《时空数据处理与组织课程实习》

实习报告

学　　院: 遥感信息工程学院

班　　级: 22F12

学　　号: 2022302131042

姓　　名: 牛驰原

实习地点: 三区 5-220

2024 年　　5 月　　21 日

1.实验目的

本次课程设计的任务就是利用大数据工具 Spark ，综合采用课程中介绍的方法进行车辆轨迹的初步分析，包括车辆停留点分析、加减速监测等，为以后进一步的海量轨迹数据的深入挖掘和应用打下基础。

2.实验环境

Python 版本：3.11.7；

Java 版本：1.8.0；

Hadoop 版本：3.2.2；

Spark: spark-3.4.0-bin-hadoop3-scala2.13;

第三方库：pyspark、findspark、psutil、math；

系统环境：Windows10/11、VSCode 的 conda 虚拟环境。

配置环境变量：SPARK_HOME、HADOOP_HOME、JAVA_HOME 至系统变量。

3.实验内容和步骤

3.1 实验内容

使用课程实习过程中搭建的 Windows 下的 Spark 环境，采用 Python 编程，数据采用微软的车辆轨迹数据，进行车辆轨迹分析。

3.1.1 实验目的

轨迹数据是时空环境下,通过对一个或多个移动对象运动过程的采样所获得的数据信息,包括采样点位置、采样时间、速度等,这些采样点数据信息根据采样先后顺序构成了轨迹数据。近年来,随着各种定位技术的发展和普及,以及无线互联网的高速发展推动下的移动智能终端的更新换代,民用 GPS 设备在移动终端上得到了非常广泛的使用,海量的轨迹数据在日常生活中正在日益积累,如何有效挖掘轨迹数据背后的信息成为一个备受关注的热点。

3.1.2 实验数据

GeoLife GPS Trajectories

该 GPS 轨迹数据集出自微软研究 GeoLift 项目。从 2007 年四月到 2012 年八月收集了 182 个用户的轨迹数据。这些数据包含了一系列以时间为序的点，每一个点包含经纬度、海拔等信息。包含了 17621 个轨迹，总距离 120 多万公里，总时间 48000 多小时。这些数据不仅仅记录了用户在家和在工作地点的位置轨迹，还记录了大范围的户外活动轨迹，比如购物、旅游、远足、骑自行车。

这个数据集可以用来进行用户活动相似度估算，移动模型挖掘，用户活动推荐，基于位置的社交网络，位置隐私，位置推荐。

一个文件夹存储一个用户的 GPS 日志，这些日志文件都被转换成了 plt 格式。为了避免时间区间问题，统一使用了 GMT 格式的时间表示。我们实验采用的是第 180 号数据的第一个 plt 文件，编号为 20090415134400，其具体格式为：

```
Line 1…6 are useless in this dataset, andcan be ignored. Points are described in following lines, one for each line.
Field 1: Latitude in decimal degrees.
Field 2: Longitude in decimal degrees.
Field 3: All set to 0 for this dataset.
Field 4: Altitude in feet (-777 if notvalid).
Field 5: Date - number of days (withfractional part) that have passed since 12/30/1899.
Field 6: Date as a string.
Field 7: Time as a string.
Note that field 5 and field 6&7represent the same date/time in this dataset. You may use either of them.

Example:
39.906631,116.385564,0,492,40097.5864583333,2009-10-11,14:04:30
39.906554,116.385625,0,492,40097.5865162037,2009-10-11,14:04:35
```

图 1. 数据样式

而且每个 plt 文件的前六行是有关该点集数据的一些说明。

3.1.3 主要内容

（1）车辆速率计算

从原始轨迹数据中获取每个轨迹点的经纬度坐标，利用经纬度坐标计算两点距离，通过该轨迹点与前一个点的距离和两个轨迹点的时间差，计算该点的即时速率。根据实际意义，规定起点和终点的即时速率为零。

（2）车辆停留点分析

停留点识别的算法描述如下：

遍历轨迹点，检查速率值，设置停留认定阈值，若小于阈值，则将上一个点作为停留开始点。继续遍历轨迹点，只要轨迹点的速率值仍小于阈值，则将该点

作为停留点。

（3）车辆加减速分析

加速和减速都是持续性的过程，而突发的速率变化则可能是数据采集或预处理阶段的误差引起的正常波动，因此加/减速检测方法可以采用寻找至少连续 N（N ≥ 2）个点速率单调变化的片段；除此之外，部分车辆在匀速行驶过程中也可能存在微小的速度差异，因此我们需要设置判定其加减速的加速度阈值，当加速度绝对值大于该阈值时，我们再运用加/减速检测方法认定其是否处于加/减速状态。

3.2 实验步骤

实验的主要步骤为：数据读取与预处理——数据清洗——时间序列分析——车辆瞬时速度计算——停留点判断——加减速分析——结果输出。具体实现过程如下：

3.2.1 数据读取与预处理

在读取数据之前要进行环境配置与初始化，导入必要的库和函数并初始化 Spark Session。

```
#coding=utf-8

import findspark
findspark.init()

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, FloatType, DoubleType, IntegerType, StringType
from pyspark.sql import Row
from pyspark.sql.functions import col, radians, lag, when, udf, lead
from pyspark.sql.window import Window
import math
from math import radians, sin, cos, sqrt, atan2

# 初始化 Spark Session
spark = SparkSession.builder.appName("Trajectory Analysis").getOrCreate()
```

图 2. 初始化代码截图

接着进行数据模型的定义，定义数据架构，明确每列数据的类型。

```
# 定义数据架构
schema = StructType([
    StructField("latitude", DoubleType(), True),
    StructField("longitude", DoubleType(), True),
    StructField("unused", IntegerType(), True),
    StructField("altitude", DoubleType(), True),
    StructField("date_numeric", DoubleType(), True),
    StructField("date", StringType(), True),
    StructField("time", StringType(), True)
])
```

图 3. 数据架构定义代码截图

在进行数据结构定义之前，我们要先分析数据文件，可知第三列值均为 0，而第六第七列的数据又是数据记录时刻的日期和时间，是 String 型数据，而另外四列分别是经度、纬度、高程以及时间戳，这四列数据是我们用于计算车辆速度的关键，他们均为浮点型。

在初次尝试的时候，我定义经度、纬度、高程以及时间戳的数据类型均为单精度浮点型也就是 FloatType，但是由于数据本身的原因，各数据间的差值都非常小，如使用单精度浮点型读取数据，得到的结果会非常不精准，可能会出现时间序列下相邻数据时间差值为 0 的情况，因此我们均采用 DoubleType 读取数据。

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLo
24/06/03 00:44:15 WARN NativeCodeLoader: Unable to load native-hadoop libra
+---------+----------+------+---------+-----------+----------+--------+
| latitude| longitude|unused| altitude|datenumeric|      date|    time|
+---------+----------+------+---------+-----------+----------+--------+
|26.162203| 119.94379|     0|492.13153|   39918.57|2009-04-15|13:44:00|
|26.161528| 119.94324|     0|491.84198|  39918.574|2009-04-15|13:44:05|
|  26.1619| 119.94323|     0|491.88004|  39918.574|2009-04-15|13:44:10|
|26.161776|119.943275|     0|491.61948|  39918.574|2009-04-15|13:44:15|
|26.161667| 119.94328|     0| 491.7133|  39918.574|2009-04-15|13:44:20|
| 26.16167| 119.94313|     0| 491.5529|  39918.574|2009-04-15|13:44:25|
|26.161566| 119.94235|     0|203.65495|  39918.574|2009-04-15|13:44:30|
|26.161558|  119.9424|     0| 193.4336|  39918.574|2009-04-15|13:44:35|
|26.161482|119.942474|     0| 184.9649|  39918.574|2009-04-15|13:44:40|
| 26.16146| 119.94249|     0|184.06723|  39918.574|2009-04-15|13:44:45|
| 26.16143| 119.94253|     0|182.18839|  39918.574|2009-04-15|13:44:50|
|26.161413| 119.94255|     0|178.28241|  39918.574|2009-04-15|13:44:55|
|26.161411| 119.94256|     0|178.63869|  39918.574|2009-04-15|13:45:00|
|26.161407| 119.94256|     0| 178.4208|  39918.574|2009-04-15|13:45:05|
|26.161406|119.942566|     0|177.91125|  39918.574|2009-04-15|13:45:10|
|26.161404| 119.94257|     0| 177.6602|  39918.574|2009-04-15|13:45:15|
|26.161404| 119.94257|     0|177.59462|  39918.574|2009-04-15|13:45:20|
|26.161402| 119.94257|     0|177.75227|  39918.574|2009-04-15|13:45:25|
|  26.1614| 119.94258|     0|177.63004|  39918.574|2009-04-15|13:45:30|
|  26.1614| 119.94258|     0|177.47443|  39918.574|2009-04-15|13:45:35|
+---------+----------+------+---------+-----------+----------+--------+
only showing top 20 rows
```

图 4. FloatType 读取数据

```
24/06/03 17:40:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your
+---------+----------+------+----------------+---------------+----------+--------+
| latitude| longitude|unused|        altitude|   date_numeric|      date|    time|
+---------+----------+------+----------------+---------------+----------+--------+
|26.162202|119.943787|     0|492.131541994751|39918.5722222222|2009-04-15|13:44:00|
|26.161528|119.943234|     0|491.841984908136|39918.5722800926|2009-04-15|13:44:05|
|  26.1619|119.943228|     0|491.880032808399| 39918.572337963|2009-04-15|13:44:10|
|26.161775|119.943276|     0|491.619461942257|39918.5723958333|2009-04-15|13:44:15|
|26.161667|119.943281|     0|491.713277559055|39918.5724537037|2009-04-15|13:44:20|
| 26.16167|119.943128|     0|491.552877296588|39918.5725115741|2009-04-15|13:44:25|
|26.161566|119.942352|     0|203.654947506562|39918.5725694444|2009-04-15|13:44:30|
|26.161558|119.942401|     0|193.433599081365|39918.5726273148|2009-04-15|13:44:35|
|26.161482|119.942477|     0|184.964908136483|39918.5726851852|2009-04-15|13:44:40|
|26.161461|119.942491|     0|184.067227690289|39918.5727430556|2009-04-15|13:44:45|
| 26.16143| 119.94253|     0|182.188389107612|39918.5728009259|2009-04-15|13:44:50|
|26.161413|119.942552|     0|178.282404855643|39918.5728587963|2009-04-15|13:44:55|
|26.161411|119.942557|     0|178.638694225722|39918.5729166667|2009-04-15|13:45:00|
|26.161408|119.942562|     0|178.420800524934| 39918.572974537|2009-04-15|13:45:05|
|26.161405|119.942565|     0| 177.91125984252|39918.5730324074|2009-04-15|13:45:10|
|26.161404| 119.94257|     0|177.660200131234|39918.5730902778|2009-04-15|13:45:15|
|26.161403|119.942574|     0|177.594619422572|39918.5731481481|2009-04-15|13:45:20|
|26.161401|119.942577|     0|177.752276902887|39918.5732060185|2009-04-15|13:45:25|
|  26.1614|119.942578|     0|177.630039370079|39918.5732638889|2009-04-15|13:45:30|
|26.161399|119.942579|     0|177.474422572178|39918.5733217593|2009-04-15|13:45:35|
+---------+----------+------+----------------+---------------+----------+--------+
only showing top 20 rows
```

图 5. DoubleType 读取数据

3.2.2 数据清洗

使用其 textFile 函数读取相关数据，得到 RDD 格式的数据 rdd。然后进行数据清洗：首先使用 filter 函数过滤表头和异常数据（去除表头前六行，且过滤海拔为-777 的数据），然后以逗号为分隔符进行数据分割，将每行数据生成 row 对象，再转为 DataFrame 格式的 data，便于后续计算。

```
# 使用RDD来读取并跳过前6行
rdd = spark.sparkContext.textFile('Spark/data/20090415134400.plt')

# 跳过前6行
filtered_rdd = rdd.zipWithIndex().filter(lambda x: x[1] > 5).map(lambda x: x[0])

# 将RDD转换为Row类型，以符合Schema
row_rdd = filtered_rdd.map(lambda x: x.split(",")).map(lambda p: Row(
    latitude=float(p[0]),
    longitude=float(p[1]),
    unused=int(p[2]),
    altitude=float(p[3]),
    date_numeric=float(p[4]),
    date=p[5],
    time=p[6]
))

# 创建DataFrame
data = spark.createDataFrame(row_rdd, schema)

# 过滤海拔为-777的数据
data = data.filter(data.altitude != -777)

data.show(20)
```

图 6. 数据清洗代码截图

对清洗过后得到的数据进行展示，检查是否有误。

3.2.3 时间序列分析

通过分析数据结构，每行数据都有唯一数据标识，也就是 data 和 time，每隔 5s 记录一条数据，时间可作为每条数据的唯一标识，因此对数据采用时间序列分析非常合适。

```
# 准备分析窗口
windowSpec = Window.orderBy("date_numeric")
```

图 7. 时间序列分析窗口代码截图

将所有数据按照时间戳升序排列，得到时间序列分析窗口，方便后续计算，在计算过程中通过 withColumn 函数将速度与加速度的计算结果添加在每条数据中，用于检查更新与展示。

3.2.4 车辆瞬时速度计算

瞬时速度为当前时刻速率，为方便计算，我们将速度计算定义为：速度=距离/时间间隔。因此这个部分我们需要经历距离计算与时间间隔计算，最后带入公式即可求出瞬时速度，即对 DataFrame 进行计算生成 speed 列。

（1）距离计算

我们一般定义距离计算是欧氏距离，又称欧几里得范数，通过计算两点坐标差值的平方和的开平方得到两点间的距离，但是我们的轨迹数据中给出的是经纬度坐标以及高程，因此我们需要从经纬度坐标中计算实际距离。

Haversine 公式常被用于处理经纬度坐标点，本实验使用该公式计算两点距离，公式如下：

$$a = \sin^2\left(\frac{\Delta \text{lat}}{2}\right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2\left(\frac{\Delta \text{lon}}{2}\right)$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

图 8. Haversine 公式

按照如上公式，编写获得实际距离的函数 haversine。这里地球半径取值 6371km，最后结果为 km 单位，转为 m 返回。同时定义并使用 Haversine 公式的 UDF 来计算相邻点之间的距离。

```
63  # 定义 Haversine 公式的 UDF
64  def haversine(lon1, lat1, lon2, lat2):
65      if None in (lon1, lat1, lon2, lat2):
66          return None
67      lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
68      dlon = lon2 - lon1
69      dlat = lat2 - lat1
70      a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
71      c = 2 * atan2(sqrt(a), sqrt(1-a))
72      r = 6371  # 地球平均半径，千米
73      return c * r * 1000  # 返回米
74
75  haversine_udf = udf(haversine, FloatType())
76
```

图 9. 定义 Harversin 公式的 UDF 代码截图

在计算过程中，我们涉及到了 sin、cos、atan2 等三角函数用于计算，pyspark.sql.functions 库中也有此函数，但与数据并不兼容，经尝试我们选用 math 库中的这些函数用于计算。

（2）时间间隔计算

由于我们先前建立了时间序列，只需调用 lag 函数获取上一条数据的时间戳并与当前数据作差运算即可得到时间间隔，记作 time_diff 成为新添加列。

```
data = data.withColumn("prev_time",
                       lag("date_numeric").over(windowSpec))

data = data.withColumn("time_diff",
                       (col("date_numeric") - col("prev_time")) * 24 * 3600)  # 转换为秒
```

图 10. 计算时间间隔的代码截图

同理，距离也通过在时间序列中调用 haversine_udf 函数计算得到距离，记作 distance 成为新添加列。

```
data = data.withColumn("prev_latitude",
                       lag("latitude").over(windowSpec))
data = data.withColumn("prev_longitude",
                       lag("longitude").over(windowSpec))
data = data.withColumn("distance",
                       haversine_udf(col("longitude"), col("latitude"), col("prev_longitude"), col("prev_latitude")))
```

图 11. 计算距离的代码截图

至此，我们就得到了时间序列中各数据的距离与时间间隔。

由于距离与时间间隔均为两条数据相对得到的数据，为方便后续运算，我们将距离与时间间隔定义为当前点相对上个点而言通过计算得到的数据，也就是说，第一个点的 distance 与 time_diff 值均为 null。

接下来我们通过简单的瞬时速率计算公式得到各个点的速度值，并记作 speed 成为新添加列。

```
data = data.withColumn("speed",
                       (col("distance") / col("time_diff")))  # 米/秒
```

图 12. 计算速率的代码截图

由此，我们成功完成了各数据点瞬时速度的计算。

3.2.5 停留点判断

遍历轨迹点，检查速率值，设置停留认定阈值，若小于阈值，则将上一个点作为停留开始点。继续遍历轨迹点，只要轨迹点的速率值仍小于阈值，则将该点作为停留点。

我们可以将该算法步骤简化为，当同时满足该点上个时刻的速度值与当前时刻的速度值均小于阈值时，认定该点为停留点，这样就避免了停留开始点的判定，简化了算法步骤。

```
# 标记停留点
data = data.withColumn("is_stop",
                       when((col("speed") < 0.5) & (lag("speed").over(windowSpec) < 0.5),
                            True).otherwise(False))  # 假定速度小于0.5米/秒为停留

```

图 13. 停留点判断的代码截图

如果该点满足条件，则将其数据的 bool 型值 is_stop 设为 true，反之设为 false，从而得到了 is_stop 停留点判断列。

3.2.6 加减速分析

由题目意思可以得知，运动状态的判断需要连续 N 个区间的速度单调变化，才可认为是加速或者减速。

为了判断方便，我们首先求解每个点的加速度（实际上是前一个点到当前点的平均加速度）。其公式为 a=（v2-v1）/（t2-t1），直接在时间序列中通过下一时刻点速度与当前点速度作差再除以下一时刻点的时间间隔得到加速度值，记作 acceleration 作为新添加列，代码如下：

```
# 计算加速度
data = data.withColumn("acceleration",
                       (lead("speed").over(windowSpec) - col("speed")) / lead("time_diff").over(windowSpec))

```

图 14. 计算加速度的代码截图

得到加速度之后，我们就可以在时间序列中进行加减速分析，根据算法要求，我们要检测至少三个点的速度，也就是分析连续两点的加速度即可（速度变化），因此我们可以将算法简化为连续两个点的加速度大于加速判定阈值认定为加速状态，小于减速判定阈值认定为减速状态，对前后两点的后点的加速减速状态值进行改动。

```
# 标记加速与减速
data = data.withColumn("is_accelerating",
                       when((col("acceleration") > 0.1) & (lag("acceleration").over(windowSpec) > 0.1),
                            True).otherwise(False))

data = data.withColumn("is_decelerating",
                       when((col("acceleration") < -0.1) & (lag("acceleration").over(windowSpec) < -0.1),
                            True).otherwise(False))

```

图 15. 加减速分析的代码截图

类似的，我们将满足条件的点数据的 bool 型值 is_ accelerating/ is_decelerating 设为 true，反之设为 false，从而得到了 is_ accelerating/ is_decelerating 加减速判断列。

4.实验成果

本次实验成果为一个 DataFrame。其格式为 row_num（行号也即点序）、date（日期）、time（时间）latitude（纬度）、longitude（经度）、speed（速度）、acceleration（加速度）、is_accelerating（加速状态）、is_decelerating（减速状态）、is_stop（停留点标志）。其前三十行部分数据如下：

row_num	date	time	latitude	longitude	speed	acceleration	is_accelerating	is_decelerating	is_stop
1	2009-04-15	13:44:00	26.162202	119.943787	0.0	3.722979540040798	false	false	false
2	2009-04-15	13:44:05	26.161528	119.943234	18.61490673250718	-2.068227268123859	false	false	false
3	2009-04-15	13:44:10	26.1619	119.943228	8.273765374171417	-1.0666832133399906	false	true	false
4	2009-04-15	13:44:15	26.161775	119.943276	2.940356107473505	-0.10729482871178968	false	true	false
5	2009-04-15	13:44:20	26.161667	119.943281	2.4038816361569615	0.1301658537706168	false	false	false
6	2009-04-15	13:44:25	26.16167	119.943128	3.0547112208048124	2.521308459821934	true	false	false
7	2009-04-15	13:44:30	26.161566	119.942352	15.661237446817559	-2.933421091139979	false	false	false
8	2009-04-15	13:44:35	26.161558	119.942401	0.9941230302846551	0.25539882188457097	false	false	false
9	2009-04-15	13:44:40	26.161482	119.942477	2.271117759329393	-0.34537514691089627	false	false	false
10	2009-04-15	13:44:45	26.161461	119.942491	0.5442411868618886	0.09912297752006315	false	false	false
11	2009-04-15	13:44:50	26.16143	119.94253	1.0398554425628352	-0.0920794684995707	false	false	false
12	2009-04-15	13:44:55	26.161413	119.942552	0.5794578187863262	-0.09403833460265562	false	false	false
13	2009-04-15	13:45:00	26.161411	119.942557	0.1092659176270858	0.002156777042684049	false	false	false
14	2009-04-15	13:45:05	26.161408	119.942562	0.12004978909126156	-0.00608007148647...	false	false	true
15	2009-04-15	13:45:10	26.161405	119.942565	0.08964941308585678	0.002520373929607...	false	false	true
16	2009-04-15	13:45:15	26.161404	119.94257	0.10225128884856227	-0.00387379881551...	false	false	true
17	2009-04-15	13:45:20	26.161403	119.942574	0.08288231946609557	-0.00165780415225...	false	false	true
18	2009-04-15	13:45:25	26.161401	119.942577	0.0745932946828104	-0.00894202578669...	false	false	true
19	2009-04-15	13:45:30	26.1614	119.942578	0.029883138433783	6.959163650646133...	false	false	true
20	2009-04-15	13:45:35	26.161399	119.942579	0.029883141913366513	-0.0015882759181411	false	false	true
21	2009-04-15	13:45:40	26.161398	119.942579	0.0222390137004236	0.001528824128020...	false	false	true
22	2009-04-15	13:45:45	26.161397	119.94258	0.0298831390106838	-0.00152883214925...	false	false	true
23	2009-04-15	13:45:50	26.161396	119.94258	0.022238974555338107	-4.55645154709821...	false	false	true
24	2009-04-15	13:45:55	26.161396	119.942581	0.019960751686482675	0.001984477241768231	false	false	true
25	2009-04-15	13:46:00	26.161395	119.942582	0.029883142709854753	-0.00597662489056...	false	false	true
26	2009-04-15	13:46:05	26.161395	119.942582	0.0	0.004447792738767348	false	false	true
27	2009-04-15	13:46:10	26.161394	119.942582	0.022238974484605833	7.843173553239072E-9	false	false	true
28	2009-04-15	13:46:15	26.161393	119.942582	0.0222390137004236	-0.00741300904744...	false	false	true
29	2009-04-15	13:46:18	26.161393	119.942582	0.0	0.03735376510160567	false	false	true
30	2009-04-15	13:46:20	26.161392	119.942583	0.07470771204220264	-1.26996921321315...	false	false	true

only showing top 30 rows

图 16. DataFrame 结果图

通过分析这个部分 DataFrame，我们可以将这段行车轨迹分为两段，第一段是行驶阶段，从 13:44:00 到 13:45:00，第二段是停车状态，从 13:45:00 到 13:46:20。

在第一段行驶阶段中，车辆经历了先减速在加速的状态，我们可以分析 speed 值，由 18 到 8 到 2 再到 3 到 15（单位为 m/s），而这些点后面所对应的加减速状态也符合我们的要求。

5.问题及解决

第一个问题就是在 3.2.1 数据读取与预处理中提到的浮点数精度问题，通过将 FloatType 更改为 DoubleType 即可成功读取数据。

第二个问题是增加点号作为新列以及将第一行和最后一行的速度值数据设

置为 0。

通过调用 row_number 函数和 count 函数记录每一条数据的行号与总数据行数，在计算 speed 的步骤通过 when().otherwise 设置速度值为 0 的判断条件，代码如下：

```
windowSpecRows = Window.rowsBetween(Window.unboundedPreceding, Window.unboundedFollowing)
data = data.withColumn("row_num", row_number().over(windowSpec))
data = data.withColumn("total_rows", count("date_numeric").over(windowSpecRows))
```

```
data = data.withColumn("speed",
                       when((col("row_num") == 1) | (col("row_num") == col("total_rows")) ,0)
                      .otherwise(col("distance") / col("time_diff")))
```

图 17. 设置首尾行速度值为 0 代码截图

第三个问题是 data.write.csv 运行失败，找了好多问题原因，最终确定是 Hadoop 的版本问题，我们实验中使用的是精简发行版，而要使用文件系统的相关功能，需要完整的分布式文件系统权限，也就是要使用 Hadoop 的完整功能，需要从官方重新下载，并配置好 HADOOP_HOME 与 HADOOP_CONF_DIR（精简版只有 bin 目录，没有 conf 目录，该目录下的 core-site.xml, hdfs-site.xml, yarn-site.xml 等配置文件对于 Hadoop 集群的正常运行是必不可少的）。

因此我并没有保存为 csv 文件，而是通过 show 函数输出在终端，并手动对其进行保存。

6.实习体会

通过本次实验，主要是复习巩固了 spark 的基础知识，将所学内容运用到实际当中。

对于本次实习的学习方式我是比较适应的，由浅入深，循序渐进，在一个一个的小练习中掌握基础知识，在最后的大实验中融会贯通。对于 spark 的 RDD 编程，我认为其更加离散化，虽然也主要是逐行操作，但是通过一些自定义的函数也可以实现数据的自由。而 spark 的 DatFrame 则更加结构化严谨化，其主要是行操作甚至窗口操作。至于 sparksql 则是一个实用额的数据增删改查的工具，将 spark 和 sqsl 语句结合从而提供更加多样化的功能。sparkML 主要是关于机器学习领域，其流水线思的操作步骤，大大简化了机器学习编程的难度。

本次实验主要是分了车辆轨迹的速度、停留点、运动状态等。事实上在事件

问题中我们会碰到各种细节，为了使编程模型更加贴近现实，我们必须处理好这些细节，这也有助于我们提高思维能力和编程习惯。尽管在实验中遇到很多困难，但是在老师的耐心帮助下和自己的细心探索之下，完美的完成了实验任务。

而此次大作业也让我回想起了今年的第十六届“华中杯”大学生数学建模竞赛，该题目数据也是行车轨迹，在该题目中分析的问题更加复杂，要使用行车轨迹估计交通信号灯周期，题目与论文见附录，核心内容是：

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。附件 1 中是 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，尝试求出这些路口相应方向的信号灯周期。

针对问题一，要求根据附件 1 中所有车辆轨迹图识别出该路口一个方向信号灯的周期。首先，对数据进行预处理和可视化，根据车辆速度求得车辆等待红灯的停车时长并剔除较小值。接着对车辆的启动时刻进行 DBSCAN 聚类得到每一批在红灯前停车等待的车辆 ID，找出每一类中等待时长的最大值合为一个序列。然后对这个序列进行筛选和中值滤波去噪处理后，取其中的最大值作为红灯的周期。依据相邻两批车流启动时刻的差值统计值，计算出信号灯的总周期以及绿灯周期。在此之上，考虑司机看到红灯后的减速过程，记作时间偏移量，引入到模型当中，增强了模型的鲁棒性和准确性。

这个数学建模问题与我们的大作业有许多相似之处，都要基于时间序列处理数据，都要分析每辆车的停止状态与加减速状态，通过计算停止时刻与启动时刻的时间差得到等待红灯时长，在对这些等待红灯时长进行聚类分析，误差处理，得到路口的红灯周期。

在如何判断其停止时刻时，我们考虑到车辆在驶入路口后由看到红灯到完全停下有一段减速的过程，为了使结果更加精确，需要加上这段减速过程对应的时间。我们将车辆减速的时间进行建模，用时间偏移量 Ta 表示，原理如下：

Step1：由附件数据分析知，车辆未减速时接近于匀速行驶，则需要找到速度突变的时刻。

Step2：经测试得知多数车辆减速的时间为 2-3s，为了提高测试结果的敏感

性，在车辆完全停止前设置了 7s 的时间长度。

Step3：求出 7s 内每一秒的加速度，其中加速度用速度的差分表示。

Step4：求出第一个加速度小于 7s 内所有加速度均值的时刻，这个时刻即为司机开始减速的时刻，该时刻与车辆停止时刻的差值即为减速时间，记为 Ta。

Step5：将减速时间 Ta 添加到中值滤波后的数据中，取序列中的最大值作为最终的红灯周期, 如下：

$$TNRed = max\{Ta + DU\_m\}$$

这个分析过程和我们大作业的内容非常相似，且难度更大，而正是引入了这个时间偏移量的想法，帮助我们在比赛中获得一等奖的成绩。

Spark 在大数据的处理上具有许多优势，比如：

1.速度：Spark 使用内存计算（in-memory computing），相比于 Hadoop MapReduce 的磁盘读写，它能更快地处理大规模数据。Spark 能提供高达 100 倍于 MapReduce 的速度。

2.易用性：Spark 支持多种编程语言，如 Scala、Python 和 Java，提供了丰富的 APIs，允许开发者使用这些语言轻松编写应用程序。此外，Spark 提供了简洁的 API 用于处理复杂的数据管道和数据聚合任务。

3.优化计算：Spark 的 Catalyst 优化器和 Tungsten 执行引擎为数据处理提供了深度优化，优化了执行计划并提高了内存和 CPU 的使用效率。

因此我在以后的学习过程中，遇到大数据处理或者深度学习问题，会选择尝试使用 Spark 进行数据挖掘或者复杂数据分析应用。本次实验让我初步认识了大数据处理框架，初步熟悉了大数据处理流程，为以后的学习工作奠定了良好的基础。最后感谢老师的教导，致此。

附录：

第十六届“华中杯”大学生数学建模挑战赛题目

B 题 使用行车轨迹估计交通信号灯周期问题

某电子地图服务商希望获取城市路网中所有交通信号灯的红绿周期，以便为司机提供更好的导航服务。由于许多信号灯未接入网络，无法直接从交通管理部门获取所有信号灯的数据，也不可能在所有路口安排人工读取信号灯周期信息。所以，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。请帮助该公司解决这一问题，完成以下任务。已知所有信号灯只有红、绿两种状态。

1. 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。附件 1 中是 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，尝试求出这些路口相应方向的信号灯周期，并按格式要求填入表 1。
2. 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。附件 2 中是另外 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，尝试求出这些路口相应方向的信号灯周期，按同样的格式要求填入表 2。
3. 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？附件 3 中是另外 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，尝试求出周期切换的时刻，以及新旧周期参数，按格式要求填入表 3，并指明识别出周期变化所需的时间和条件。

4. 附件 4 是某路口连续 2 小时内所有方向样本车辆的轨迹数据，请尝试识别出该路口信号灯的周期。

- 附件 1：路口 A1、A2、A3、A4、A5 各自一个方向连续 1 小时内车辆轨迹数据
- 附件 2：路口 B1、B2、B3、B4、B5 各自一个方向连续 1 小时内样本车辆轨迹数据
- 附件 3：路口 C1、C2、C3、C4、C5、C6 各自一个方向连续 2 小时内样本车辆轨迹数据
- 附件 4：路口 D 所有方向连续 2 小时内样本车辆轨迹数据
- 附件 5：数据文件说明及结果表格

1、轨迹数据文件格式。适用于附件 1-附件 4 所有轨迹数据文件。纯文本文件，第一行为标题行，各列以英文逗号分隔，共 5 列，分别为时间点、车辆 ID、当前位置 X 坐标、当前位置 Y 坐标。时间点单位为秒，第 0 秒开始，每 1 秒采样一次。坐标单位为米。车辆 ID 仅用于区分同一个文件中的不同车辆。车辆 ID 不一定是连续编号。不同文件中，相同 ID 的车辆没有任何联系。同一车道可能只允许一个方向前进，也可能允许两个方向前进，如直行或左转、直行或右转等。

2、表 1：路口 A1-A5 各自一个方向信号灯周期识别结果

路口	A1	A2	A3	A4	A5
红灯时长（秒）					
绿灯时长（秒）					

3、表 2：路口 B1-B5 各自一个方向信号灯周期识别结果

路口	B1	B2	B3	B4	B5
红灯时长（秒）					
绿灯时长（秒）					

4、表 3：路口 C1-C6 各自一个方向信号灯周期识别结果

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）						
周期 2 绿灯时长（秒）						
周期切换时刻						
周期 2 红灯时长（秒）						
周期 2 绿灯时长（秒）						
……						

说明：“周期切换时刻”是指信号灯周期发生变化的具体时间点，以第一个变化后的时长区间的起点计。如果信号灯周期没有变化，则“周期切换时刻”填写“无”。如果信号灯周期多次切换，按照上述格式，自行延长表格依次填写。