维度,命名为DAY,方便后续可视化和数据分类

##### 3.2.1.6 数据完善小结

**对于自行车数据**

新增字段: GRID, BID, NEAREST\_FENCE, DISTANCE, DAY

GRID: 自行车点位对应的geohash

BID: 根据index形成的编号, 使每条数据都具有独立性

NEAEREST\_FENCE: 最近的共享单车围栏对应的FID

DISTANCE: 到最近共享单车围栏(边缘)的距离

DAY: 当前所属的日(21, 22, 23, 24, 25)

部分数据的NEAREST\_FENCE 和DISTANCE距离为-1,意味着该点位周围153-306米内没有任何共享单车围栏.在实际生活中可将其作为”乱停车”数据的一部分  
MKTIME为当前时间离era的秒数,后续没有得到使用.

**对于围栏数据**

新增中心点坐标LATITUDE, LONGITUDE. 将原有的五个坐标解包为10个维度(LATITUDE\_0, LATITUDE\_1等),方便读取

新增FID字段(type: int),功能同FENCE\_ID

新增ROAD字段:当前共享单车围栏所属的街道名

新增LENGTH, WIDTH字段:当前共享单车围栏长宽

新增AREA字段:当前共享单车围栏的面积

#### 3.2.2 数据清洗

##### 3.2.2.1 锁具异常行为检测及异常数据清理

使用Microsoft SQL Server进行数据预览时, 发现共有4136辆车存在连续锁车, 连续开车等锁具异常(原因可能时锁具损坏,电量不足等)现象

以BICYCLE\_ID为001ca978928d0e762aaede9118e3c7e6的自行车为例:

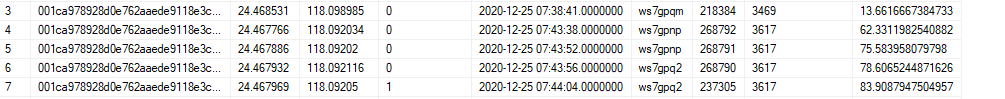


图3.2.2.1.1 开/关锁异常数据示例

7时43分的四个连续开锁数据经matplotlib可视化后, 可以看出该车在异常数据发生期间是持续移动的, 而从静止状态转为移动状态的起始数据点是第一个开锁数据 (7时38分)故可以去除由7:43:38开始的开锁记录(保留7:38:41的记录和7:44的关锁记录),并将该车加入异常列表.

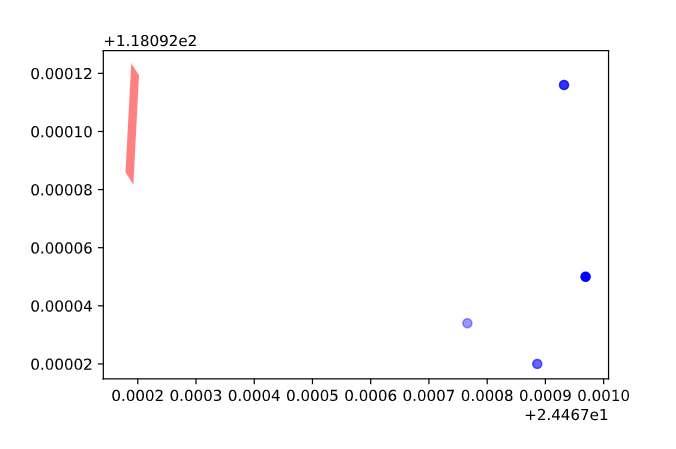


图3.2.2.1.2 异常数据可视化.红色方框为编号3617的围栏,自行车轨迹点按时间先后顺序由浅至深

同理,对于连续的关锁数据,仅保留最后一条,即由移动状态转为静止状态的数据.

以上步骤的代码位于tools/SQL/find\_ubnormal\_orders.sql 与 tools/remove\_abnormal\_script.py, 由于长度问题并不在报告中表现.

车体物理异常状态也可用SQL进行检测:若开-关间隔过短(在SQL中为min<x,x可以是1, 2等较短的分钟数)且多次进行, 则该车有可能因损坏而被用户放弃,需要进行检查.

##### 3.2.2.2 去除7时前与9时后的数据

经数据预览发现原始自行车订单数据中存在的订单时间为早6时至10时, 而本次比赛需要研究的是早高峰时段(7:00-9:00)之间存在潮汐现象的停车围栏,故使用SQL直接移除7:00之前和9:00之后的数据

##### 3.2.2.3 数据清洗结果

可用于后续数据挖掘与算法分析的共享单车开/关锁数据数量由原先的585292个下降到了339810个,原始数据中共有约41.9%的数据不可用于后续分析(所处时间不在研究范围内,或为异常数据)

### 3.3 数据安全保障

## 四. 数据挖掘与算法分析(./tools/model.ipynb)

数据预处理即数据的完善与清洗详见3.2.1数据完善与3.2.2数据清洗

在此阶段由于数据显示需求量与代码量的减少和对数据图形化需求的增加, 我们将Python Script + SQL进行的数据挖掘与清洗转至基于Python的Jupyter Notebook以便进行可视化.

经过3.2.1和3.2.2的预处理后,我们得到了两个csv文件: bikes\_data.csv 与 fence\_position.csv

文件的字段与字段说明如下表所述

|  |  |  |  |
| --- | --- | --- | --- |
| 文件名 | 字段名 | 数据类型 | 字段说明 |
| bikes\_data.csv  (数据量:339810) | DAY | INT | 数据所在的日(21-25) |
| BID | INT(UNIQUE) | 唯一订单代号 |
| BICYCLE\_ID | STRING | 自行车ID |
| LATITUDE | FLOAT | 当前纬度 |
| LONGITUDE | FLOAT | 当前经度 |
| LOCK\_STATUS | INT(BOOL) | 开/关锁状态(0开1关) |
| UPDATE\_TIME | STRING | 数据发生时间 |
| GRID | STRING | 坐标geohash编码 |
| NEAREST\_FENCE | INT | 离坐标点最近的围栏编码 |
| DISTANCE | FLOAT | 离最近围栏的距离 |
| MKTIME | FLOAT | 数据发生时间离era的秒数(后续没有使用) |
| FENCE\_ID | STRING | 围栏名 |
| fence\_position.csv | LATITUDE\_X | FLOAT | 围栏顶点经度（X：0-4） |
| LONGITUDE\_X | FLOAT | 围栏顶点纬度（X：0-4） |
| LATITUDE | FLOAT | 围栏中心点经度 |
| LONGITUDE | FLOAT | 围栏中心点纬度 |
| ROAD | STRING | 围栏所在路名 |
| AREA | FLOAT | 围栏面积 |
| FID | INT | 围栏唯一标识（作用同FENCE\_ID） |
| LENGTH | FLOAT | 围栏长度 |
| WIDTH | FLOAT | 围栏宽度 |

### 4.1 停车围栏区域划分策略的选择

目前较为常用的区域划分策略有:

1. 网格划分

2. 街道名划分

3. Kmeans聚类

4. DBSCAN

这几种策略在进行地理区域划分时均存在一定缺陷, 所以我们最终采用了HDBSCAN进行区域划分

#### 4.2.1 网格划分方法原理及缺陷

网格划分的原理是将地图变成一个个”方块”, 将同方块内的点作为同一个类进行处理, 在数据预处理阶段的geohash其实就是网格划分的方法之一.

这里可以再次使用图2.3.2(重命名为图4.2.1)进行举例.

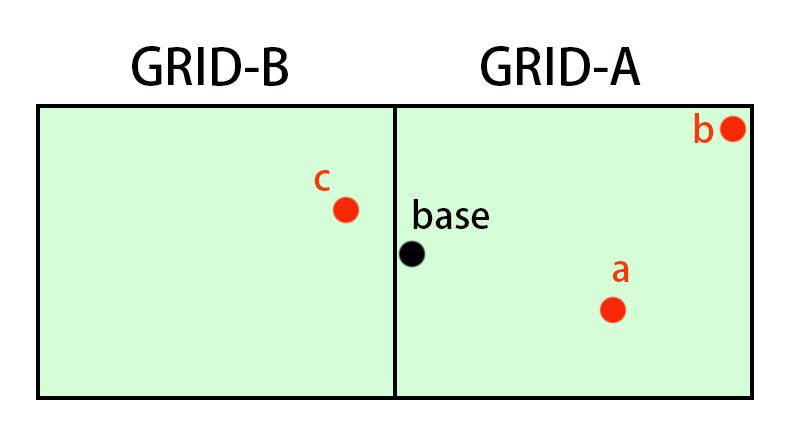


图4.2.1 基于网格的划分

在使用固定地理网格划分时, a, b, c, base四个点会出现以下错误划分:

Cluster A: (a, b, base)

Cluster B: (c)

而事实上base与c距离远远小于base与a, b间距离,不应将base与c归为不相同的划分而将base与a, b当作同一个簇

#### 4.2.2 街道名划分法的缺陷

以 ”长乐路” 与 “长浩路” 为例(图4.2.2)

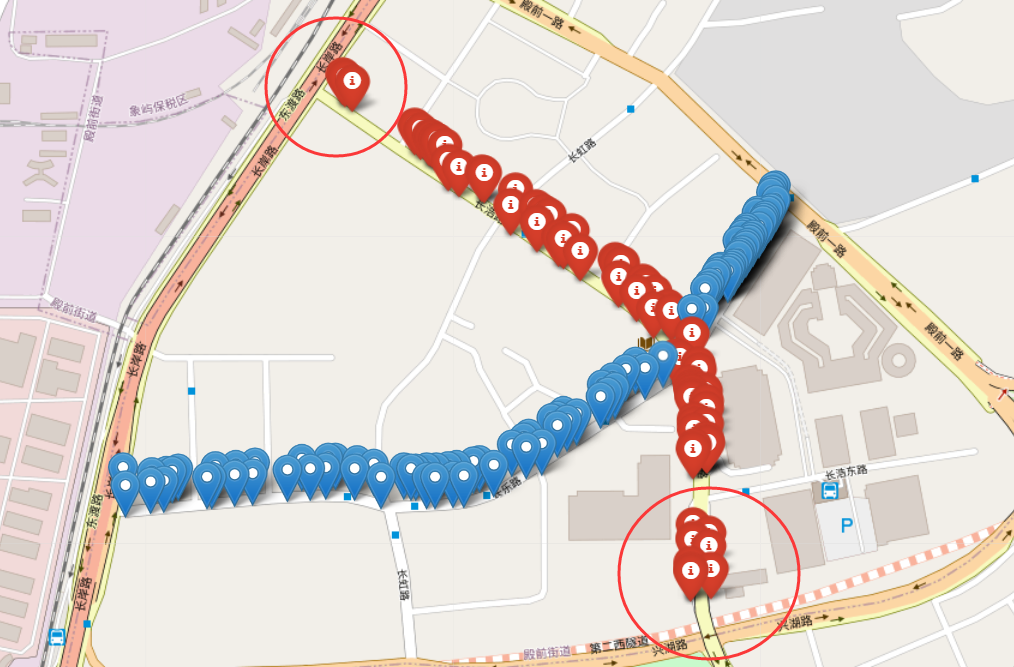


图4.2.2 长乐路(蓝色标记)与长浩路(红色标记)停车围栏位置

部分区域(如红圈所示)的共享单车围栏存在非常明显的,与同街道其余点分离的情况, 而在长浩路与长乐路的交界处(包含部分长乐路停车围栏与部分长浩路停车围栏),停车围栏的离散程度极低, 应作为同一簇处理.所以根据路名简单地将长浩路作为一簇, 将长乐路作为另一个簇无法确保簇内数据的密集程度.

#### 4.2.3 Kmeans聚类划分方法的原理及缺陷

Kmeans非常适合处理空间直线距离=实际距离, 且已知分类树凸集, 然而在实际地理信息中, 经常出现”内外环”, “两条直线距离极近的平行街道不互通”的状况.

即, 出现不连通的非凸集合时,会出现错误的分类



图4.2.3.1 Kmeans 误分类情形



图 4.2.3.2 正确分类

并且在本次项目中,我们无法确定簇数量,即n\_clusters无法定义.

所以在此不考虑使用在其他领域较常使用的Kmeans, 而需要使用可聚类任意形状数据, 参数中不需要簇数量的算法.

#### 4.2.4 DBSCAN(密度聚类)缺陷

DBSCAN是目前较为常用的,聚类不规则形状簇的算法.

本项目曾经尝试过使用该算法进行聚类, 发现了以下问题:

1. DBSCAN对epsilon参数极为敏感(该参数的细微变化都会导致结果截然不同, 而如果不知道恰当的epsilon, 则无法进行聚类)

2. 存在链式传导, 即只要有少量的点断开, 就会导致本应被分为同一个簇的点分裂为多个簇

如,在聚类”乱停车”点的时候我们发现,箭头所指处两个簇距离极近,本应进行合并却被错误地分为不同的簇, 中间部分被错误划分为离群点



图4.2.4.1 DBSCAN下错误分裂的簇

所以,虽然我们在之前的DBSCAN(eps=33米, min\_samples = 1)中取得了20.2221分的算法分数,但仍然不将此算法作为区域划分策略(之前未经清洗的数据获得过21.4681分, 但是基于科学角度无法将其作为可用答案)



图4.2.4.2 未清洗的数据获得的算法分数



图4.2.4.3 DBSCAN(eps = 33米, min\_samples = 1)采用留存流量密度与留存流量zscore之和排序获得的算法分数

#### 4.2.5 最终方案采取的划分方式: HDBSCAN

HDBSCAN(Hierarchical Density-Based Spatial Clustering) 是DBSCAN与层次聚类算法的结合体, 可用于发现任意形状的簇, 且在存在少量不连续点时,也不会错误地将簇分裂.

HDBSCAN与其他算法的效果对比可见HDBSCAN官方文档

<https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html>

在这里仅展示其与DBSCAN聚类的效果区别

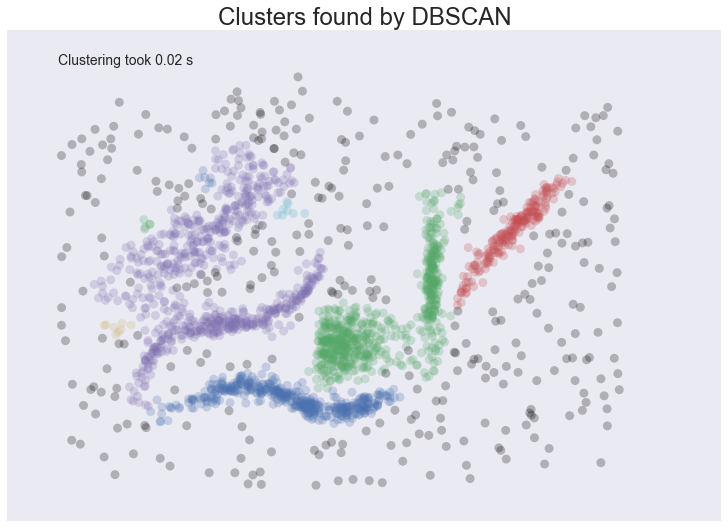


图4.2.5.1 DBSCAN聚类效果(eps:0.025)

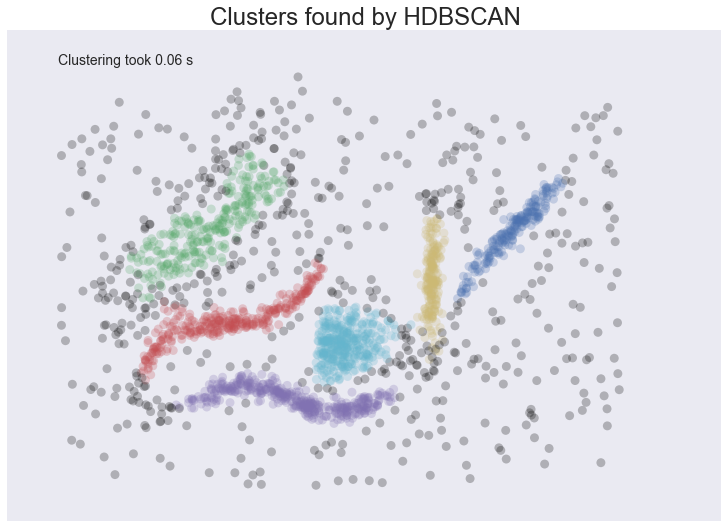


图4.2.5.2 HDBSCAN聚类效果(min\_cluster\_size:15)

在聚类本项目”乱停车”数据点时,也获得了更好的聚合效果(DBSCAN的错误分类见图4.2.4.1 DBSCAN下错误分裂的簇)



图4.2.5.3 HDBSCAN下的簇,分离性更好

#### 4.2.6 基于HDBSCAN区域划分结果

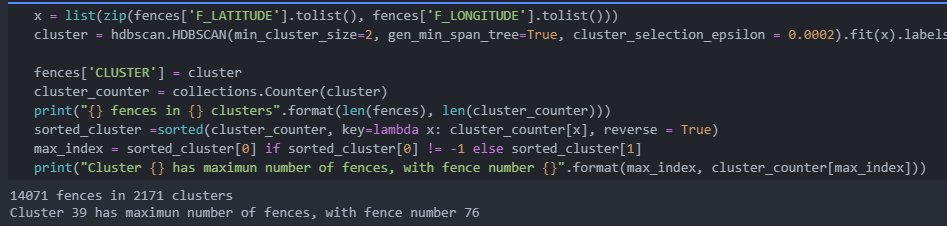


图4.2.5.1 区域划分结果

参数:

min\_cluster\_size = 2 – 意为临近的两个共享单车停车围栏即可单独成为一个区域

gen\_min\_span\_tree=True – 生成最小生成树

cluster\_selection\_epsilon = 0.0003 – 若两个簇间距小于33米,则强行合并. 这个操作的意义是,如果两个围栏中心点距离在33米内, 那么是很容易通过步行达到的(即使是33米车道也为双向六车道以内规格,较易通行). 该参数可不填, 但填写之后可以达到更好的聚类效果

使用中心点作为一个围栏的特征点进行划分. 大型围栏可自成一簇,合并”小而紧”的围栏

最终生成了1733个簇和cluster = -1的离群点集, 其中包含个体数量的一个簇中有100个停车围栏.在folium中进行观察,发现该区域的停车围栏是非常密集的,符合一个”区域”的定义

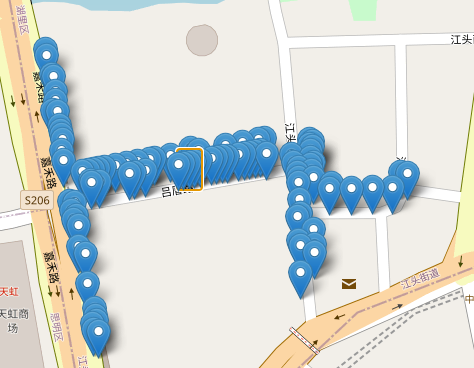


图4.2.5.2 围栏数量最大的簇的 folium可视化结果

在这之后,我们还将每个离群点单独作为一个簇,以便其后进行统计

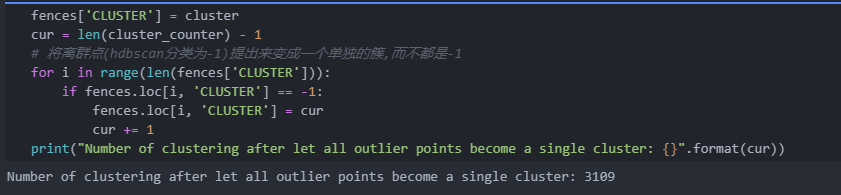


图4.2.5.3 最终的簇数量

最终簇数量为3109

### 4.2 基于围栏大小设置停车允许范围(范围外车辆属于乱停车, 不计入)

在观察fence\_position.csv时, 我们发现不同围栏的大小差距非常之大. 而以大型共享单车围栏边界为基础, 延伸相同长度之后覆盖的额外范围是远远大于小型共享单车围栏的.

以 1 \* 1 和10 \* 10 的围栏为例:

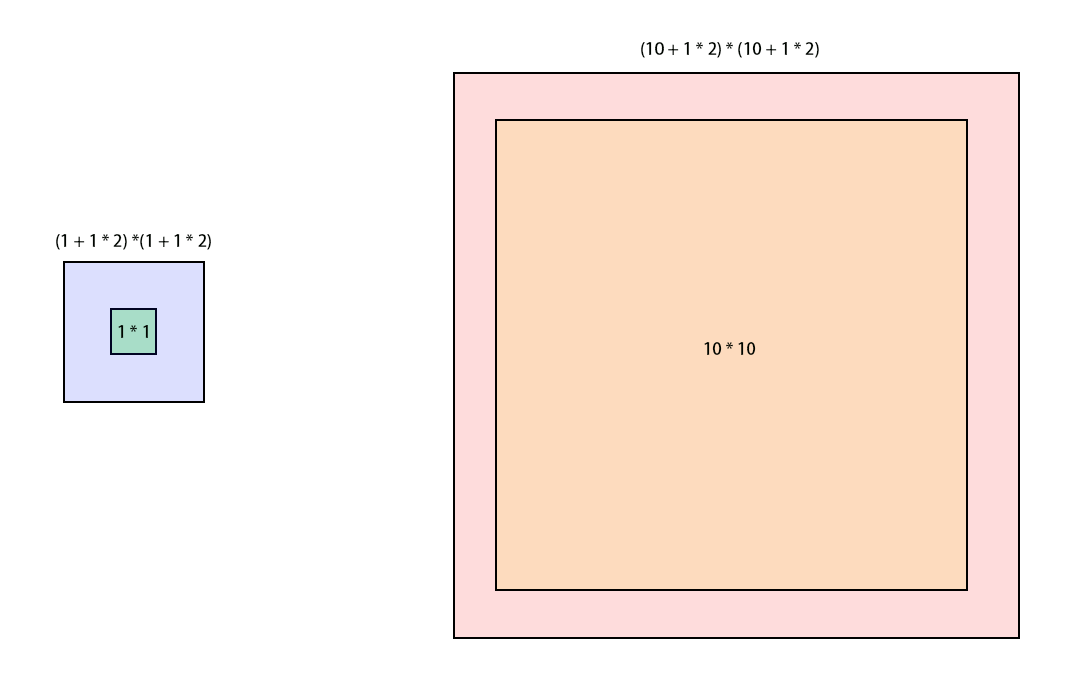


图4.3.1.1 1 \* 1 停车围栏和10 \* 10停车围栏分别往外扩张1米

1 \* 1围栏向外扩张1米后,额外增加的面积只有9 – 1 = 8 平方米,而 10\* 10围栏向外扩张1米后,额外增加的面积变为144-100=44平方米

由此可知, 大型围栏外围调度难度远远高于中小型围栏, 因此对于大型围栏,应做特殊处理减少外围可容忍的额外停车距离.

基于此理论, 为了方便分类,我们加入了CENTER\_DISTANCE维度: 车辆与围栏中心点的距离

对于bikes\_data.csv中的所有非异常数据, 我们将其拆分为可计入停车围栏流量的点和无法计入的点.

在实际的自行车调度中, 对于中小型围栏,调度人员步行搜集车辆的活动范围几乎一致,而对于大型共享单车围栏,调度人员更趋向于使用驾车+步行的方式收集该围栏附近所有的自行车.

所以,我们可将”无法计入”(也可称乱停车)点定义为: 离停车围栏中心点40米外(针对中小型共享单车围栏),或离停车边界20米外(针对大型共享单车围栏)的点.这两种范围外的共享单车是调度人员较不易搜集的.

1. bikes\_valid = bikes[((bikes['CENTER\_DISTANCE'] <= 40) & (bikes['DISTANCE'] != -1)) | (bikes['DISTANCE'] <=20) & (bikes['DISTANCE'] != -1)]
2. bikes\_outliers = bikes[~bikes['BID'].isin(bikes\_valid['BID'])]

代码中DISTANCE为-1的数据是无法被geohash网格覆盖到的停车点,即距离共享单车围栏边缘最少有153米的开车/锁车点, 显然属于”乱停车”范畴.

进行定义后, 不同面积的共享单车围栏所容忍的停车范围如图4.3.1.2所示:

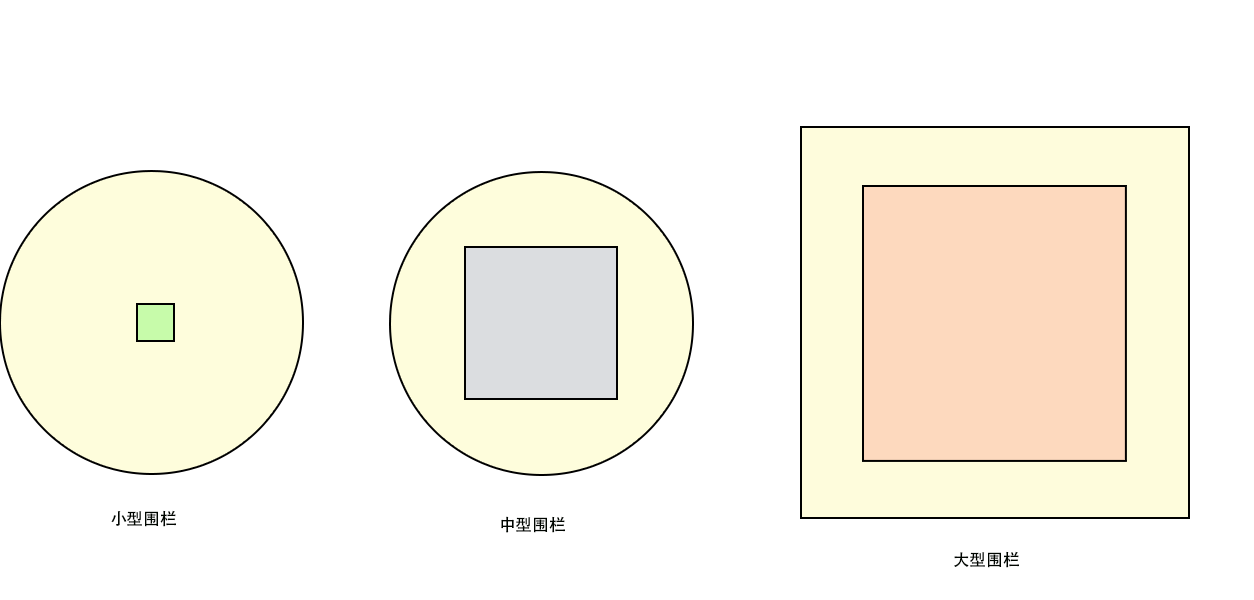


图4.2.1.2 不用面积共享单车围栏对应的停车容忍范围(中小型一致,大型往外扩张一个较短距离)

进行剔除后, 数据量由339810降低为282383, 产生了57427个停车区域外(不计入流量)的点

### 4.3 开/关锁数据的进一步挖掘与可视化

#### 4.3.1 观察不同日期下的开/关锁数据分布

我们在之前提出过疑问:12.25为西方圣诞节, 那么这一天的早高峰时段的开关锁地区分布是否会与其他日期不同? 为此我们使用了matplotlib进行了简单可视化

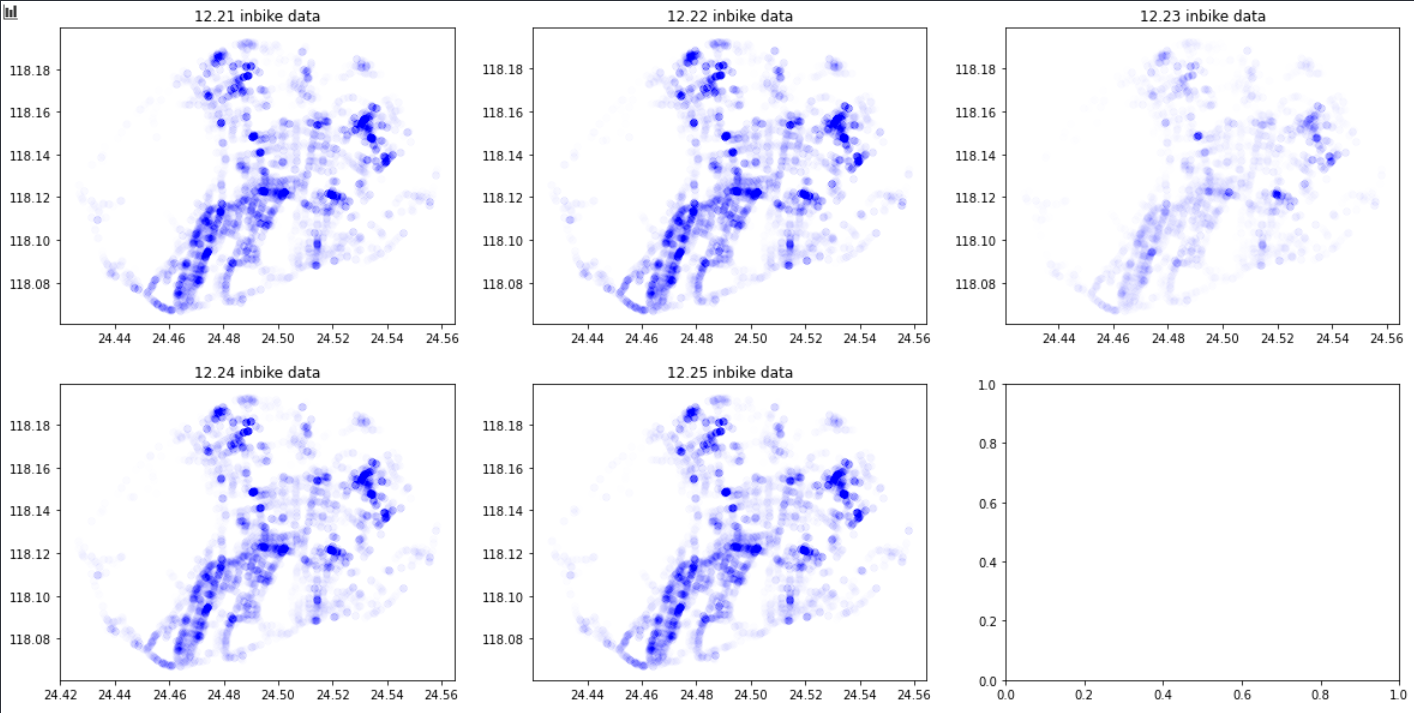


图4.3.1 12.21-12.25早高峰关锁地点分布

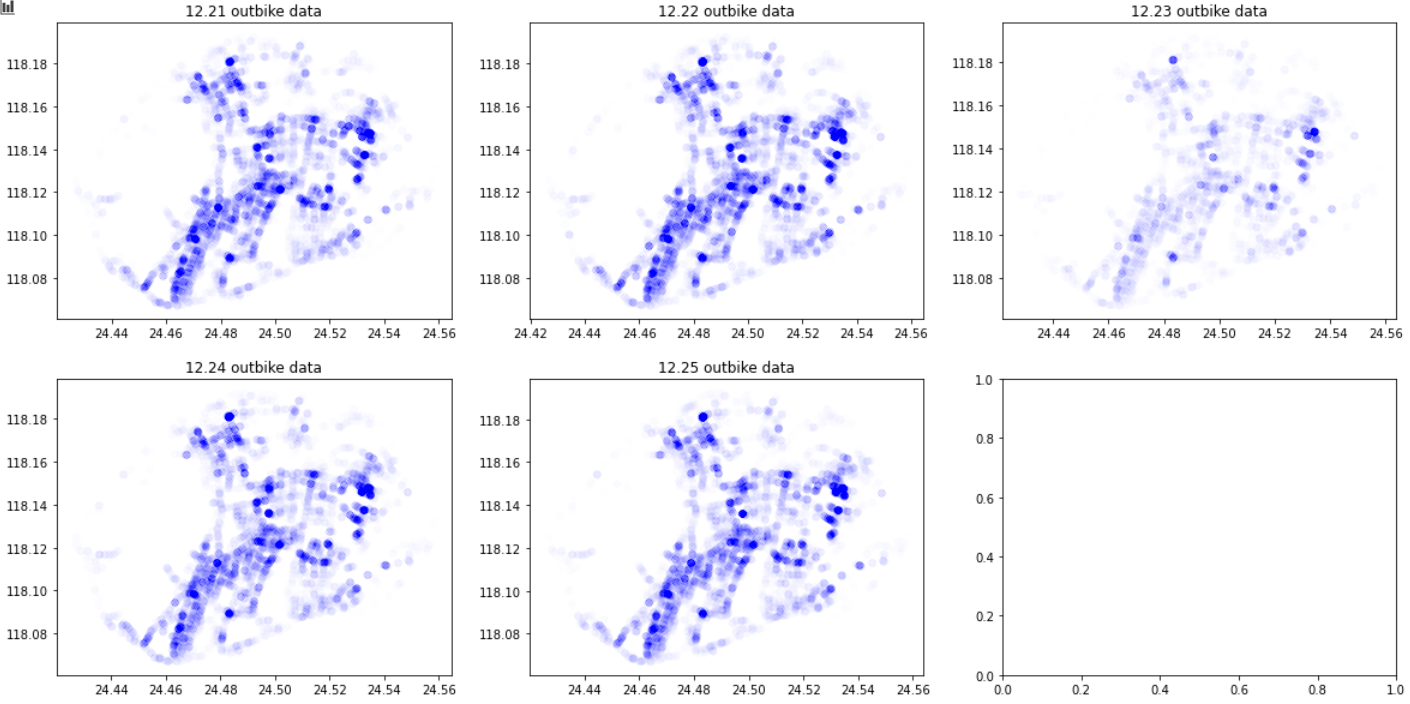


图4.3.2 12.21-12.25早高峰开锁地点分布

从图中可以看出,12.21, 12.22, 12.24, 12.25四天开/关锁数据点分布都十分类似,而12.23自行车使用情况明显较少.为了防止某一地区有两天以上天自行车滞留量为负,而剩下的时间为一个极大的数字导致的将该点误认为潮汐点的错误发生,我们在后续将停车围栏聚类前/类后对应地加了一个ACTIVE\_DAYS(有多少天流量为正)维度观察这一个停车围栏/一个簇中的停车围栏是否存在该种错误,对于单个围栏的ACTIVE\_DAYS,计算代码如下:

1. **def** peek\_day(a, b, c, d, e):
2. cnt = 0
3. **for** day **in** [a, b, c, d, e]:
4. **if** day > 0:
5. cnt += 1
6. **return** cnt
7. fences['ACTIVE\_DAYS'] = fences.apply(**lambda** x: peek\_day(x['21\_FLOW'], x['22\_FLOW'], x['23\_FLOW'], x['24\_FLOW'], x['25\_FLOW']), axis = 1)

由于需要研究的是以7时至9时为一个整体的共享单车潮汐状态,所以在此不对单日时间进行切片. 单日时间维度上的自行车流量预测方式将在第八节实施建议中进行描述.

#### 4.3.2 “乱停车”现象集中时段和地区的探索

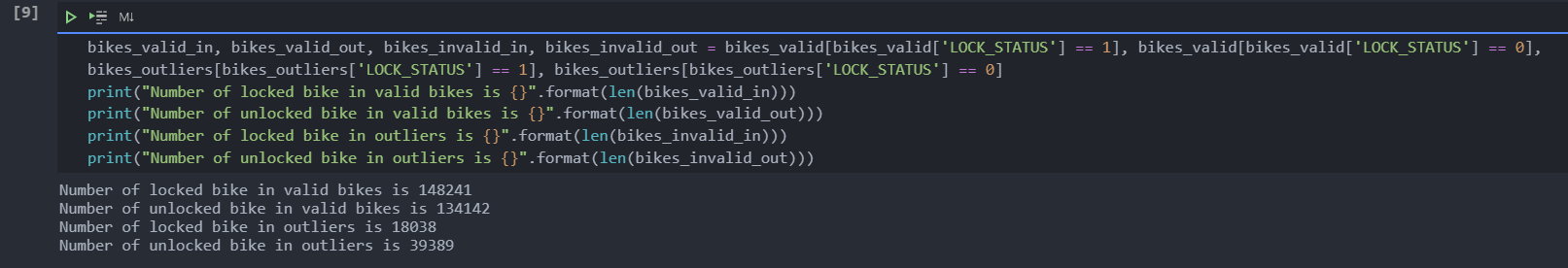


图4.3.2.1 开/关锁数据在停车范围内/外的数量

可以看出在区域内的关锁车辆较开锁车辆多出约10.5%, 在区域外的关锁车辆则比关锁车辆少约54.2%.

我们对此数据产生了浓厚兴趣: 开关锁数据在停车范围内/外比例差距如此之大,是否与7-9时之间的时间段有关?

就这个问题, 再次对数据按时间-停放量进行可视化处理



图4.3.2.2 范围内外开/关锁数量随时间的变化

可以看出在允许范围外停车/取车量与在允许范围内停车/取车量的变化趋势没有特别明显的变化

对停车/取车数据的空间分布进行可视化:

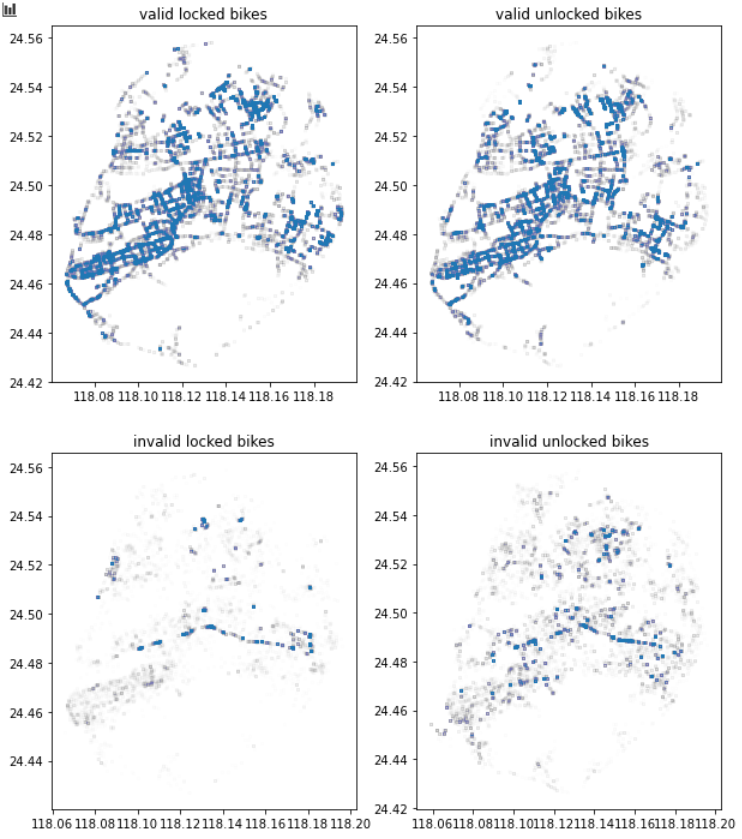


图4.3.2.3 范围内外开/关锁数量与空间关系, 颜色越深重叠越多

可以看出有部分区域”乱停车”现象较为严重,具体细节将在8.2 基于早高峰及早高峰之前”乱停车”数据的建议进行讨论

## 五. 潮汐点识别方法及结果分析

基于之前的数据可视化, 整理及聚类结果(见4.2.6. 基于HDBSCAN的区域划分结果), 我们新建了一个名为cluster\_detail的pandas dataframe.

1. *# 单个cluster的总占地面积*
2. cluster\_area = fences.groupby('CLUSTER').agg({'AREA':'sum'})
3. *# 单个cluster中的围栏数量*
4. cluster\_cnt = fences.groupby('CLUSTER').agg({'FID':'count'})
5. *# 21 - 25日单个cluster中的留存流量*
6. cluster\_21\_flow = fences.groupby('CLUSTER').agg({'21\_FLOW':'sum'})
7. cluster\_22\_flow = fences.groupby('CLUSTER').agg({'22\_FLOW':'sum'})
8. cluster\_23\_flow = fences.groupby('CLUSTER').agg({'23\_FLOW':'sum'})
9. cluster\_24\_flow = fences.groupby('CLUSTER').agg({'24\_FLOW':'sum'})
10. cluster\_25\_flow = fences.groupby('CLUSTER').agg({'25\_FLOW':'sum'})
11. *# 汇总*
12. cluster\_detail = pd.merge(cluster\_cnt, cluster\_area, how = 'left', left\_index = True, right\_index = True)
13. for i in range(21, 26):
14. cluster\_detail = pd.merge(cluster\_detail, eval('cluster\_' + str(i) + '\_flow'), how = 'left', left\_index = True, right\_index = True)
15. cluster\_detail['FLOW'] = cluster\_detail.apply(lambda x: x['21\_FLOW'] + x['22\_FLOW'] + x['23\_FLOW'] + x['24\_FLOW'] + x['25\_FLOW'], axis = 1)
16. cluster\_detail['ACTIVE\_DAYS'] = cluster\_detail.apply(lambda x: peek\_day(x['21\_FLOW'], x['22\_FLOW'], x['23\_FLOW'], x['24\_FLOW'], x['25\_FLOW']), axis = 1)
17. cluster\_detail['FLOW\_DENSITY'] = cluster\_detail.apply(lambda x: x['FLOW']/x['AREA'], axis = 1)

fences中的21\_FLOW, 22\_FLOW, 23\_FLOW, 24\_FLOW, 25\_FLOW 为之前通过每日的流入/流出数据计算的, 每个围栏的留存流量

计算公式: 留存流量 = 当日流入流量- 当日流出流量

若流出流量大于流入流量,则留存流量为负数

FLOW为簇中所有日期的留存流量总和, ACTIVE\_DAYS为簇中留存流量大于0的天数统计, FLOW\_DENSITY为簇中留存流量与共享单车围栏面积的比, 即留存密度

完成后的cluster\_detail信息如图5.0.1所示:

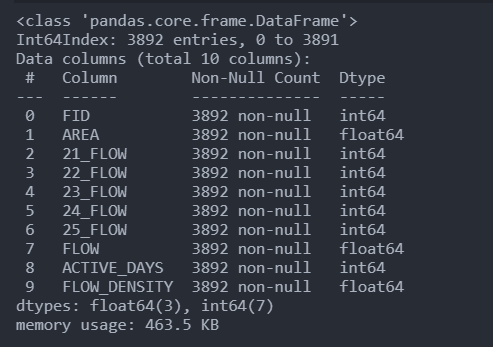


图5.0.1 cluster\_detail

我们按照以下三种不同的指标识别了7时至9时潮汐现象最突出的40个区域

1. “车多” – 按区域内自行车”留存数量”进行排序

2. “车挤” – 按区域内自行车”留存密度”进行排序

3. “又多又挤” – 按区域内自行车留存数量与留存密度的综合指标进行排序

### 5.1 “车多” - 按区域内自行车”留存数量”进行排序的结果

将cluster\_detail按照FLOW字段(自行车留存数量)降序排列

1. sorted\_cluster\_detail = cluster\_detail.sort\_values("FLOW", ascending = False)
2. sorted\_cluster\_detail.head(40)

观察前40个区域



图5.1.1 留存数量最大的40个区域信息

发现ACTIVE\_DAYS均为5,即这些地区在12.21至12.25的自行车留存流量均为正数(若ACTIVE\_DAYS < 3则代表该地区自行车流量不稳定), 所以可以直接作为流量最大的40个地区计入答案

这40个地区中,一共包括559个共享单车围栏数量

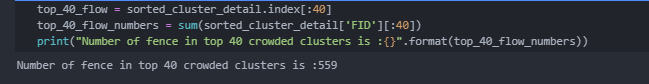


图 5.1.2 留存流量最大的40个区域所包含的共享单车围栏数量

用folium可视化潮汐停车围栏位置:

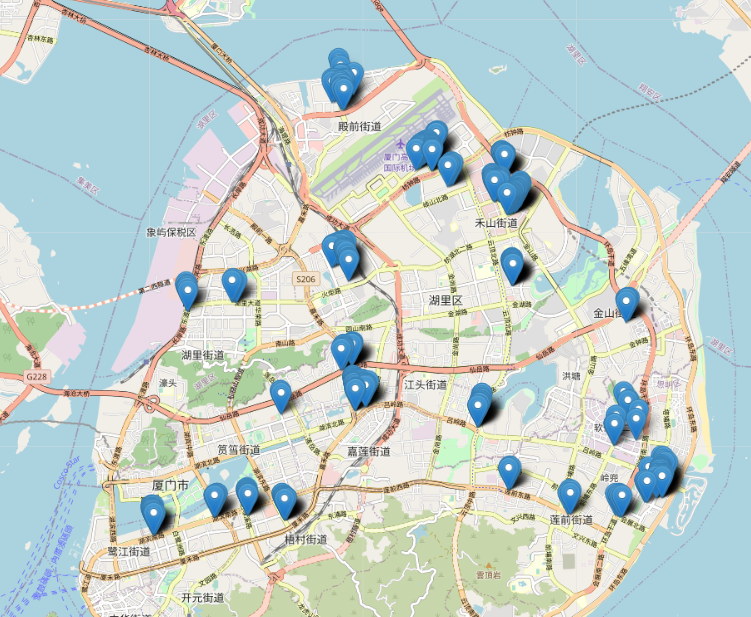


图5.1.3 留存流量最大的40个区域的地图位置

提交结果:



图5.1.4 提交结果

### 5.2 “车挤” - 按照区域内自行车”留存密度”进行排序的结果

将cluster\_detail按照FLOW\_DENSITY字段进行排序, 流量密度最大的前40个地区的ACTIVE\_DAYS也均为5

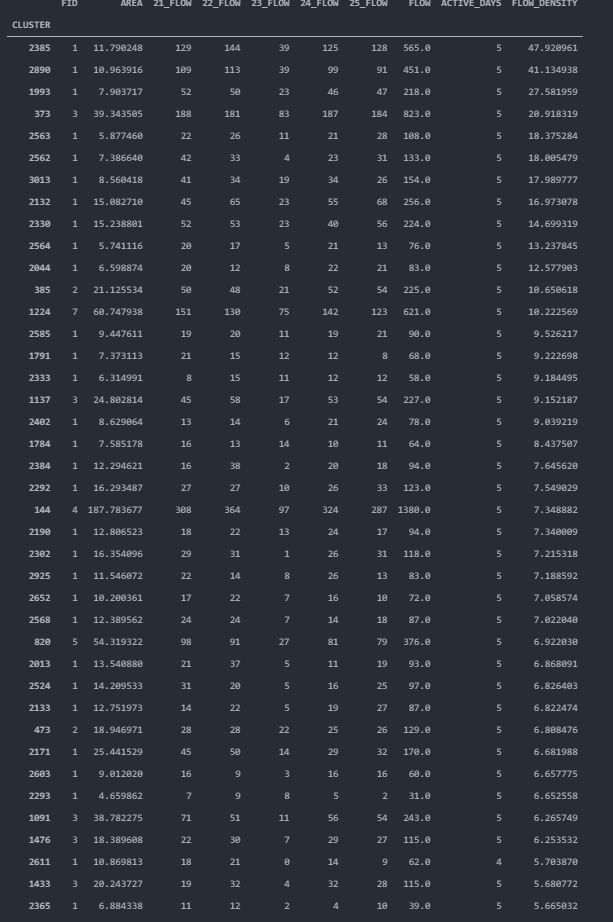


图5.2.1 留存密度最大的40个区域信息

密度前40区域内只有64个围栏. 由图5.1.1中FID的数值,可以看出大部分留存密度较大的围栏都是单独成簇,即距离其他围栏较远

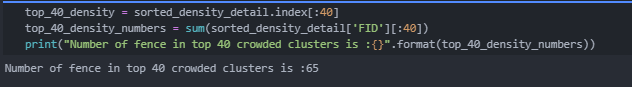


图 5.2.2 留存密度最大的40个区域所包含的停车围栏数量

经Folium可视化后得到这些围栏在地图上的分布

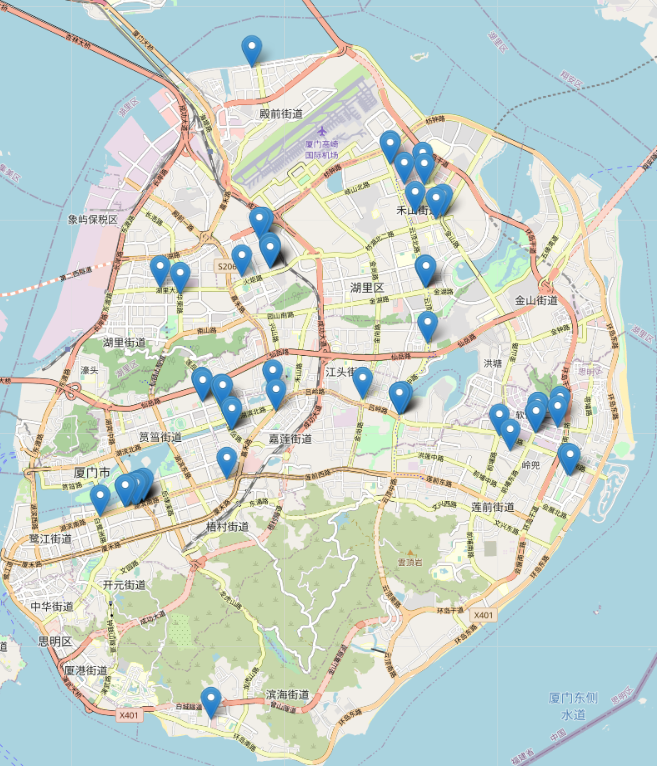


图5.2.3 留存密度最大的40个区域的地图位置

提交结果:



图5.2.4 提交结果

该方法提交结果较不理想

### 5.3 “又多又挤” – 按“留存数量”与”留存密度”的综合指标进行排序的结果

将留存流量和留存流量密度采用z-score标准化后相加, 以此排序. 这样可以兼顾留存流量和留存流量密度两个指标.

因数据存在极端值, 所以不采用归一化.

代码实现及排名前40的簇如下:

1. flow\_zscore = (cluster\_detail['FLOW'] - cluster\_detail['FLOW'].mean())/cluster\_detail['FLOW'].std()
2. density\_zscore = (cluster\_detail['FLOW\_DENSITY'] - cluster\_detail['FLOW\_DENSITY'].mean())/cluster\_detail['FLOW\_DENSITY'].std()
3. cluster\_detail['MIXED\_SCORE'] = flow\_zscore + density\_zscore

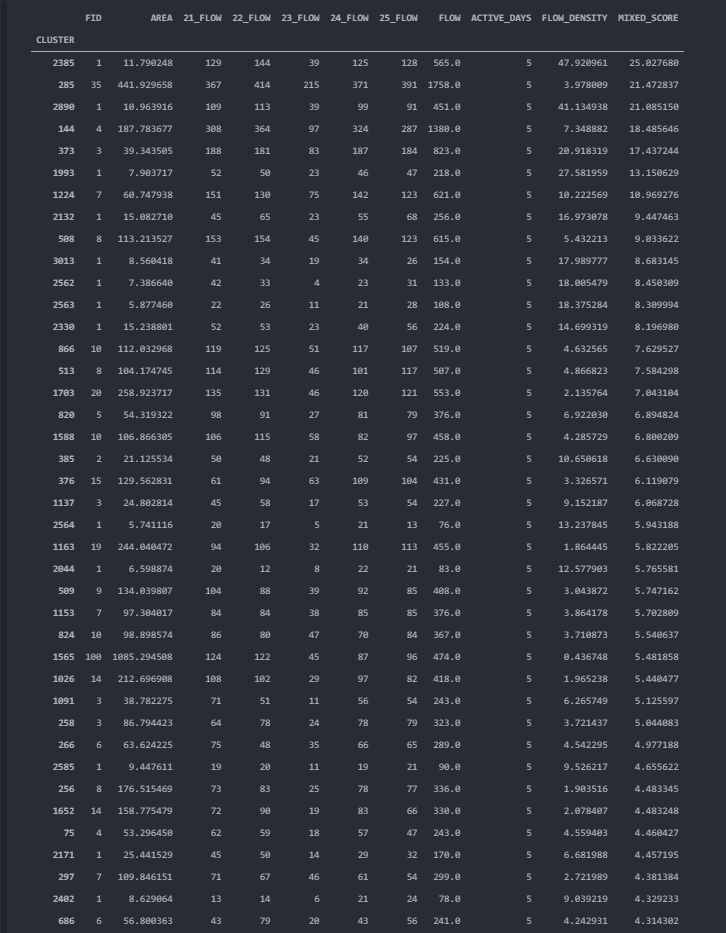


图5.3.1 混合方法分数最高的40个区域信息

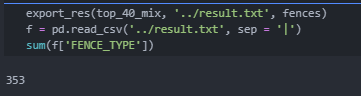


图 5.3.2 混合方法分数最高的40个区域所包含的停车围栏数量

前40区域包含353个围栏

Folium 可视化后的围栏分布:



图5.3.3 混合方法分数最高的40个区域的地图位置

提交结果:



图5.3.4 提交结果

## 六.应用成果

## 七. 作品价值

## 八. 实施建议

就早高峰时期部分潮汐地区”停车难”的问题,我们建议采用两种解决方式:

1. 引导用户前往较近的, 流量或预计流量较小(“不繁忙”)的围栏

2. 在部分”乱停车”现象集中且离停车围栏过远的地区增设停车围栏

### 8.1 针对”还不进”问题的用户调度策略

#### 8.1.1 寻找”不繁忙” 单车围栏的方法

经过对哈啰单车调度人员的调查,我们得知共享单车调度是全天进行的.所以,我们希望将潮汐点用户引导至较近的:

1.现存车辆较少(流量小或者刚刚进行过调度的)

2. 预计入流量较小

的停车围栏

在本项目数据中, 距离较近但流量远少于同簇其它围栏的共享单车停车围栏是存在的,可通过fences中的FLOW和ACTIVE\_DAYS进行观察

以留存流量指标得到的结果为例:

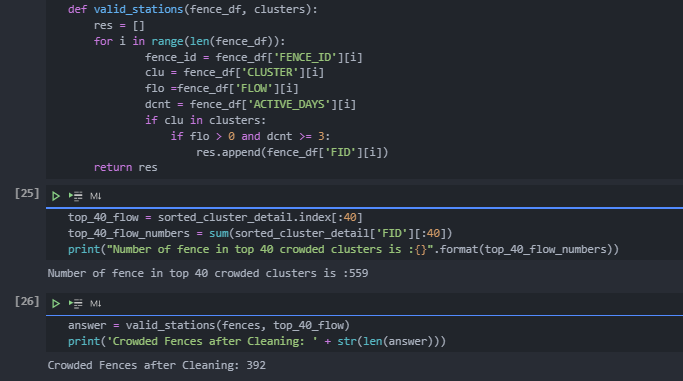


图8.1.1.1 前40繁忙的簇中依然存在流量较小/流量为正天数小于3的围栏

预计流量可通过将共享单车围栏的流量数据按照时及分划分后进行时间序列数据分析. 由于只有五天的数据,所以这里不进行代码实现,仅提供方法.

#### 8.1.2 调度过程实施步骤

调度过程的实现步骤如下:

1. 采用geohash算法获取当前坐标点的geohash并列举出当前geohash单元格及临近八个单元格的所有围栏

2. 使用Dijkstra算法算出当前自行车所在点与这些围栏之间在有向有权图中的最短路径, 抛弃用户接受范围(max\_distance)外的点.

3. 移除最短路径距离大于用户可接受距离的围栏

4. 将停车围栏现存车辆 – 流量, 当前距离, 正流量天数标准化, 分别乘以不同的正经验权重k1, k2, k3. 设现存车辆-流量为x1, 当前距离为x2, 正流量天数为x3, 那么最终得到一个分数s = k1x2 + k2x2 + k3x3, 将用户引导至s最小的点.

我们在./tools/suggestions.ipynb 针对望海路0\_R\_1旁的一个停车点做了模拟引导, 其中因为缺乏图数据无法使Dijkstra算法, 所以用地理直线距离作为代替. 由于无法得知现存车辆数,所以使用留存流量代替该指标进行计算.

模拟引导算法中的参数为: max\_distance = 300, k1 = 0.3, k2 = 0.5, k3 = 0.2 (实际应用时应修改)

模拟结果:



图8.1.2.1 模拟引导结果

### 8.2基于早高峰”乱停车”数据的建议

使用HDBSCAN(min\_cluster\_size=500, 意为一个簇的最小数据量为500)对前面所定义的停车允许范围外的自行车(包括开锁和关锁.开锁处若为允许范围外,说明在7时之前出现了”乱停车”现象)进行聚类后,发现存在19处较为集中的”乱停车”点

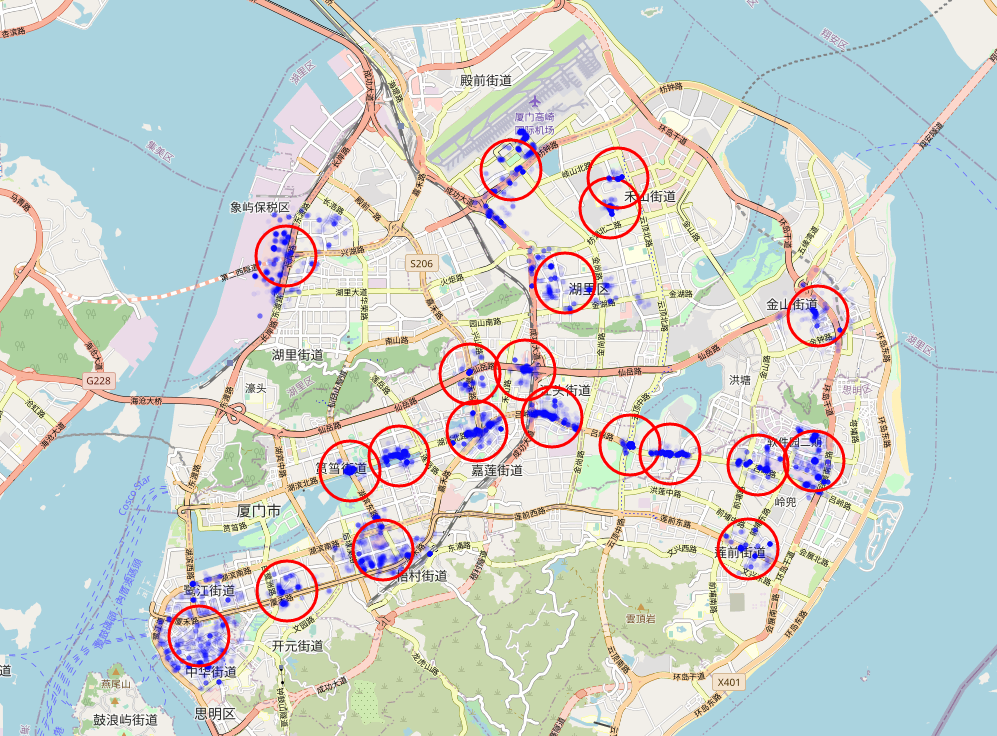


图8.2.1 “乱停车”现象集中点(为了绘图方便,使用红色大圈圈出.实际为不规则形状)

以成功大道与吕岭路交界处深绿色”乱停车”集中点为例, 将”乱停车”点重新绘制为红色



图8.2.2 成功大道与吕岭路交界处”乱停车”点(红色小点)与停车围栏位置(蓝色大点)

可见停车围栏与”乱停车”现象密集区域直线距离在50米外,且没有直线连通路径. 需要跨过吕岭路并转至对面龙伏路.

若交通条件允许,应在泰和花园处设置停车围栏, 以方便市民停放共享单车.

在中山路步行街附近的”乱停车”现象也较为类似: 较近的共享单车围栏分布在镇海路,厦禾路和轮渡一侧,中华街道内测. 而中山路步行街处道路狭窄,车流量大, 店铺众多, 共享单车是此处最为便捷的出行方式.



图8.2.3 中山路步行街附近”乱停车”现象

为此我们前往该处进行了实地调查, 调查发现:

1. 咨询执勤交警得知, 中山路步行街内不允许骑自行车进入

2. 思明南路与思明西路部分未开业的商铺门口存在不少随意停放的共享单车

3. 思明南路,思明北路,思明东路,思明西路道路狭窄,两侧均为商铺,不适合设立共享单车停车围栏

4. 该区域小巷极多, 调度车辆(在之前对于调度人员的调查中发现调度车辆均为小型卡车)难以进入, 调度人员也很难在纵横交错的小巷中准确找到被随意停放的共享单车

为此给出几点可以改进猜想

1. 观察到中山路步行街入口处较为宽敞, 是否可以在市政规划及市容允许的前提下于此处设立共享单车围栏?

2. 因小巷内散落共享单车的收集难度过高, 应适当提高”乱停车”的处罚力度

# 九. 参考文献

HDBSCAN Library Documentation <https://hdbscan.readthedocs.io/en/latest/index.html> and its source code

Scikit-Learn Documentation <http://scikit-learn.github.io/stable> and its source code

Malzer, C., & Baum, M. (2019). A Hybrid Approach To Hierarchical Density-based Cluster Selection. arxiv preprint 1911.02282.

聚类算法大盘点

<https://liangyaorong.github.io/blog/2017/%E8%81%9A%E7%B1%BB%E7%AE%97%E6%B3%95%E5%A4%A7%E7%9B%98%E7%82%B9/#3.2>

Geohash Generator <https://www.movable-type.co.uk/scripts/geohash.html>

Geohash Source code <https://pypi.org/project/python-geohash/#description>

Coggle DCID 2021 教程<https://coggle.club/learn/dcic2021/>