# Real-Time Network Assessment and Updating Using Vehicle-Locating Data

A Thesis
Presented to
The Academic Faculty

by

Zachary T. Roberts

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Civil and Environmental Engineering

Georgia Institute of Technology
December 2022

# Real-Time Network Assessment and Updating Using Vehicle-Locating Data

Approved by:

Dr. Iris Tien, Advisor
School of Civil & Environmental Engineering
*Georgia Institute of Technology*

Dr. John Taylor, Committee Member
School of Civil & Environmental Engineering
*Georgia Institute of Technology*

Dr. Michael Hunter, Committee Member
School of Civil & Environmental Engineering
*Georgia Institute of Technology*

Date Approved: December 11, 2022

## ACKNOWLEDGMENTS

Support for this project from the Georgia Department of Transportation (GDOT) through Award No. RP 20-01 is acknowledged.

Discussions with GDOT staff, including John Hibbard, Larry Barnes, and Emily Fish, regarding project scope is acknowledged.

Data provided by GDOT staff, including Teague Buchanan, Hong Liang, and their respective team members, is also acknowledged.

# TABLE OF CONTENTS

# LIST OF FIGURES

## SUMMARY

This project explores the ability to use vehicle-locating data to assess the state of the road network, including identifying road blockages along different segments of the transportation system. Compared to prior work using stationary data sources, such as loop detectors, traffic cameras, or traffic monitoring stations, or individual human-collected data collected either directly or through third-party sources, this project utilizes the mobile sources of Georgia Department of Transportation (GDOT) vehicles and their associated vehicle-tracking information to infer the state of the road network and perform transportation network assessment. These data are already currently being collected, demonstrating the utility of these data in performing road network assessment without the need to invest in new technologies, dedicate additional resources, or implement new instrumentation or infrastructure.

The raw dataset of vehicle-locating data is large and, in many cases, messy. In this project, we develop and implement multiple data trimming and processing methods using ArcGIS-specific Python algorithms to transform this initially large dataset into a usable format for network assessment. To utilize the vehicle-locating data in particular, we create a workflow to enable comparison of the vehicle routes with optimal routes to detect suboptimal routing decisions that may be indicative of blockages in the road network. This workflow includes the creation of vehicle route segments based on the individual vehicle-locating data points, the linking of segments into routes, the identification of optimal routes between these points, and the comparison of distances between the actual taken routes and the optimal routes to detect the degree of suboptimal routing and its association with the likelihood of the presence of a road blockage.

We use the resulting datasets as inputs and create machine learning models with multiple variables to detect the presence of a road blockage. We explore both regression-based and classification-based models, and find that the classification model performs particularly well for this task. In this project, through the use of multiple data processing and data analysis methods combined with machine learning approaches, we show how the vehicle-locating data can be used to perform network assessment and accurate detection of blockages in the road network.

# CHAPTER 1
# INTRODUCTION

Road infrastructure makes up a crucial component of Georgia's asset network. Throughout the state, connections link different areas to each other, providing access to employment, social, and health services, thereby supporting state activities and stimulating economic development. These services are interrupted, however, by the presence of road blockages, including those due to vehicular accidents, debris, and flooding, among other factors, which limit and can prohibit travel along certain routes. Providing real-time information on the state of the transportation network is a way for state agencies to understand the state of the network at any point in time, deploy resources as needed to resolve any road blockages, and prioritize specific areas of the road network for recovery.

An increasing number of data sources are available to potentially provide such information on the state of the road network. However, these often require significant resources to implement, including to install certain infrastructure or hardware to collect data, or in changing specific practices by the public or individual workers to ensure reliable data collection. These challenges potentially limit the utility of these data sources for road network assessment, both by the amount of data that can be collected, and in how accurate or reliable these data turn out to be.

In this project, rather than using these types of data sources (such as data collected from fixed infrastructure installations, or individual human-collected data) with their accompanying challenges and limitations, we (research team) use data that are already currently being collected by the Georgia Department of Transportation (GDOT). Specifically, we use data that are already implemented through hardware on GDOT vehicles that track GDOT vehicle locations as they travel over the network to infer the state of the road network and perform continuous network assessment and updating. The idea is that as GDOT vehicles travel over the network, they are continuously collecting data on the state of the network in the road segments they are traveling over. For example, if a GDOT vehicle travels over a certain route, it can be inferred from the vehicle-locating information that the particular route that the vehicle traveled over is unblocked, and open for passage. In contrast, if a vehicle makes an unexpected detour around a certain part of the network, there is some likelihood that the vehicle was avoiding a blocked part of the network, indicating a potential road blockage in the area avoided.

Thus, the GDOT vehicles provide valuable information on the real-time network state. The benefit of using these vehicle-locating data for the network assessment is that these data are already being collected by GDOT assets, so no additional investment in assets or infrastructure needs to be implemented to perform the network assessment. In addition, the GDOT equipment that collects the vehicle-locating data is implemented passively rather than actively, meaning that it will collect these data without the need for operators to turn on certain instruments or capabilities. The result is that there is less risk that something will occur to disrupt data collection and that there is increased reliability that the data will be continuously collected. The GDOT vehicle-locating data are also being collected continuously, enabling the network assessments that are made based on the data to be continuously updated as new information is recorded and received about the locations and routes of GDOT vehicles across the network. Finally, because the data are being collected by GDOT rather than by a third party or by the public, and they are being used for GDOT purposes, there are no issues regarding data security or privacy in order to collect or use the data. Also, GDOT has control over how the data are collected moving forward, rather than relying on potential changing data collection strategies, rules, and regulations from third-party owners.

The objective of this project is to create a system and investigate the feasibility of such a system that is able to utilize currently collected GDOT vehicle-locating data to provide real-time assessment and updating of the state of the transportation network assessment. Doing so will provide GDOT with important information to support increased situational awareness of the state of the network, as well as support resource allocation, hazard mitigation, and network recovery operations to resolve any road blockages across the transportation network.

To accomplish this, in this project, we utilize data sources provided by GDOT as inputs into the system. The main data inputs are vehicle routing information and traffic incident data representing road blockage information. Next, we perform a series of preprocessing, data-modification, and data-processing operations in order to make the data usable and consistent for the full data-processing system. It is noted that the datasets investigated are large, and require several transformations to enable operational viability and provision of use as geographic information system (GIS) intelligence. A workflow has been developed to efficiently create and utilize the vehicle-locating points (VLPs). This includes the processing of the large vehicle-locating datasets using data trimming and buffering methods, as well as the identification and connection of specific vehicle-locating data points into individual vehicle route segments. These operations refine and process the datasets.

Next, the goal is to create a model that is able to use the vehicle-locating data as inputs to detect road blockages in the transportation network. We utilize machine learning methods, which involve building of the models and both training and testing of the datasets with the models. The training step—training a dataset using machine learning models—enables us to understand how traffic conditions and vehicle-routing information interact with each other to be able to infer the presence of a road blockage based on the vehicle-locating information. The goal is then to use the trained dataset to apply to a real-time detection system with the presence of processing capabilities. The specific machine learning methods investigated include ordinary least squares (OLS) linear regression and decision tree classification, both of which are explored to learn the trends of traffic across the network based on the vehicle-locating information to accurately predict the likelihoods of road blockages. The resulting model provides intelligence and learning about how two large datasets, containing VLPs and georeferenced traffic incident data, interact with one another over the time scope of the study.

The results of this study demonstrate the novelty and utility of a mobile detection system across a broad network utilizing currently collected GDOT vehicle-locating data to provide information about the state of the transportation network as it changes over time.

# CHAPTER 2

# LITERATURE REVIEW

Previous research includes work in the area of using new technologies to facilitate evacuation decisions after a disaster (Iliopoulou et al. 2020); however, this project focuses on transportation network assessment rather than evacuation routing. While many previous studies focus on traffic estimation and prediction (e.g., Mena-Yedra et al. 2018), this project focuses on real-time network assessments with outcomes facilitating resource allocation and network recovery through identification and detection of road blockages. In terms of specific technologies, previous research often utilizes fixed data-collection sources, such as loop detectors and traffic monitoring stations providing traffic count information (Singh et al. 2018). Compared to that work on utilizing stationary data sources (i.e., loop detectors, traffic cameras, traffic monitoring stations) for transportation network analysis, this study focuses on the mobile sources of GDOT vehicles and their associated vehicle-tracking information, which is wider-reaching with lower operational costs, and the other benefits previously described.

Recently, movement has been toward the use of increased mobile data sources (e.g., Meng et al. 2017). However, that work focuses on traffic flow modeling rather than actual network assessment, which is the focus of this project. Finally, regarding the use of mobile data for post-disaster network assessment, much of the recent work uses crowdsourced information for infrastructure assessment (Basu et al. 2016, Astarita et al. 2020). Compared to crowdsourced data, the mobile vehicle-locating data utilized in this project represents a more trustworthy, detailed, and accurate geolocated data source for transportation network assessment.

This effort contributes to the previous studies that have explored utilizing vehicle-locating data to perform real-time transportation network assessment. The anticipated benefit is that data that are currently collected by agencies, such as GDOT, can be used and leveraged for use in transportation network assessment and updating as vehicles, routes, and network conditions change.

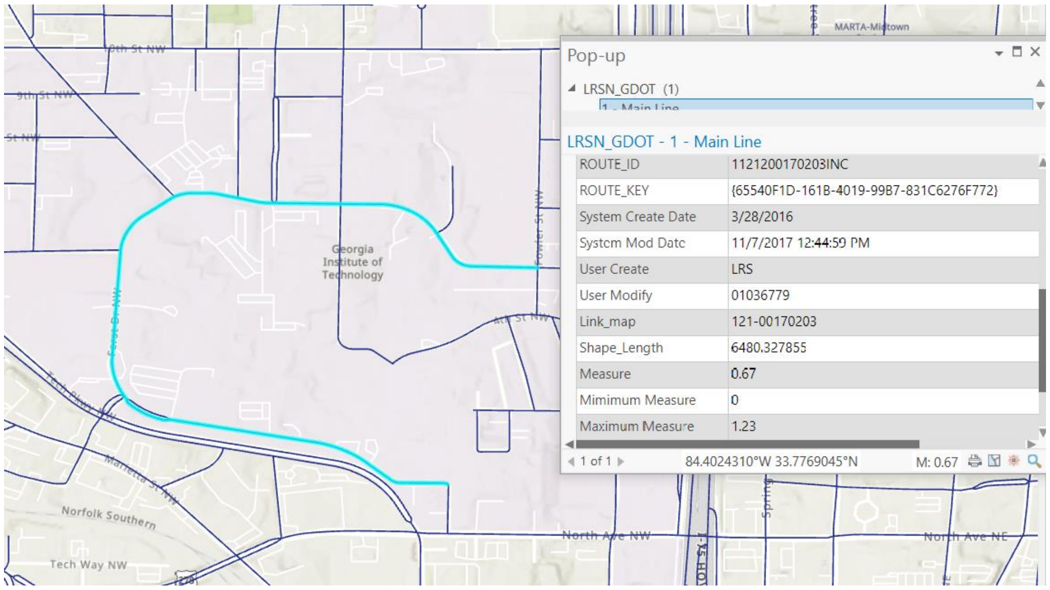
# CHAPTER 3

# METHODOLOGY

## 3.1. Data Input Sources

### Georgia Road Network Shapefile

To assess the geospatial relations of all the utilized datasets related to the transportation network, it is necessary to have a base file of the road network. The first data input is the shapefile of the State of Georgia's road network. This shapefile consists of a series of interconnected polylines representing the midpoint of Georgia roads. The shapefile includes the identification code, geolocation, and width of 205,351 road segments.



LRSN_GDOT - 1 - Main Line

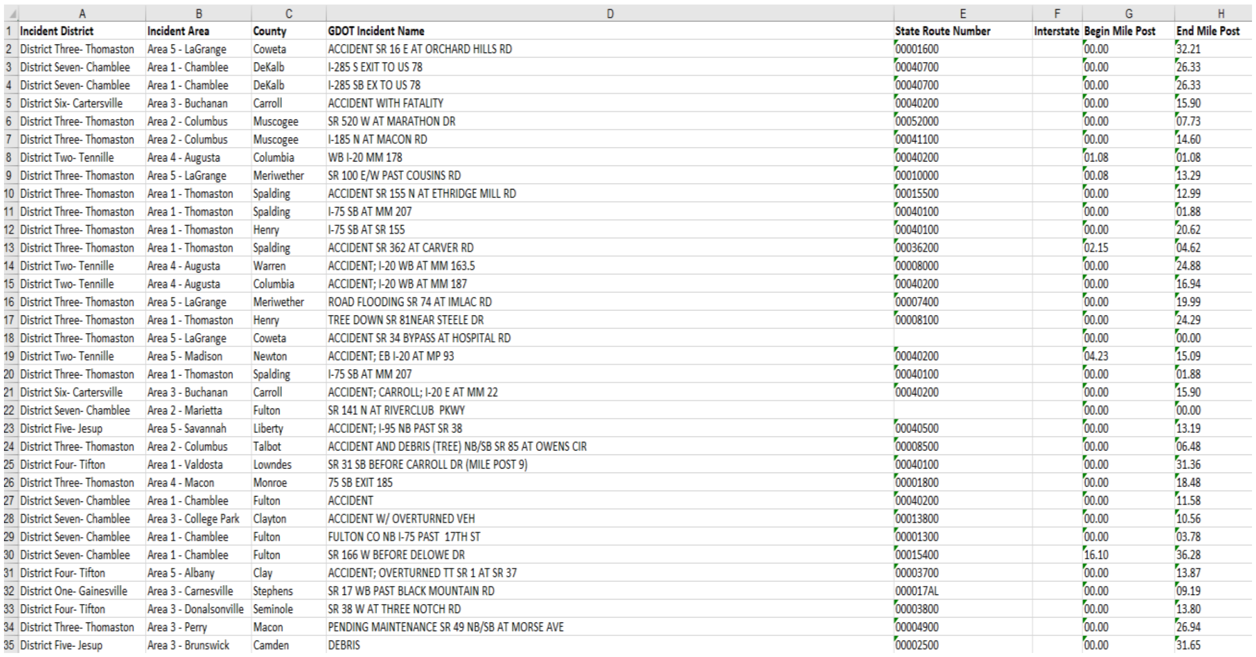
| | |
|---|---|
| ROUTE_ID | 1121200170203INC |
| ROUTE_KEY | {65540F1D-161B-4019-99B7-831C6276F772} |
| System Create Date | 3/28/2016 |
| System Mod Date | 11/7/2017 12:44:59 PM |
| User Create | LRS |
| User Modify | 01036779 |
| Link_map | 121-00170203 |
| Shape_Length | 6480.327855 |
| Measure | 0.67 |
| Mimimum Measure | 0 |
| Maximum Measure | 1.23 |

84.4024310°W 33.7769045°N

**Figure 1.** Map. The LRSN_GDOT feature is representative of the provided Georgia Road Network Shapefile. Shown in cyan is Ferst Drive on the Georgia Tech campus in Atlanta, GA. Selected feature displaying information on the polyline shape length, county, road identification code, and direction (increasing/decreasing).

## WebEOC Executive Report

To match the vehicle-locating data with identified incidents on the road network leading to potential road blockages, it is necessary to know where and when the road incidents occurred. This information is attained through the WebEOC Executive Report, which is exported as a spreadsheet. The report includes state route location, incident description, direction, and the number of lanes passable. For this study, there are approximately 4,000 incidents reported within the Fulton County boundaries, spanning between January 2016 and September 2021.



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | **Incident District** | **Incident Area** | **County** | **GDOT Incident Name** | **State Route Number** | **Interstate** | **Begin Mile Post** | **End Mile Post** |
| 2 | District Three- Thomaston | Area 5 - LaGrange | Coweta | ACCIDENT SR 16 E AT ORCHARD HILLS RD | 00001600 | | 00.00 | 32.21 |
| 3 | District Seven- Chamblee | Area 1 - Chamblee | DeKalb | I-285 S EXIT TO US 78 | 00040700 | | 00.00 | 26.33 |
| 4 | District Seven- Chamblee | Area 1 - Chamblee | DeKalb | I-285 SB EX TO US 78 | 00040700 | | 00.00 | 26.33 |
| 5 | District Six- Cartersville | Area 3 - Buchanan | Carroll | ACCIDENT WITH FATALITY | 00040200 | | 00.00 | 15.90 |
| 6 | District Three- Thomaston | Area 2 - Columbus | Muscogee | SR 520 W AT MARATHON DR | 00052000 | | 00.00 | 07.73 |
| 7 | District Three- Thomaston | Area 2 - Columbus | Muscogee | I-185 N AT MACON RD | 00041100 | | 00.00 | 14.60 |
| 8 | District Two- Tennille | Area 4 - Augusta | Columbia | WB I-20 MM 178 | 00040200 | | 01.08 | 01.08 |
| 9 | District Three- Thomaston | Area 5 - LaGrange | Meriwether | SR 100 E/W PAST COUSINS RD | 00010000 | | 00.08 | 13.29 |
| 10 | District Three- Thomaston | Area 1 - Thomaston | Spalding | ACCIDENT SR 155 N AT ETHRIDGE MILL RD | 00015500 | | 00.00 | 12.99 |
| 11 | District Three- Thomaston | Area 1 - Thomaston | Spalding | I-75 SB AT MM 207 | 00040100 | | 00.00 | 01.88 |
| 12 | District Three- Thomaston | Area 1 - Thomaston | Henry | I-75 SB AT SR 155 | 00040100 | | 00.00 | 20.62 |
| 13 | District Three- Thomaston | Area 1 - Thomaston | Spalding | ACCIDENT SR 362 AT CARVER RD | 00036200 | | 02.15 | 04.62 |
| 14 | District Two- Tennille | Area 4 - Augusta | Warren | ACCIDENT; I-20 WB AT MM 163.5 | 00008000 | | 00.00 | 24.88 |
| 15 | District Two- Tennille | Area 4 - Augusta | Columbia | ACCIDENT; I-20 WB AT MM 187 | 00040200 | | 00.00 | 16.94 |
| 16 | District Three- Thomaston | Area 5 - LaGrange | Meriwether | ROAD FLOODING SR 74 AT IMLAC RD | 00007400 | | 00.00 | 19.99 |
| 17 | District Three- Thomaston | Area 1 - Thomaston | Henry | TREE DOWN SR 81NEAR STEELE DR | 00008100 | | 00.00 | 24.29 |
| 18 | District Three- Thomaston | Area 5 - LaGrange | Coweta | ACCIDENT SR 34 BYPASS AT HOSPITAL RD | | | 00.00 | 00.00 |
| 19 | District Two- Tennille | Area 5 - Madison | Newton | ACCIDENT; EB I-20 AT MP 93 | 00040200 | | 04.23 | 15.09 |
| 20 | District Three- Thomaston | Area 1 - Thomaston | Spalding | I-75 SB AT MM 207 | 00040100 | | 00.00 | 01.88 |
| 21 | District Six- Cartersville | Area 3 - Buchanan | Carroll | ACCIDENT; CARROLL; I-20 E AT MM 22 | 00040200 | | 00.00 | 15.90 |
| 22 | District Seven- Chamblee | Area 2 - Marietta | Fulton | SR 141 N AT RIVERCLUB PKWY | | | 00.00 | 00.00 |
| 23 | District Five- Jesup | Area 5 - Savannah | Liberty | ACCIDENT; I-95 NB PAST SR 38 | 00040500 | | 00.00 | 13.19 |
| 24 | District Three- Thomaston | Area 2 - Columbus | Talbot | ACCIDENT AND DEBRIS (TREE) NB/SB SR 85 AT OWENS CIR | 00008500 | | 00.00 | 06.48 |
| 25 | District Four- Tifton | Area 1 - Valdosta | Lowndes | SR 31 SB BEFORE CARROLL DR (MILE POST 9) | 00040100 | | 00.00 | 31.36 |
| 26 | District Three- Thomaston | Area 4 - Macon | Monroe | 75 SB EXIT 185 | 00001800 | | 00.00 | 18.48 |
| 27 | District Seven- Chamblee | Area 1 - Chamblee | Fulton | ACCIDENT | 00040200 | | 00.00 | 11.58 |
| 28 | District Seven- Chamblee | Area 3 - College Park | Clayton | ACCIDENT W/ OVERTURNED VEH | 00013800 | | 00.00 | 10.56 |
| 29 | District Seven- Chamblee | Area 1 - Chamblee | Fulton | FULTON CO NB I-75 PAST 17TH ST | 00001300 | | 00.00 | 03.78 |
| 30 | District Seven- Chamblee | Area 1 - Chamblee | Fulton | SR 166 W BEFORE DELOWE DR | 00015400 | | 16.10 | 36.28 |
| 31 | District Four- Tifton | Area 5 - Albany | Clay | ACCIDENT; OVERTURNED TT SR 1 AT SR 37 | 00003700 | | 00.00 | 13.87 |
| 32 | District One- Gainesville | Area 3 - Carnesville | Stephens | SR 17 WB PAST BLACK MOUNTAIN RD | 000017AL | | 00.00 | 09.19 |
| 33 | District Four- Tifton | Area 3 - Donalsonville | Seminole | SR 38 W AT THREE NOTCH RD | 00003800 | | 00.00 | 13.80 |
| 34 | District Three- Thomaston | Area 3 - Perry | Macon | PENDING MAINTENANCE SR 49 NB/SB AT MORSE AVE | 00004900 | | 00.00 | 26.94 |
| 35 | District Five- Jesup | Area 3 - Brunswick | Camden | DEBRIS | 00002500 | | 00.00 | 31.65 |

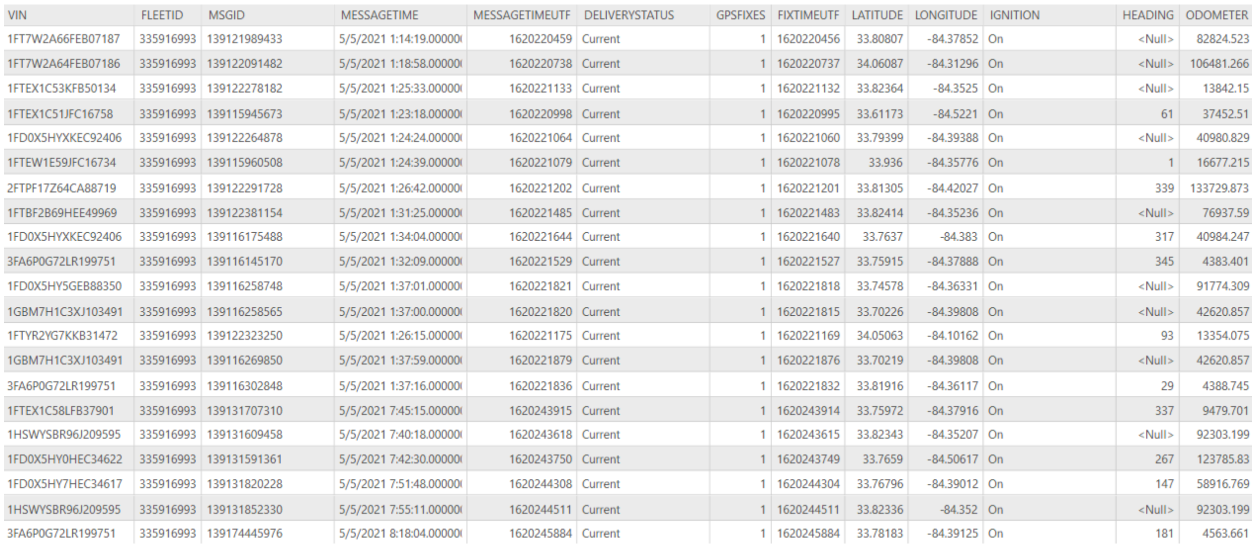
**Figure 2.** WebEOC Spreadsheet Data. Important features include Incident Type, Time of Occurrence, and Geolocation (latitude/longitude).

## Verizon Network Fleet Geodatabase

The vehicle-locating data utilized in the project as tracking information for the GDOT-owned vehicles as they travel over the road network is from the Verizon Network Fleet system installed and operational on the vehicles. These data are output as a Verizon Network Fleet Attribute Table displaying vehicle locating point identifying information. An example of the converted Excel spreadsheet of the GDOT vehicle-tracking information from the Verizon Network Fleet geodatabase, located in the appendices, includes vehicle ID, location, time, and ignition status of the vehicle (On/Off). The location and time information are used to create the vehicle tracks over the network. The "Ignition" column provides the information about the state of the vehicle being turned on or off. These data are used to cut the large dataset into individual vehicle route segments, indicating when to cease a vehicle route segment in the created function *Valid.py*. Vehicles are grouped and identified using the data in the "VIN" column, representing the vehicle identification number. This enables us to identify and locate individual vehicles over the network. Vehicles are tracked with a frequency of 2 minutes until the ignition of the car is turned off.

Vehicle-locating points are restricted to Fulton County and subdivided into 19 separate ArcGIS feature classes. The Verizon Network Fleet data are subdivided to improve the processing time of our user developed *Valid.py* function, which creates the vehicle route segments. For the study, approximately 44,000 vehicle locating points were randomly selected across the 19 ArcGIS feature classes. These points, spanning between April and May of 2021, were then merged for processing.

As the Verizon Network Fleet data span across Georgia, our data need to be extracted from Fulton County, which is chosen for its centrality of vehicle traffic in the state. To initiate this process, a feature class is used named *Counties.gdb* containing the shapes of all 159 counties in the state of Georgia. To extract the Fulton County shape, the attribute is selected in the Feature Class Attribute Table, and scrolling over to the layers in the ArcGIS project, we create a layer via the *"Make Layer From Selected Features"* function. The Selected Feature Class is then named "Fulton County". The ArcGIS function *Clip* is then used to create new feature classes containing only the vehicle-locating points from Fulton County. The input is the Vehicle Locating Point Feature Class and Fulton County selection feature class, with the output being of the name VLP_FC[number of data subdivision]. Here, VLP stands for vehicle-locating points, and FC stands for Fulton County.



| VIN | FLEETID | MSGID | MESSAGETIME | MESSAGETIMEUTF | DELIVERYSTATUS | GPSFIXES | FIXTIMEUTF | LATITUDE | LONGITUDE | IGNITION | HEADING | ODOMETER |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1FT7W2A66FEB07187 | 335916993 | 139121989433 | 5/5/2021 1:14:19.00000 | 1620220459 | Current | 1 | 1620220456 | 33.80807 | -84.37852 | On | <Null> | 82824.523 |
| 1FT7W2A64FEB07186 | 335916993 | 139122091482 | 5/5/2021 1:18:58.00000 | 1620220738 | Current | 1 | 1620220737 | 34.06087 | -84.31296 | On | <Null> | 106481.266 |
| 1FTEX1C53KFB50134 | 335916993 | 139122278182 | 5/5/2021 1:25:33.00000 | 1620221133 | Current | 1 | 1620221132 | 33.82364 | -84.3525 | On | <Null> | 13842.15 |
| 1FTEX1C51JFC16758 | 335916993 | 139115945673 | 5/5/2021 1:23:18.00000 | 1620220998 | Current | 1 | 1620220995 | 33.61173 | -84.5221 | On | 61 | 37452.51 |
| 1FD0X5HYXKEC92406 | 335916993 | 139122264878 | 5/5/2021 1:24:24.00000 | 1620221064 | Current | 1 | 1620221060 | 33.79399 | -84.39388 | On | <Null> | 40980.829 |
| 1FTEW1E59JFC16734 | 335916993 | 139115960508 | 5/5/2021 1:24:39.00000 | 1620221079 | Current | 1 | 1620221078 | 33.936 | -84.35776 | On | 1 | 16677.215 |
| 2FTPF17Z64CA88719 | 335916993 | 139122291728 | 5/5/2021 1:26:42.00000 | 1620221202 | Current | 1 | 1620221201 | 33.81305 | -84.42027 | On | 339 | 133729.873 |
| 1FTBF2B69HEE49969 | 335916993 | 139122381154 | 5/5/2021 1:31:25.00000 | 1620221485 | Current | 1 | 1620221483 | 33.82414 | -84.35236 | On | <Null> | 76937.59 |
| 1FD0X5HYXKEC92406 | 335916993 | 139116175488 | 5/5/2021 1:34:04.00000 | 1620221644 | Current | 1 | 1620221640 | 33.7637 | -84.383 | On | 317 | 40984.247 |
| 3FA6P0G72LR199751 | 335916993 | 139116145170 | 5/5/2021 1:32:09.00000 | 1620221529 | Current | 1 | 1620221527 | 33.75915 | -84.37888 | On | 345 | 4383.401 |
| 1FD0X5HY5GEB88350 | 335916993 | 139116258748 | 5/5/2021 1:37:01.00000 | 1620221821 | Current | 1 | 1620221818 | 33.74578 | -84.36331 | On | <Null> | 91774.309 |
| 1GBM7H1C3XJ103491 | 335916993 | 139116258565 | 5/5/2021 1:37:00.00000 | 1620221820 | Current | 1 | 1620221815 | 33.70226 | -84.39808 | On | <Null> | 42620.857 |
| 1FTYR2YG7KKB31472 | 335916993 | 139122323250 | 5/5/2021 1:26:15.00000 | 1620221175 | Current | 1 | 1620221169 | 34.05063 | -84.10162 | On | 93 | 13354.075 |
| 1GBM7H1C3XJ103491 | 335916993 | 139116269850 | 5/5/2021 1:37:59.00000 | 1620221879 | Current | 1 | 1620221876 | 33.70219 | -84.39808 | On | <Null> | 42620.857 |
| 3FA6P0G72LR199751 | 335916993 | 139116302848 | 5/5/2021 1:37:16.00000 | 1620221836 | Current | 1 | 1620221832 | 33.81916 | -84.36117 | On | 29 | 4388.745 |
| 1FTEX1C58LFB37901 | 335916993 | 139131707310 | 5/5/2021 7:45:15.00000 | 1620243915 | Current | 1 | 1620243914 | 33.75972 | -84.37916 | On | 337 | 9479.701 |
| 1HSWYSBR96J209595 | 335916993 | 139131609458 | 5/5/2021 7:40:18.00000 | 1620243618 | Current | 1 | 1620243615 | 33.82343 | -84.35207 | On | <Null> | 92303.199 |
| 1FD0X5HY0HEC34622 | 335916993 | 139131591361 | 5/5/2021 7:42:30.00000 | 1620243750 | Current | 1 | 1620243749 | 33.7659 | -84.50617 | On | 267 | 123785.83 |
| 1FD0X5HY7HEC34617 | 335916993 | 139131820228 | 5/5/2021 7:51:48.00000 | 1620244308 | Current | 1 | 1620244304 | 33.76796 | -84.39012 | On | 147 | 58916.769 |
| 1HSWYSBR96J209595 | 335916993 | 139131852330 | 5/5/2021 7:55:11.00000 | 1620244511 | Current | 1 | 1620244511 | 33.82336 | -84.352 | On | <Null> | 92303.199 |
| 3FA6P0G72LR199751 | 335916993 | 139174445976 | 5/5/2021 8:18:04.00000 | 1620245884 | Current | 1 | 1620245884 | 33.78183 | -84.39125 | On | 181 | 4563.661 |

**Figure 3.** Spreadsheet data. Verizon Network Fleet Attribute Table displaying vehicle locating point identifying information. The "Ignition" column shows the state of the vehicle being turned on or off. This will indicate when to cease a vehicle route segment for *Valid.py*. Vehicles are grouped and identified using the data in the "VIN" column, representing the vehicle identification number.

## 3.2. Data Pre-Processing

With the set of data inputs, certain preprocessing and data-modification operations need to be conducted to properly prepare the data for processing. The purpose in this preprocessing stage is to prepare the workspace within ArcGIS for the later route segmentation and analysis stages.

The first step in the data preprocessing is to convert the Verizon Network Fleet Excel files into ArcGIS geopoints. This is done through the function *XY Table To Point*, where the X field specifies longitude, and the Y field specifies latitude. This latitude and longitude information is included in the Network Fleet .csv table. All other columned information is transferred into and associated with each VLP. The output of this function is a feature class of georeferenced points corresponding with the vehicle-locating data points.

Next, a network analysis layer that identifies the Georgia road network must also be set up. This is completed through the creation of a "New Network Database" (ND), which inputs the LRSN_GDOT (i.e., the Georgia Road Network Shapefile). This input is composed of all the center points of the Georgia road network strung together as separate polylines.

Additionally, in working with the datasets, given the large size of the datasets, the data points are trimmed to reduce the processing time and remove any redundant information in the datasets. Looking closely at the data, the data in this preprocessing time are trimmed based on the location of the points relative to the locations of the road segments. Here, we trim the data based on proximity to the road network using the ArcGIS *Buffer* analysis function. Vehicle-locating data points sufficiently far from the road network indicate that the vehicle is not actually on the road or traveling along a road segment.

A buffer of 30 ft (which covers large highways) is chosen from the center of the road network layer to cover all data points within the road network. Parking lot areas and driveways are examples of data points that occur outside of the buffer and are trimmed and not included in the analysis. These cases can be neglected in our objective for vehicle routing and vehicle tracking along road segments. Once points are converted into vehicle routes, additional trimming will be conducted. The redundant data points are removed as they do not represent information about vehicles traveling on the road network, and therefore, are not of use in the vehicle route-tracking analysis process to detect blockages along the road network for this project.

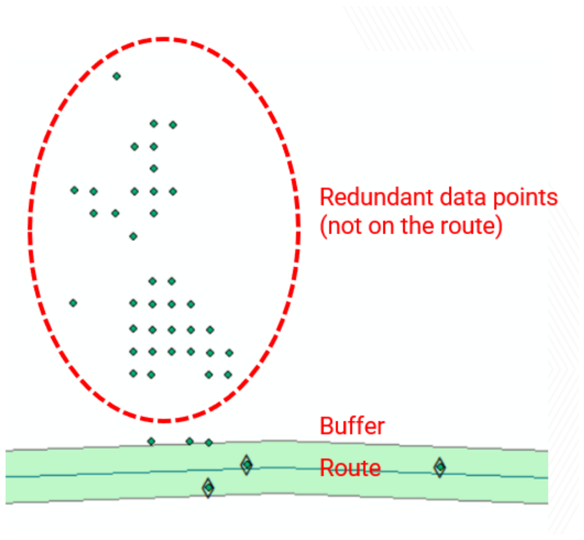


**Figure 4.** Diagram. Demonstration of vehicle-locating points removed by the ArcGIS *Buffer* function in the data-trimming step. Redundant data points (outlined in red) most likely represent stationary vehicles due to their proximity to each other and distance from the buffered route segment (shown in green).

## 3.3. Data Processing

With the data preprocessed, this chapter describes the many functions developed as part of this project to process the data in the created data analysis and processing pipeline. These functions are written in Python to facilitate the interoperability of datasets and use with ArcGIS for the geolocated data. There are two main processing steps, each with an associated Python function written. The first is to obtain the desired vehicle routing and incident segments such that they can be overlaid for analysis; this function is called *GetSegments.py*. The second is to ensure that the vehicle-locating data points are valid and to connect consecutive valid points as nodes to create individual vehicle routing segments; this function is called *Valid.py*.

Our first step of processing involves coding a function to retrieve the vehicle routing and WebEOC incident segments for further analysis. The first function, ***GetSegments.py***, is enabled by the ArcPy function *Segment Along Line* and utilizes ArcPy. *GetSegments* inputs the WebEOC dataset and the Road Network layer to output incident segments. Therefore, the WebEOC points with length of segment blockage (Begin mile of segment to End mile of segment) are converted to line segments.

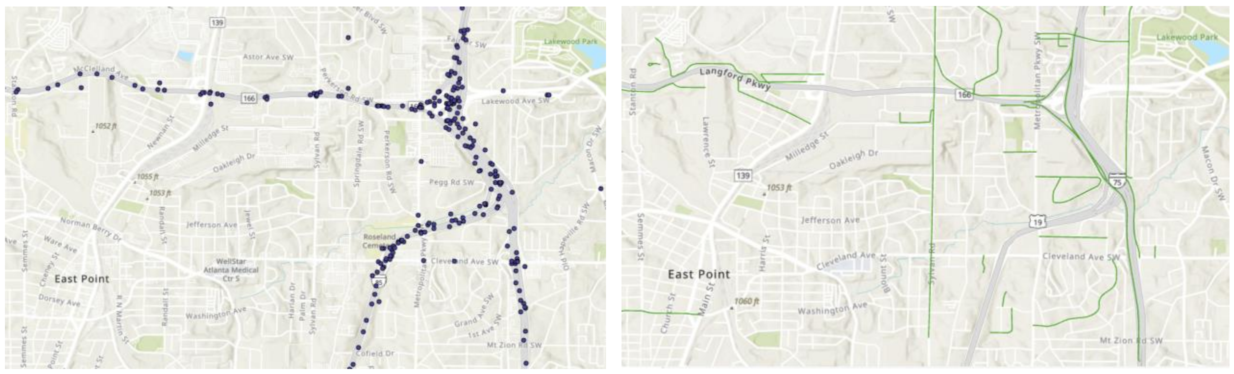


**Figure 5.** Screenshot. *GetSegments.py* output.

The second function, ***Valid.py***, utilizes the ArcGIS *Network Analyst* feature and ArcPy to find connected vehicle-locating points by similar Vehicle ID (VIN) and FixTime to create vehicle segments across the Georgia road network. The function ensures that the VLPs used in the analysis are all valid points as part of individual GDOT vehicle routes. *Valid.py* also utilizes a function file named *mxFindRoutes* that finds the VLPs to be connected and linked together through the *Valid.py* function. The VLPs are sequential points for a single vehicle as it travels over the network. Each point is separated by a 2-minute time interval. As such, a certain number of points in sequence are of interest to construct the full detailed routes of the vehicles.

The objective here is to use the individual VLPs to construct a continuous route of the vehicle as it travels over the network. In doing so, there is a tradeoff between the number of consecutive points used (to construct a continuous route), and the computational cost of storing all the points in an increasingly large dataset for processing. Therefore, from an investigation of the data, and the typical distance covered between points, up to six points are connected at a time to create a vehicle route segment. *mxFindRoutes* links these points, up to six in number, to input into *Solve* and create a vehicle segment of as many nodes.

Given the breadth and number of vehicle tracks that are collected as part of the datasets, *Valid.py* unintentionally creates certain extraneous segments, including multiple route segments on the same path, some routes contained within others, and segments with no length. These attributes are often found when working with field-collected datasets and need to be addressed to ensure the resulting data points used for analysis are all representative of the data that are desired to be collected, and are accurate and reliable in reflecting vehicle routing actions in the field.

To detect and filter out the data, and in particular remove the extraneous routing segments, a custom program is developed in MATLAB to identify such segments (specifically those where multiple route segments are identified on the same path and the segments that are of zero length). Once identified, the rows containing extraneous segments are selected with the *Select By Attributes* tool and deleted to eliminate their effect on the subsequent machine learning model developed.

Finally, given the density and amount of data collected as part of the vehicle-locating equipment, significant computational times are required to process the datasets. An initial processing and analysis were conducted through the *Valid.py* function.

Therefore, to reduce computational times, we created an additional step in the data processing and analysis, which is to iterate by vehicle numbers to match the correct vehicle segments and complete the code. A list of VINs are compiled, and the input in Valid Track Points are single-vehicle selections (using ArcGIS *Make Layer From Selected Features*), repeated for each vehicle. Because the code is iterative through each VIN, a queue of *Valid.py* codes is made for each vehicle, which significantly improves the data processing times. To iterate through the necessary data points for processing, the computer was left on over night and took multiple days to run.

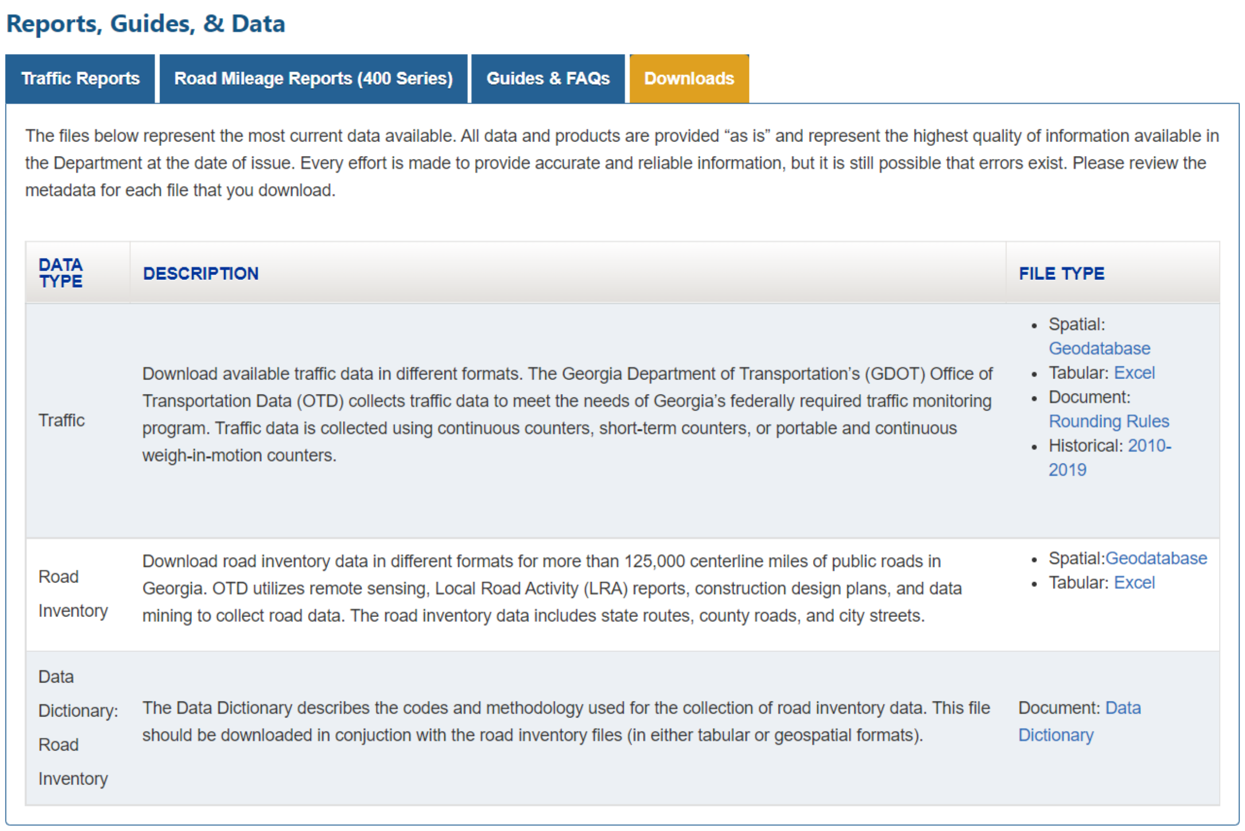
# CHAPTER 4

# RESULTS

## 4.1. Model Variables

### Average Annual Daily Traffic (Ind. Variable)

The first independent variable included is the average annual daily traffic (AADT) on specific road segments. The amount of traffic on a given road segment will affect the likelihood of a potential incident on that segment. GDOT provided road and traffic data that are publicly available to be used in this project, such as a shapefile of traffic counts along the Georgia road network (GDOT 2021). Included in these data are geolocated AADT for given road segments. The most recent numbers from 2019 are used, as 2021–2022 data have yet to be published. Under the Traffic Data Type, the Spatial Geodatabase is used for this project. Attribute traffic information is later linked based on the Feature ID (FID) closest to a vehicle route segment (representing the road the vehicle is on) using the ArcGIS *Merge* function.



Reports, Guides, & Data

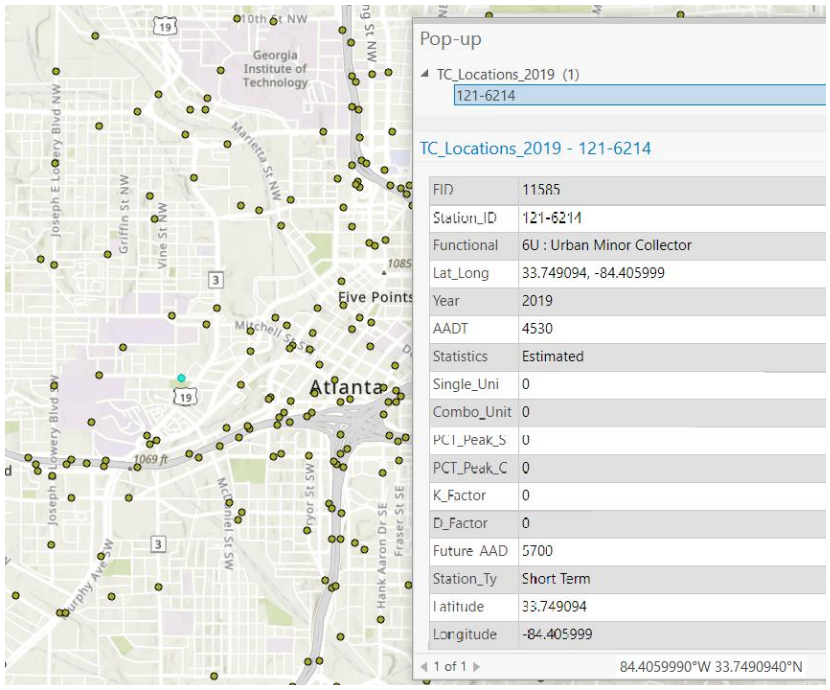
Traffic Reports | Road Mileage Reports (400 Series) | Guides & FAQs | Downloads

The files below represent the most current data available. All data and products are provided "as is" and represent the highest quality of information available in the Department at the date of issue. Every effort is made to provide accurate and reliable information, but it is still possible that errors exist. Please review the metadata for each file that you download.

| DATA TYPE | DESCRIPTION | FILE TYPE |
|---|---|---|
| Traffic | Download available traffic data in different formats. The Georgia Department of Transportation's (GDOT) Office of Transportation Data (OTD) collects traffic data to meet the needs of Georgia's federally required traffic monitoring program. Traffic data is collected using continuous counters, short-term counters, or portable and continuous weigh-in-motion counters. | • Spatial: Geodatabase<br>• Tabular: Excel<br>• Document: Rounding Rules<br>• Historical: 2010-2019 |
| Road Inventory | Download road inventory data in different formats for more than 125,000 centerline miles of public roads in Georgia. OTD utilizes remote sensing, Local Road Activity (LRA) reports, construction design plans, and data mining to collect road data. The road inventory data includes state routes, county roads, and city streets. | • Spatial:Geodatabase<br>• Tabular: Excel |
| Data Dictionary: Road Inventory | The Data Dictionary describes the codes and methodology used for the collection of road inventory data. This file should be downloaded in conjuction with the road inventory files (in either tabular or geospatial formats). | Document: Data Dictionary |

**Figure 6.** Screenshot. GDOT Road and Traffic Data. The most recent numbers from 2019 are used, as 2021 data have yet to be published. Under the Traffic Data Type, the Spatial Geodatabase is used for this project.

The ArcGIS *Near* function is used to link the VLPs with this geolocated traffic information. Inputs are a feature class and target feature class to find the closest feature of the target feature class. The output is the FID of the closest feature in the Traffic Data into the VLP Attribute Table. A search radius of 100 ft is set, which is intended to avoid incorrect closest features from being linked to the data.



Pop-up

TC_Locations_2019 (1)

121-6214

TC_Locations_2019 - 121-6214

| | |
|---|---|
| FID | 11585 |
| Station_ID | 121-6214 |
| Functional | 6U : Urban Minor Collector |
| Lat_Long | 33.749094, -84.405999 |
| Year | 2019 |
| AADT | 4530 |
| Statistics | Estimated |
| Single_Uni | 0 |
| Combo_Unit | 0 |
| PCT_Peak_S | 0 |
| PCT_Peak_C | 0 |
| K_Factor | 0 |
| D_Factor | 0 |
| Future_AAD | 5700 |
| Station_Ty | Short Term |
| Latitude | 33.749094 |
| Longitude | -84.405999 |

1 of 1 84.4059990°W 33.7490940°N

**Figure 7.** Screenshot. Visualization of Traffic Location Data (left) with georeferenced information (pop-up on right). AADT is used for this study. For our example here, 4,530 vehicles is the AADT for 2019. Attribute information is later linked based on the FID closest to a vehicle route segment (representing the road the vehicle is on) using the ArcGIS *Merge* function.

## Optimal Route Length Difference (Ind. Variable)

Key to the investigations performed in this project is the detailed analysis of the vehicle routes that are identified based on the vehicle-locating data collected from the GDOT vehicles traveling over the road network. We consider the difference between the actual routes taken by the GDOT vehicles (identified by the vehicle-locating data) and the determined optimal routes in traveling between points in the network, where optimality is measured by the shortest route length, where we analyze these differences by identified vehicle route segments. In order to compare the taken routes and optimal routes, we first need to determine the optimal routes; then, we look at the difference between these two routes by length.

To identify and create the optimal route segments, the start and end points are taken for each vehicle segment and input into the Network Database Analyst as “Stops” using the *Import Stops* function. We then use the function *Solve* to create the optimal vehicle segments. As the ArcGIS *Feature Compare* cannot be utilized for polylines but rather points, we use the length difference between the optimal route and the taken route as the model variable.

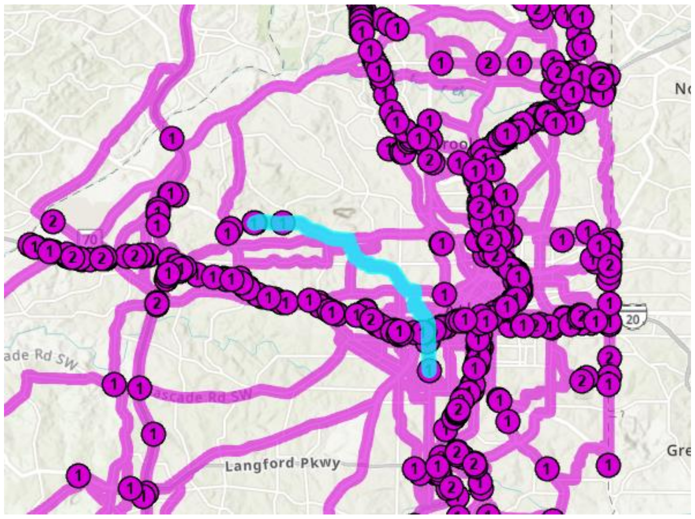


**Figure 8.** Screenshot. ArcGIS *Solve* function output. Each purple line, with one highlighted in cyan as a demonstration, represents an optimal path. 1’s represent the starting point and 2’s represent the ending point of a vehicle route segment.

As identification and comparison with the optimal route is key to the analysis of the vehicle route segments and use of the vehicle-locating data, the optimal route workflow is now described in more detail. To explain the workflow of creating the Optimal Route from the *Valid.py* vehicle-locating points layer, we must first make a separate layer of the start and end points for each route segment. With this separate layer, we can then run *Solve* similar to that of *Valid.py*, except we use the ArcGIS code rather than our developed code as in *Valid.py* to find the output. Once this step is completed, we compare the actual taken route and the optimal route by using the taken route segments and the optimal segments (from *Solve*) to create an Optimal Route Length Difference field. The output is the length difference between the taken route and the optimal route.

**Daily Precipitation (Ind. Variable)**

The website *Weather Underground* provides historical daily weather condition values, including precipitation, humidity, temperature, and wind speed. Precipitation is utilized for our model, pairing the daily precipitation total (in inches) to the vehicle routes. Wind speed is also said to have a significant impact on road conditions, but wind speed never reached critical values to become hazardous for the road network. The thought with adding a precipitation variable is that the chance of vehicle hydroplaning combined with a lack of driver visibility with increased precipitation would intuitively increase likelihood of vehicle collisions, a major contributor to road blockages.

**2020 GDOT Traffic Factors (Ind. Variable)**

The Georgia Department of Transportation publishes traffic factors every year that correspond to daily and monthly impact on traffic. The most recent published copy comes from 2020, and our values focus on Fulton County and on Minor/Major Arterials and Freeways. From the data, we see, for example, that Weekends (daily) and the Spring months of March to May (monthly) have the highest (on average) traffic factor contributions into the model.

**Historical Route Danger (Ind. Variable)**

Considering that certain parts of the road network are more likely to experience blockages, we define a new variable called the Historical Route Danger that evaluates along the sum of historical blockages/incidents that have occurred along the route in recent years. Using the same method for collecting the model input, two separate danger indexes are collected from the WebEOC Executive Report and from the *Numetric* Crash Data website. WebEOC Executive Report represents the

GDOT-reported road blockages along the Georgia road network before the first point of our vehicle geolocated points, spanning January 2016 to March 2021. Numetric data spans January 2013 to the present day, and includes privately collected vehicle crashes, but in much higher quantity than that of the WebEOC Report data. Note that Numetric data does not represent all road blockages along the road network, just vehicle collisions.

The procedure for collecting the historical blockages is first that each of the routes is given a distance buffer, hence creating a polygon which contains the route segment. Next, the ArcGIS *Spatial Join* feature is used, which allows the route layer to be contrasted with the WebEOC Report and Numetric Crash Data. The "*Match Option* = *Contains Within*" parameter within ArcGIS is used to find all instances of the Join Feature (WebEOC/Numetric layer) contained within each of the polygons (representing each route with a buffer). With these collected values along each vehicle route, we characterize the danger (or likelihood of blockage) of roads traveled along the route.

**WebEOC Incident Presence (Dep. Variable)**

Finally, the objective is to use the independent variables described above in order to predict road blockages in the network, in this case measured by road incidents as recorded by WebEOC. Thus, we use the WebEOC data as our model output, with the presence of an incident being the binary dependent variable for prediction. We run classification models for the datasets, split on the binary dependent variable between ten different time buffers: 0 hours, 1 hour, 3 hours, 6 hours, 12 hours, 1 day, 2 days, 1 week, 2 weeks, and 1 month, where the buffers are subtracted from each vehicle point's start time to allow for greater time in a route between the (Start Time – Buffer) and End Time. These time frames are selected to ensure sufficient data (i.e., sufficient numbers of individual data points) for the training and testing of the models. If a vehicle route segment is within 100 ft of a WebEOC incident within the time frames, a “1” is given for the presence of the traffic incident. A “0” represents the vehicle route segment not being in the presence of a traffic incident with the same constraints. Note that with an increase in data and processing capabilities, the model should be trained on the incident being within an hour or less of the vehicle driving by, to be able to pinpoint when a traffic incident has occurred and the application of a real-time monitoring system. This study focuses on a particular subset of the data, with the scope focused on Fulton County and the longer time frames to demonstrate the model’s feasibility and applicability.

## 4.2. Classification Model Evaluation

We chose machine learning as a method of detecting road blockages because of the ability of the models to learn indicators/trends. Additionally, the model, through training sets has the ability to improve itself. Specifically, decision tree-based classification was the method selected, as the nature of binary prediction was well-suited for our project. Decision Trees are a supervised machine learning algorithm that use strings of rules to make classifications. To run the classification model, the shapefile was exported to MATLAB, where a script was written to handle and transform the ArcGIS feature classes to usable *double* formatted matrices.

We analyze the accuracy of our classification results using a confusion matrix, which displays the true result (blockage or no blockage) and the model-predicted result (blockage or no blockage) on a 2x2 matrix. If we take the presence of a route blockage as the “positive” class, and no route blockage present as the “negative” class, then True Class = 1 and Predicted Class = 1 indicates a true negative (TN), upper left in the confusion matrix; True Class = 1 and Predicted Class = 2 indicates a false positive (FP), upper right in the confusion matrix; True Class = 2 and Predicted

Class = 2 indicates a true positive (TP), lower right in the confusion matrix; and True Class = 2 and Predicted Class = 1 indicates a false negative (FN), lower left in the confusion matrix.

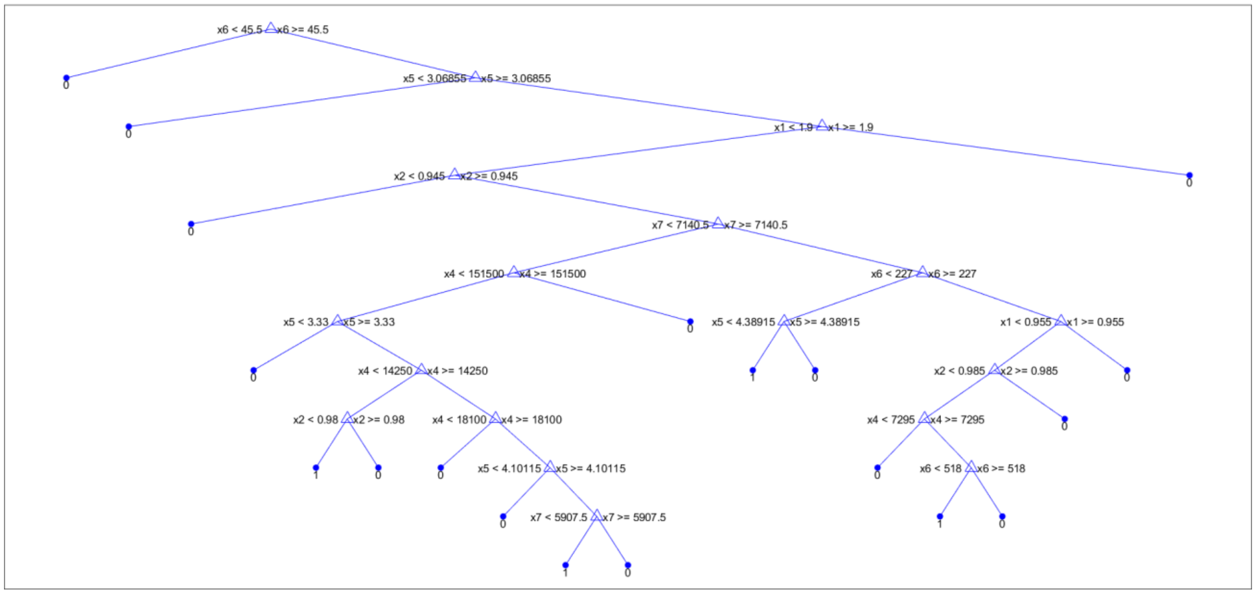


**Figure 9.** Screenshot. Classification decision tree visualized in MATLAB for 3-hour buffer. Each node represents a predictive decision made by the model to arrive at an estimate for whether or not a route blockage is present. End nodes (leaves) represent these binary predictions. In this model, x1 = Precipitation, x2 = Daily Traffic Factor, x3 = Monthly Traffic Factor, x4 = Average Annual Daily Traffic, x5 = Optimal Length Difference, x6 = Historical Route Danger – WebEOC, x7 = Historical Route Danger – Numetric.

From these values, we can calculate the accuracy and performance of the classification model. In particular, we are interested in the recall and precision of the models. Recall indicates the ability of a classification model to identify the data points in a relevant class and is calculated as $\frac{TP}{TP+FN}$. Precision, on the other hand, indicates the ability of a classification model to return only the data points in a class and is calculated as $\frac{TP}{TP+FP}$.

In this case, we will look at the 2-day and 1-month buffers for a comparison of the classification models. For the 2-day model, Recall = 9.9 percent and Precision = 16.7 percent. The 1-month yields stronger results, with Recall = 68.3 percent and Precision = 64 percent. Out of the 44,143 segments, there were 1,746 (3.96%) blockages for the 2-day buffer (3.96% of the route segments had an incident occur on the route between their start and end times, or to a month before) and there were 16,337 (37.01%) blockages for the 1-month buffer. It can be concluded that the 1-month classification model has more information about blockages so it can develop more specifications/decisions to develop a classification. As the 2-day buffer model only has ~ 4% true classification result (blockage present), it has less information about what produces a blockage from the explanatory variables.

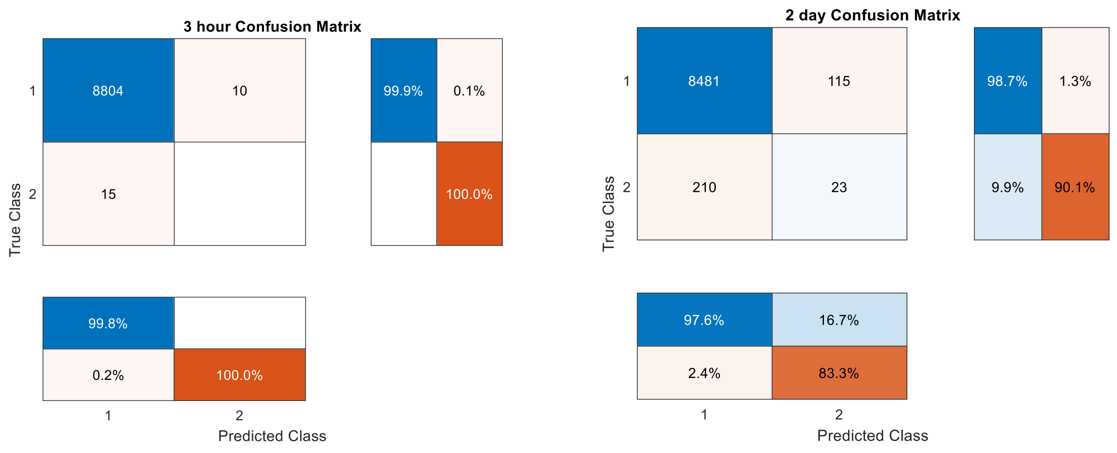


**Figure 10.** Screenshots. Confusion matrices for 3-hour and 2-day classification models, with row summaries (right of each matrix) also shown. With less WebEOC intersections in the 3-hour classification model, the tree-based algorithm predicts far fewer road blockages than that of the 1-month, with a higher percentage of WebEOC intersections.

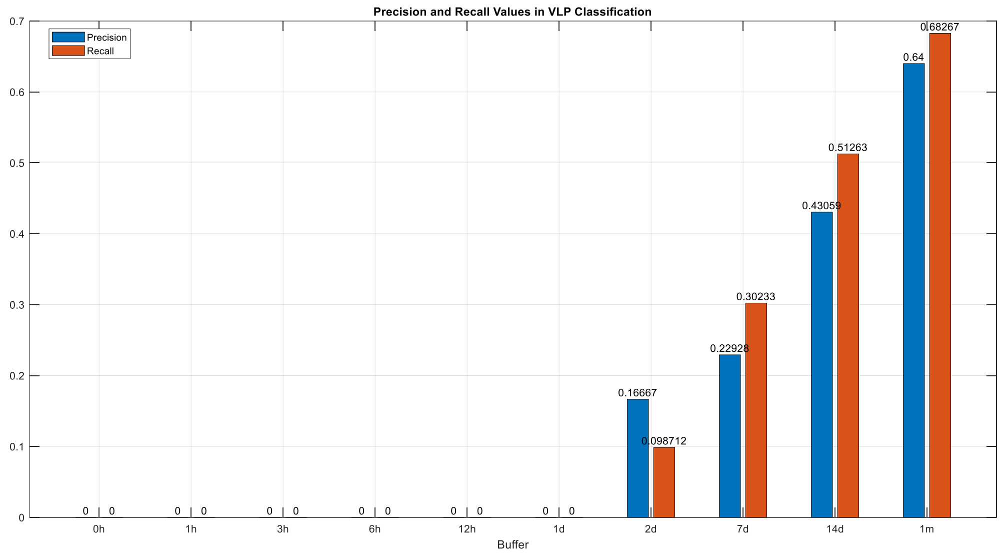


**Figure 11.** Screenshot. Calculated Precision and Recall values in the Classification model for each time buffer, derived from confusion matrices. 1-month time buffer shows the best results for both precision and recall results.

Recall and Precision can be compiled as well into an F1 score, which is equal to $2 \times \frac{Precision \times Recall}{Precision+Recall}$. The score is scaled between 0 and 1, where 1 represents a perfect classification. Similar to the recall/precision results, and intuitively, the F1 scores increase the larger the buffer is. The smallest F1 score is that of the 2-day model, which produced a 0.12, where the 1-month classification model performed quite well at a 0.66 value. It should be noted that the 0-hour through 1-day models did not have a true positive, so therefore cannot have a real F1 score. As these models have the goal of being implemented real-time, this was highly discouraging, but with more factors identified and more data input into the model, we could see more positive results.

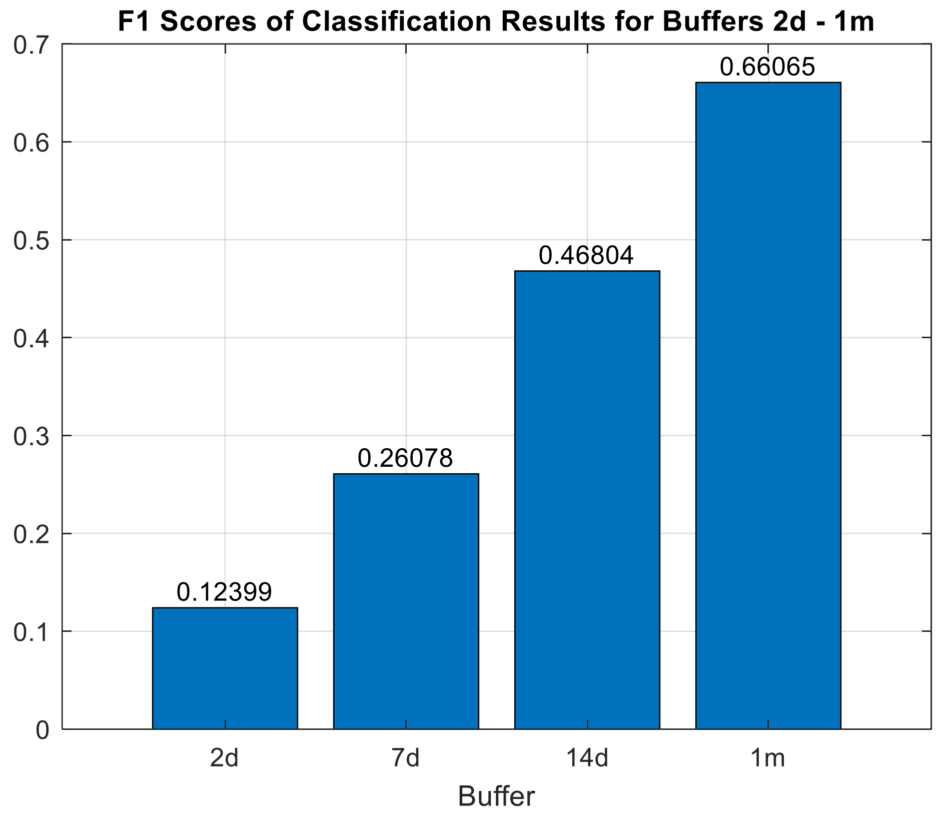


**Figure 12.** Screenshot. F1 scores in the Classification Model for each time buffer, derived from confusion matrices. Scores are based on a scale between 0-1, where 1 shows a perfectly fit classification model that made no false guesses in class. The model was unable to guess a true positive for the buffers 0-hour to 1-day, which have been omitted on the chart for that reason.

## Upsampling Classification Method with Imbalanced Data

We see poor F1 results with our model due to the data containing a high number of vehicle locating routes without WebEOC incident overlap. With this large imbalance in the dataset, the model is hindered from the ability to learn trends about the data, favoring erring on the side of guessing no WebEOC overlap for the classification. This leads us to explore ways that the model is able to detect these trends even with the imbalanced dataset.

Upsampling is a method which generalizes the classification boundary of the minority class to help improve the predictive performance of the model (Tran et al 2022). Looking at the 3-hour buffer, we see $^{102}/_{44,142}$ (or ~ 0.23 percent) of the routes containing a WebEOC blockage within the time frame. With such an imbalance and the previously stated low F1 scores, upsampling is deemed to be an important method to implement to reform the training dataset. We are able to customize our model further, including defining the concentration of upsampling in the training set, the time buffer to lengthen vehicle routes, and adjusting the misclassification cost (where the model can assign varying weights to false results to potentially increase accuracy in the predictive results). Each of these factors were tested under statistically significant trialing to maximize predictive capability in our model.

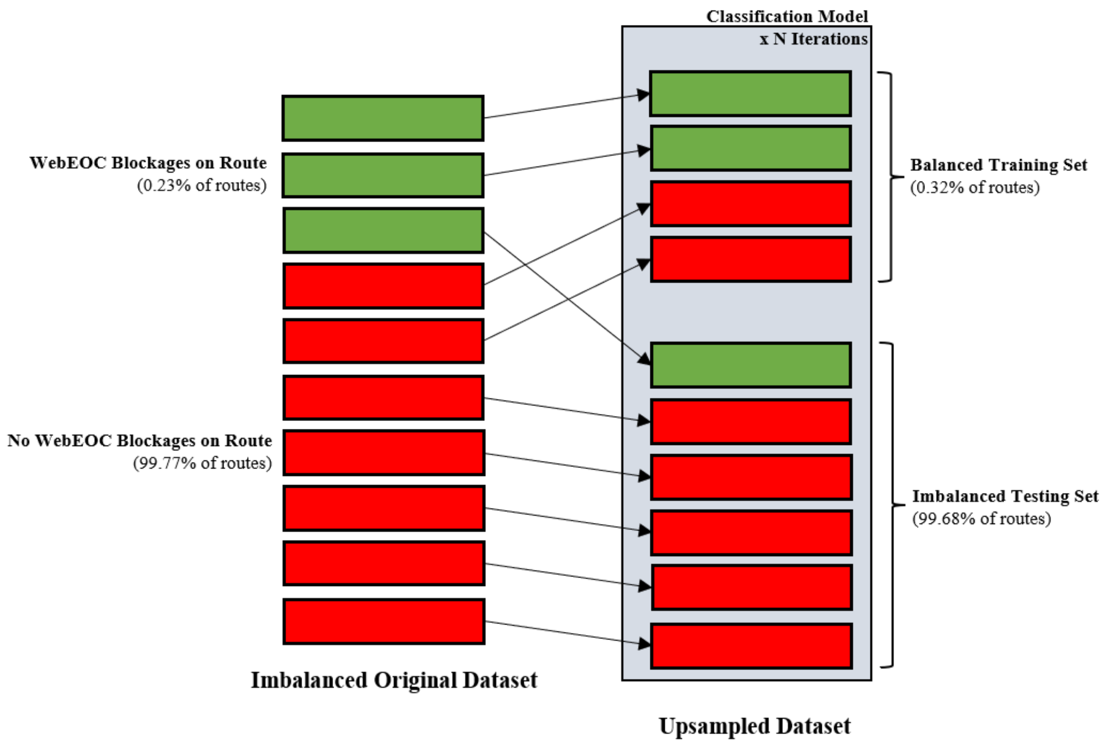


**Figure 13.** Upsampling approach with development of balanced training set for higher predictive capability [based on methodology of Tran 2022].

In an effort to generate meaningful results, we choose the 3-hour time buffer, meaning that a blockage spatially detected along the route between 3 hours before the route and at the end of the route is counted as a "true" outcome (WebEOC_3h = 1). Within this time buffer, we choose a training set consisting of 50 percent true outcomes (blockage detected) and 50 percent false outcomes (no blockage detected). True and false outcomes for this training set are both randomly selected. For the misclassification cost, after testing a variety of cost ratios, we choose the cost for a false negative (FN) result as being 5 times higher than that of a false positive (FP). False negative, in the scope of our model, means that the model predicts that there is no blockage, but there actually is a blockage along the route. False positives, on the other hand, represent the model predicting there is a blockage along the route, but not being correct in doing so. The goal is to have the model be able to learn the trends of when blockages occur. A strong recall value is indicative that the model is properly applying data trends towards accurate positive class prediction, and is the selected measure of importance for this study.

Across 50 classification trials, the model outputs 97.13 percent recall and 0.53 percent precision. From these results, the recall is high, and it can be said that the model is at least grouping the common characteristics of the data behind a road blockage where it is not missing many "true" outcomes. To address the precision, we find in our trials that the higher the ratio of true outcomes to false outcomes in our training set, the more the model will predict true outcomes. With 50 percent of our training set being true outcomes, the model ends up predicting 15.41 percent of all routes as having a WebEOC blockage along the route on average, which is much less than the 50 percent of routes having a WebEOC blockage as contained within the training set. The model is able to sort the characteristics of a possible road blockage, at a rate that is very much not random. Applied to the larger dataset, it is difficult to predict the few true outcomes among all the total outcomes. We see that the model is predicting true cases very well and compared with the overall data composition, at a much lower rate than what is presented in the training set. We would anticipate that with an expanded dataset and an increase in predictive variables that these results would only increase.

# CHAPTER 5

# CONCLUSIONS

This project explores the ability to use GDOT vehicle-locating data to assess the state of the road network in Georgia, including identifying road blockages along different segments of the transportation system. The goal is to determine if we are able to utilize data that are currently being collected to perform this network assessment. This novelty is in using a different data source than has been used or explored in the past, specifically mobile vehicle-locating data collected from GDOT-owned vehicles, rather than using stationary data sources such as loop detectors, traffic cameras, or traffic monitoring stations; or public crowdsourced data sources that rely on third parties for data collection and curation.

Through the course of the project, we made several discoveries. First, the data are crucial to the ability to create such a system. The raw dataset of vehicle-locating data is large and, in many cases, messy, with cases of missing data, zero-length data segments, and redundant route segments. Through multiple data trimming and processing methods developed and implemented using ArcGIS-specific Python algorithms, this initially large dataset is made into a usable format to run machine learning models to see the importance of multiple variables, including the vehicle-locating data and associated routing decisions, on the likelihood of road blockage detection. The steps for transforming the data that have been established as part of this project are described in detail and are reliable and repeatable methods that can be implemented with new datasets.

Second, to utilize the vehicle-locating data, we create a workflow to enable comparison of the vehicle routes with optimal routes to detect suboptimal routing decisions that may be indicative of blockages in the road network. This requires multiple steps in the workflow, including the creation of vehicle route segments based on the individual vehicle-locating data points, the linking of segments into routes, the identification of optimal routes between these points, and then the comparison of distances between the actual taken routes (processed from the vehicle-locating data points) and the optimal routes to detect the degree of suboptimal routing and its association with the likelihood of the presence of a road blockage.

Finally, to use this vehicle routing information to assess the state of the road network, we create machine learning models with multiple variables as input to detect the presence of a road blockage. The classification model is quite simple with the set of input variables and a binary output. The inputs, daily traffic on a given road, the difference between the taken route and the optimal route, precipitation, traffic factors, and historical route danger are clear indicators of whether or not a road blockage can be detected, but it is also clear that there are additional variables at play in leading to road blockages, and this set of variables cannot be relied upon to produce a tell-all classification model, capable of being run continuously with real-time data sources. For this model to run in real-time, it must be complemented with other variables, such as data from live traffic feeds. Therefore, the results from this work are an indication of the potential use of this specific new data source and application, but this is not the end result. At the very minimum, with Upsampling, the 3-hour model outputted significant recall results at 97.13%.

As different transportation-related data sources emerge, there is the opportunity to leverage these data sources for monitoring of the conditions of a transportation network. Rather than relying on external third-party data, this project explores the use of GDOT-collected data for this purpose. In addition, it focuses on the use of data that are already currently being collected, demonstrating the utility of these data in performing road network assessment without the need to invest in new technologies, dedicate additional resources, or implement new instrumentation or infrastructure. Through the use of multiple data processing methods combined machine learning approaches, we show how the vehicle-locating data can be used to perform network assessment and detection of blockages in the road network.

# CHAPTER 6

# FUTURE WORK

There are a variety of factors that, if and when applied, are sure to increase the model's predictive capability. Increased power of processing would result in more data, stemming from more Georgia counties included, more GDOT vehicles on record, a broader range of dates analyzed (resulting in more fluctuation in variables such as precipitation and traffic factors), and an increased road network analyzed. The project, due to processing limitations, was constrained to a randomly sampled subset of Fulton County specifically (a small subset of the overall data from WebEOC and Verizon Network Fleet). Once the model receives inputs beyond Fulton County and for a broader range of times, more WebEOC and historical Numetric blockages will enable broader assessment and characterization of the Georgia road network. It is then that we will be able to see true effect of the supervised model, although the run model does show positive outlooks.

Including more variables would also likely increase the model's performance, and could be customizable to the use of what factors are deemed important in a road network, entirely at the discretion of the operator. An expanded analytical approach to the way road infrastructure (and specifically, vehicular traffic) is managed will lead to data-driven solutions. For example, incorporating live data feeds from a service such as *Waze* (via public-use API) may give a new edge to our model's predictive capability, pairing live traffic feeds with historically trained predictions. Worth looking into as well are unsupervised learning techniques, such as anomaly detection, k-means clustering, and other methods. Unsupervised learning is focused on determining data patterns, which may be a better fit for the datasets analyzed.

# APPENDIX A

# ARCPY GLOSSARY

Below are the definitions of varying functions used in ArcGIS and ArcPy as part of the ArcGIS and Python functions and codes that have been developed in this project.

a) Add Join

    arcpy.management.AddJoin(in_layer_or_view, in_field, join_table, join_field, {join_type}, {index_join_fields})

b) Buffer

    arcpy.analysis.Buffer(in_features, out_feature_class, buffer_distance_or_field, {line_side}, {line_end_type}, {dissolve_option}, {dissolve_field}, {method})

c) Calculate Field

    arcpy.management.CalculateField(in_table, field, expression, {expression_type}, {code_block}, {field_type}, {enforce_domains})

d) Clip

    arcpy.analysis.Clip(in_features, clip_features, out_feature_class, {cluster_tolerance})

e) Convert Time Field

    arcpy.management.ConvertTimeField(in_table, input_time_field, {input_time_format}, output_time_field, {output_time_type}, {output_time_format})

f) Delete Selection

    arcpy.management.DeleteFeatures(in_features)

g) Feature Compare

    arcpy.management.FeatureCompare(in_base_features, in_test_features, sort_field, {compare_type}, {ignore_options}, {xy_tolerance}, {m_tolerance}, {z_tolerance}, {attribute_tolerances}, {omit_field}, {continue_compare}, {out_compare_file})

h) Feature Class To Feature Class

arcpy.conversion.FeatureClassToFeatureClass(in_features, out_path, out_name, {where_clause}, {field_mapping}, {config_keyword})

i) Feature To Point

arcpy.management.FeatureToPoint(in_features, out_feature_class, {point_location})

j) Feature Class To Shapefile

arcpy.conversion.FeatureClassToShapefile(Input_Features, Output_Folder)

k) Generalized Linear Regression (GLR)

arcpy.stats.GeneralizedLinearRegression(in_features, dependent_variable, model_type, output_features, explanatory_variables, {distance_features}, {prediction_locations}, {explanatory_variables_to_match}, {explanatory_distance_matching}, {output_predicted_features})

l) Make Route Analysis Layer (*Import Stops*, *Run*, *Routes*)

arcpy.na.MakeRouteAnalysisLayer(network_data_source, {layer_name}, {travel_mode}, {sequence}, {time_of_day}, {time_zone}, {line_shape}, {accumulate_attributes}, {generate_directions_on_solve}, {time_zone_for_time_fields}, {ignore_invalid_locations})

m) Merge

arcpy.management.Merge(inputs, output, {field_mappings}, {add_source})

n) Segment Along Line

arcpy.segmentAlongLine (start_measure, end_measure, {use_percentage})

o) Select By Attributes

arcpy.management.SelectLayerByAttribute(in_layer_or_view, {selection_type}, {where_clause}, {invert_where_clause})

p) Solve

arcpy.na.Solve(in_network_analysis_layer, {ignore_invalids}, {terminate_on_solve_error}, {simplification_tolerance}, {overrides})

q) Spatial Join

```
arcpy.analysis.SpatialJoin(target_features, join_features, out_feature_class, {join_operation},
{join_type}, {field_mapping}, {match_option}, {search_radius}, {distance_field_name})
```

r) Summary Statistics

```
arcpy.analysis.Statistics(in_table, out_table, {statistics_fields}, {case_field})
```

s) XY Table To Point

```
arcpy.management.XYTableToPoint(in_table, out_feature_class, x_field, y_field, {z_field},
{coordinate_system})
```

## APPENDIX B

## USER CODES

The below functions and descriptions apply to the user-generated processing code:

```
import arcpy
import pandas as pd

# get parameters from the toolbox interface
tbl_Segments = arcpy.GetParameterAsText(0)
field_route_name_tbl = arcpy.GetParameterAsText(1)
field_start_position = arcpy.GetParameterAsText(2)
field_end_position = arcpy.GetParameterAsText(3)
field_last_updated = arcpy.GetParameterAsText(4)

lyr_route = arcpy.GetParameterAsText(5)
field_route_name_lyr = arcpy.GetParameterAsText(6)

workspace_output = arcpy.GetParameterAsText(7)
result_route_name = arcpy.GetParameterAsText(8)

# read segment table from the csv file
df_segment_position = pd.read_csv(tbl_Segments)

# detect the workspace type
dec_workspace = arcpy.Describe(workspace_output)

type_workspace = dec_workspace.workspaceType

# if workspace is a folder, export a shapefile (.shp)
if type_workspace == "FileSystem":
    result_route_name = result_route_name + ".shp"

result_route_layer_path = workspace_output + "\\" + result_route_name

# get spatial reference of the route layer
spRf = arcpy.Describe(lyr_route).spatialReference

# create segments feature class
arcpy.management.CreateFeatureclass(workspace_output, result_route_name,
geometry_type="POLYLINE", spatial_reference=spRf)
arcpy.management.AddField(result_route_layer_path, "Route_name", "TEXT", None, None,
100)
arcpy.management.AddField(result_route_layer_path, "S_Position", "DOUBLE")
arcpy.management.AddField(result_route_layer_path, "E_Position", "DOUBLE")
arcpy.management.AddField(result_route_layer_path, "LastUpdated", "Date")

# insert segments into segments feature class
in_Cur = arcpy.da.InsertCursor(result_route_layer_path, ["SHAPE@", "Route_name",
"S_Position", "E_Position", "LastUpdated"])
```

```
# no. of routes for counting and defining progressor
n_route = df_segment_position.count()[0]


arcpy.SetProgressor("Step", "processing....", 0, n_route, 1)

i = 0

# loop segments' row in segments table, get the segment from road network.
for index, row in df_segment_position.iterrows():

    route_name = row[field_route_name_tbl]
    start_position = row[field_start_position]
    end_position = row[field_end_position]
    last_updated = row[field_last_updated]

    i = i + 1

    arcpy.SetProgressorPosition()
    arcpy.SetProgressorLabel("processing " + route_name + "...... " + "{}/{}, {:.1f}%".format(i,
n_route, i * 100.0 / (n_route)))

    s_cur_route = arcpy.da.SearchCursor(lyr_route, ["Shape@", field_route_name_lyr], "{0}
='{1}'".format(field_route_name_lyr, route_name))

    try:
        s_c_route = s_cur_route.next()

        segments_shp = s_c_route[0].segmentAlongLine(start_position, end_position)
        in_Cur.insertRow((segments_shp, route_name, start_position, end_position,
last_updated))
        arcpy.AddMessage("{}: start from {}, end at{}, last updated {}, OK".format(route_name,
start_position, end_position, last_updated))

# if the route in the csv file does not have corresponding name in the route network layer,
return a "NotOK"
    except StopIteration:
        arcpy.AddMessage("{}: start from {}, end at{}, last updated {},
NotOK".format(route_name, start_position, end_position, last_updated))
        continue

del in_Cur

arcpy.ResetProgressor()
```
*GetSegments Code*

```
import arcpy
import os

# to export direction information using "AddLocation method", xml format should be
used
import xml.dom.minidom as xmld

class FindRoutes():

    def __init__(self, track_points, track_points_name, order_field, ignition_field,
network_NAlyr, output_db_path, output_routes_name, output_statistical_tbl_name):
        self.track_points = track_points
        self.track_points_name = track_points_name
        self.order_field = order_field
        self.ignition_field = ignition_field
        self.network_NAlyr = network_NAlyr
        self.output_db_path = output_db_path
        self.output_routes_name = output_routes_name
        self.output_statistical_tbl_name = output_statistical_tbl_name

        self.lyr_tem = arcpy.MakeFeatureLayer_management(self.track_points, "layer_tem")

        self.dic_points_name = {}
        self.dic_points_ignition = {}
        self.dic_points_counter = {}
        self.dic_points_fixtime = {}


        self.result_routes = os.path.join(output_db_path, output_routes_name)
        self.result_statistical_table = os.path.join(output_db_path, output_routes_name)

        # add counter field to layer
        # counter is defined to account for individual travels (judging from Ignition status)
        arcpy.AddField_management(track_points, "Counter", "SHORT")

        # assign counters
        i = 1
        up_cur = arcpy.da.UpdateCursor(track_points, [order_field, ignition_field, "Counter"],
sql_clause=(None, "ORDER BY " + order_field))
        for up_c in up_cur:
            up_c[2] = i
            if up_c[1] == "Off":
                i = 0
            i = i + 1
            up_cur.updateRow(up_c)
        # create the statistical table
        self.result_table = arcpy.CreateTable_management(output_db_path,
output_statistical_tbl_name)
        self.statis_tbl_fields = ["Track_Point_Name",
                                  "Before2p",
                                  "Before2p_dif",
                                  "Before3p",
                                  "Before3p_dif",
                                  "Before4p",
                                  "Before4p_dif",
                                  "Before5p",
                                  "Before5p_dif",
                                  "Ignition",
                                  "Counter",
                                  "FixTime"]

        # define the format of the cell using AddField function
        for add_f in self.statis_tbl_fields:
            if add_f == "Track_Point_Name" or add_f == "Ignition":
                arcpy.AddField_management(self.result_table, add_f, "TEXT")
            elif add_f == "FixTime":
                arcpy.AddField_management(self.result_table, add_f, "Date")
            else:
                arcpy.AddField_management(self.result_table, add_f, "DOUBLE")

        # scan the track points
        s_cur = arcpy.da.SearchCursor(track_points, ["shape@", track_points_name, order_field,
ignition_field, "Counter"], sql_clause=(None, "ORDER BY " + order_field))

        n = 0
        for s_c in s_cur:
            n = n + 1
            self.dic_points_name[n] = s_c[1]
            self.dic_points_fixtime[n] = s_c[2]
            self.dic_points_ignition[n] = s_c[3]
            self.dic_points_counter[n] = s_c[4]

        RouteSubLayer = arcpy.na.GetNAClassNames(network_NAlyr)

        self.routeSlyr_stops = RouteSubLayer["Stops"]
        self.routeSlyr_route = RouteSubLayer["Routes"]

        arcpy.na.AddFieldToAnalysisLayer(network_NAlyr, self.routeSlyr_route, "Direction",
"TEXT", field_length=100000)
        arcpy.na.AddFieldToAnalysisLayer(network_NAlyr, self.routeSlyr_route,
"FixTime_Start", "Date", field_length=1000)
```

```
        arcpy.na.AddFieldToAnalysisLayer(network_NAlyr, self.routeSlyr_route,
"FixTime_Current", "Date", field_length=1000)

    # function for insert data into statistial table
    def insert_statis_table(self,
                            Track_point_name,
                            Before2p=None,
                            Before2p_dif=None,
                            Before3p=None,
                            Before3p_dif=None,
                            Before4p=None,
                            Before4p_dif=None,
                            Before5p=None,
                            Before5p_dif=None,
                            Ignition=None,
                            Counter=None,
                            FixTime=None):

        inst_val = [Track_point_name,
                    Before2p,
                    Before2p_dif,
                    Before3p,
                    Before3p_dif,
                    Before4p,
                    Before4p_dif,
                    Before5p,
                    Before5p_dif,
                    Ignition,
                    Counter,
                    FixTime]


        inst_statis_tbl = arcpy.da.InsertCursor(self.result_table, self.statis_tbl_fields)
        inst_statis_tbl.insertRow(inst_val)
        del inst_statis_tbl

    # function for setup stops for "AddLocation" and "Solve"; set up messages
    def stopsSetup(self, current_point=3, checkBack=2):

        stops1 = None
        stops2 = None
        message1 = None
        message2 = None

        if current_point >= 3 and checkBack == 2:
            stops1 = [self.dic_points_name[current_point - 2],
                    self.dic_points_name[current_point]]
            message1 = "{0}<--{1}, ignition:{2},
counter:{3}".format(self.dic_points_name[current_point],
                                                              self.dic_points_name[current_point - 2],
                                                              self.dic_points_ignition[current_point],
                                                              self.dic_points_counter[current_point])

            stops2 = [self.dic_points_name[current_point - 2],
                    self.dic_points_name[current_point - 1],
                    self.dic_points_name[current_point]]
            message2 = "{0}<--{1}<--{2}, ignition:{3},
counter:{4}".format(self.dic_points_name[current_point],
                                                                   self.dic_points_name[current_point - 1],
                                                                   self.dic_points_name[current_point - 2],
                                                                   self.dic_points_ignition[current_point],
                                                                   self.dic_points_counter[current_point])


        if current_point >= 4 and checkBack == 3:
            stops1 = [self.dic_points_name[current_point - 3],
                    self.dic_points_name[current_point]]
            message1 = "{0}<--{1}, ignition:{2},
counter:{3}".format(self.dic_points_name[current_point],
                                                              self.dic_points_name[current_point - 3],
                                                              self.dic_points_ignition[current_point],
                                                              self.dic_points_counter[current_point])

            stops2 = [self.dic_points_name[current_point - 3],
                    self.dic_points_name[current_point - 2],
                    self.dic_points_name[current_point - 1],
                    self.dic_points_name[current_point]]
            message2 = "{0}<--{1}<--{2}<--{3}, ignition:{4},
counter:{5}".format(self.dic_points_name[current_point],
                                                                       self.dic_points_name[current_point - 1],
                                                                       self.dic_points_name[current_point - 2],
                                                                       self.dic_points_name[current_point - 3],
                                                                       self.dic_points_ignition[current_point],
                                                                       self.dic_points_counter[current_point])
        if current_point >= 5 and checkBack == 4:
            stops1 = [self.dic_points_name[current_point - 4],
                    self.dic_points_name[current_point]]
            message1 = "{0}<--{1}, ignition:{2},
counter:{3}".format(self.dic_points_name[current_point],
                                                              self.dic_points_name[current_point - 4],
                                                              self.dic_points_ignition[current_point],
                                                              self.dic_points_counter[current_point])
            stops2 = [self.dic_points_name[current_point - 4],
                      self.dic_points_name[current_point - 3],
                      self.dic_points_name[current_point - 2],
                      self.dic_points_name[current_point - 1],
                      self.dic_points_name[current_point]]
            message2 = "{0}<--{1}<--{2}<--{3}<--{4}, ignition:{5}, 
counter:{6}".format(self.dic_points_name[current_point],
                                                                     self.dic_points_name[current_point - 1],
                                                                     self.dic_points_name[current_point - 2],
                                                                     self.dic_points_name[current_point - 3],
                                                                     self.dic_points_name[current_point - 4],
                                                                     self.dic_points_ignition[current_point],
                                                                     self.dic_points_counter[current_point])
        if current_point >= 6 and checkBack == 5:
            stops1 = [self.dic_points_name[current_point - 5],
                      self.dic_points_name[current_point]]
            message1 = "{0}<--{1}, ignition:{2}, 
counter:{3}".format(self.dic_points_name[current_point],
                                                       self.dic_points_name[current_point - 5],
                                                       self.dic_points_ignition[current_point],
                                                       self.dic_points_counter[current_point])

            stops2 = [self.dic_points_name[current_point - 5],
                      self.dic_points_name[current_point - 4],
                      self.dic_points_name[current_point - 3],
                      self.dic_points_name[current_point - 2],
                      self.dic_points_name[current_point - 1],
                      self.dic_points_name[current_point]]
            message2 = "{0}<--{1}<--{2}<--{3}<--{4}<--{5}, ignition:{6}, 
counter:{7}".format(self.dic_points_name[current_point],
                                                                          self.dic_points_name[current_point 
- 1],
                                                                          self.dic_points_name[current_point 
- 2],
                                                                          self.dic_points_name[current_point 
- 3],
                                                                          self.dic_points_name[current_point 
- 4],
                                                                          self.dic_points_name[current_point 
- 5],

self.dic_points_ignition[current_point],

self.dic_points_counter[current_point])
```

```
        return stops1, stops2, message1, message2

    # xml format is adopted to extract the direction information
    def getDirection(self):
        dr = arcpy.na.Directions(self.network_NAlyr, "XML").getOutput(0)
        dr_list = []
        dom = xmld.parse(dr)
        root = dom.documentElement
        rs = root.getElementsByTagName("STRING")

        for r in rs:
            if r.getAttribute("style") in ["depart", "normal", "arrive"]:
                dr_list.append(r.getAttribute("text"))

        dr_str0 = " --> ".join(dr_list)

        return dr_str0

    # function for finding routes from point 1-2
    def findRoutesBack1p(self, current_point=2):
        stops = [self.dic_points_name[current_point - 1],
                 self.dic_points_name[current_point]]

        # In ArcGIS Pro, the FindRoutes function can only be completed by online routing
service, for which the result is not extractable. Therefore, we now use Addlocation and
Solve to solve for the routes between points
        sql = str(stops).replace("[", "").replace("]", "").replace("u", "")
        arcpy.management.SelectLayerByAttribute(self.lyr_tem, "NEW_SELECTION", "{}
in({})".format(self.track_points_name, sql))
        arcpy.na.AddLocations(self.network_NAlyr,
                              self.routeSlyr_stops,
                              self.lyr_tem,
                              "Name {} #".format(self.track_points_name),
                              sort_field=self.order_field,
                              append="CLEAR")
        arcpy.na.Solve(self.network_NAlyr)


        result_route = self.routeSlyr_route

        dr_str = self.getDirection()

        message = "{0}<--{1}, ignition:{2},
counter:{3}".format(self.dic_points_name[current_point],
                                                self.dic_points_name[current_point - 1],
                                                        self.dic_points_ignition[current_point],
                                                        self.dic_points_counter[current_point])

        up_cur = arcpy.da.UpdateCursor(result_route, ["shape@", "Name", "Direction",
"FixTime_Start", "FixTime_Current"])

        route_shp = None
        for up_c in up_cur:
            up_c[1] = message
            up_c[2] = dr_str
            up_c[3] = self.dic_points_fixtime[current_point - 1]
            up_c[4] = self.dic_points_fixtime[current_point]
            route_shp = up_c[0]
            up_cur.updateRow(up_c)


        del dr_str

        if not arcpy.Exists(self.result_routes):
            arcpy.CopyFeatures_management(result_route, self.result_routes)
        else:
            arcpy.Append_management(result_route, self.result_routes)

        return route_shp, message



    # function for finding routes from point >= 3
    def findRoutesBack2p(self, current_point, backward, checkIndex):
        stops = None
        message = None

        stops1, stops2, message1, message2 = self.stopsSetup(current_point, backward)

        if checkIndex == 1:
            stops = stops1
            message = message1
        elif checkIndex == 2:
            stops = stops2
            message = message2

        sql = str(stops).replace("[", "").replace("]", "").replace("u", "")
        arcpy.management.SelectLayerByAttribute(self.lyr_tem, "NEW_SELECTION", "{}
in({})".format(self.track_points_name, sql))
        arcpy.na.AddLocations(self.network_NAlyr,
                              self.routeSlyr_stops,
                        self.lyr_tem,
                        "Name {} #".format(self.track_points_name),
                        sort_field=self.order_field,
                        append="CLEAR")
        arcpy.na.Solve(self.network_NAlyr) # route solving algorithm, arcgis function

        result_route = self.routeSlyr_route


        dr_str = self.getDirection()

        up_cur = arcpy.da.UpdateCursor(result_route, ["shape@", "Name", "Direction",
"FixTime_Start", "FixTime_Current"])

        route_shp = None
        for up_c in up_cur: # just recording
            up_c[1] = message
            up_c[2] = dr_str
            up_c[3] = self.dic_points_fixtime[current_point - backward]
            up_c[4] = self.dic_points_fixtime[current_point]
            route_shp = up_c[0]
            up_cur.updateRow(up_c)

        del dr_str

        arcpy.Append_management(result_route, self.result_routes)

        return route_shp, message
```
*mxFindRoutes Code (Valid.py Function File)*

```
import arcpy
import mxFindRoutes_pro

# get parameters from tool
track_points = arcpy.GetParameterAsText(0)
track_points_name = arcpy.GetParameterAsText(1)
order_field = arcpy.GetParameterAsText(2)
ignition_field = arcpy.GetParameterAsText(3)
NDS = arcpy.GetParameterAsText(4)
output_db = arcpy.GetParameterAsText(5)
result_routes_name = arcpy.GetParameterAsText(6)
result_table_name = arcpy.GetParameterAsText(7)

# setup the workspace can be overwrote
overw = arcpy.env.overwriteOutput

arcpy.env.overwriteOutput = True


# beginning of the tool
myFindRoute = mxFindRoutes_pro.FindRoutes(track_points, track_points_name, order_field,
ignition_field, NDS, output_db, result_routes_name, result_table_name)

# get the numbers of the track points, to set up the progressor
n = len(myFindRoute.dic_points_name)

arcpy.SetProgressor("step", "calculating route...", 0, n - 1, 1)

for i in range(1, n + 1):

    point_name = myFindRoute.dic_points_name[i]

    arcpy.SetProgressorPosition()
    arcpy.SetProgressorLabel("calculating route of point " + point_name + "......")

    bf2p = bf2p_dif = bf3p = bf3p_dif = bf4p = bf4p_dif = bf5p = bf5p_dif = None
    ignition = myFindRoute.dic_points_ignition[i]
    counter = myFindRoute.dic_points_counter[i]
    fixtime = myFindRoute.dic_points_fixtime[i]

    # 1st point of the travel
    if counter == 1:
        arcpy.AddMessage("p{0}:ignition:{1}, counter:{2}".format(str(i), ignition, counter))
    # 2nd point of the travel
    if counter == 2:
        r_shp, msg = myFindRoute.findRoutesBack1p(i)
        arcpy.AddMessage("p" + str(i) + ":" + msg)


    # 3rd point of the travel
    if counter >= 3:

        r_shp1_1, msg = myFindRoute.findRoutesBack2p(i, 2, 1)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route1_1 = r_shp1_1

        r_shp1_2, msg = myFindRoute.findRoutesBack2p(i, 2, 2)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route1_2 = r_shp1_2


        bf2p = 1
        bf2p_dif = 0

        # Sometimes the recorded points are too close to produce a route and if not
specified, it will cause errors and the termination of the program. Basically, the error is
from the result; when the result is none (no routes were returned), it will be impossible
for comparison.
        # To solve for this problem, the following codes were added (same for below):
        if shp_route1_2 is None and shp_route1_1 is not None:
            bf2p = 0
            bf2p_dif = 0 - shp_route1_1.length

        elif shp_route1_2 is not None and shp_route1_1 is None:
            bf2p = 0
            bf2p_dif = 0 - shp_route1_2.length

        elif shp_route1_2 is None and shp_route1_1 is None:
            bf2p = 1
            bf2p_dif = 0

        elif not shp_route1_2.equals(shp_route1_1):
            bf2p = 0
            bf2p_dif = shp_route1_2.length - shp_route1_1.length

    if counter >= 4:
        # check 4-1--> 2 or 3
        # --route 4-1
        r_shp2_1, msg = myFindRoute.findRoutesBack2p(i, 3, 1)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route2_1 = r_shp2_1


        r_shp2_2, msg = myFindRoute.findRoutesBack2p(i, 3, 2)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route2_2 = r_shp2_2

        bf3p = 1
        bf3p_dif = 0

        if shp_route2_2 is None and shp_route2_1 is not None:
            bf3p = 0
            bf3p_dif = 0 - shp_route2_1.length

        elif shp_route2_2 is not None and shp_route2_1 is None:
            bf3p = 0
            bf3p_dif = 0 - shp_route2_2.length

        elif shp_route2_2 is None and shp_route2_1 is None:
            bf3p = 1
            bf3p_dif = 0

        elif not shp_route2_2.equals(shp_route2_1):
            bf3p = 0
            bf3p_dif = shp_route2_2.length - shp_route2_1.length



    if counter >= 5:
        # check 5-1--> 2 or 3 or 4
        # --route 5-1
        r_shp3_1, msg = myFindRoute.findRoutesBack2p(i, 4, 1)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route3_1 = r_shp3_1


        r_shp3_2, msg = myFindRoute.findRoutesBack2p(i, 4, 2)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route3_2 = r_shp3_2
        bf4p = 1
        bf4p_dif = 0

        if shp_route3_2 is None and shp_route3_1 is not None:
            bf4p = 0
            bf4p_dif = 0 - shp_route3_1.length

        elif shp_route3_2 is not None and shp_route3_1 is None:
            bf4p = 0
            bf4p_dif = 0 - shp_route3_2.length

        elif shp_route3_2 is None and shp_route3_1 is None:
            bf4p = 1
            bf4p_dif = 0

        elif not shp_route3_2.equals(shp_route3_1):
            bf4p = 0
            bf4p_dif = shp_route3_2.length - shp_route3_1.length



    if counter >= 6:
        # check 6-1--> 2 or 3 or 4 or 5
            # --route 6-1
        r_shp4_1, msg = myFindRoute.findRoutesBack2p(i, 5, 1)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route4_1 = r_shp4_1


        r_shp4_2, msg = myFindRoute.findRoutesBack2p(i, 5, 2)
        arcpy.AddMessage("p" + str(i) + ":" + msg)

        shp_route4_2 = r_shp4_2

        bf5p = 1
        bf5p_dif = 0

        if shp_route4_2 is None and shp_route4_1 is not None:
            bf5p = 0
            bf5p_dif = 0 - shp_route4_1.length

        elif shp_route4_2 is not None and shp_route4_1 is None:
            bf5p = 0
            bf5p_dif = 0 - shp_route4_2.length
        elif shp_route4_2 is None and shp_route4_1 is None:
            bf5p = 1
            bf5p_dif = 0

        elif not shp_route4_2.equals(shp_route4_1):
            bf5p = 0
            bf5p_dif = shp_route4_2.length - shp_route4_1.length



    myFindRoute.insert_statis_table(point_name,
                                    bf2p, bf2p_dif,
                                    bf3p, bf3p_dif,
                                    bf4p, bf4p_dif,
                                    bf5p, bf5p_dif,
                                    ignition, counter, fixtime)



arcpy.env.overwriteOutput = overw
arcpy.ResetProgressor()
arcpy.SelectLayerByAttribute_management(track_points, "CLEAR_SELECTION")
#arcpy.RefreshActiveView()
```
*Valid Code*

# REFERENCES

Astarita, V., Giofrè, V.P., Guido, G., Stefano, G., and Vitale, A. (2020). "Mobile Computing for Disaster Emergency Management: Empirical Requirements Analysis for a Cooperative Crowdsourced System for Emergency Management Operation." *Smart Cities*, *3*(1), pp.31–47. Available online: https://doi.org/10.3390/smartcities3010003.

Bailey, K.D. (2005). "Typology Construction, Methods and Issues." In Kempf-Leonard, K. (Ed.), *Encyclopedia of Social Measurement*, Elsevier, London, *3*, pp. 889–898.

Basu, M., Bandyopadhyay, S., and Ghosh, S. (2016). "Post Disaster Situation Awareness and Decision Support Through Interactive Crowdsourcing." *Procedia Engineering*, *159*(2016), pp.167–173. Available online: http://dx.doi.org/10.1016/j.proeng.2016.08.151.

ESRI. (2021). *GIS Dictionary*. (website) Available online: http://webhelp.esri.com/arcgisserver/9.3/java/geodatabases/definition_frame.htm, last accessed May 1, 2021.

Georgia Department of Transportation (GDOT). (2021). "Road and Traffic Data." (website) Atlanta, GA. Available online: http://www.dot.ga.gov/ds/data#tab-4, last accessed September 1, 2021.

Iliopoulou, C., Konstantinidou, M.A., Kepaptsoglou, K.L., and Stathopoulos, A. (2020). "ITS Technologies for Decision Making During Evacuation Operations: A Review." *Journal of Transportation Engineering, Part A: Systems*, *146*(4), 04020010. Available online: https://doi.org/10.1061/JTEPBS.0000329.

Mena-Yedra, R., Casas, J., and Gavaldà, R. (2018). "Assessing Spatiotemporal Correlations from Data for Short-term Traffic Prediction Using Multi-task Learning." *Transportation Research Procedia*, *34*, pp.155–162. Available online: http://dx.doi.org/10.1016/j.trpro.2018.11.027.

Meng, F., Wong, S.C., Wong, W., and Li, Y.C. (2017). "Estimation of Scaling Factors for Traffic Counts Based on Stationary and Mobile Sources of Data." *International Journal of Intelligent Transportation Systems Research*, *15*(3), pp.180–191. Available online: http://dx.doi.org/10.1007/s13177-016-0131-1.

Tran T, Le U, Shi Y. (2022). An effective up-sampling approach for breast cancer prediction with imbalanced data: A machine learning model-based comparative analysis. *PLoS ONE* 17(5): e0269135. https://doi.org/10.1371/journal.pone.0269135.

Singh, N.K., Vanajakashi, L., and Tangirala, A.K. (2018). "Segmentation of Vehicle Signatures from Inductive Loop Detector (ILD) Data for Real-time Traffic Monitoring." *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, January, pp. 601–606. Available online: https://doi.org/10.1109/COMSNETS.2018.8328281.

Weather Underground. (2021). "Atlanta, GA Weather History." TWC Product and Technology, Brookhaven, GA. Available online: https://www.wunderground.com/history/monthly/us/ga/atlanta/KATL/date/2021-6, last accessed September 1, 2021.

Yale University. (2021). "Linear Regression." (website) Available online: http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm, last accessed September 1, 2021.