Predicting Traffic Accident Severity in Montgomery County, Maryland

Jacob Rodgers

Loyola University of Maryland: College of Arts and Sciences

DS796: Data Science Project

Dr. Bu Hyoung, Lee

April 12th, 2024

Table of Contents

Figures

Predicting Traffic Accident Severity in Montgomery County, Maryland

Abstract

The purpose of this project is to develop predictive models for automobile accident severity in Montgomery County, Maryland between the years of 2015 and 2023. Accident severity is determined on a tri-fold basis consisting of driver injury severity, driver vehicle damage, and number of drivers involved in a collision. The types of machine learning models used include tree-based models: random forest and xgboost, and non-tree-based models: logistic regression, Bernoulli and complement naïve bayes. Hyperparameters were selected using a 10-fold grid search cross-validation and models were scored on accuracy, weighted f1 score and confusion matrix analysis. In most cases, the selected target variables struggled with class imbalances. This is reflected in the performance of the models as the major class values deemed easier to predict than the minor class ones. Between the three target variables, the results demonstrate that the tree-based models performed the best in general. Visualizations were created to compare model results and to identify trends between the selected variables. Drawbacks to the study include problems with computer resources and environment setup which did not allow for the construction of neural networks to be compared against the other selected model types. Overall, the project proves the benefits of using tree-based models to predict accident severity, and provides Montgomery County, Maryland with valuable information in regard to the safety of the public.

Keywords

Injury Severity, Vehicle Damage, Automobile Accidents, Machine learning, Random Forest, XGBoost, Montgomery County, Crash Reports, Data Visualization, Python, GeoPandas, Scikit-Learn

1. Motivation and Background

Motivation

Since the dawn of the modern-day automobile, people have struggled with the challenge of passing around or avoiding collisions on the road. Whether one is involved in or witnesses an automobile accident, they are affected by the overall outcome of the collision. Thus, the topic of automobile incidents is a significant one, due to the impact on transportation and public safety that it carries. This study aims to analyze accident data with the goal of predicting the severity of an accident in hopes of improving the safety of the roads in Montgomery County, Maryland.

The dataset at hand was chosen given the proximity of Montgomery County, Maryland to Loyola and the highly dense metropolitan area of Washington DC. Also, before selecting the associated topic, perceptions on the severity of automobile accidents were skewed heavily toward being more severe than the actual data suggests. While there is no lack of automobile accidents in Montgomery County, the proportion of highly severe accidents to the overall whole is quite marginal. However, this project's interest still lies in attempting to generate predictive models such that factors may be identified to further reduce the number of severe accidents in an area with already acceptable public safety record. Deployment of such models may prove to be

essential so that county authorities can adapt and instigate policies, further reducing the effect of automobile accidents on county residents and commuters alike.

Background and Challenges

For the purposes of this data science project, Montgomery County, Maryland is assuming the role of the client exclusively in theory. No officials or employees of the county were contacted or involved in the project's proposal and accomplishment, as all of the data utilized in this project has been obtained directly from the internet via the openMontgomery initiative. In late 2012, the Montgomery County Government launched its openMontgomery program under which dataMontgomery resides. "The dataMontgomery program provides residents and constituents with direct access to County datasets in consumable formats, so they may be viewed, sorted, and used…" ("Montgomery County Department of Technology Services", 2022, p. 5). These datasets were curated from the "…31 Executive Branch departments, offices and agencies, which collectively employ more than 600 applications, databases, and spreadsheets…" containing Montgomery County data ("Montgomery County Department of Technology Services", 2022, p. 13). Every dataset that has been released by the county has undergone a comprehensive evaluation process in which any data containing "personal identifying information, private information such as HIPPA, medical and other factors such as sensitive or confidential data" will never be published or has been redacted in ways that do not contain any unique identifiable information to link back to a specific associated individual (p. 57). Although the county will not be made aware of the results of this project, the research conducted may be of importance to the county, as a local governments first and foremost priority is to the safety of its citizens. These findings are multi-faceted in that they could be used by the county to potentially

decrease accident severity and also educate the public about the safety of the roadways that they frequently maneuver.

Nevertheless, there are intrinsic challenges associated with analyzing automobile accidents – mainly the fact that they are accidents, and to some extent unpredictable. This project does not aim to predict the precise time and space and accident will occur, as that task is practically impossible. Instead the project focuses only on analyzing the outcomes of accidents after they occur. While this significantly narrows the focus of project into something more predictable, the leading causes of the accidents may not be captured in any given data, thus swaying model calculations in a less ideal direction. In reality, every accident is its own unique circumstance that cannot be quantified by a limited number of standardized data points. For instance, the health condition, mood, amount of sleep, hunger levels, or a startling sound could all directly contribute to an accidents outcome but be overlooked or unavailable when performing data collection. Therefore, the results obtained from this project should not be regarded as definitive when predicting accident severity, but rather as a limited perspective into a wider image.

Dataset

There are three main datasets that were used as the foundation for the features and target variables selected in this project. All three datasets contain crash report information on accidents that have occurred within Montgomery County, Maryland from 2015 through the end of 2023. Each datapoint was collected and entered into a shared database utilized by multiple police departments within Montgomery County. The main dataset that was utilized is the drivers dataset which contains more than 170,000 observations and 43 features, in which each observation

represents a single driver who was involved in an accident. The remaining two datasets were merged into the drivers dataset based on the accidents corresponding report number. The incident dataset contains more than 96,000 observations and 44 features, in which each observation represents a single accident involving one or more drivers. The non-motorist dataset contains roughly 5600 observations and 32 features, in which each observation represents a living entity that was involved in the accident but was not operating a motor vehicle at the time of occurrence. Supplementary datasets containing geographic information about the county were also used during this project but were mostly confined to the data cleaning portion of the process.

2. Overall Analysis and Research Platform

Overall Analysis

There are four main pillars to the data analysis done for this project: data cleaning, data modeling, data visualization and data interpretation. The most time-consuming parts of the project were utilizing a plethora of techniques to clean the data in a way that was most practical, as well as generating predictive models for each of the selected target variables. The original plan was to start with data cleaning, move into visualizations and end with modeling, yet due to the anticipated time it would take to generate and fine-tune the models, data visualization was moved toward the end instead. Unfortunately, prior to modeling many of the features had to be removed due to over correlation, lack of real-word practicality, or initial cleanliness/completeness of the features. In the end, each target variable that was selected had 4-5 different machine learning methods applied to them in attempts to see which could produce the best results. Afterwards, only the important discoveries made during the cleaning and modeling

process were visualized as to stay consistent with the overall key points the project is attempting to convey.

Platforms

Python was selected as the single software component that would be used to complete all of the necessary steps for this project from end-to-end. To facilitate the creation of python code, Jupyter Notebooks would be chosen as the integrated development environment (IDE). The cell-based approach of the notebooks allows for code to be broken down into smaller more manageable chunks which complements iterative development methods and easy to read documentation all in one singular space. Outputs are also stored directly in the notebooks as plain text which is perfect for being able to run computations once and referencing them multiple times later.

Base python does not support all of the data science techniques that needed to be applied to this project, thus supporting packages were installed. In terms of cleaning the data, the main libraries used were pandas, geopandas, shapely, numpy and fuzzywuzzy. Of these libraries listed, the only two that were covered in course content at Loyola include pandas and numpy. Nonetheless, this project would not have been possible without the utilization of the other three packages. Overall, pandas and numpy were both used as a means of storing and manipulating the dataset. With a portion of the data being geospatial, geopandas and shapely were needed to directly work with and filter the coordinate features. In fact, not only were geopandas and shapely more accurate and efficient in removing outliers from the dataset than attempted clustering techniques, but they also helped bypass the out of memory exceptions that were thrown when performing the clustering approach. On the other hand, fuzzywuzzy was integral in

recategorizing the nominal features whose values tended to have many spelling mistakes scattered about.

For the modeling portion of the project, the main libraries that were used include scipy, scikit-learn and xgboost. Scipy was used pre-modeling to determine normality of the target variables and to calculate the Cramer's V statistic between all of the selected features. This allowed for features with strong correlations to the target variables to be selected, and features depicting multicollinearity with other features to be removed. Scikit-learn and xgboost provided all of the machine learning models that were used to predict the target variables. Originally, a pytorch implementation was attempted to model a neural network, however it was quickly scratched from the project altogether. Along these lines, the laptop that was being used for the computations does include a cuda compatible nvidia graphical processing unit (GPU). Cuda and cudNN drivers were installed onto the computer and were recognized by the systems command line interface, however the data scientist could never successfully get the drivers to load within the selected IDE. Without these drivers, the speed in which the neural network model could be calculated was significantly hindered resulting in run times upwards of 16 hours when performing hyperparameter tunings. Ultimately, while not the same type of model as a neural network, xgboost was chosen as an alternative modeling type. Like a pytorch implementation, xgboost also has the luxury of being able to run on a GPU, though without the necessary drivers, computation times across a CPU for xgboost were much less complex than the neural network alternative. With other aspects of the project taking priority, troubleshooting GPU driver installation was not attempted given that this aligns more with system administration and is beyond the scope of the project. In the future, having this capability would be desirable, but for now, the models produce similar results whether computed on a CPU or GPU.

Finally, for the data visualization portion of the project, matplotlib, seaborn, plotly and geopandas were chosen. Matplotlib and seaborn were applied together to generate heatmaps, line graphs and bar plots for exploratory data analysis and feature selection. These libraries also proved instrumental for determining and exploring trends within the data and their impact on Montgomery County's public safety. Moving along, plotly and geopandas were vital in displaying datapoints directly onto a map of Montgomery County. This approach was extremely helpful in visualizing the results of outlier removal from the dataset and provided insights into how particular features interact with each other in a geographical space.

3. Related Works

A 2014 study out of the United Arab Emirates (UAE) by Mohamed Elfadil utilized multi-class support vector machines to predict the underlying causes of traffic accidents within their country. According to Elfadil (2014), the top 5 causes for automobile accidents include a lack of consideration for other drivers, over-speeding, entering an unclear roadway, running a red light, and not maintaining a safe driving distance between other vehicles (p.445). This is surprising, as none of these contributing factors were included in the Montgomery County, Maryland Crash Reports dataset. Nonetheless, the model performance of Elfadil's study reported both a 75% f1-score and accuracy score, making it an acceptable but not commendable model. Before generating the model, Elfadil decided to implement a bootstrapping method to resample the data with replacements, thus creating a more balanced class distribution for the main causes of automobile accidents.

Another research study predicted accident injury severity throughout England and Scotland. A K-means clustering technique was utilized to separate the two countries into four different

regions geographically. The main model type used was a stacked sparse autoencoder (SSAE), with supplementary model types being a deep neural network and a standard autoencoder (Ma, et al., 2021). An autoencoder is “a typical generative model used in deep learning that consists of three layers, i.e., the input layer, hidden layer and output layer” (Ma, et al., 2021, p.6). Being too simple of a model, a penalty was added, and outputs of previous hidden layers were passed in as inputs to create the enhanced SSAE model. Each of these model types were ran against each other in the four different geographical regions. In the end the SSAE model performed the best scoring a 0.79 f1 score in the weakest region and a 0.84 f1 score in the strongest. For comparison, the autoencoders lowest performance was 0.75 and best was 0.79, while the DNN’s lowest performance was 0.75 and best was 0.80 (Ma, et al. 2021).

A third study conducted out of Beijing, China decided to create a multi-task DNN model to predict the number of injured, the number killed, and the value of property lost across different traffic accidents (Yang, et al., 2022). The study’s main purpose however was to compare the selected multi-task DNN to the performance of other baseline models – which include FCM support vector machines, single-task deep neural networks, random forests, and logistic regressions. Overall, the researches remarked that the “… proposed model outperforms all baseline models on accuracy and AUC on all three-traffic accident severity prediction tasks” (Yand, et al., 2022, p.9). With that being said, the proposed model’s performance was nothing to write home about as the prediction of the injury task reported an accuracy of 47% with an AUC of 70% and for the property task prediction performance was slightly better with an accuracy of 55% and an AUC of 74% (Yand, et al., 2022).

Another study from the Universities of Connecticut and Central Florida, took a more traditional approach when predicting injury severity and vehicle damage. All of the collected

data used to build the predictive models occurred at intersection in non-rural areas. Intersections were classified into two types – signalized and sign-controlled. The modeling methods that were observed include multivariate Poisson-lognormal (MPVLN) and joint negative binomial-generalized ordered probit fractional split (NB-GOPFS) models (Wang, et al. 2021). Wang and others state that "in the MPVLN model, we assume the crash counts are correlated among all severity levels" while "in the NB-GOPFS model, the total crash counts and crash proportions by each severity are jointly estimated, by accounting for the correlations between total crashes and crash severity proportions" (2021, p.46, p.47). Overall both of the models selected performed similarly across all test cases, however the NB-GOPFS model was significantly better at predicting low severity crashes whereas the MPVLN was only slightly better at predicting high severity ones (Wang, et al., 2021).

The next study is by far the most extensive one that was done, though it only focused on single vehicle accidents and the severity of their outcomes. The features in this study were also the most similar to the ones that were collected in the Montgomery County, Maryland Crash Reports dataset. An extensive table was created depicting the class of each feature and the distribution of severity levels for each class. The modeling methods decided upon can be broken down into two types – tree-based and non-tree-based models. Of the tree-based models, decision trees, random forests, adaptive boosting, gradient boosting, and extreme gradient boosting (XGBoost) representations were constructed. Of all the studies mentioned so far, this is the first to use XGBoost - a gradient descent theory booster which connects multiple base estimators and integrates "…block and regularization idea[s] to enhance the accuracy and speed of the model…" and thwart overfitting (Yan, et al., 2021, p.4). As for the non-tree-based models, quadratic discriminant analysis, support vector machines, k-nearest neighbors, Bernoulli naïve

bayes, and multi-layer perceptron representations were all constructed. Of all the studies mentioned so far, this is also the first to use Bernoulli naïve bayes whose "… basic idea… is to apply Bayes' theorem with the "naïve" assumption of conditional independence between every pair of features given the value of the class variable" (Zhang and Su, 2008, as cited in Yan, et al., 2021, p.5). Lastly, this is also the first of the studies mentioned so far to utilize a grid search technique for hyperparameter tuning.

The last of the conducted studies, done by Xu, et al. (2013), focuses on the real-time aspect of predicting injury and property damage crashes, with a heavy emphasis on severe injuries. The data collected was obtained from 119 loop detectors placed along a large freeway segment in San Francisco for 794 automobile crashes in the year 2008 (Xu, et al, 2013). The SAS software was used to generate binary sequential logit models on a 20-fold cross validation. The results of the model were measured via ROC curves all of which performed between 0.75 and 0.80 for the selected target variables. The study reporters stated that if such a model were to be implemented to generate real time preemptive accident alerts the false-positive rate would need to be further minimized, and a generalized model could not be deployed in areas where the model was not trained (Xu, et al, 2013).

Up until this point, all of the studies mentioned have created their own models to answer questions surrounding accident severity. The last related work found is instead a research study which collected and analyzed a plethora of different machine learning studies surrounding the prediction of automobile injury severities. In total there were "… 56 studies analyzed, from 2001 to 2021, [and] more than 20 different machine learning techniques were applied" (Santos, et al., 2022, p.257). The results the researchers found regarding model types best suited for predicting injury severity are quite similar to the results found within this specific project. Of the studies

collected, decision trees were utilized 26 times, random forests 23 times, logistic regressions 19 times, support vector machines 17 times, multi-layer perceptron models 9 times ans Bayesian networks 3 times. Of these models, the percentage of times the algorithm was determined to achieve the best performance ranks with Random Forest leading at 69.5% of the time, Bayesian networks with 66.6%, support vector machines with 52.9%, multi-layer perceptron with 33.3%, decision trees with 30.7% and logistic regression with 10.5%. Thus, with the random forest being utilized the second most amount of times and having been the model with the best performance 16 (or 69.5%) of those times, it by far outperforms the other models selected for predicting injury severity in accidents (Santos, et al., 2022).

4. Data Analysis

Research Question

The overall research question that this project intends to answer is: Once an automobile accident occurs within Montgomery County, Maryland, can the outcome/severity of that accident be predicted based on contributing factors?

The criteria for assessing accident severity will be broken down into three separate classes – the level to which the drivers involved in the accident were injured, the extent to which the vehicles involved in the accident were damaged, and the number of individual drivers involved in the crash report identifier. Success will be determined by how well each of these classes can be predicted, especially relating to the most severe outcomes for each class.

Hypotheses

In relation to the research question, hypotheses were formed for each of the three classes that will be predicted. These hypotheses are as follows:

Driver Injury Severity – Within Montgomery County, Maryland, drivers who are not injured or are minorly injured will be easily predictable while drivers who are severely injured or experience fatality will be harder to predict.

Vehicle Damage – Within Montgomery County, Maryland, vehicle damage across all categories will be easily predictable as it relates to the accidents contributing factors.

Number of Driver's Involved – Within Montgomery County, Maryland, single or double vehicle accidents will be easily predictable while accidents involving more than two vehicles will be harder to predict.

Data Loading

The main Montgomery County, Maryland Crash Report dataset utilized for this project was obtained in three separate CSV (comma-separated values) files – drivers, incidents, and non-motorists. The drivers CSV file was selected to be the primary data source with incidents and non-motorists as the secondary. Each of the three datasets were loaded into their respective pandas dataframes. Relevant features were selected and aggregated from the incidents and non-motorist dataframes before being merged with the drivers dataframe on the report number assigned to the accident. Given that the data has geographic components, shapely was used to create a geometry between the latitude and longitude features, and the master dataframe was converted into a geopandas geo-dataframe. A handful of other datasets were obtained as

GeoJSON (Geographic JavaScript Object Notation) files and loaded throughout the project to aid with data cleaning, exploratory analysis, and feature engineering.

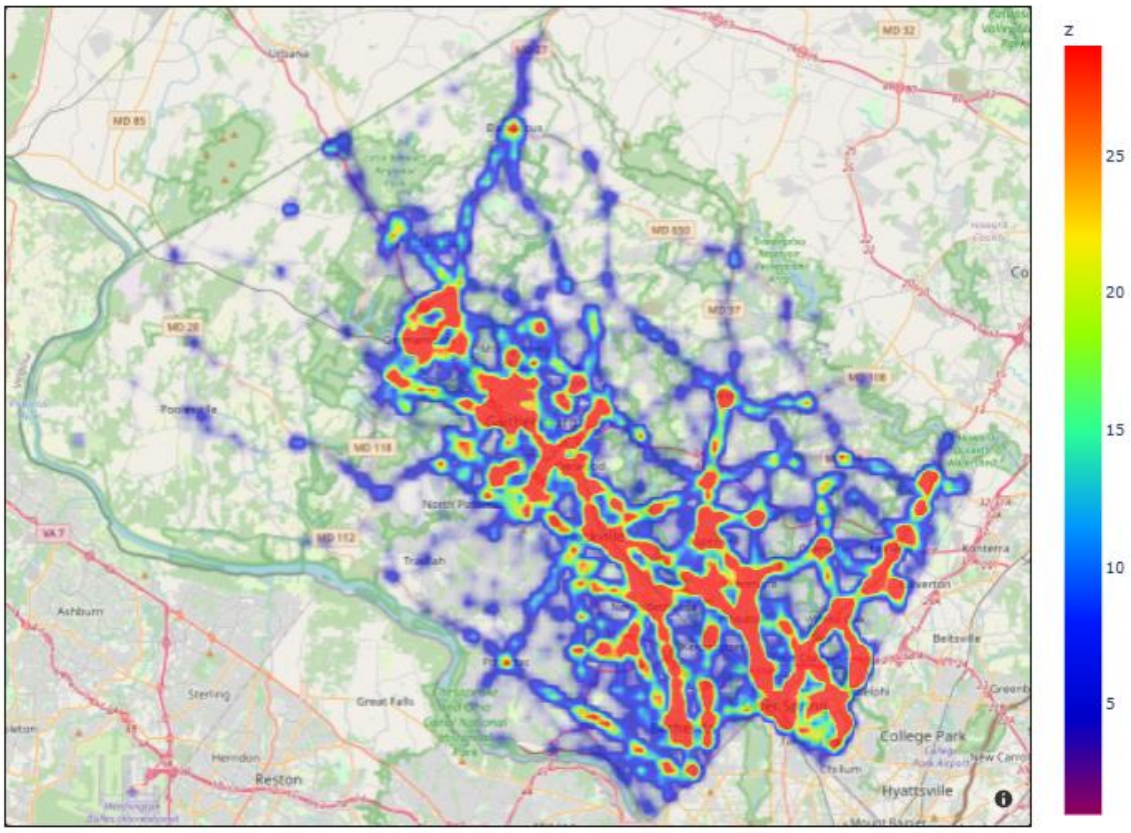
Exploratory Data Analysis and Data Cleaning

Outlier Removal

In its natural states, the Montgomery County, Maryland Crash Report dataset contained a small amount of accident information for incidents occurring outside of the county's borders. Initially, clustering techniques were attempted to remove geographic outliers, however those techniques struggled greatly with memory constraints and precision, as some data points within county lines were often removed accidentally. To combat this, a supplementary dataset containing the coordinates for Montgomery County's borders was obtained and a Shapely unary union was performed to remove any outliers with the level of precision needed. Only accidents that occurred within Montgomery County, Maryland remained. A heat map representation of remaining datapoints can be found in figure 1. Each datapoint in this heat map signifies an individual driver with areas in red depicting a higher frequency of driver accidents and areas in blue depicting a lower frequency. Given the aerial view of the heat map, the density of drivers may seem greater than if it were to be zoomed in further.

Figure 1

Per Driver Density Map - Montgomery County, Maryland



Missing Data Analysis

In order to select the features that will be used when predicting the target variables, a visual analysis needed to be performed to better understand which features would not be worth imputing. A horizontal bar chart representing the missing data per feature can be found in figure 2, while a histogram representing the missing data per driver observation can be found in figure 3. Before creating each of these graphs, missing data labels across all features first had to be standardized. While empty values already existed in the dataset, there were also non-missing values that should be treated as missing and were thus converted to be so. In a numeric context,

this consisted of zero values, while in a categorical context, this consisted of values such as ‘unknown’, ‘n/a’, ‘no name’, etc.

Figure 2

Missing Data Per Feature

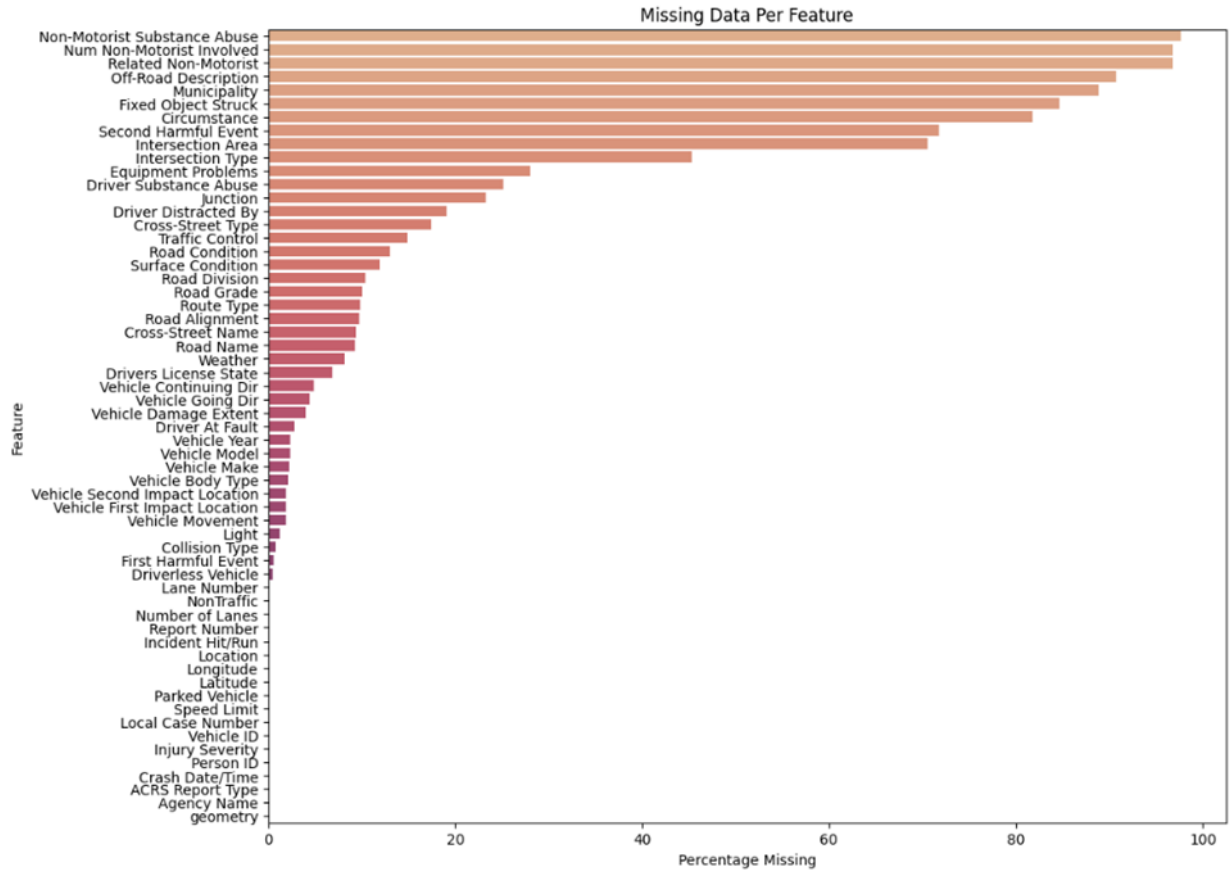
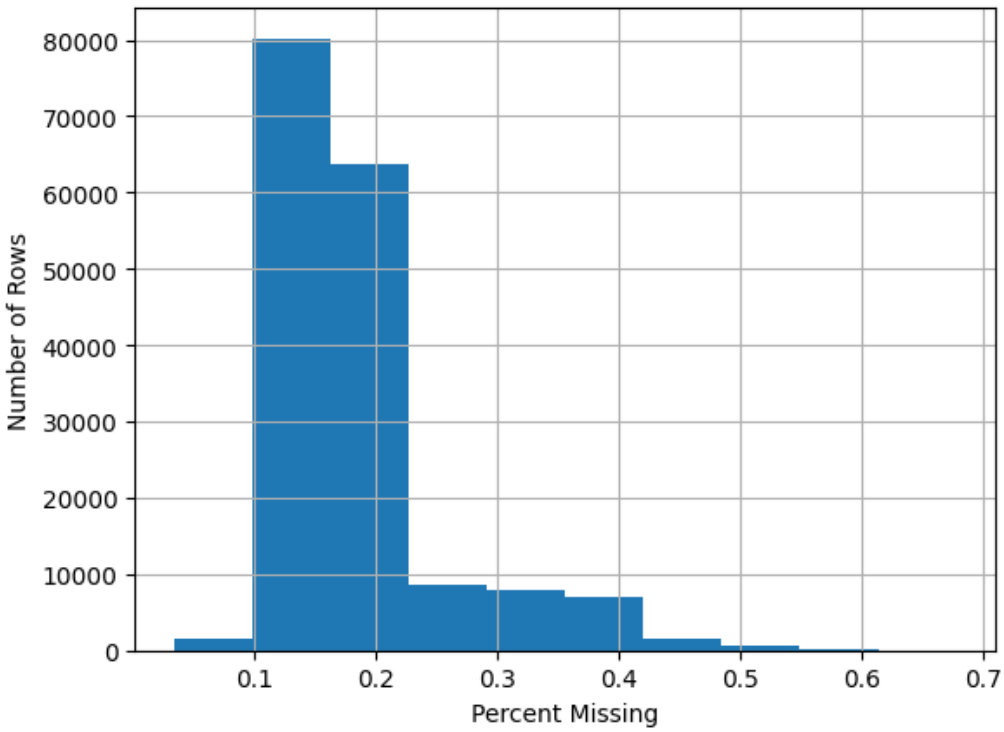


Figure 3

Missing Data Per Driver Observation



Target Variable Creation and Remapping

Two of the three target variables selected were native to the dataset, while the other one needed to be engineered based off of the dataset. In regard to driver injury severity, the injuries sustained were not nearly as severe as was to be expected. Of all observations, 82% of drivers were not injured while less than 0.01% of driver's were fatally injured, making this target variable appear extremely unbalanced. To slightly combat this, the target was remapped into 3 categories: 'no apparent injury' with 82% of the observations, 'possible or minor injury' with 17% and 'serious or fatal injury' making up the remaining 1%. To stay true to the nature of the data, and due to such a small subsample for 'serious or fatal injuries,' no bootstrapping or resampling methods were performed.

Figure 4

Injury Severity Interactive Scatter Plot - All

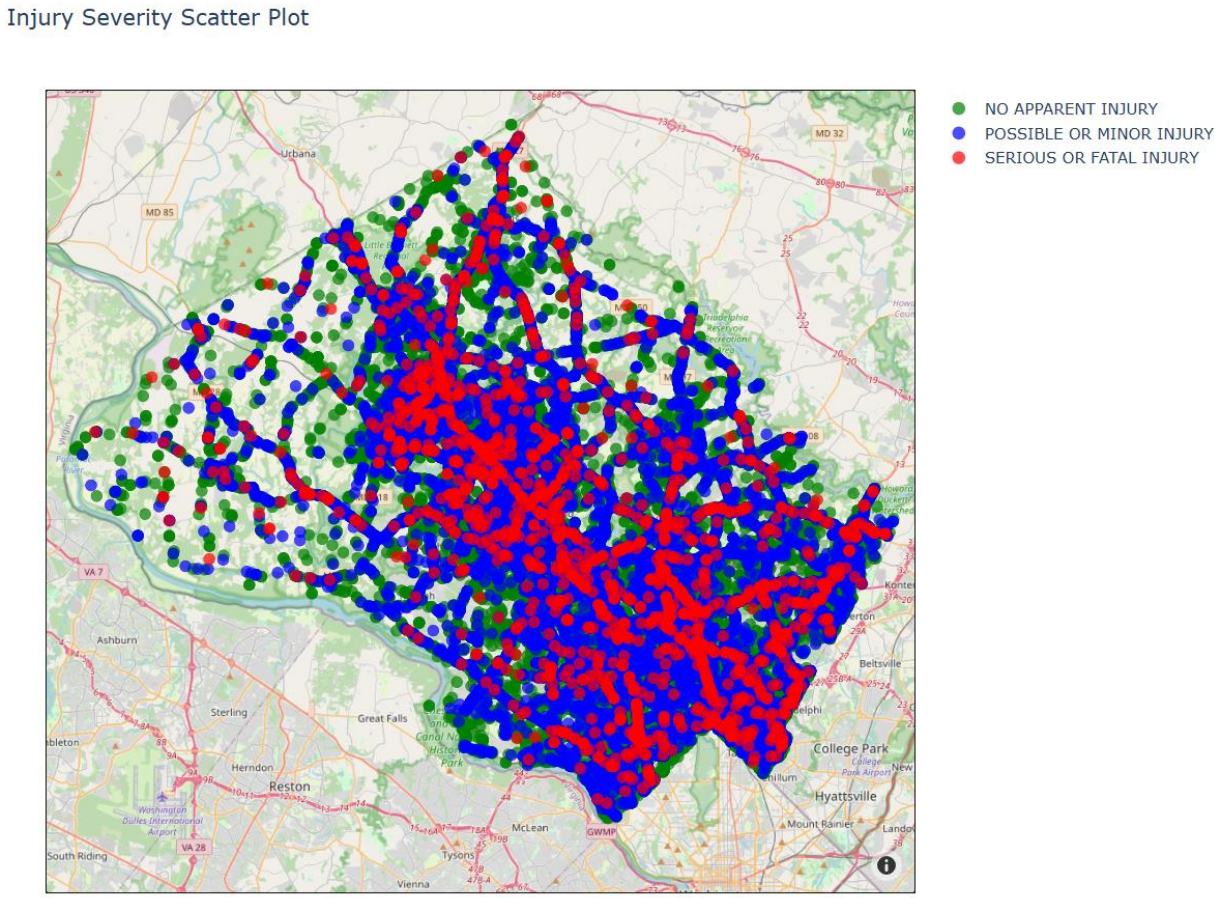


Figure 5

Injury Severity Interactive Scatter Plot - No Apparent Injury

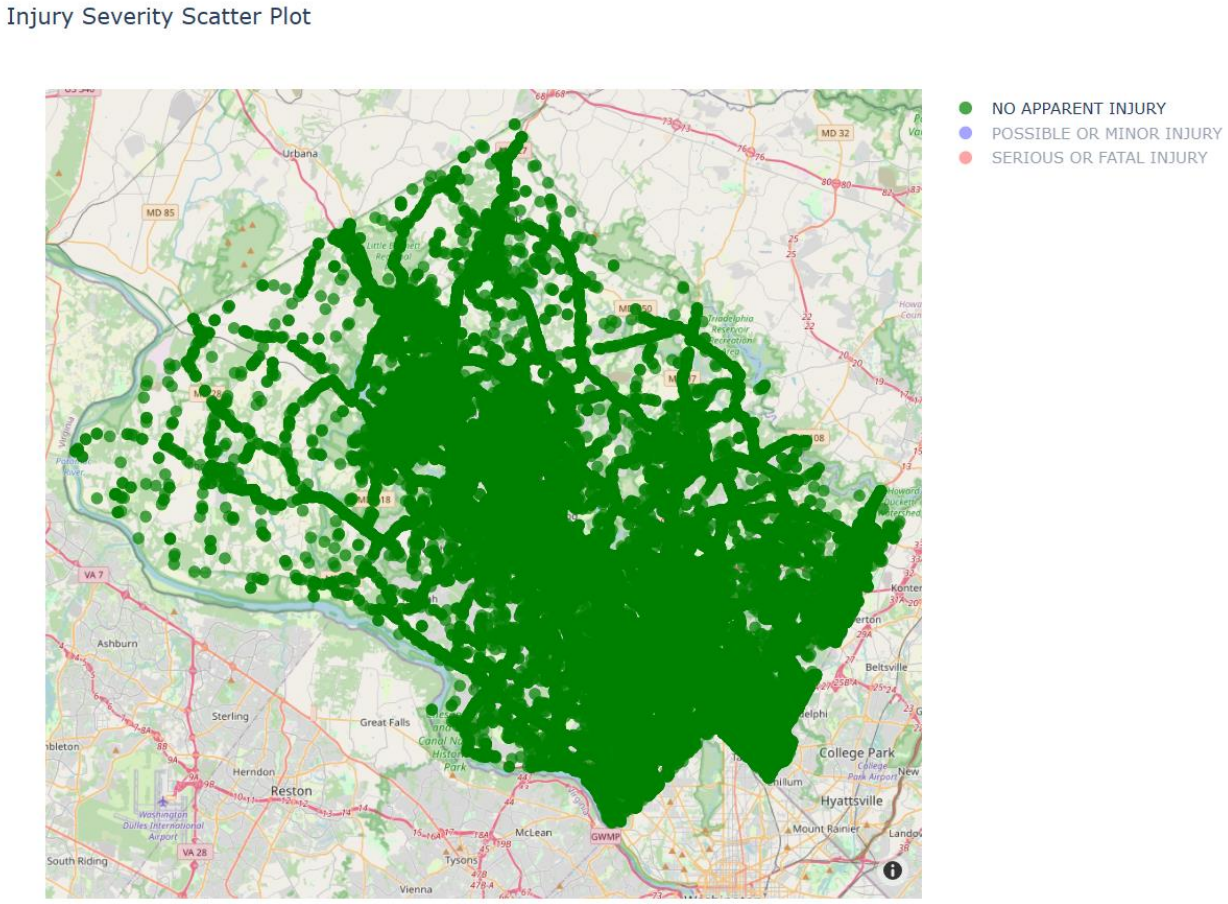


Figure 6

Injury Severity Interactive Scatter Plot – Possible or Minor Injury

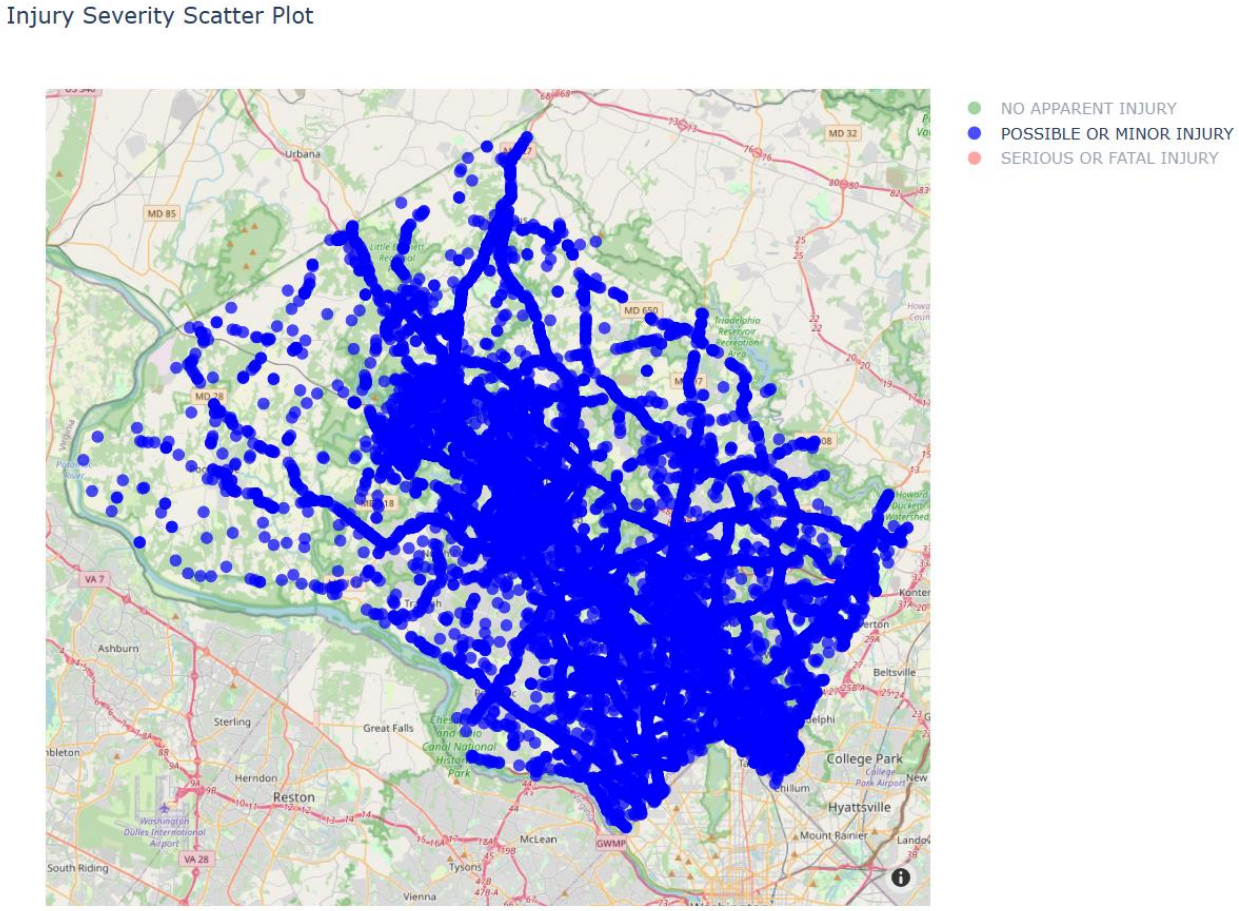
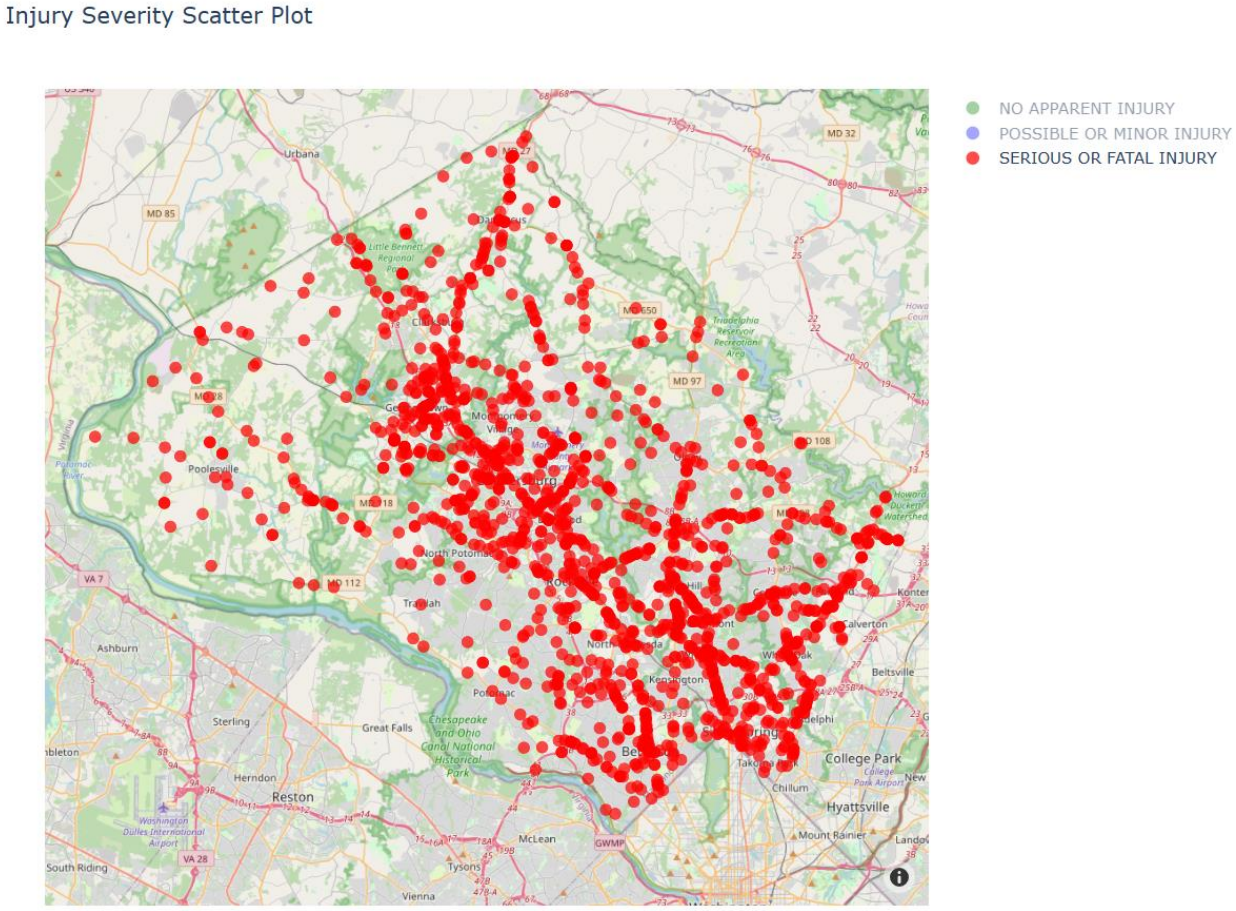


Figure 7

Injury Severity Interactive Scatter Plot – Serious or Fatal Injury



While not as extremely unbalanced, the vehicle damage target variable also struggled to illustrate a balanced distribution. To better combat this, the target was remapped into 3 categories: ‘destroyed or disabling’ with 43.5% of observations, ‘functional’ with 30% of observations and ‘no damage or superficial’ with 26% of observations. Unfortunately, once the modeling portion of the project was attempted, the performance in predicting these three categories was sub-par and variable remapping was performed once again. The ‘functional’ and ‘no damage or superficial’ categories were combined to create a new ‘drivable’ category making

up 56% of the observations and ‘destroyed or disabling’ was renamed to ‘not drivable,’ resulting in a binary distribution.

Figure 8

Vehicle Damage Interactive Scatter Plot - All

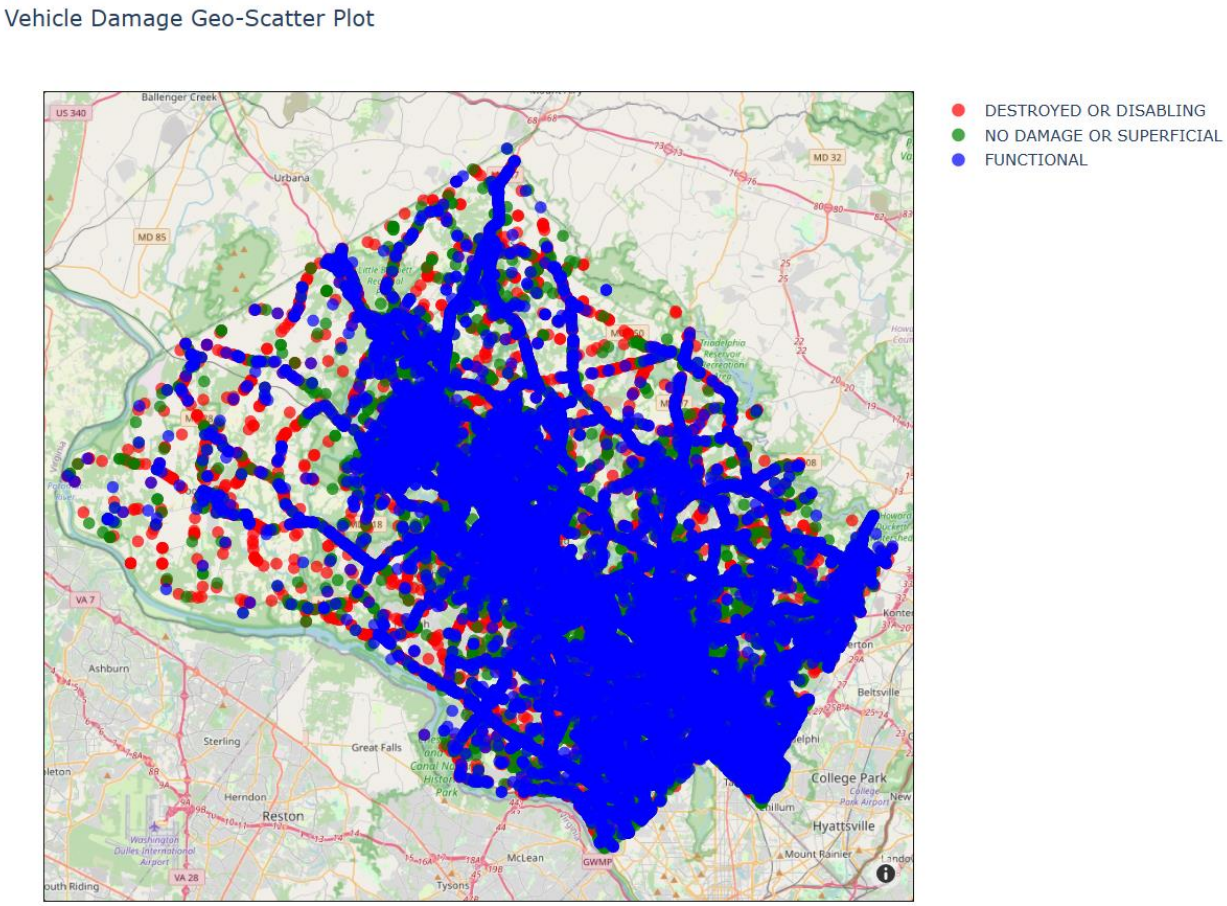


Figure 9

Vehicle Damage Interactive Scatter Plot – No Damage or Superficial

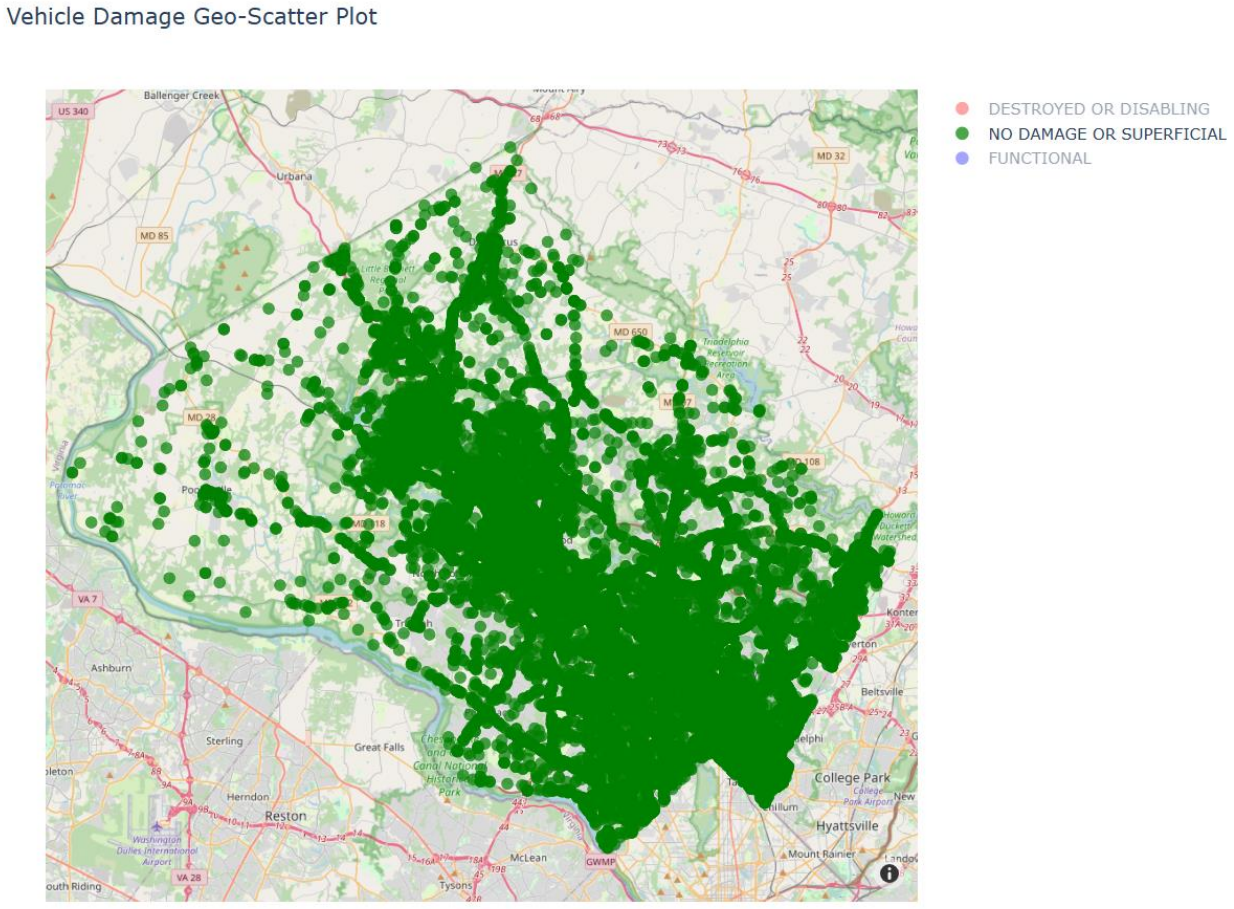


Figure 10

Vehicle Damage Interactive Scatter Plot – Functional

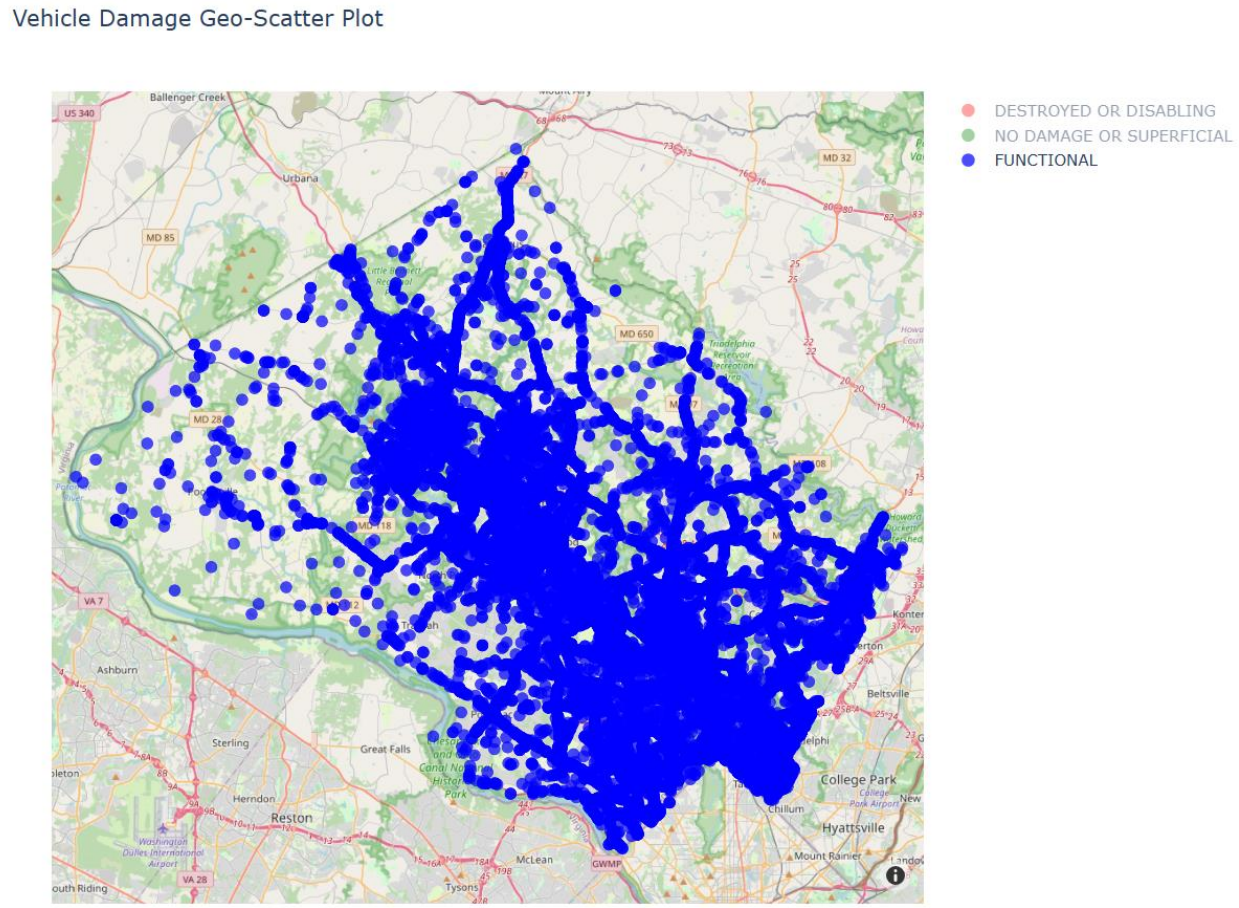
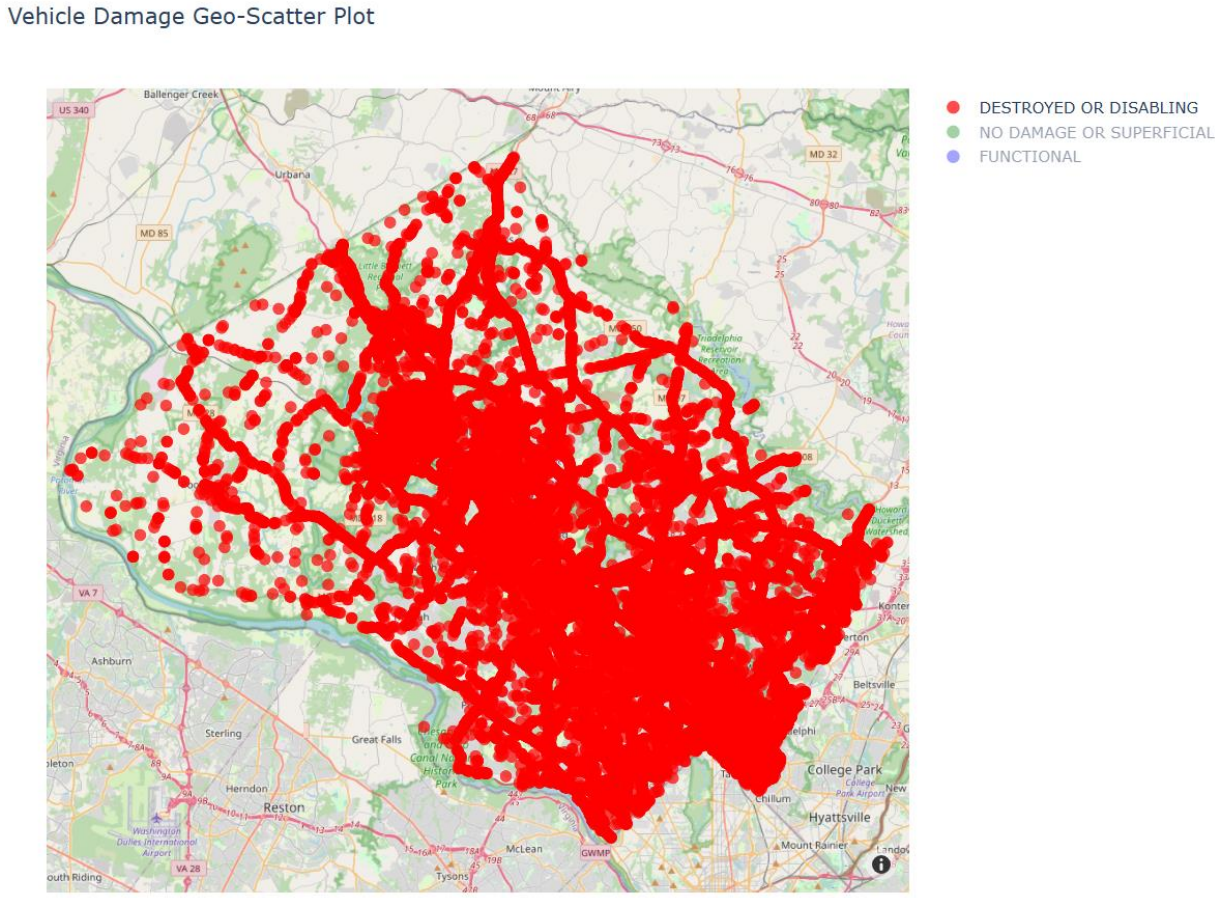


Figure 11

Vehicle Damage Interactive Scatter Plot – Destroyed or Disabling



The number of motorist involved target variable was created from the driver's dataset by aggregating the number of times each report number occurred and then merging that number back onto the main dataset for the corresponding rows. Since each row represents a specific driver, and each report number represents a specific accident, the number that resulted indicates the total sum of drivers involved in that particular accident. Of all the observations, 18% included a single driver, 67.5% included two drivers, 11.4% included three drivers and 3.2%

included four or more drivers. While this target variable was also unbalanced, in order to stay true to the nature of the data, no bootstrapping or resampling methods were performed.

Figure 12

Number of Drivers Involved Interactive Scatter Plot - All

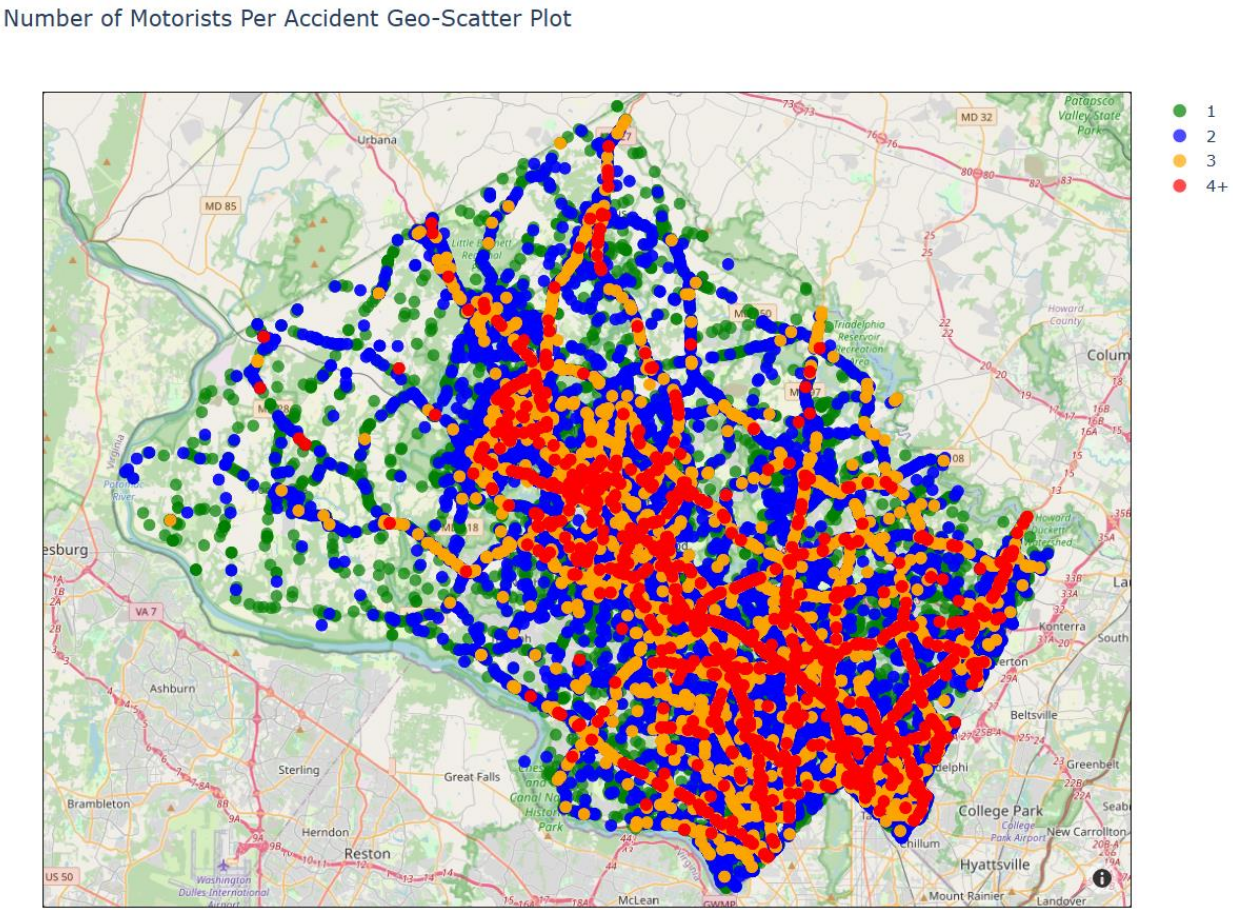


Figure 13

Number of Drivers Involved Interactive Scatter Plot – 1 Driver Involved

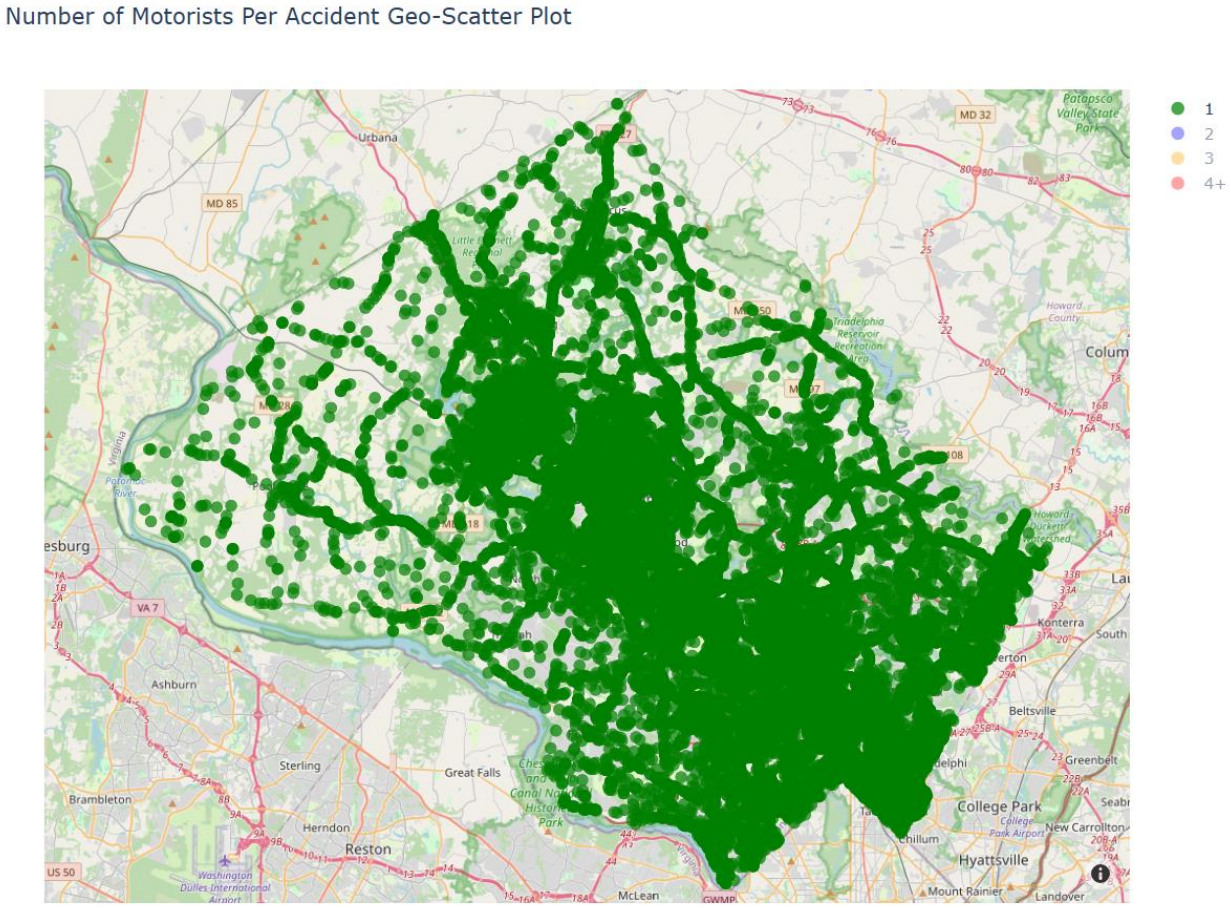


Figure 14

Number of Drivers Involved Interactive Scatter Plot – 2 Drivers Involved

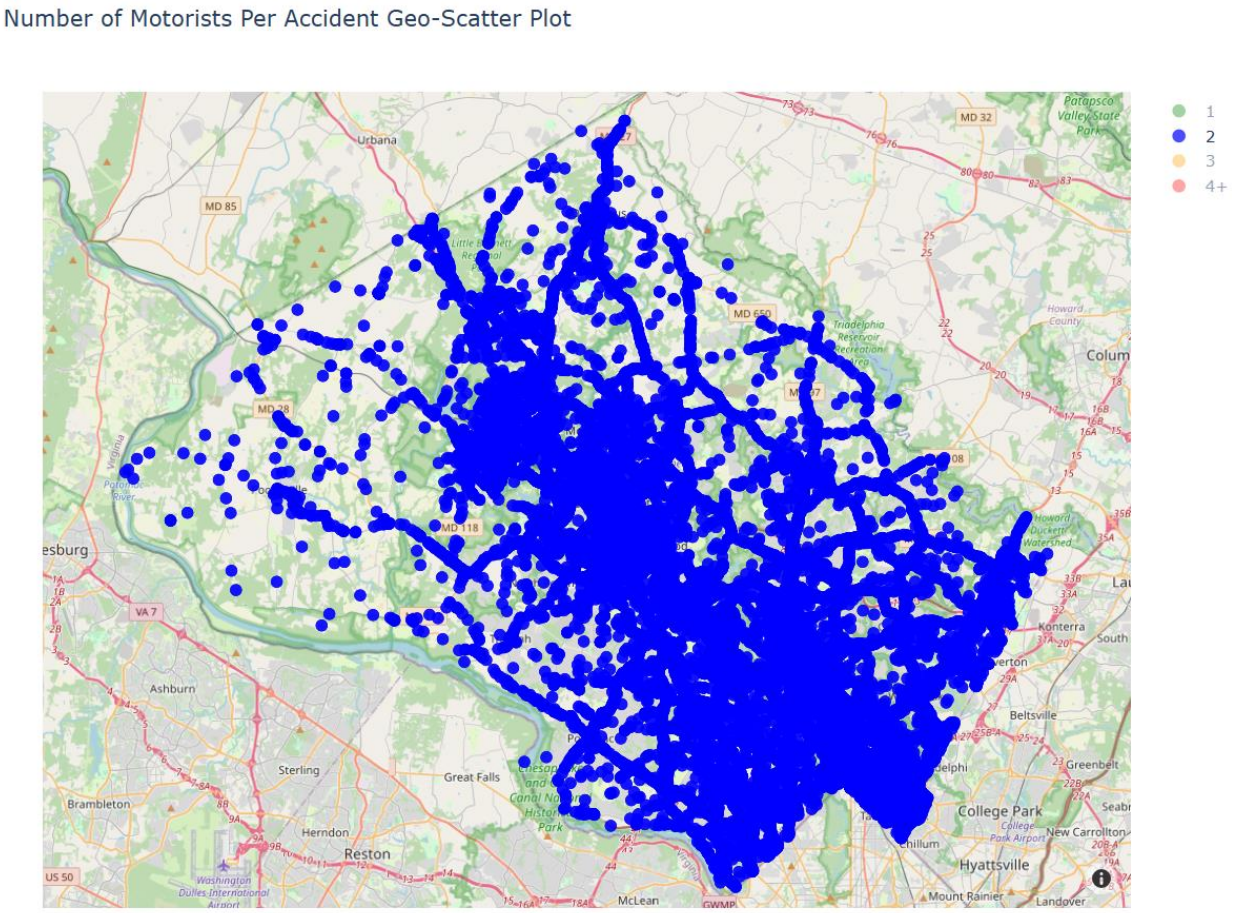


Figure 15

Number of Drivers Involved Interactive Scatter Plot – 3 Drivers Involved

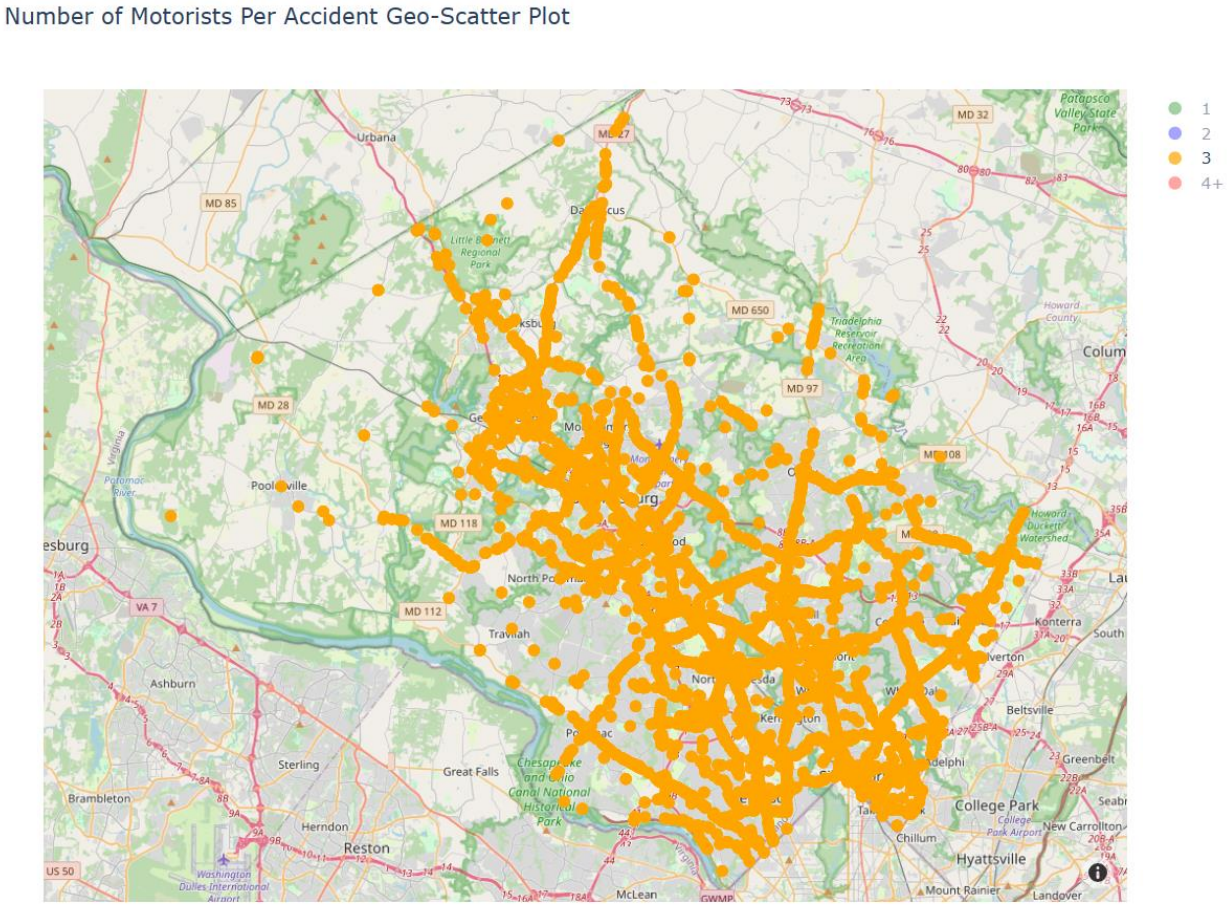
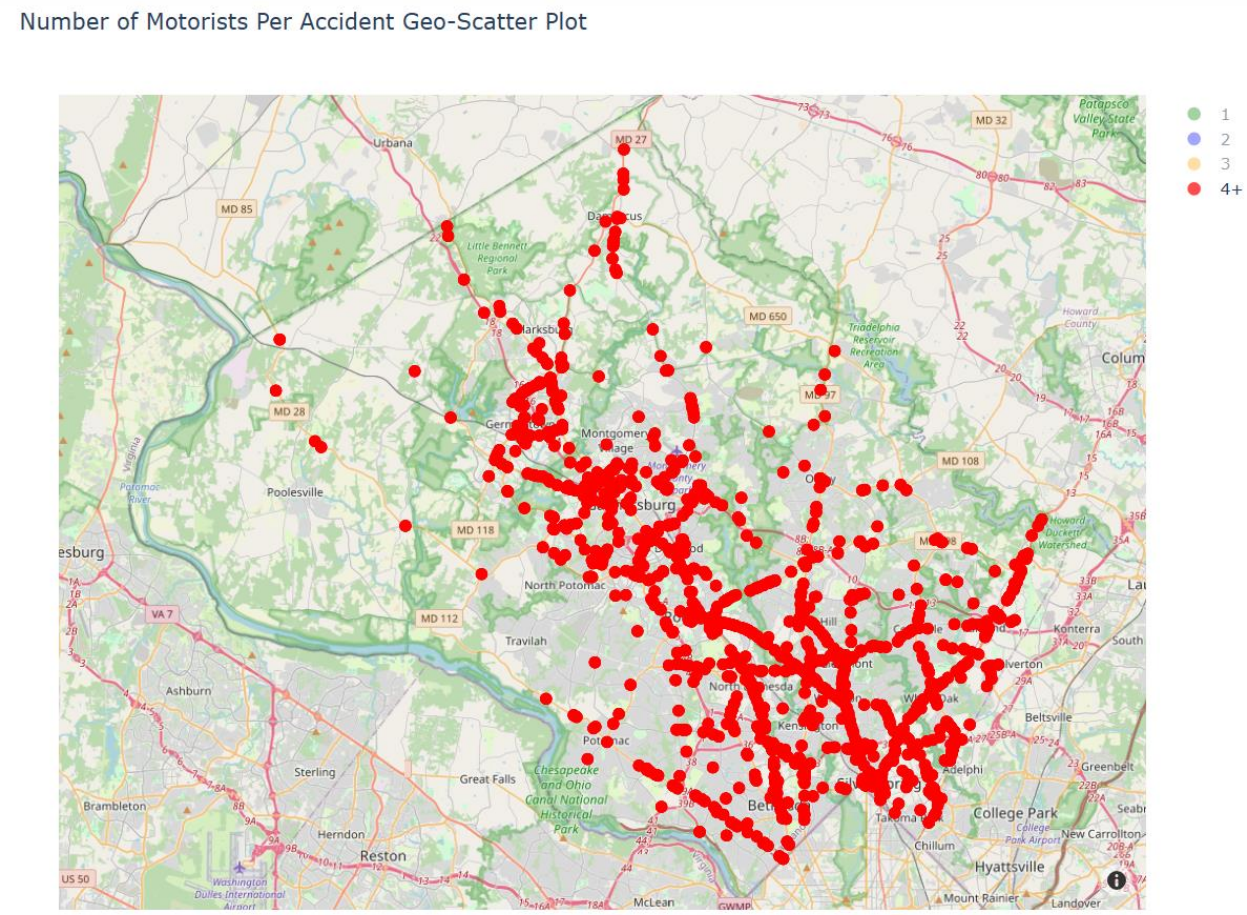


Figure 16

Number of Drivers Involved Interactive Scatter Plot – 4+ Drivers Involved



Feature Selection

Alongside examining the amount of missing data a feature has, how the feature interacts with the variable that is going to be predicted is also a heavy consideration. Thus, a statistical measure needs to be chosen to interpret the association between a feature and the target variable. Given that a large majority of features in the Montgomery County, Maryland Crash report dataset are categorical, the Cramer's V statistic was chosen. To generate the Cramer's V statistic, pandas was used to calculate a contingency table between each feature and the selected target variable. Next, the scipy library was loaded to calculate the chi-squared statistic based on the previously

generated contingency table. Finally the Cramer's V formula was applied to normalize the chi-squared values such that they would exist between 0 (representing absolutely no relationship between the two variables) and 1 (representing a perfect relationship between the two variables).

Three horizontal bar charts were created to visually display the Cramer's V statistic as it relates to each target variable – injury severity in orange, vehicle damage in green, and number of drivers involved in black (see figures 17-19). Unfortunately, in all three cases, none of the features that scored above a 0.8 were selected. Majority of these features contained identifiable markers that were too highly associated with individual datapoints, which would directly contribute to substantial model overfitting if they were used. Other features within this range had too much missing data to be considered worthwhile for use. After further analysis of the Cramer's V values and amount of missing data per feature, the following 17 unique features were selected between the three models:

Collision type, driver at fault, driver substance abuse, first harmful event, lane number, light, nontraffic, number of lanes, parked vehicle, route type, speed limit, vehicle body type, vehicle first impact location, vehicle second impact location, vehicle make, and vehicle movement.

Figure 17

Cramer's V Statistic for Injury Severity Remapped Target Variable

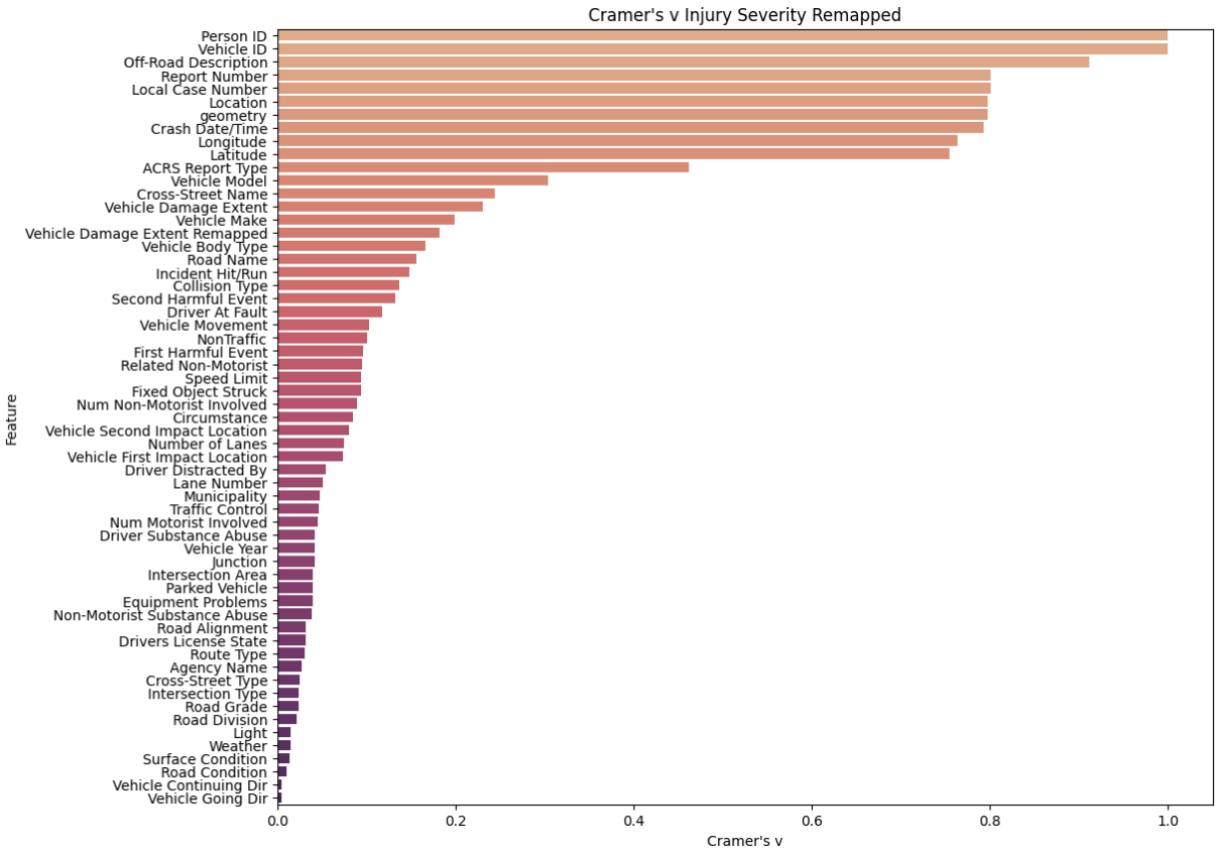


Figure 18

Cramer's V Statistic for Vehicle Damage Remapped Target Variable

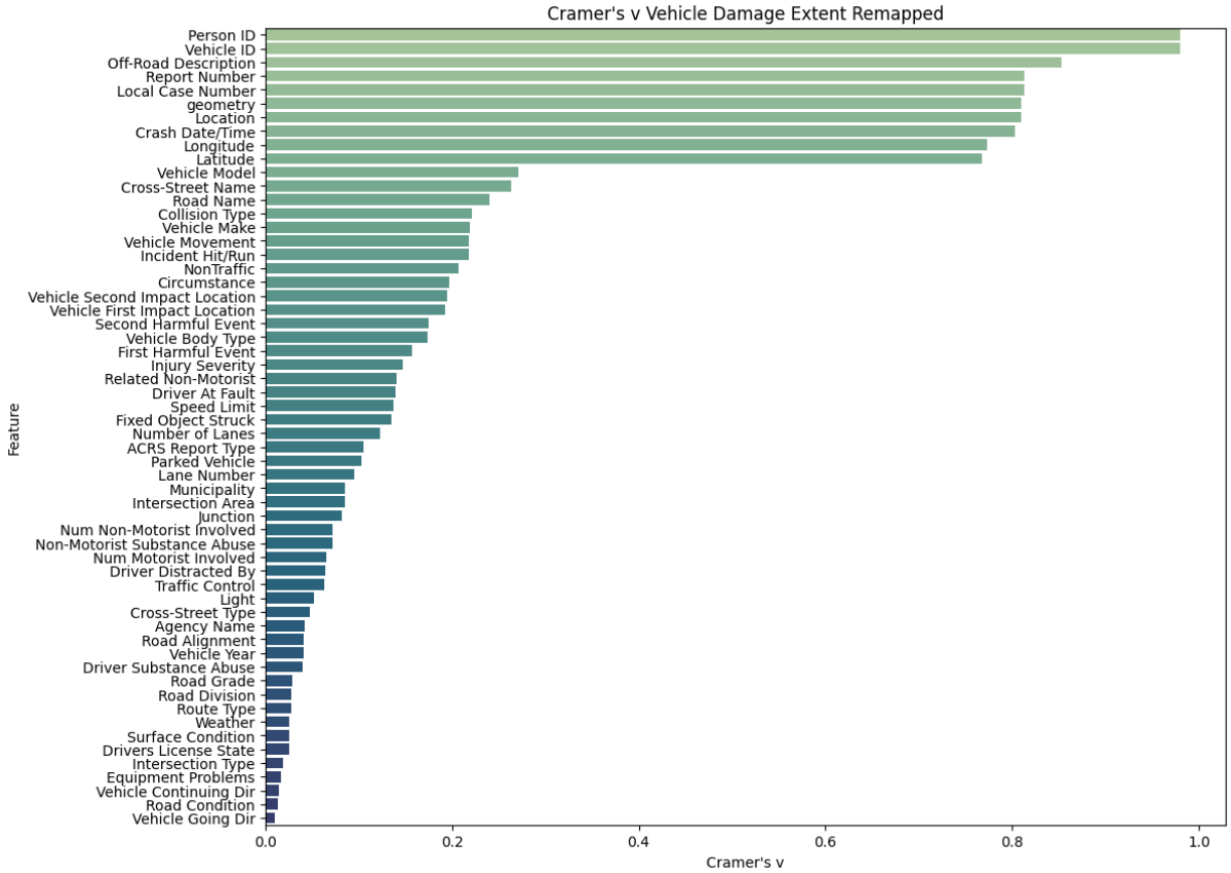
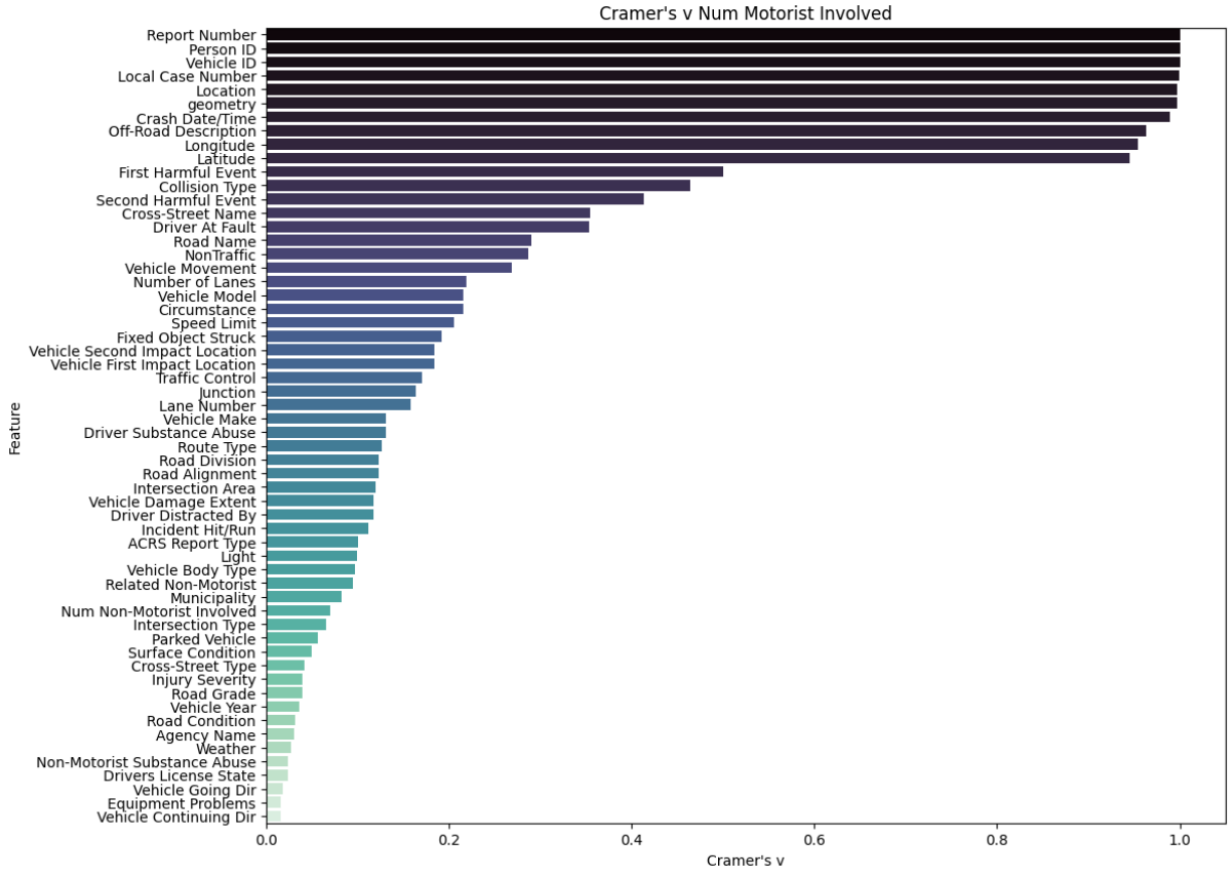


Figure 19

Cramer's V Statistic for Number of Drivers Involved Target Variable

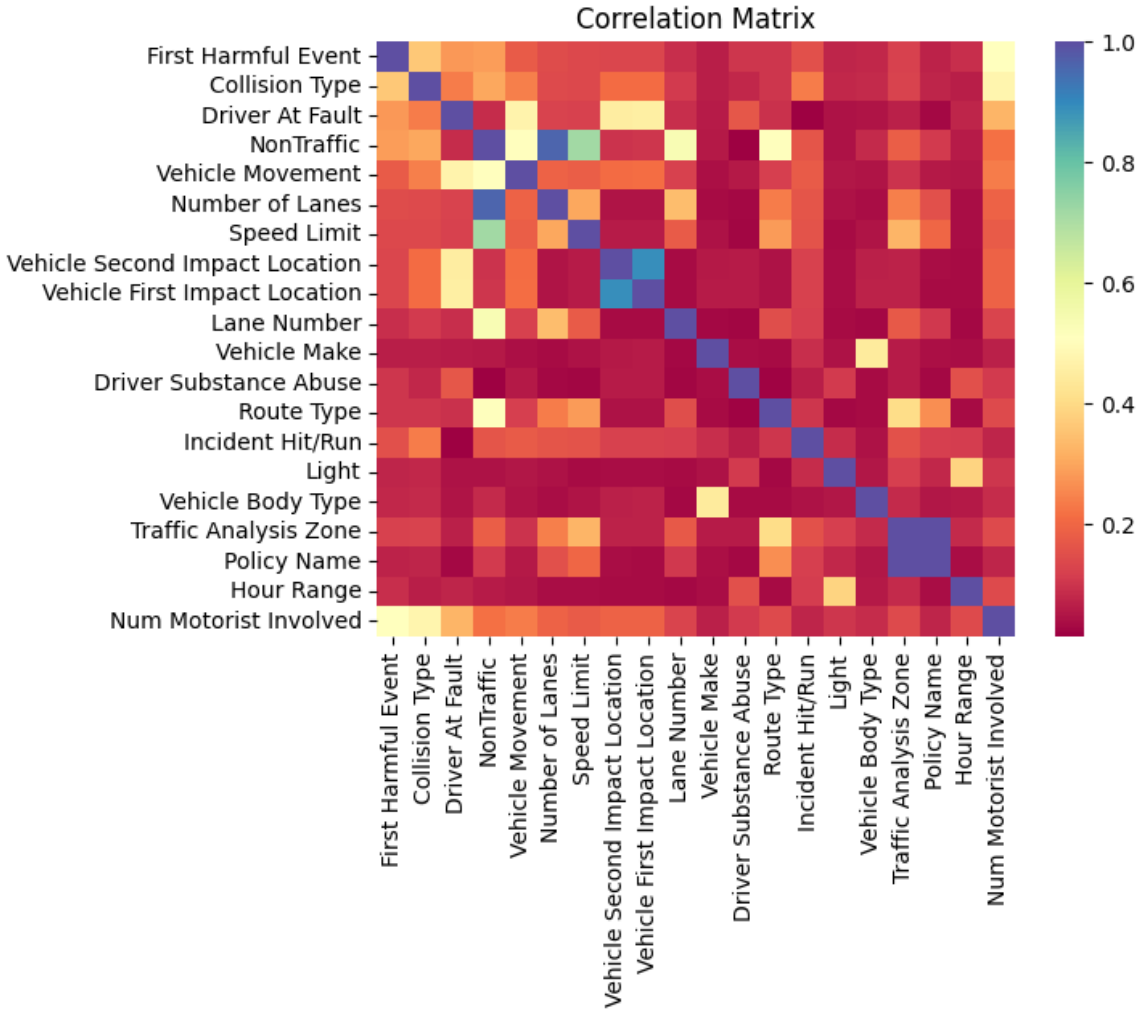


After initial feature selection for each target variable, three Cramer's V correlation matrices were created to assess for multicollinearity. While each matrix utilized slightly different features, the same three features were removed across the board for each target variable. Looking at figure 20, high correlations can be seen between number of lanes and nontraffic, vehicle first impact location and vehicle second impact location, as well as policy name and traffic analysis zone (both features that will be engineered later). It was decided that number of lanes would be removed as nontraffic stores data less complex being a binary variable. Vehicle second impact location was removed as the nature of the variable was always in response to vehicle first impact location's data entry. Lastly, although traffic analysis zone had a better correlation with the target

variables, it was removed purely because policy name is less complex and derived from the same supplementary dataset as traffic analysis zone.

Figure 20

Correlation Matrix for the Number of Driver's Involved Target Variable



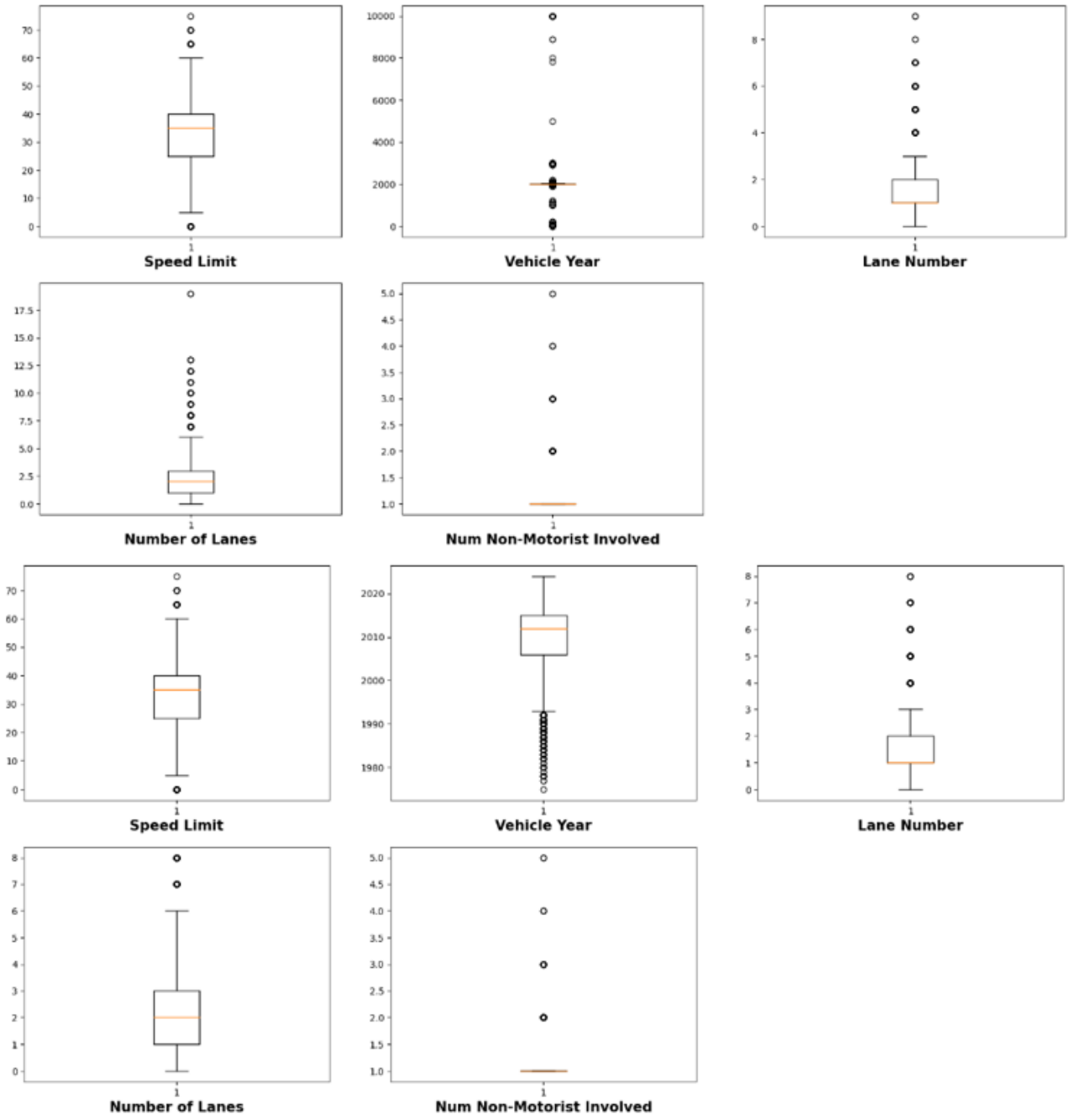
Feature Cleaning

Only the initial variables selected were cleaned to make them model ready, however, due to such a small amount of numeric data, each of these variables were cleaned/visualized as seen in figure 21. Based on these boxplots, the variables were cleaned through clipping and recategorizing some of the more outlandish values. All of the outliers for the speed limit category

appear as valid real-life limits, and thus the outliers on each end were grouped together and the variable was converted to be categorical. The vehicle's model year variable suffered greatly from data mis-entry where majority of the years often times had wrong or missing numbers. Most roadways in Maryland do not exceed 8 lanes (if counting both directions). With the lane number feature representing the main lane where the accident occurred, it would be statistically improbable for a road to have above 9 lanes and not have any accidents in the upper numbers. Thus both the lane number and number of lanes features were clipped to 8 lanes. The outliers in the number of non-motorists involved feature are valid such that any number of non-motorist getting stuck would be considered an outlier across the whole number of observations in the dataset to begin with, as only around 3% of accidents reported such a thing.

Figure 21

Numeric Boxplots (Top=Before Cleaning, Bottom=After Cleaning)

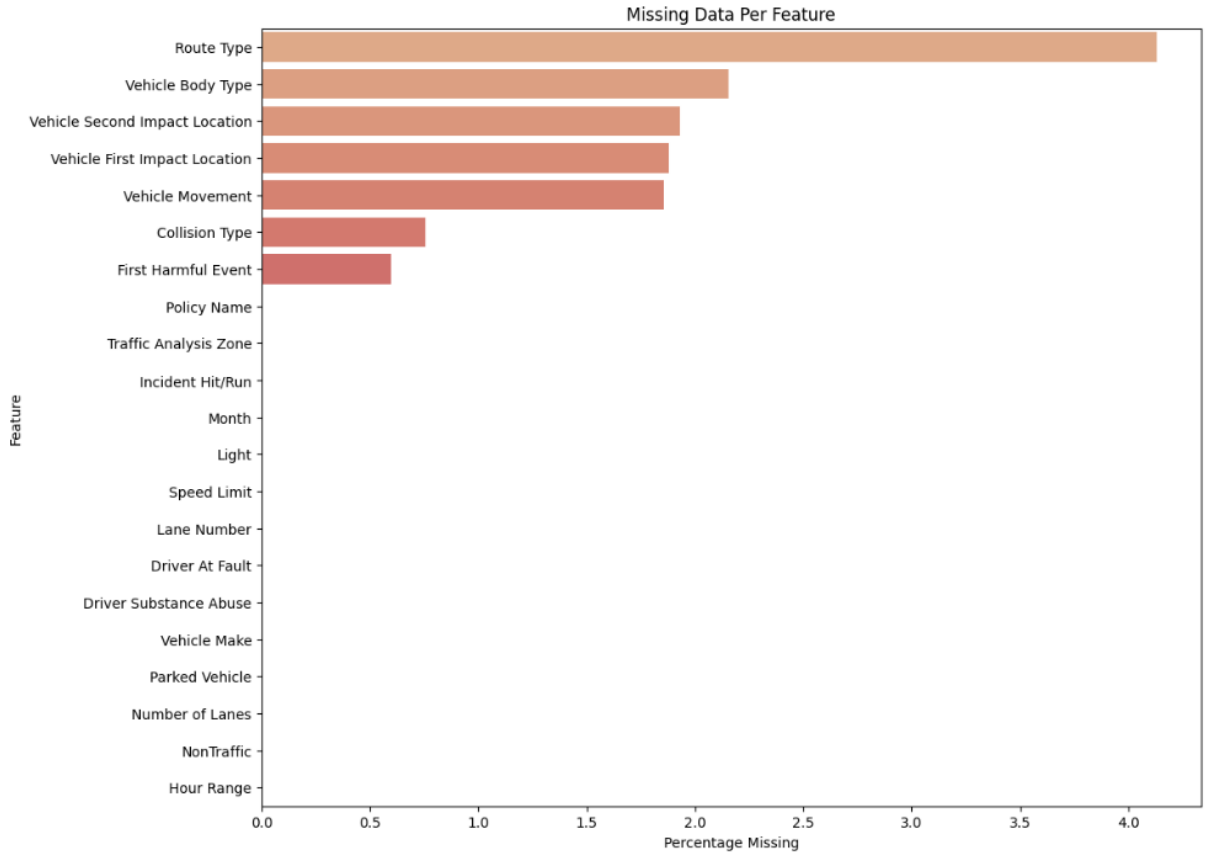


For the categorical variables, re-binning techniques were applied such that categories would be combined and renamed to decrease the variable complexity. For categorical variables that contained many misspelled words, fuzzywuzzy was utilized to conduct a fuzzy word

comparison and match the misspelled entries to properly spelled categories of higher counts. Next, to account for any missing values in the selected features, a multitude of imputing techniques were utilized. For some features, a distribution of category counts were created, and missing values were inserted randomly based on the category percentages in the generated distribution. Other features were imputed based on supplementary features in the dataset that were not chosen due to substantial amounts of missing data themselves, or lower correlations with the target variables. For the route type feature specifically, a slightly different technique was utilized. Given that the nature of the Montgomery County, Maryland Crash Reports dataset is geographic, many environmental attributes of the county are well defined such that they can be found elsewhere. Three new datasets were acquired as GeoJSON files relating to Montgomery County maintained roads, municipality-maintained roads, and Maryland (state) maintained roads. Each of these files were read into their own geo-dataframes consisting of shapely geometries which defined the roads geographically. Next, an intersection between these geometries and the crash coordinates in the drivers dataset were performed using geopandas spatial indexing to significantly speed up processing times. After all three of these techniques were applied, some of the selected variables still consisted of small amounts of missing data, as seen in figure 22. Given the large size of the driver's dataset, it was determined that before modeling any rows that still contained missing values would be dropped entirely, permitting the class distribution of the target variables did not change dramatically.

Figure 22

Missing Data Per Selected Feature After Data Cleaning

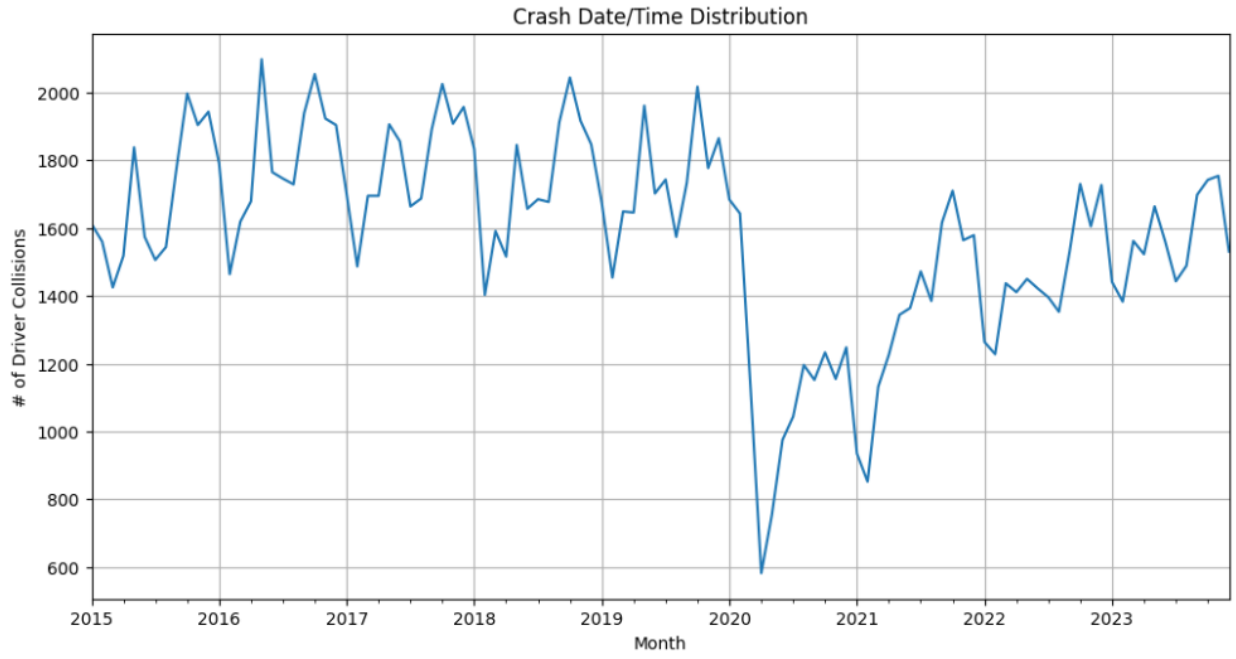


Feature Engineering

Due to the linear aspect of time, and important observation can be made in figure 23, which shows a decrease in the number of overall driver collisions starting in early 2020 and most likely due to the onset of COVID-19 and the government mandated quarantine. Since then, the overall number of collisions per month has not fully recovered to the state that it was in pre-quarantine. Aside from this, the decision was made to remove the date and time of the crash from the model due to its high potential in creating overfitting in addition to the time series component of the feature being mostly irrelevant. Figures 24 through 26 show relative consistency between target class occurrences per month since the start of the dataset, with the caveat that mostly all of the categories follow the dip and rebound caused by the COVID-19 quarantine.

Figure 23

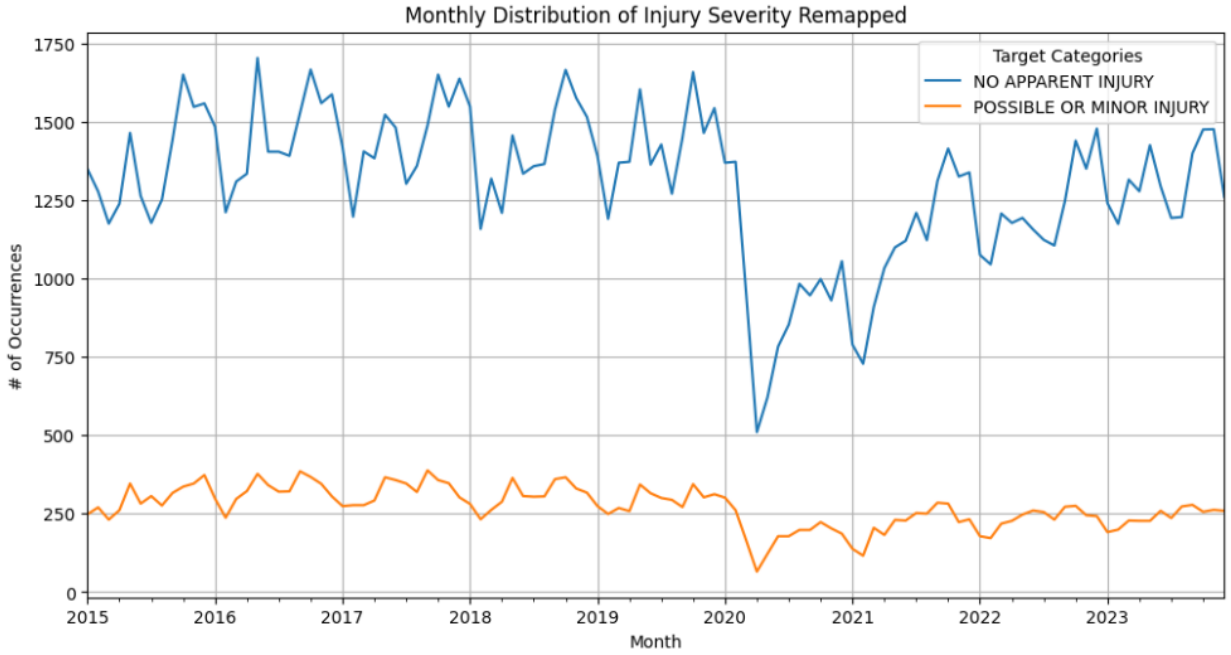
Overall Number of Driver Collisions per Month



At any rate, while the hours of the day and months of the year repeat cyclically, years increase linearly thus making it a poor model predictor. Therefore, hour range and month features were engineered based on the original crash data and time feature. The other features engineered were traffic analysis zone and policy name, each of which were generated by another GeoJSON file storing spatial information about Montgomery County. The traffic analysis zones are essentially spatial representations of traffic presumably based on number of individuals while the policy name feature can be more closely affiliated with the community and/or city names located within Montgomery County, Maryland.

Figure 24

Injury Severity Collisions Per month



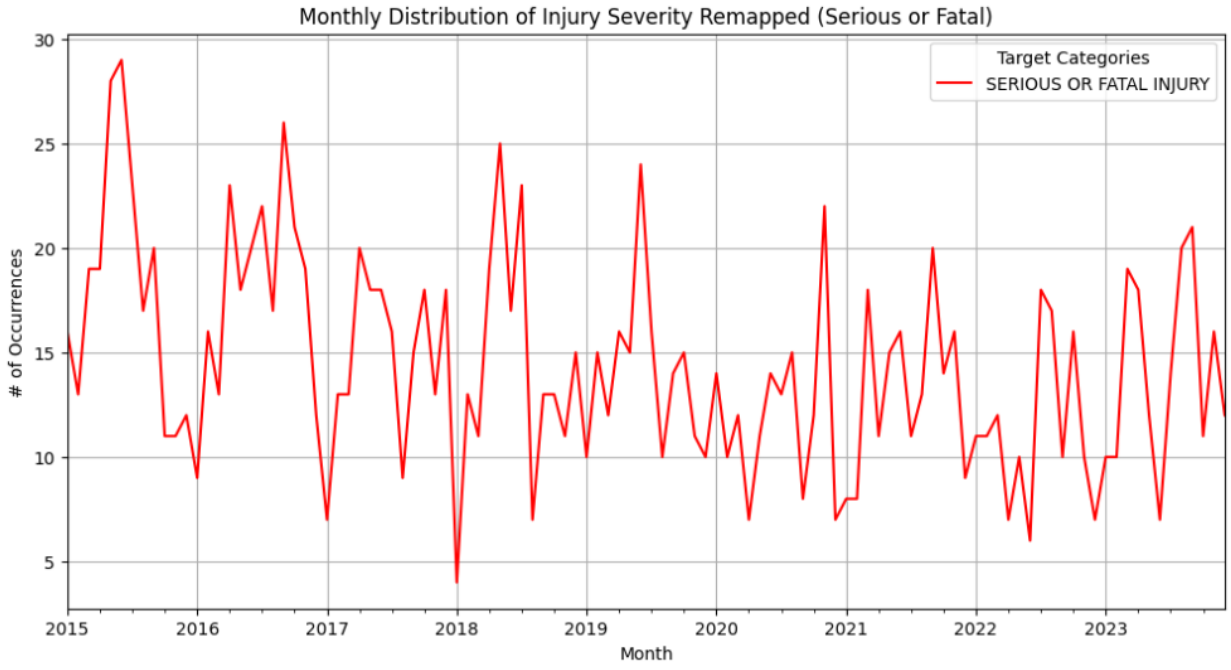


Figure 25

Vehicle Damage Collisions Per Month

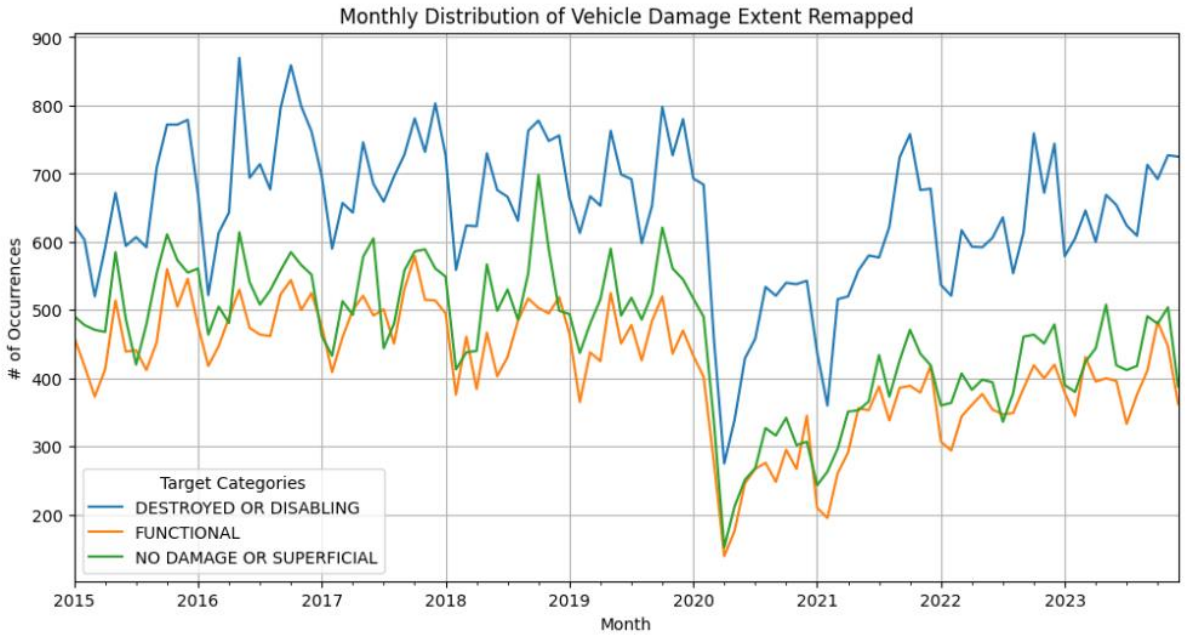
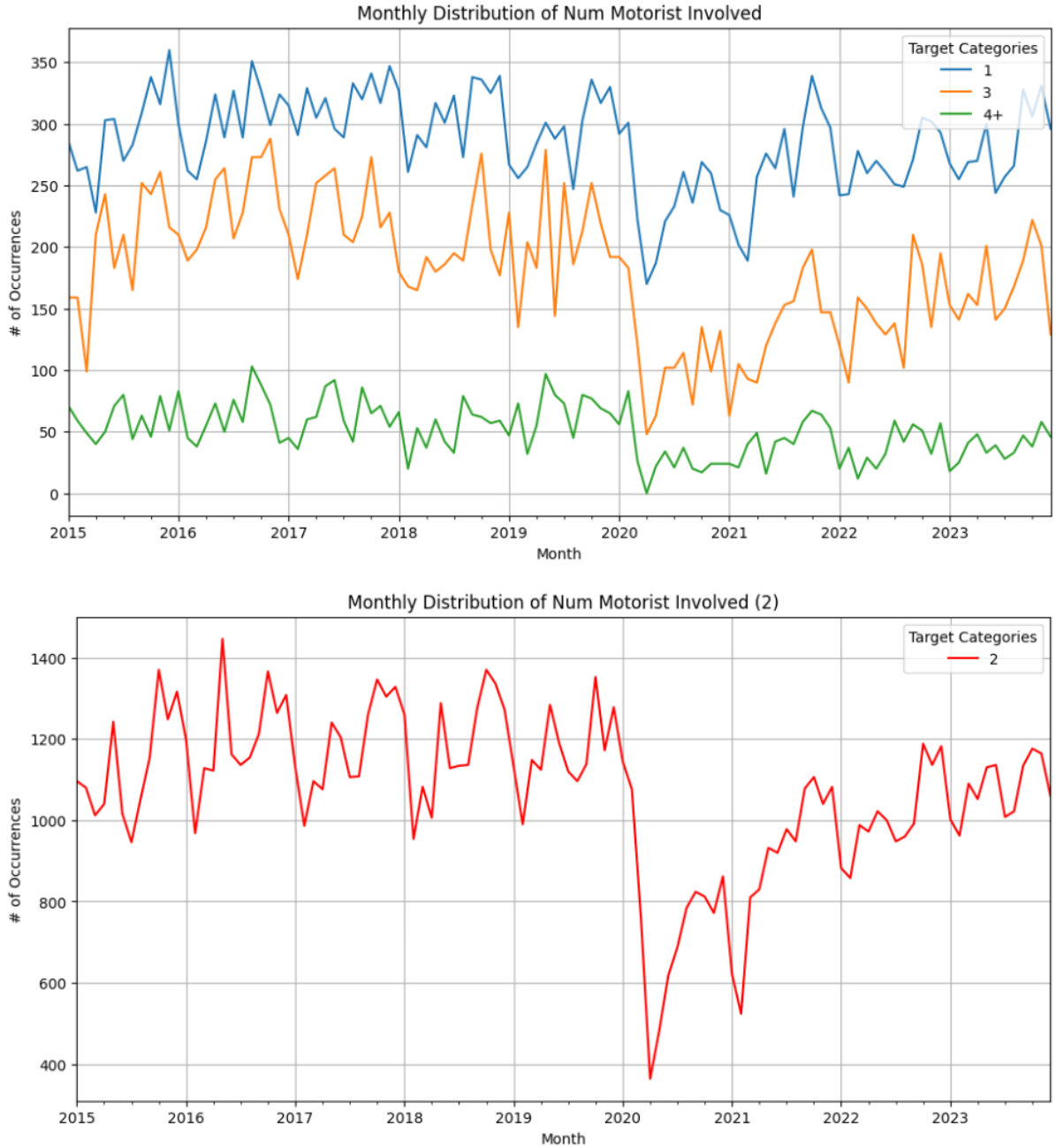


Figure 26

Number of Driver's Per Collision Per Month



Model Building and Results

Background

Across the board, each of the three target variables underwent the same modeling process, albeit with minor variations at certain points. First and foremost, for the machine learning models to predict the target variables, the features had to be converted into a format that is decipherable by the model. Any nominal features were one-hot encoded while binary and ordinal features were categorically encoded. During the hyperparameter tuning and final modeling processes a ten-fold cross validation was performed as a precaution to model overfitting. The five machine learning techniques used to predict the target variables were Random Forests, Complement and Bernoulli Navie Bayes models, XGBoost and lastly a singular Logistic Regression. A multilayer perceptron model was also attempted on one of the targets originally but was quickly scrapped for the other model types mentioned.

The metrics used to score all five of the model types consisted of weighted f1 score and accuracy. Given the unbalanced nature and occasionally binary aspects of the target variables, f1 score was chosen as the main modeling metric. Essentially f1 score combines precision and recall values into a singular number; However utilizing the weighted attribute for imbalanced datasets skews this assumption slightly. Accuracy on the other hand is simply the amount of observations the model predicted correctly to the total number of observations attempted to be predicted, making it a worse metric for analyzing imbalanced data. Both of these metrics were supplemented by visualizing the confusion matrix of the cross-validation fold which reported the highest weighted f1 score for the final models. With the exception of the MLP and XGBoost models, majority of these preprocessing, modeling, and scoring techniques were explained and implemented by utilizing the Scikit-Learn libraries and API.

Each of the selected models have hyperparameters, which when applied properly, can significantly influence the outcome of the models. This means that the hyperparameters selected for the first target variable may be suited well for a favorable outcome, however those same parameters applied to a different target variable of the same model type may see a lack in performance. The way this was combated was through utilizing Scikit-Learn's Grid Search Cross-Validation method. Mappings were created for a model's hyperparameters storing multiple values for each which were then passed into the Grid Search. The Grid Search then runs through all possible combinations of supplied hyperparameters and reports back the best combination based on the scoring metric selected. For the purposes of this project, machine learning techniques with a higher time complexity only utilized 10% of the dataset to perform the cross validations, while models that were less time complex utilized the entire dataset. The combination of hyperparameters with the highest weighted f1 score were passed into the final models which ran a 10-fold cross validation on all of available data.

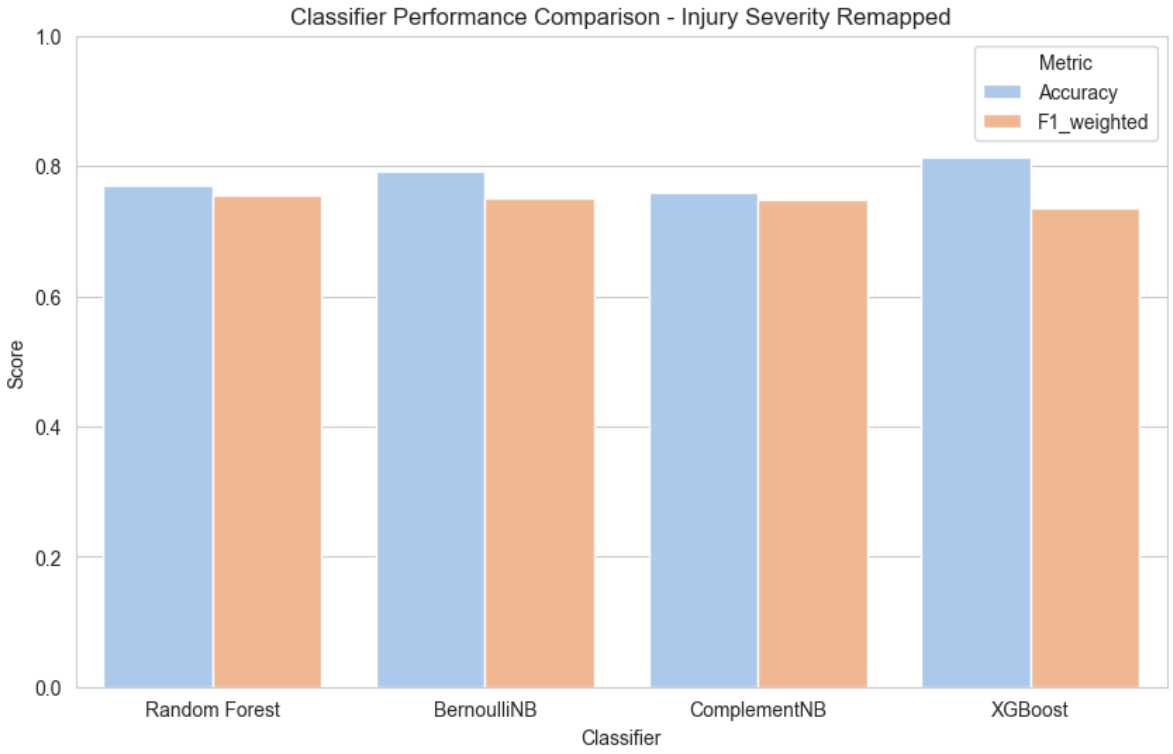
Results

Starting with the Injury Severity target variable, four machine learning models were put head-to-head. In the end the model selected for best performance was the Random Forest Classifier. A Random Forest model is an ensemble machine learning technique which generate a multitude of decision tree classifiers, pruning them across multiple iterations in hopes to improve scoring and minimize model overfitting. Overall the final Random Forest model reported an accuracy score of 77.1% and a weighted f1 score of 75.5%, which can be seen in the first column position of figure 27. This model had the highest performance across all models in the weighted f1 category, though it only had the third highest performance for accuracy score. However, if time complexity is of importance, the Bernoulli model may want to be selected as it performed

similarly to the Random Forest but only took 11 seconds to compute compared to the 5 minutes of the Random Forest.

Figure 27

Injury Severity Model Results



Also, by viewing the confusion matrix for the best cross validation fold in figures 28-31, the Random Forest did fairly well to accurately predict no injuries and minor injuries whilst still classifying a small amount of serious and fatal injuries properly. The same cannot be said for the other three models as they for the most part performed worse in the minor and serious injury categories (the Complement Naïve Bayes model did not even attempt to predict serious or fatal injuries). Overall the initial hypothesis was correct in that drivers with no injuries were easy to predict and drivers with serious or fatal injuries were harder to predict; However, it was incorrect in stating that predicting drivers who received minor injuries would be easy as in most cases all

three models heavily misclassified minor injuries as being not injured. In terms of the selected Random Forest model's best weighted f1 fold, no apparent injury was predicted correctly around 90% of the time, possible of minor injury was predicted correctly around 25% of the time and serious or fatal injury was only predicted correctly around 9% of the time.

Figure 28

Injury Severity Best Fold CV Confusion Matrix – Random Forest

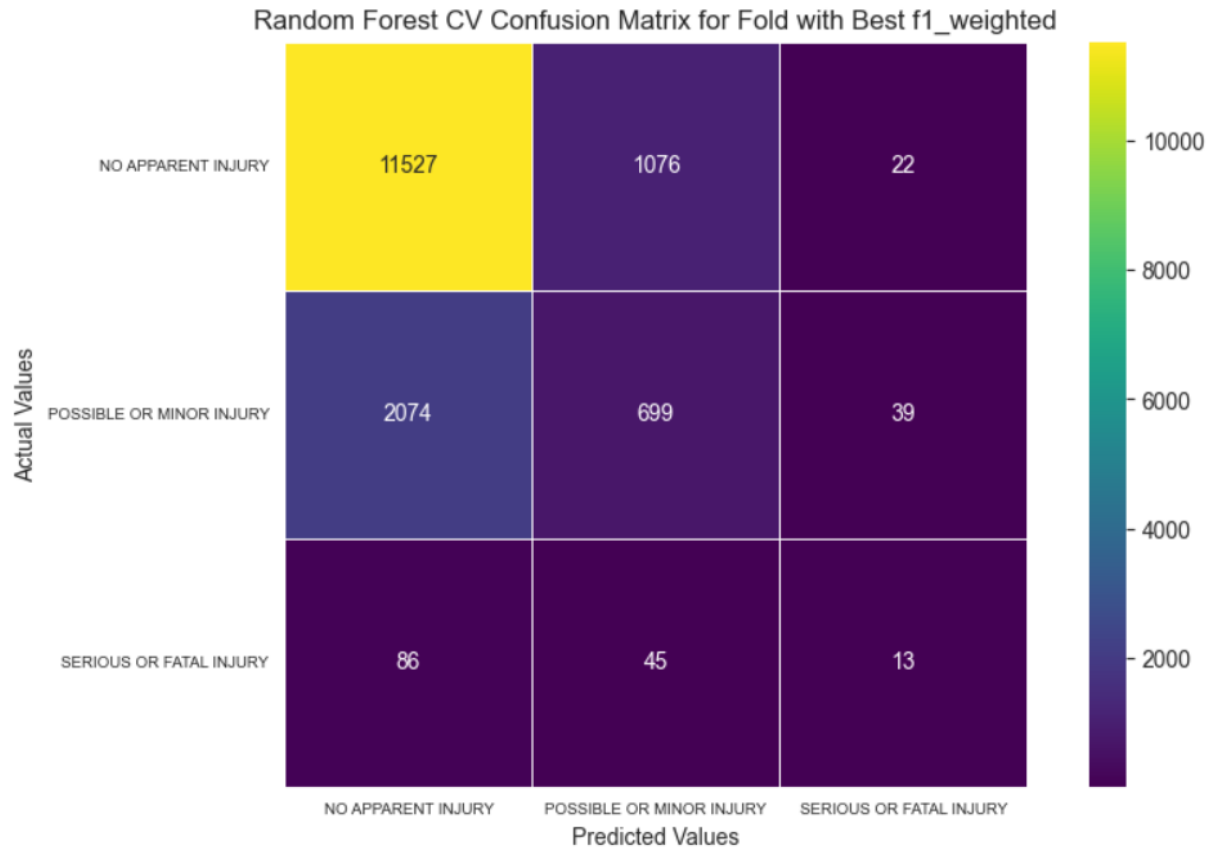


Figure 29

Injury Severity Best Fold CV Confusion Matrix – Bernoulli Naïve Bayes

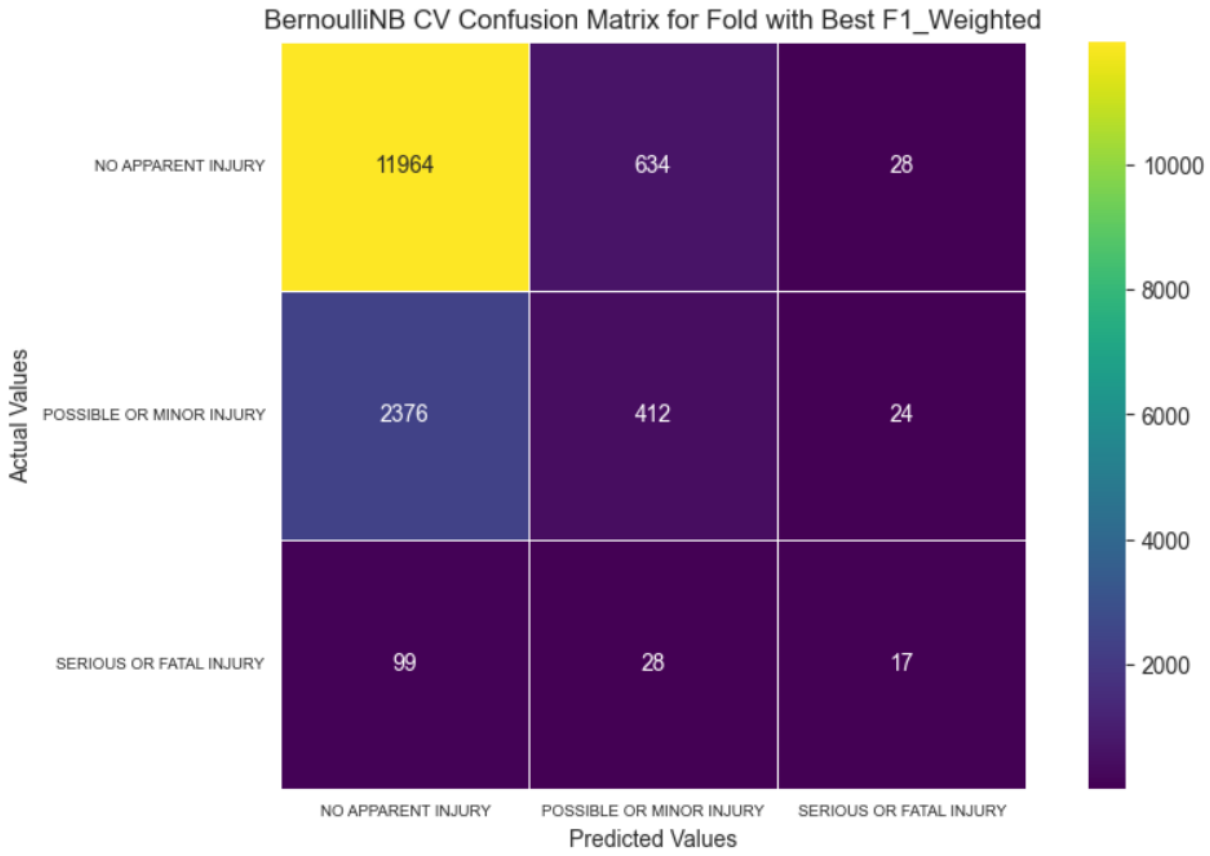


Figure 30

Injury Severity Best Fold CV Confusion Matrix – Complement Naïve Bayes

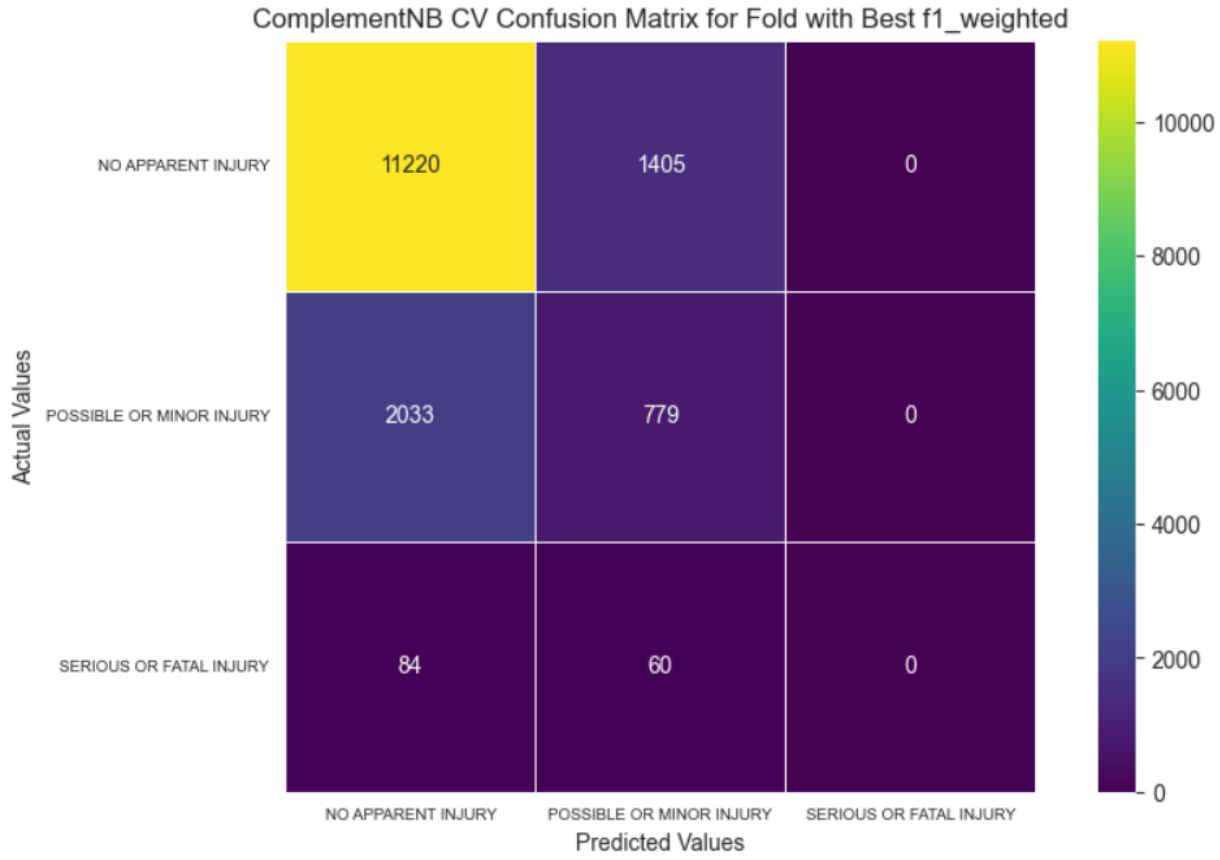
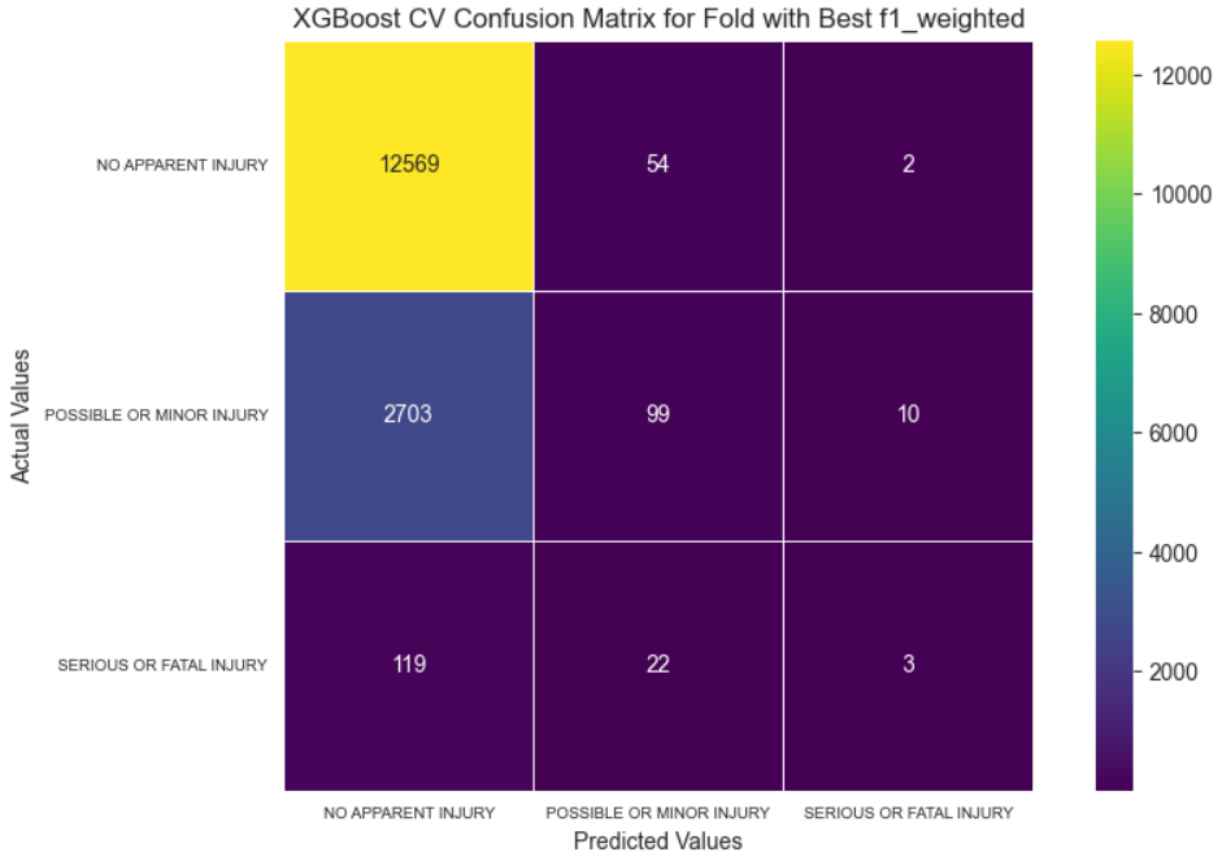


Figure 31

Injury Severity Best Fold CV Confusion Matrix - XGBoost



Moving on to the Driver Vehicle Damage target variable, models were generated for the predictive class being three categories and also binary. When the target still consisted of three categories (no damage or superficial, functional, and destroyed or disabling) the highest weighted f1 score across all the models was not even 60%. By observing the confusion matrices in figures 32-35, it can be easily distinguished that the models each did a decent job at classified no damage or disabling damage, but often times miscategorized when a driver's vehicle was damaged but not to the level of it being disabling. Thus the target variable was recategorized to be binary with the no damage or superficial and functional categories being combined as the

vehicle is still drivable in all of these scenarios, while the remaining category of disabling and destroyed results in the vehicle being no longer drivable. Given that the new target variable is binary in nature, a logistic regression model was also added into the mix.

Figure 32

Vehicle Damage Best Fold CV Confusion Matrix Non-Binary – Random Forest

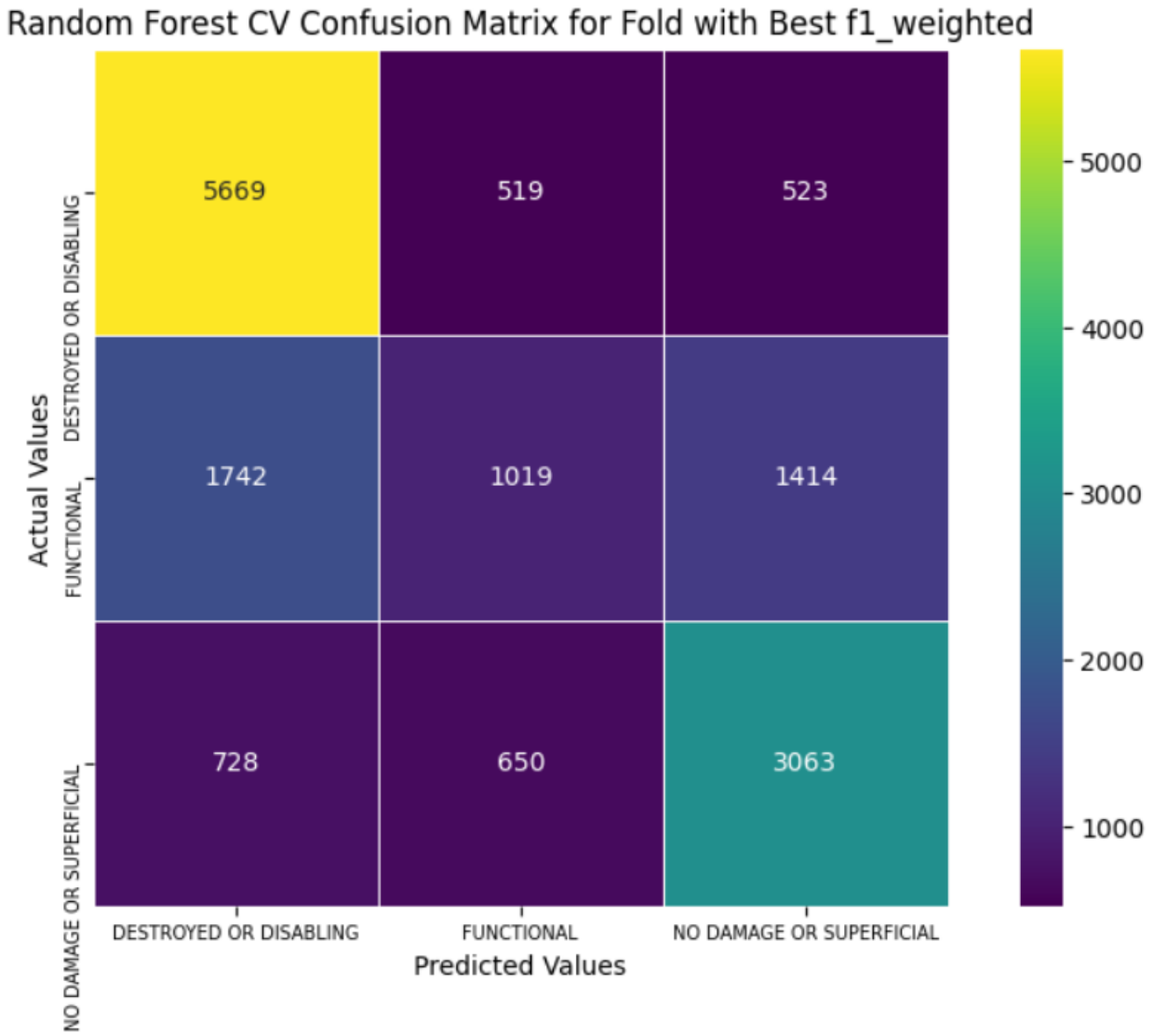


Figure 33

Vehicle Damage Best Fold CV Confusion Matrix Non-Binary – Bernoulli Navie Bayes

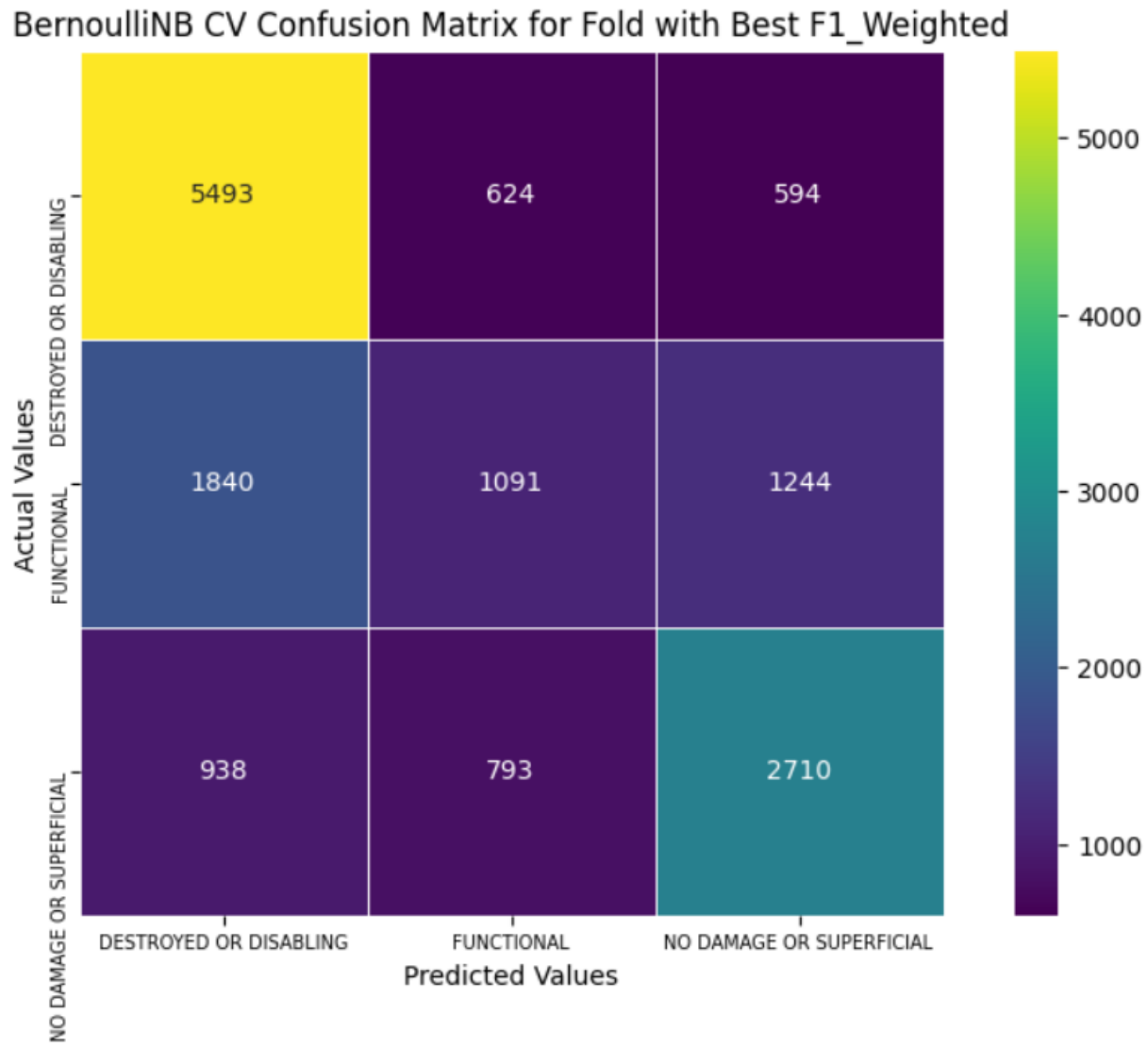


Figure 34

Vehicle Damage Best Fold CV Confusion Matrix Non-Binary – Complement Naïve Bayes

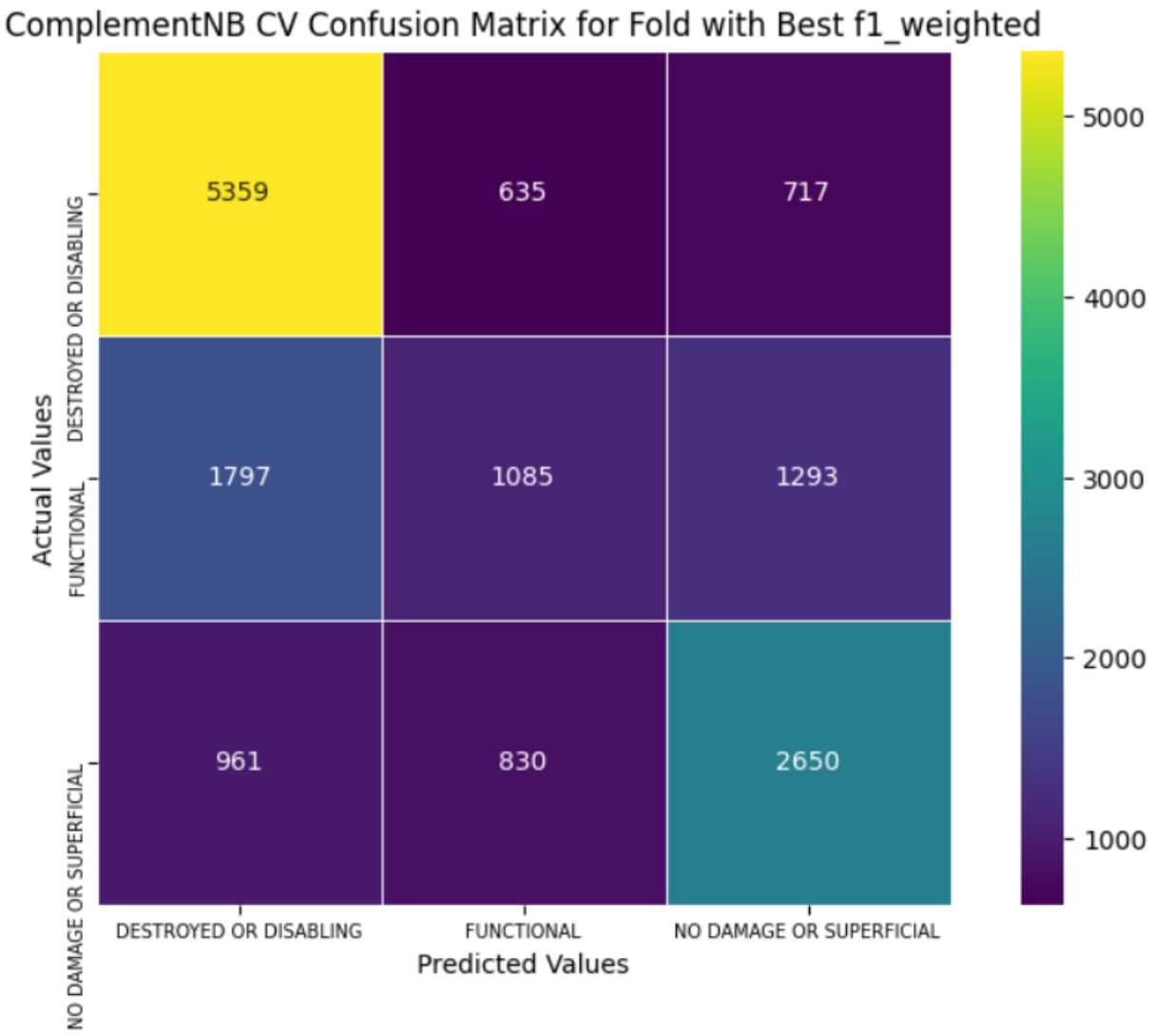
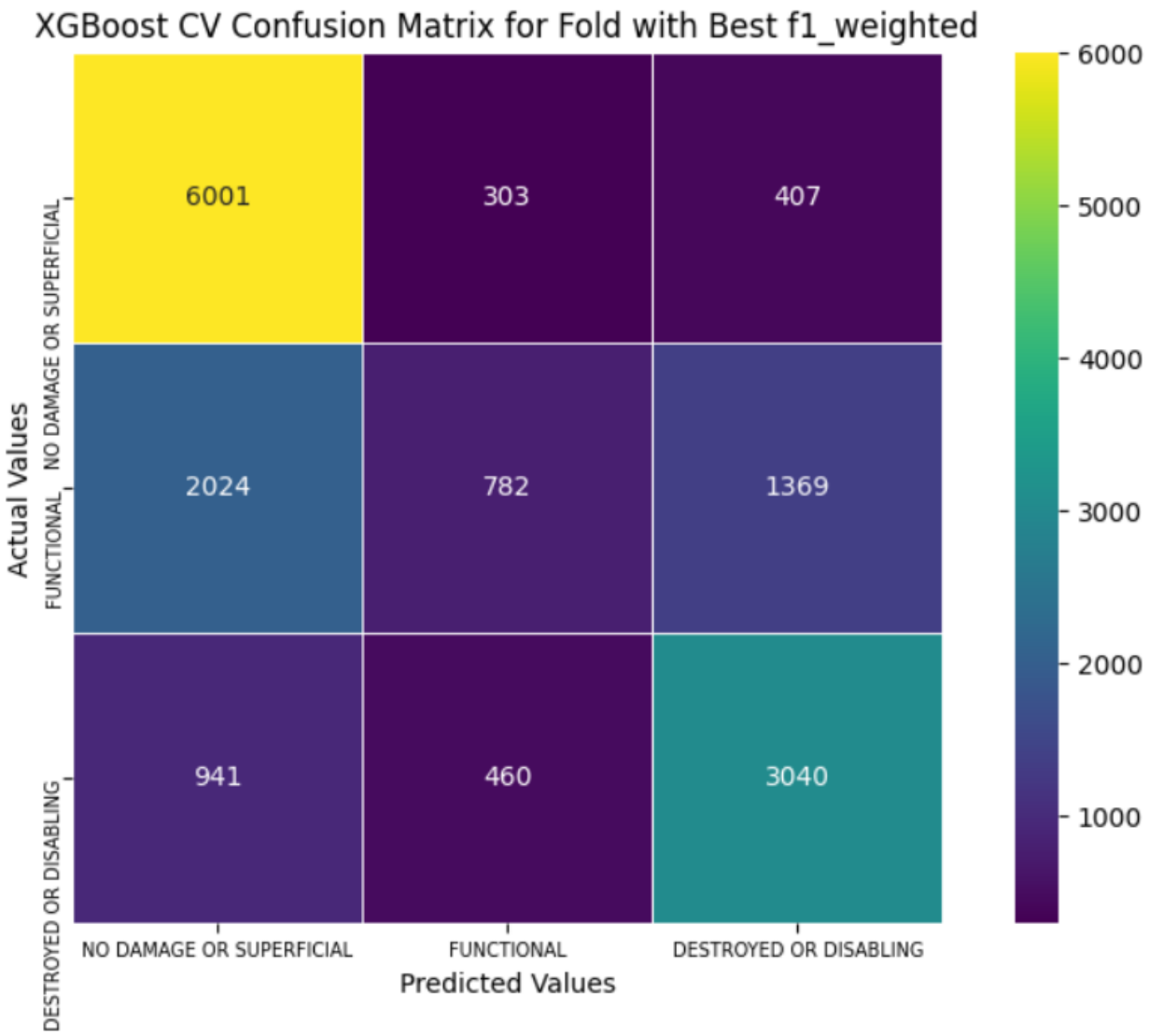


Figure 35

Vehicle Damage Best Fold CV Confusion Matrix Non-Binary - XGBoost

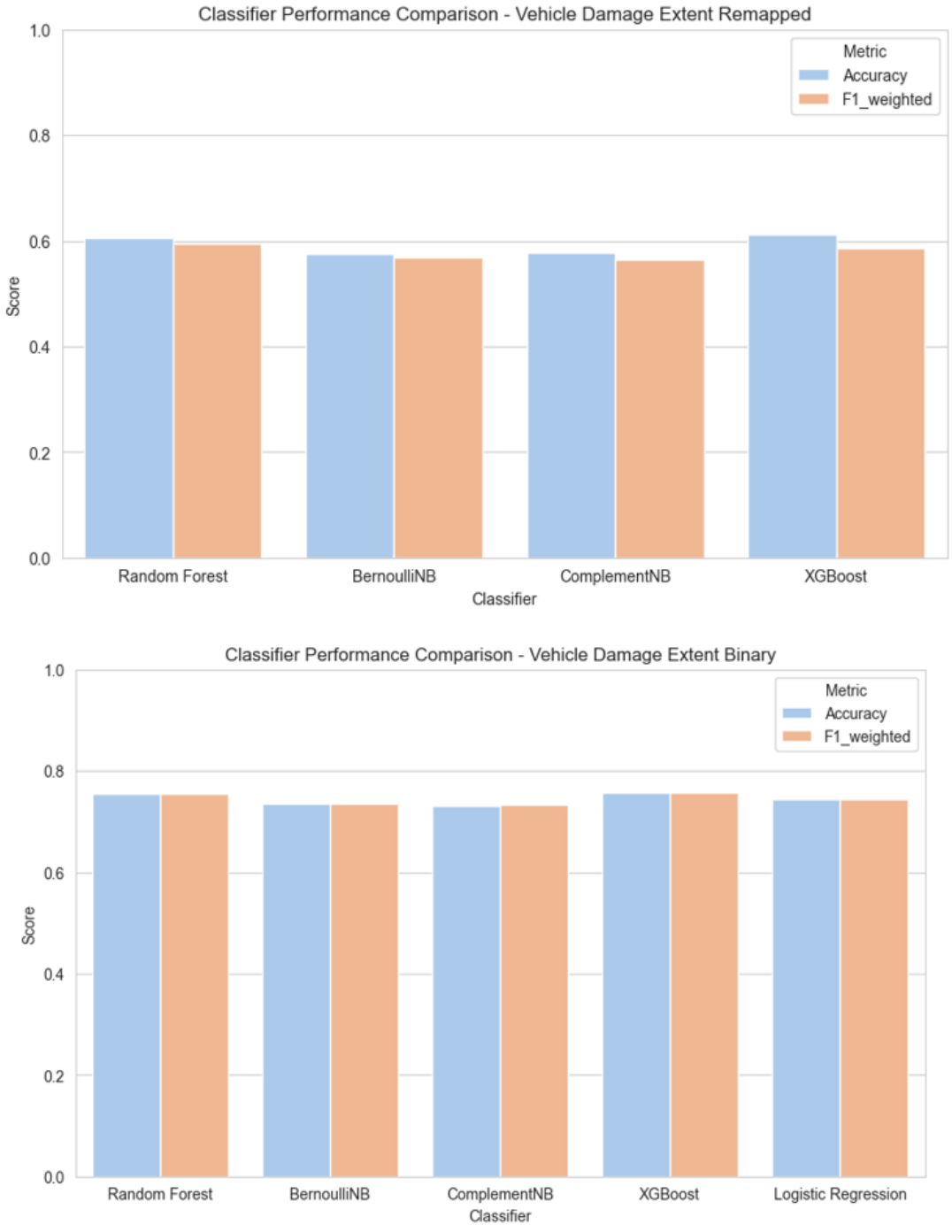


In the end the model selected for best performance was a toss-up between the XGBoost and the Random Forest. XGBoost or Extreme Gradient boosting is similar to the Random Forest in that it is also an ensemble which works with tree-based structures. For the purposes of this project only the gradient boosted decision tree was attempted with XGBoost, however utilization of a dart booster may have further increased model performance but was not incorporated due to

the significant increase in time complexity involved. In terms of metrics the XGBoost performed ever so slightly better than the Random Forest, with an accuracy score of 75.7% and a weighted f1 score of 75.8%. In comparison, the Random Forest had an accuracy score of 75.4% and a weighted f1 score of 75.5%. Comparisons of all of the models before and after recategorizing the target variable can be found in figure 36.

Figure 36

Vehicle Damage Model Results (Non-Binary = Top, Binary = Bottom)



In relation to the initial hypothesis that the Vehicle Damage target variable would be easily predictable, the results show that this is still the case. While there was trouble predicting

the target as three separate classes, as soon as it was converted to binary, the accuracy scores show average results for the ability to predict whether after an accident occurs if a vehicle will still be drivable or not. If the importance relies on predicting vehicles that will still be drivable the XGBoost model should be chosen, whereas if the importance relies on predicting vehicles that will no longer be drivable, the Random Forest model should be chosen. The confusion matrix for the best cross validation fold in figures 37 and 38 supports this statement.

Figure 37

Vehicle Damage Best Fold CV Confusion Matrix Binary – Random Forest

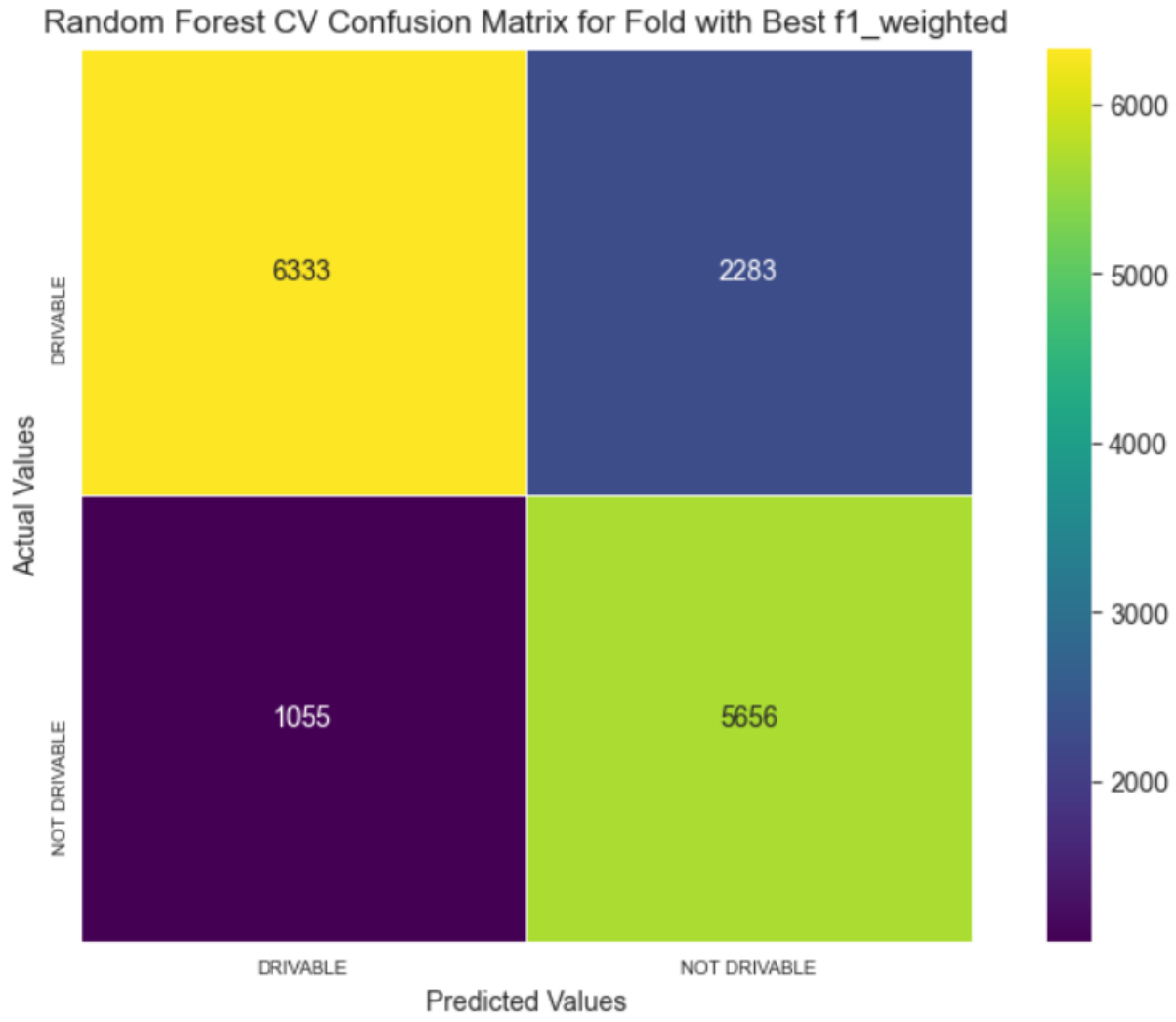
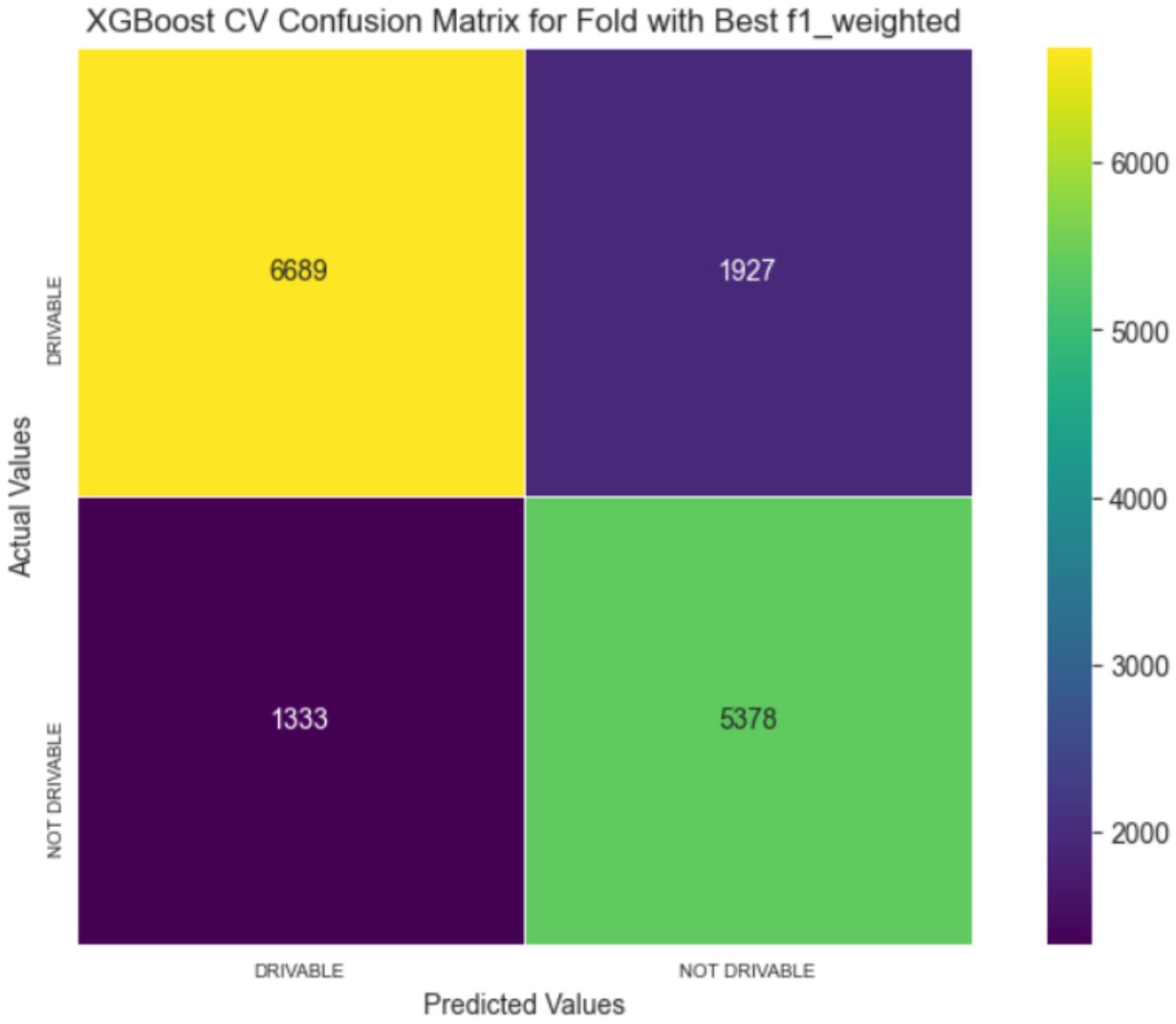


Figure 38

Vehicle Damage Best Fold CV Confusion Matrix Binary - XGBoost

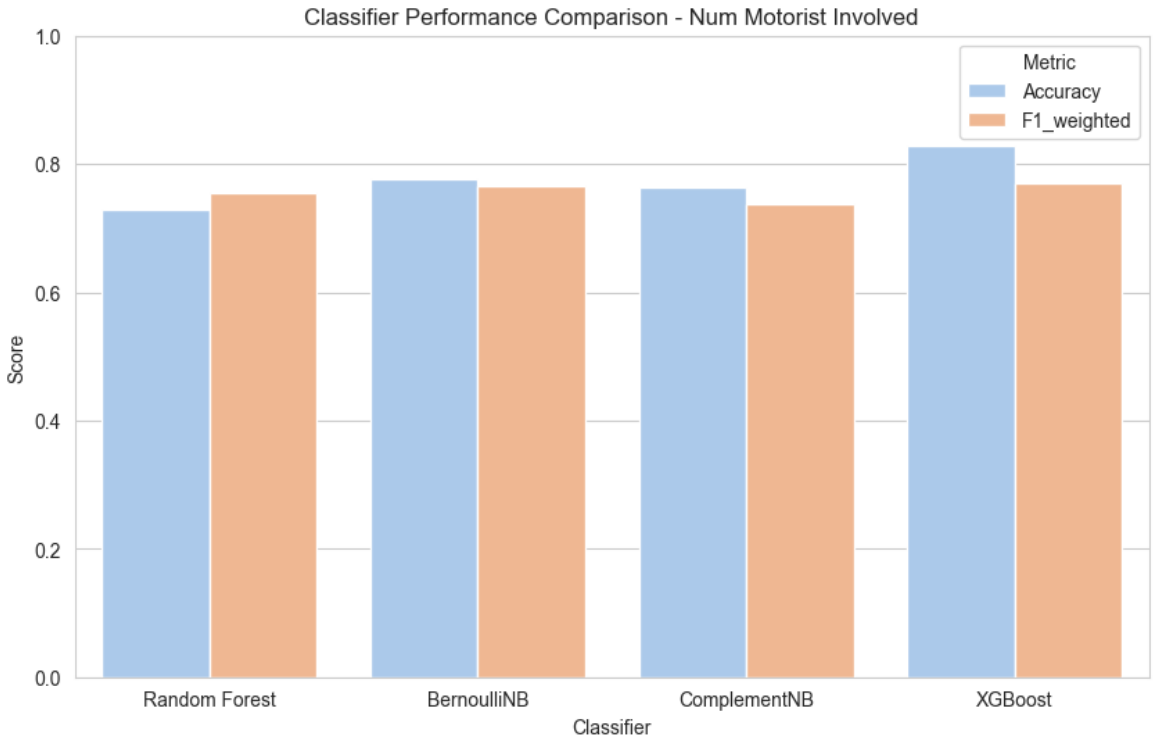


Lastly, the Number of Driver's involved target variable had a similar outcome to that of the previous target variable. The selected model that performed the best was once again the Random Forest, however strictly based on scoring metrics the XGBoost would have been chosen. The Random Forest reported an accuracy score of 72.9% and a weighted f1 score of

75.5%, while the XGBoost reported an accuracy score of 82.8% and a weighted f1 score of 77%. How all four of the models stacked up against each other can be observed in figure 39.

Figure 39

Number of Driver's Involved Model Results



The reason why the Random Forest was selected over the XGBoost as being the best model all comes down to how the confusion matrix for the best weighted f1 fold compares across the two models. Looking at figures 40-43, it can be easily seen besides the Random Forest, all three of the other models had a very difficult time predicting accidents that involved four or more drivers. Along these lines, the XGBoost was also the worst model at predicting accidents involving three drivers. Where the XGBoost excelled was at predicting accidents with one or two drivers involved, though this is automatically much easier given the unbalanced nature of the dataset. Ultimately the initial hypothesis regarding this target variable was mostly correct as accidents containing one or two drivers were much easier to predict than those containing three

or more. In terms of the selected Random Forest model's best weighted f1 fold, single driver accidents were predicted correctly around 93% of the time, double driver accidents were predicted correctly around 78% of the time, triple driver accidents were predicted correctly around 35% of the time and accidents containing four or more drivers were predicted correctly around 32% of the time.

Figure 40

Number of Driver's Involved Best Fold CV Confusion Matrix – Random Forest

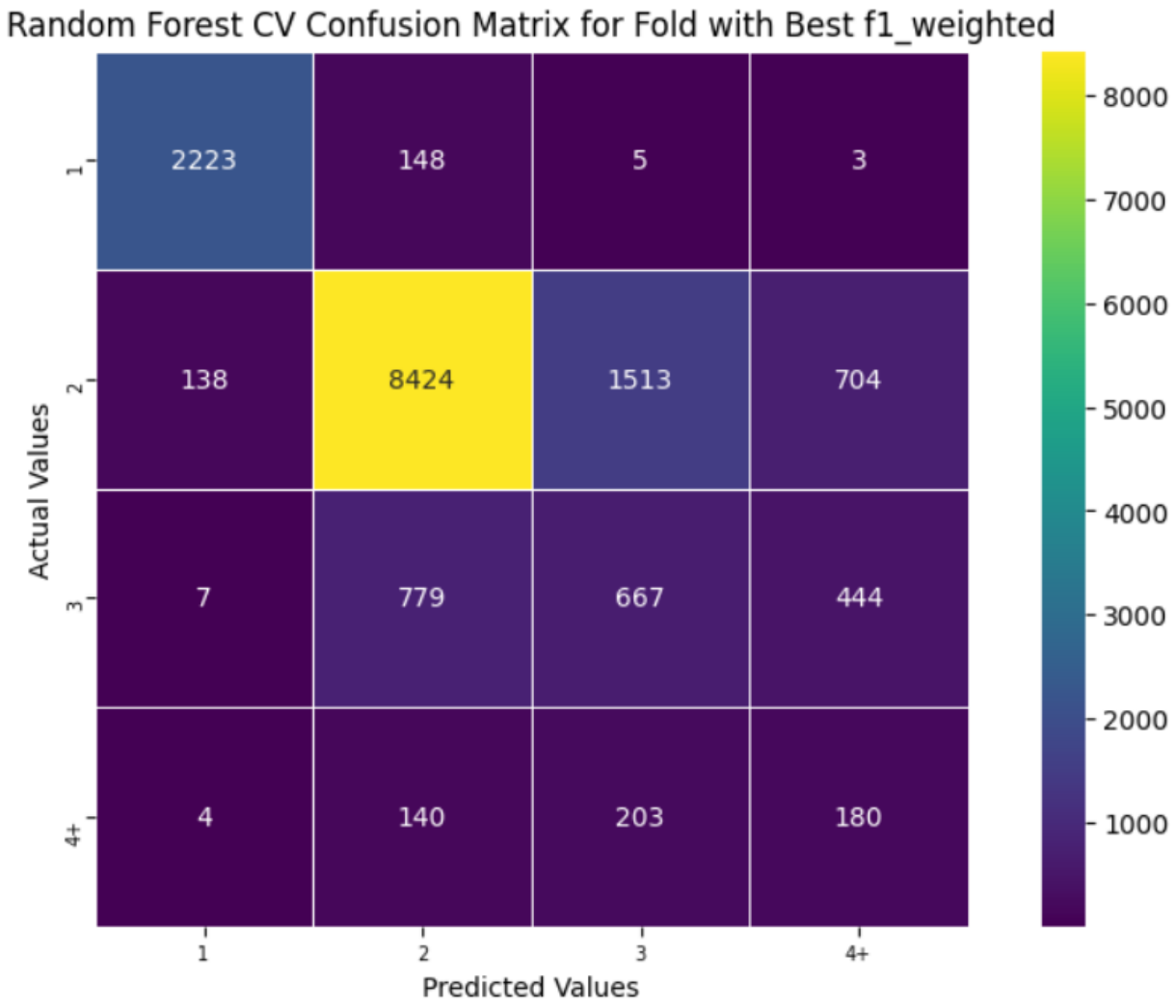


Figure 41

Number of Driver's Involved Best Fold CV Confusion Matrix – Bernoulli Naïve Bayes

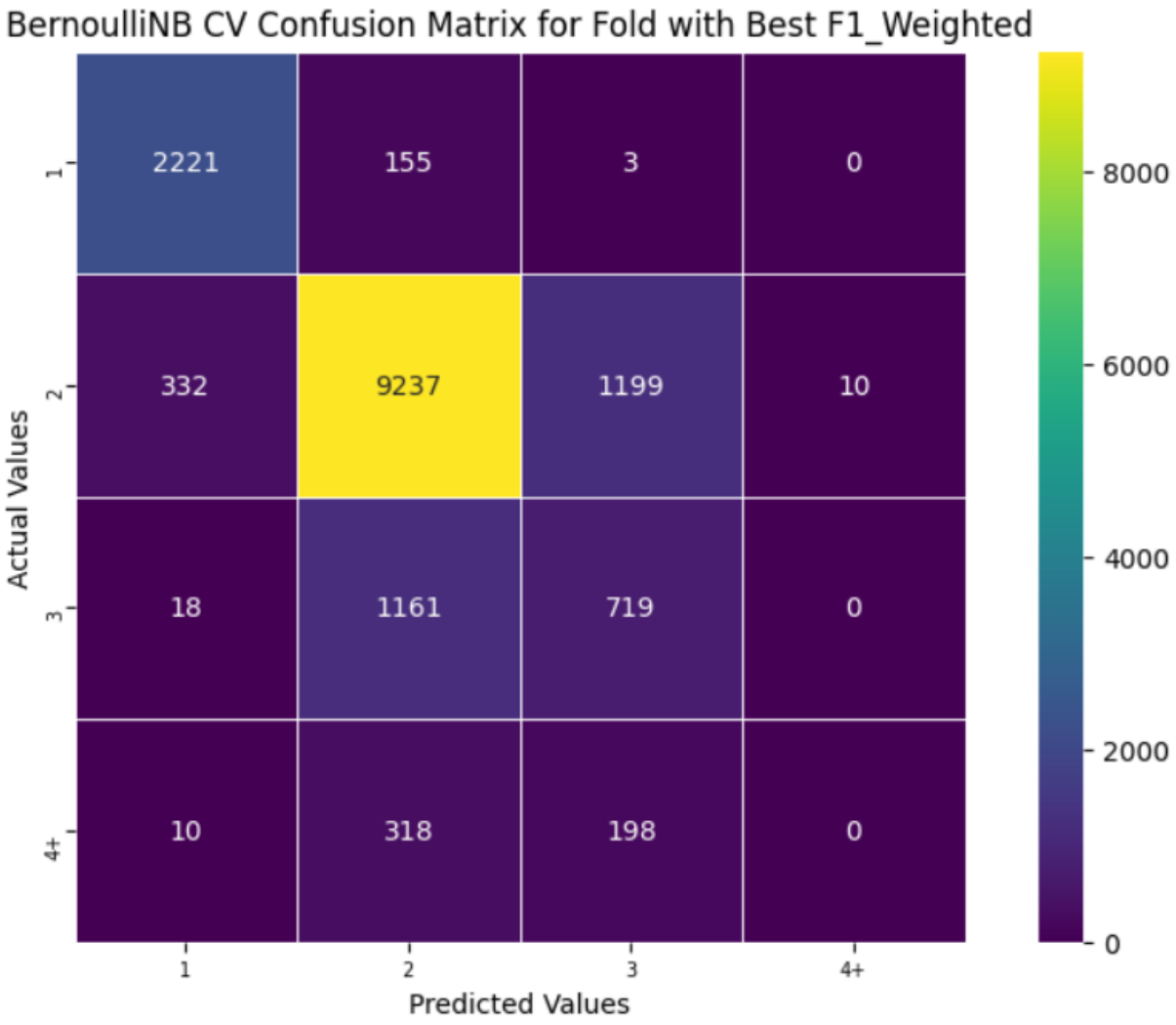


Figure 42

Number of Driver's Involved Best Fold CV Confusion Matrix – Complement Naïve Bayes

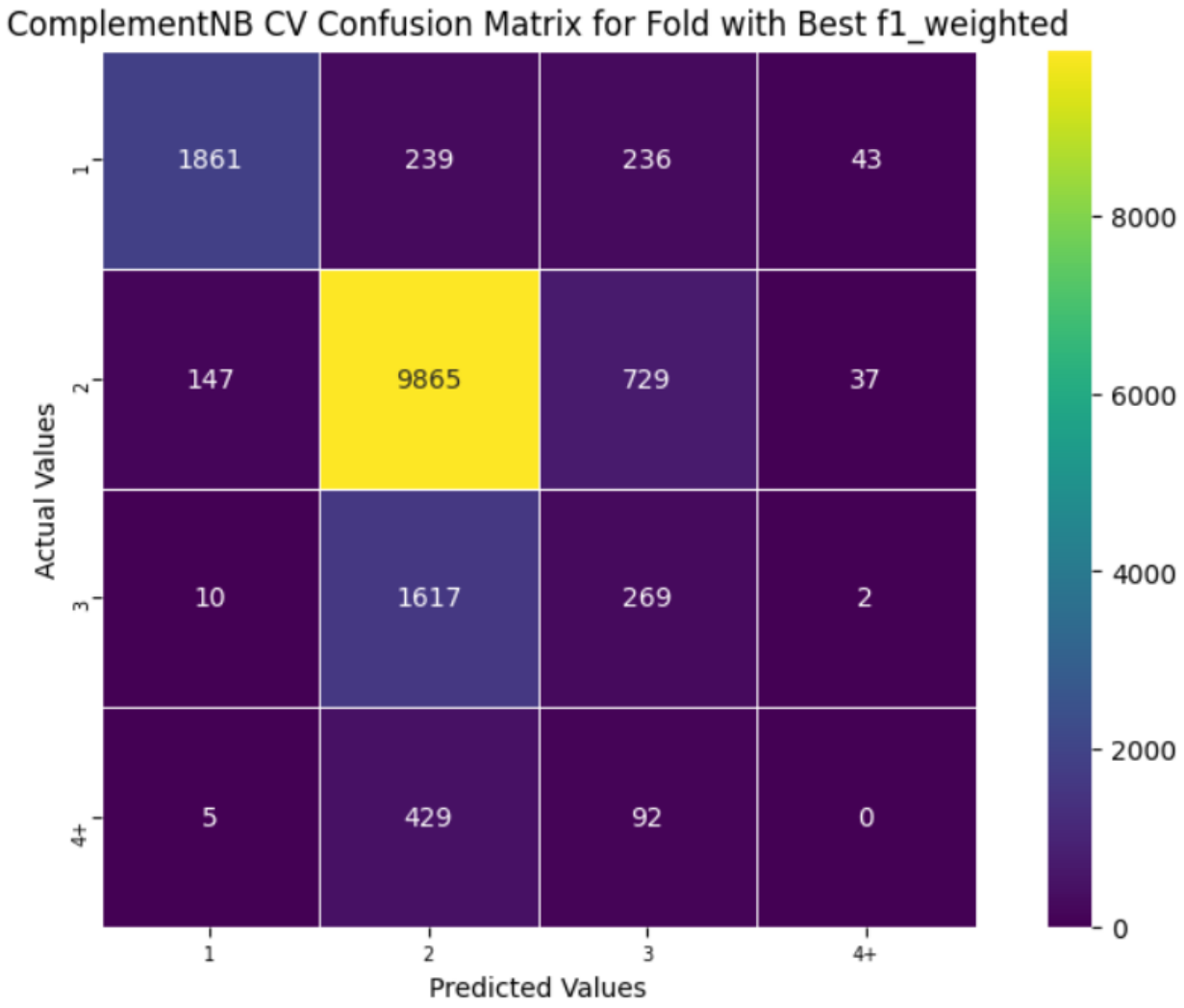
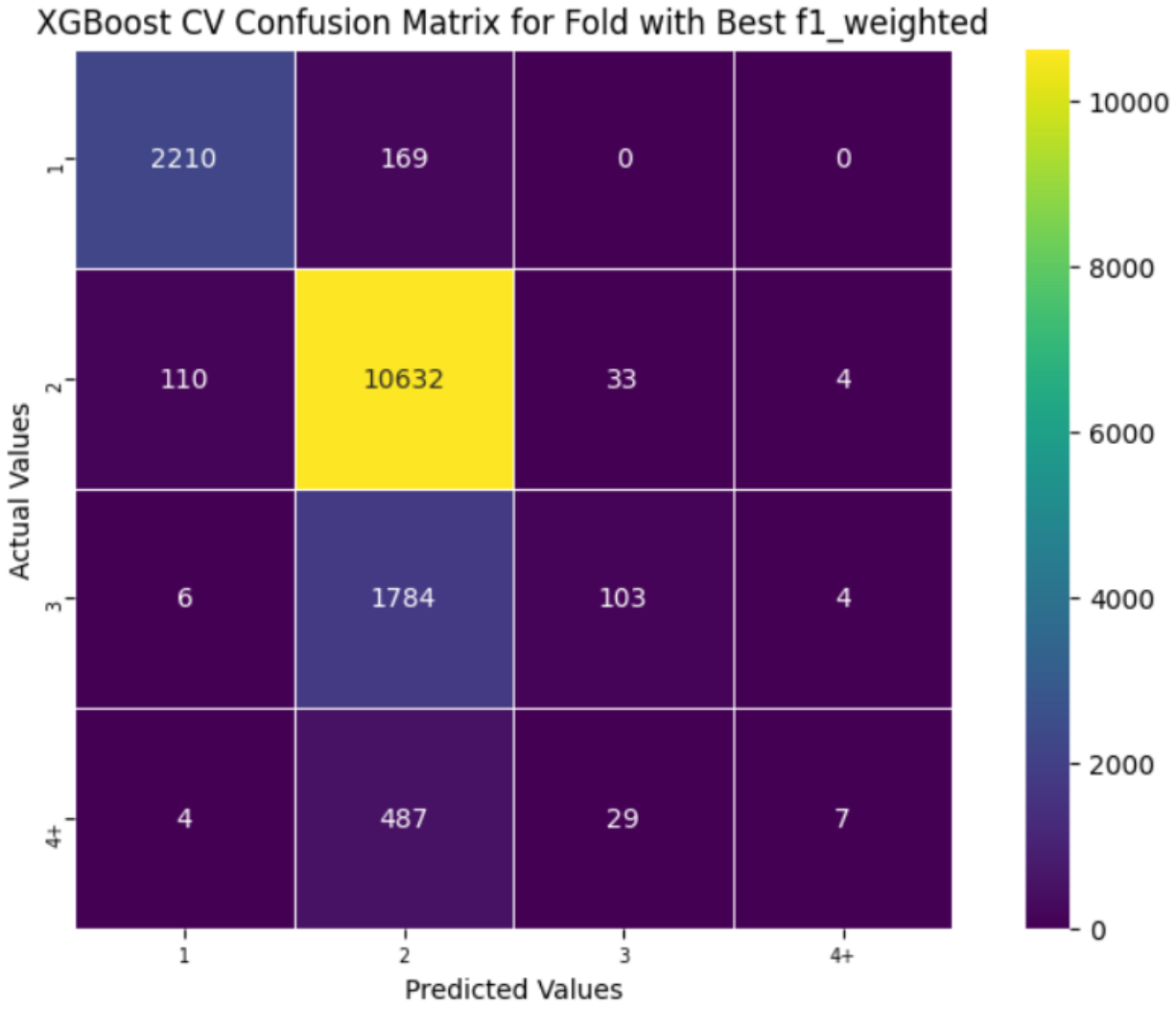


Figure 43

Number of Driver's Involved Best Fold CV Confusion Matrix - XGBoost



5. Threats to Study Validity

There are specific aspects of the dataset that were not accounted for when performing the model building process that could affect the validity of the study. To begin, accidents that have occurred closer to the current date and time should be valued more heavily than those that are furthest away from the current date. The reason being that if the model were ever deployed in a production instance, all of the severity predictions that it would be making would be for recently

occurring accidents. Unfortunately, the models developed within this project did not account for this in two separate ways. First, during the hyperparameter tuning stage of the model building process, models that had a higher time complexity were fed only 10% of the dataset to decrease run time. However, the 10% of the dataset that they were utilizing was biased in that it was obtained from the first part of the dataset, in return being the oldest accidents available. Second, models which take into account the time-series nature of the data should have been utilized in that accidents occurring more recently were weighted higher in training than those that occurred less recently. A good example of this would have been the LSTM RNN (Long Short-Term Memory Recurrent Neural Network) model.

Another aspect of the dataset that threatens the validity of this study would have to be the class imbalances across almost all of the target variables. Often times when attempting to predict severity of any sort, it is the most sever classes that are the most beneficial to be predicted. For instance, when predicting Driver Injury Severity and Driver Vehicle Damage, being able to accurately predict human fatalities, and vehicle totalities are the most important, such that if the data allows for them to be predicted than mitigations may be able to be put in place to lower these numbers in reality. Unfortunately, in the case of the data at hand, these classes often times represented such a small portion of the data that they get overpowered by more common and less severe classes, which in return are much easier for the models to accurately predict, and less focus is given to the classes of higher severity which are more beneficial for the modeler to know.

The last aspect of the dataset that threatens the validity of this study has to do with the completeness of the original dataset. Since many of the variables were missing values or had miss spellings in category names they had to be filled and interpreted though the use of statistical

estimates. Given that these estimates were made there may be places where a record was filled or changed to a value other than what actually occurred based on the accident. In other cases, there were features left out of the model due to their incompleteness and complexity. An example of this would be the road name and cross street name features. Both of these features were missing 10% of their data, and of the data that was not missing a significant amount of the values held spelling mistakes. These shortcomings could have been remedied with enough time and effort, but even with doing so the complexity of the features were too great for the machine that the models were being run on as each of these features alone would have upwards of a thousand individual classes.

6. Future Work

Incomplete Items

Items that were not able to be completed as part of this project include initializing working gpu drivers, modeling and refining different neural network techniques, further refinement of dataset features and Docker deployments for the most successful models. More time would have been spent on creating and refining multi-layer perceptron and recurrent neural networks if necessary gpu drivers could have been initialized within the Jupyter notebooks. The drivers were installed but could not be loaded into the chosen ide resulting in an attempted MLP model which took upwards of 16 hours to hyperparameter tune on a very small portion of the entire dataset running solely off of a CPU. In terms of feature selection, road name and cross street name were not included in the final models due to missing data and class complexity. Ideally the missing values would have been reverse geocoded with Nominatim and utilized in the

final models as the streets where the accidents occurred seemed to have a decent Cramer's V correlation with the target variables. Lastly, creating a Docker image containing the most successful models would have allowed for the client to easily deploy the models into their environment. This would be optimal for generating severity predictions on any new accident information that may need to be entered in real time.

Next Steps

The initial study discussed in this manuscript was only focused on accidents which occurred in Montgomery County Maryland between 2015 and 2023. Of the 170,000 observations in the initial dataset, there were in total 153 fatal injuries. While this is amazing for public safety, it makes it extremely difficult to predict for any model. Also, with such a small amount of observations for this minor class, the amount of distinct and unique datapoints are not available to make a resampling technique such as bootstrapping worthwhile to artificially increase the number of fatal observations and balance out the dataset. Thus, a solution to this problem would be to broaden the sample size and expand the range of accidents from only Montgomery County to all of Maryland or even the north easter region of the United States. This would potentially narrow the gap between target class imbalances and provide a stronger basis to perform resampling techniques upon.

Other future work that would be beneficial to this project consists of feature ablation for goal achievement which would take place after the model has been trained. Say the overall goal of the project is to reduce serious and fatal injuries by 5% within the next 2 years, or to reduce the number of multi-vehicle accidents by 10% within the next 5 years. Feature ablation is one way that a data scientist can determine what the best contributing factors are to the model that

may sway these results. A baseline test set with the most recent accidents would be run on the model and held as a control. Then features in particular observations of interest would have their values be manipulated to something other than what they actually are. The model would be rerun on the new modified test set and differences in outcomes would be noted between the new test set and the control. The process would be repeated until it can determine real life factors within the features that have an influence on accident severity outcomes in a positive light. Then the client should come up with ways to replicate these values in any new real-world data and see if in fact the goals initially set can be met.

7. Reflections

The largest takeaway from completing this data science research project is that there is no single aspect, or one size fits all technique that can be applied to any dataset - each dataset is unique and needs to be treated as such for optimal results. While many techniques learned in class were applied to this project, just as many new techniques needed to be discovered for this project to be successful. Never having used them before, the python packages Shapely, GeoPandas, FuzzyWuzzy and XGBoost were integral to this project and without them it would have been much more complex and harder to finish. Along these lines, a lot was learned about how there is no one correct way to do thing and that often times through trial and error many different tools and techniques can assist in reaching the same end goal.

The part of the project that offered the greatest reward would have to be exploring and manipulating the initial dataset itself. No amount of modeling will make up for a lack of understanding the data in its most natural and unmodified state. Thus, exploring initial trends and

learning the points where the data is deficient is also allowed for the most prevalent parts of the data to be highlighted and the most troublesome parts to be corrected.

By far the most challenging aspect of this project was the struggle of setting up the development environment and lack of the necessary computing power to compute model calculations. Attempting to install the necessary driver for gpu utilization within python, while covered in some of the machine learning courses, was not as straightforward as the documentation entailed. Thus it felt more like a system administrative task rather than a data science one and was never able to be used during this project. Because of this, the inspiration to use PyTorch for neural networks suffered greatly as the time complexity was not reasonable for someone who has priorities outside of this project in regard to their career. Even so, models that were used inside of this project, such as XGBoost, could have been sped up by processing models on a gpu. Other times, data science techniques such as DBSCAN clustering were attempted, but could not be used as a result of a lack of memory on the system. The same can be said for striving to run multiple processes simultaneously across different notebooks to minimize run times.

Since completing this data science project, suggestions for other researches include aspects that were done as well as aspects that should have been done. Starting with the should have been done, it is highly recommended that future data scientists who perform a similar project utilize cloud computing for their analysis and model building. Creating an AWS account is now easier than ever, and the price associated with the instances stood up is marginal compared to purchasing hardware. In doing so, future researches will not only be able to increase computing power when needed but may also even use multiple cloud instances to run code in parallel. This

will significantly decrease the computing times necessary for model building and will allow the researcher more time to spend on upgrades and enhancements.

The other suggestion would be to schedule short reoccurring meetings with the client, or if no client is available then to utilize the IAB partners that Loyola offers. Having met with an IAB partner during this project, the value they add in terms of accountability in adhering to the proposed project plan is insurmountable. Also, with being in the data science field professionally, the IAB partners often have specific insights into how the project can be enhanced based on what has already been constructed, as well as tips and techniques for presenting the project results to client but also the project review board at the end of the semester.

References

Related Works

Elfadil, M. A. (2014). Predicting Causes of Traffic Road Accidents Using Multi-class Support Vector Machines. *Journal of Communication and Computer, 2014*(5), 441–447. https://doi.org/10.17265/1548-7709

Ma, Z., Mei, G., & Cuomo, S. (2021). An analytic framework using Deep Learning for prediction of traffic accident injury severity based on contributing factors. *Accident Analysis & Prevention, 160*, 1-16. https://doi.org/10.1016/j.aap.2021.106322

Montgomery County Department of Technology Services. Montgomery County Government Open Data Operations Manual. (2022, July 15). https://www.montgomerycountymd.gov/open/Resources/Files/FY23%20Open%20Data%20Operations%20Manual.pdf

Santos, K., Dias, J. P., & Amado, C. (2022). A literature review of machine learning algorithms for Crash injury severity prediction. *Journal of Safety Research*, *80*, 254–269. https://doi.org/10.1016/j.jsr.2021.12.007

Wang, K., Bhowmik, T., Zhao, S., Eluru, N., & Jackson, E. (2021). Highway Safety Assessment and improvement through crash prediction by injury severity and vehicle damage using multivariate poisson-lognormal model and joint negative binomial-generalized ordered probit fractional split model. *Journal of Safety Research*, *76*, 44–55. https://doi.org/10.1016/j.jsr.2020.11.005

Xu, C., Tarko, A. P., Wang, W., & Liu, P. (2013). Predicting crash likelihood and severity on freeways with real-time loop detector data. *Accident Analysis & Prevention*, *57*, 30–39. https://doi.org/10.1016/j.aap.2013.03.035

Yan, X., He, J., Zhang, C., Liu, Z., Qiao, B., & Zhang, H. (2021). Single-vehicle crash severity outcome prediction and determinant extraction using tree-based and other non-parametric models. *Accident Analysis & Prevention*, *153*, 1–21. https://doi.org/10.1016/j.aap.2021.106034

Yang, Z., Zhang, W., & Feng, J. (2022). Predicting multiple types of traffic accident severity with explanations: A multi-task Deep Learning Framework. *Safety Science*, *146*, 1-13. https://doi.org/10.1016/j.ssci.2021.105522

Referenced APIs

Pandas - https://pandas.pydata.org/docs/reference/index.html

GeoPandas - https://geopandas.org/en/stable/docs/reference.html

Shapely - https://shapely.readthedocs.io/en/stable/manual.html

Plotly - https://plotly.com/python-api-reference/

Numpy - https://numpy.org/doc/stable/reference/

Matplotlib - https://matplotlib.org/stable/api/index.html

Seaborn - https://seaborn.pydata.org/api.html

Scipy - https://docs.scipy.org/doc/scipy/reference/

FuzzyWuzzy - https://github.com/seatgeek/thefuzz/blob/master/README.rst

Scikit-Learn - https://scikit-learn.org/stable/modules/classes.html

XGBoost - https://xgboost.readthedocs.io/en/latest/python/python_api.html

Links to Datasets

Driver's Dataset - Crash Reporting - Drivers Data | Open Data Portal (montgomerycountymd.gov)

Incident's Dataset – Crash Reporting - Incidents Data | Open Data Portal (montgomerycountymd.gov)

Non-Motorist Dataset – Crash Reporting - Non-Motorists Data | Open Data Portal (montgomerycountymd.gov)

Maryland County Boundaries – Maryland County Boundaries | Open Data | opendata.maryland.gov

Montgomery County Maintained Roads – Montgomery County Maintained Roads | Montgomery County Maintained Roads | Maryland's GIS Data Catalog

Maryland Interstates – Maryland Routes with ZIP Code (Interstates, MD State, & US State) | Maryland Routes with ZIP Code (Interstates, MD State, & US State) | Maryland's GIS Data Catalog

Maryland Municipal Maintained Roads – Municipal Maintained Roads | Municipal Maintained Roads | Maryland's GIS Data Catalog

Montgomery County Traffic Analysis Zones - Traffic Analysis Zones (TAZ) | Traffic Analysis Zones (TAZ) | Montgomery County Data Catalog (arcgis.com)

Appendix

Data Cleaning Notebook

Imports

In [1]:
```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point
import plotly.express as px
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
from fuzzywuzzy import fuzz
import plotly.graph_objects as go
```

Initializing Data

Links to Download Datasets

Drivers Data; Incidents Data; Non-Motorist Data; Maryland County Boundaries Data; Montgomery County Maintained Roads

Reading in the Data

In [2]:
```
driver_dtypes = {'Local Case Number': "string"}

drivers_df = pd.read_csv("../Datasets/Crash_Reporting-Drivers_Data.csv", dtype=driver_dtypes)
incident_df = pd.read_csv("../Datasets/Crash_Reporting-Incidents_Data.csv")
non_motorist_df = pd.read_csv("../Datasets/Crash_Reporting-Non-Motorists_Data.csv")

maryland_counties = gpd.read_file("../Datasets/Maryland_County_Boundaries.geojson")
montgomery_count_geo = maryland_counties[maryland_counties["county"] == "MONTGOMERY"]["geometry"]
```

Merge the Data

In [3]:
```
incident_df_drop_cols = incident_df[['Report Number', 'Hit/Run', 'Lane Number', 'Number of Lanes',
                                     'Road Grade', 'NonTraffic', 'First Harmful Event', 'Second Harmful Event',
                                     'Fixed Oject Struck', 'Junction', 'Intersection Type', 'Intersection Area',
                                     'Road Alignment', 'Road Condition', 'Road Division']]
incident_df_drop_cols = incident_df_drop_cols.copy()

rename_cols = {'Hit/Run': 'Incident Hit/Run', 'Fixed Oject Struck': 'Fixed Object Struck'}
incident_df_drop_cols.rename(columns=rename_cols, inplace=True)
```

In [4]:
```
merged_df = pd.merge(drivers_df, incident_df_drop_cols, on='Report Number', how='inner')
```

In [5]:
```
non_motorist_count_df = non_motorist_df['Report Number'].value_counts().reset_index()
non_motorist_count_df.rename(columns={"index": "Report Number", "Report Number": "Num Non-Motorist Involved"}, inplace=True)
```

In [6]:
```
merged_df = pd.merge(merged_df, non_motorist_count_df, on='Report Number', how='left')
```

In [7]:
```
print(merged_df.shape)
print(drivers_df.shape)
```

```
(172004, 58)
(172004, 43)
```

Check for Duplicated Data

In [8]:
```
merged_df.duplicated().value_counts()
```

Out[8]:
```
False    172004
dtype: int64
```

Filter Geographically

In [9]:
```
geometry = [Point(xy) for xy in zip(merged_df["Longitude"], merged_df['Latitude'])]
merged_gdf = gpd.GeoDataFrame(merged_df, geometry=geometry)
```

In [10]:
```
%%time
filtered_gdf = merged_gdf[merged_gdf["geometry"].within(montgomery_count_geo.unary_union)]
```

```
CPU times: total: 2min 55s
Wall time: 2min 56s
```

In [11]:
```
heatmap_df = filtered_gdf.groupby(["Latitude", "Longitude"]).size().reset_index(name="Count")
```

In [12]:
```
fig = px.density_mapbox(lat = heatmap_df.Latitude, lon = heatmap_df.Longitude, z=heatmap_df.Count,
                        radius = 9,
                        center = dict(lat = 39.128, lon = -77.202),
                        zoom = 9,
                        mapbox_style = 'open-street-map',
                        color_continuous_scale = 'rainbow',
                        opacity = 0.65)
fig.update_layout(
    width=1000,
    height=800
)
fig.show()
```

Standardize Missing Data Labels

In [13]:
```
filtered_gdf = filtered_gdf.copy()
filtered_gdf['Vehicle Year'] = filtered_gdf['Vehicle Year'].replace(0, np.nan)
cols = ['Route Type', 'Cross-Street Type', 'Road Name', 'Cross-Street Name', 'Off-Road Description', 'Collision Type',
       'Weather', 'Surface Condition', 'Light', 'Traffic Control', 'Driver Substance Abuse', 'Driver At Fault',
       'Driver Distracted By', 'Drivers License State', 'Vehicle Damage Extent', 'Vehicle First Impact Location',
       'Vehicle Second Impact Location', 'Vehicle Body Type', 'Vehicle Movement', 'Vehicle Continuing Dir',
       'Vehicle Going Dir', 'Driverless Vehicle', 'Vehicle Make', 'Vehicle Model', 'Equipment Problems', 'Road Grade',
       'First Harmful Event', 'Second Harmful Event', 'Fixed Object Struck', 'Junction', 'Intersection Type',
       'Road Alignment', 'Road Condition', 'Road Division', 'Non-Motorist Substance Abuse']
filtered_gdf[cols] = filtered_gdf[cols].replace(['Unknown', 'NO NAME', 'UNKNOWN', 'UNKNOWN LOCATION', 'XX', "N/A, UNKNOWN"],
                                                np.nan)
```

In [14]:
```
def plot_missing_feat(df, title):
    percent_missing = ((df.isna().sum() / df.shape[0]) * 100).reset_index()
    percent_missing.rename(columns={"index":"Feature", 0: "Percentage Missing"}, inplace=True)
    percent_missing.sort_values(by="Percentage Missing", ascending=False, inplace=True)
    plt.figure(figsize=(12,10))
    sns.barplot(x=percent_missing.columns[1], y=percent_missing.columns[0], data=percent_missing, palette='flare')
    plt.title(title)
    plt.xlabel(percent_missing.columns[1])
    plt.ylabel(percent_missing.columns[0])
    plt.show

def plot_missing_row(df):
    percent_missing= (df.isna().sum(axis=1) / df.shape[1])
    plt = percent_missing.hist()
    plt.set_xlabel('Percent Missing')
    plt.set_ylabel('Number of Rows')
```

Check Missing Data Per Feature

In [15]:
```
plot_missing_feat(filtered_gdf, "Missing Data Per Feature")
```

Check Missing Data Per Observation

In [16]:
```
plot_missing_row(filtered_gdf)
```

Target Variable Selection

Target 1: Injury Severity

In [17]:
```
injury_map = {"SUSPECTED SERIOUS INJURY": "SERIOUS OR FATAL INJURY", "FATAL INJURY": "SERIOUS OR FATAL INJURY",
              "SUSPECTED MINOR INJURY": "POSSIBLE OR MINOR INJURY", "POSSIBLE INJURY": "POSSIBLE OR MINOR INJURY"}
filtered_gdf["Injury Severity Remapped"] = filtered_gdf["Injury Severity"].replace(injury_map)
filtered_gdf["Injury Severity"].value_counts(normalize=True)
```

Out[17]:
```
NO APPARENT INJURY          0.820143
POSSIBLE INJURY             0.101654
SUSPECTED MINOR INJURY      0.069102
SUSPECTED SERIOUS INJURY    0.008208
FATAL INJURY                0.000893
Name: Injury Severity, dtype: float64
```

In [18]:
```
filtered_gdf["Injury Severity Remapped"].value_counts(normalize=True)
```

Out[18]:
```
NO APPARENT INJURY          0.820143
POSSIBLE OR MINOR INJURY    0.170756
SERIOUS OR FATAL INJURY     0.009101
Name: Injury Severity Remapped, dtype: float64
```

Target 2: Vehicle Damage Extent

In [19]:
```
damage_map = {"DISABLING": "DESTROYED OR DISABLING", "DESTROYED": "DESTROYED OR DISABLING",
              "NO DAMAGE": "NO DAMAGE OR SUPERFICIAL", "SUPERFICIAL": "NO DAMAGE OR SUPERFICIAL"}
filtered_gdf["Vehicle Damage Extent Remapped"] = filtered_gdf["Vehicle Damage Extent"].replace(damage_map)
filtered_gdf["Vehicle Damage Extent"].value_counts(normalize=True)
```

Out[19]:
```
DISABLING      0.377956
FUNCTIONAL     0.272333
SUPERFICIAL    0.265246
DESTROYED      0.046098
NO DAMAGE      0.037752
OTHER          0.000614
Name: Vehicle Damage Extent, dtype: float64
```

In [20]:
```
filtered_gdf["Vehicle Damage Extent Remapped"].value_counts(normalize=True)
```

Out[20]:
```
DESTROYED OR DISABLING      0.424054
NO DAMAGE OR SUPERFICIAL    0.302998
FUNCTIONAL                  0.272333
OTHER                       0.000614
Name: Vehicle Damage Extent Remapped, dtype: float64
```

Target 3: Number of Motorists Involved

In [21]:
```
num_motorist = filtered_gdf['Report Number'].value_counts().reset_index()
num_motorist.rename(columns={"index": "Report Number", "Report Number": "Num Motorist Involved"}, inplace=True)
filtered_gdf = filtered_gdf.merge(num_motorist, on='Report Number', how='left')
filtered_gdf["Num Motorist Involved"] = filtered_gdf["Num Motorist Involved"].apply(lambda x: "4+" if x >= 4 else str(x))
filtered_gdf["Num Motorist Involved"] = filtered_gdf["Num Motorist Involved"].astype(str)
```

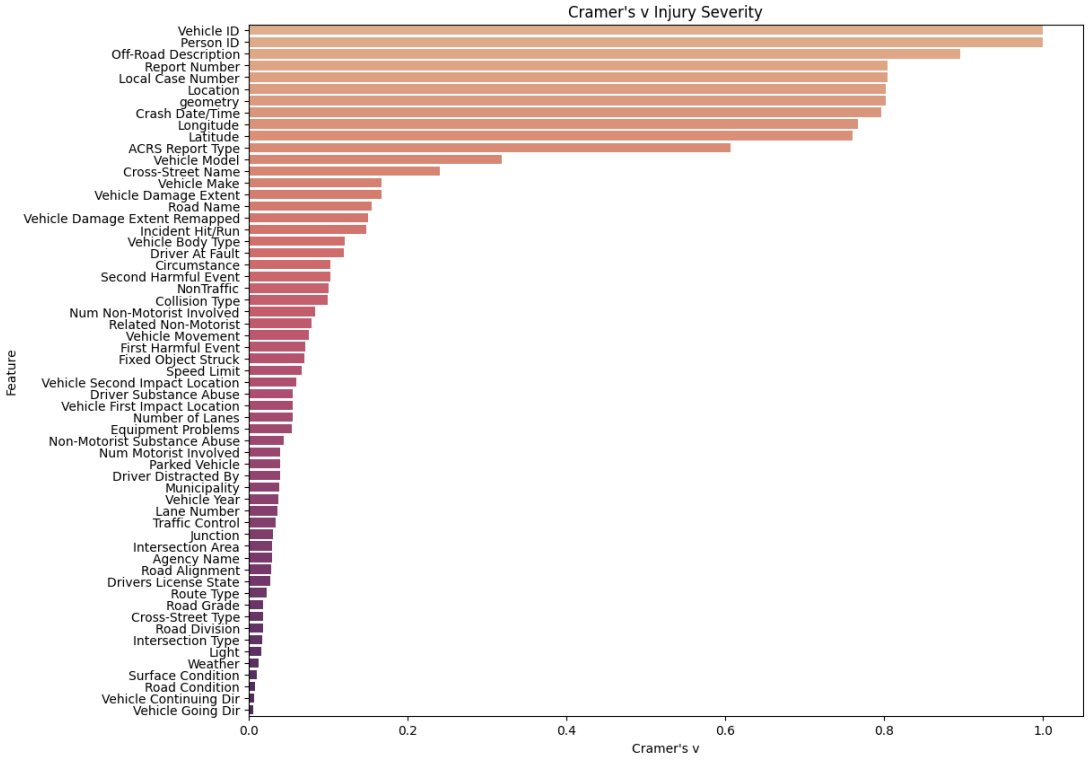
In [22]:
```
filtered_gdf["Num Motorist Involved"].value_counts(normalize=True)
```

Out[22]:
```
2     0.674402
1     0.180026
3     0.113760
4+    0.031812
Name: Num Motorist Involved, dtype: float64
```

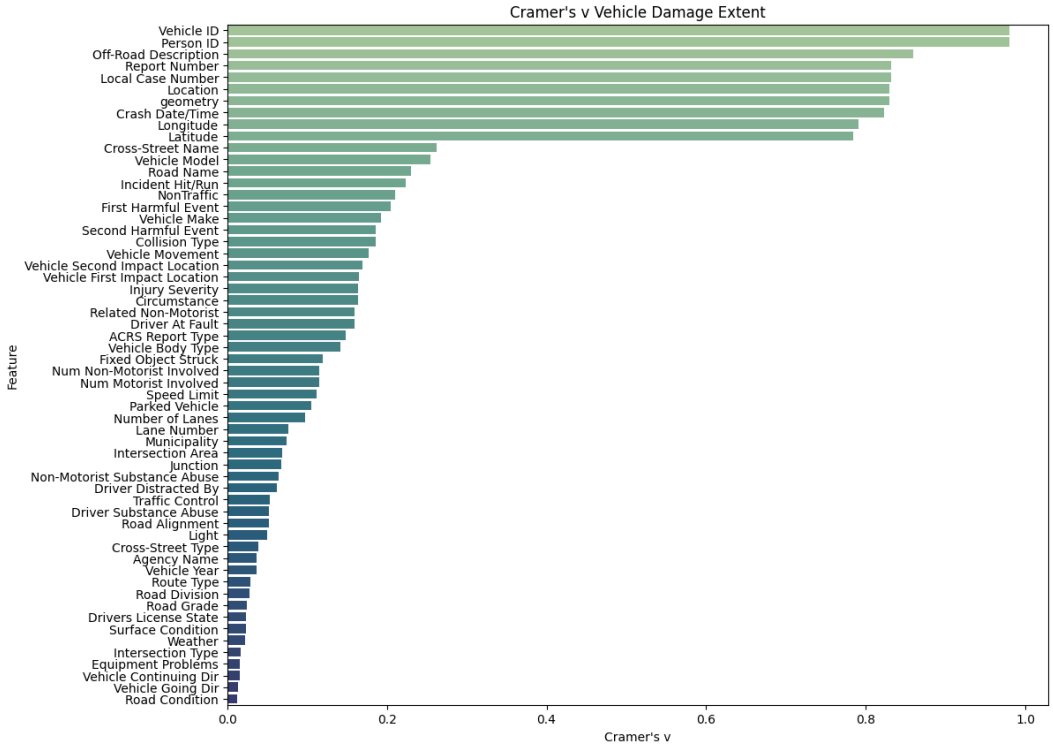
Exploratory Data Analysis

In [23]:
```
def get_cramers_v(df, target_name):
    cramers_v_df = pd.DataFrame(columns=["Feature", "Cramer's v"])

    for col in df.columns.tolist():
        if col != target_name:
            notna_df = df[df[col].notna()]
            contingency_table = pd.crosstab(notna_df[col], notna_df[target_name])
            chi2, _, _, _ = chi2_contingency(contingency_table)
            feature_shape = contingency_table.shape[0]
            target_shape = contingency_table.shape[1]
            n = notna_df.shape[0]
            cramers_v = np.sqrt(chi2 / (n * min(feature_shape-1, target_shape-1)))
            cramers_v_df = pd.concat([cramers_v_df, pd.DataFrame([{"Feature":col, "Cramer's v":cramers_v}])], ignore_index=True)
    cramers_v_df = cramers_v_df.sort_values(by="Cramer's v", ascending=False)
    return cramers_v_df

def plot_cramers_v(df, target_name, palette='flare'):
    plt.figure(figsize=(12,10))
    sns.barplot(x="Cramer's v", y="Feature", data=df, palette=palette)
    plt.title(f"Cramer's v {target_name}")
    plt.xlabel("Cramer's v")
    plt.ylabel("Feature")
    plt.show
```

In [24]:
```
tar1_cv_df = get_cramers_v(filtered_gdf.drop(columns=['Driverless Vehicle', "Injury Severity Remapped"]), "Injury Severity")
tar1_cv_df2 = get_cramers_v(filtered_gdf.drop(columns=['Driverless Vehicle', 'Injury Severity']), "Injury Severity Remapped")

plot_cramers_v(tar1_cv_df, "Injury Severity")
plot_cramers_v(tar1_cv_df2, "Injury Severity Remapped")
```



In [25]:
```
tar2_cv_df = get_cramers_v(filtered_gdf.drop(columns=['Driverless Vehicle', "Vehicle Damage Extent Remapped",
                        "Injury Severity Remapped"]), "Vehicle Damage Extent")
tar2_cv_df2 = get_cramers_v(filtered_gdf.drop(columns=['Driverless Vehicle','Vehicle Damage Extent',
                        "Injury Severity Remapped"]),"Vehicle Damage Extent Remapped")

plot_cramers_v(tar2_cv_df, "Vehicle Damage Extent", palette="crest")
plot_cramers_v(tar2_cv_df2, "Vehicle Damage Extent Remapped", palette="crest")
```



In [26]:
```
tar3_cv_df = get_cramers_v(filtered_gdf.drop(columns=['Driverless Vehicle', "Vehicle Damage Extent Remapped",
                        "Injury Severity Remapped"]), "Num Motorist Involved")
plot_cramers_v(tar3_cv_df, "Num Motorist Involved", palette="mako")
```

Feature Selection and Engineering

Feature Selection & Cleaning based on Cramer's v and amount of data missing

In [27]:
```
target1_feats = ["Vehicle Make", "Vehicle Body Type", "Incident Hit/Run", "Collision Type", "Route Type",
                 "Driver At Fault", "Vehicle Movement", "NonTraffic", "First Harmful Event", "Speed Limit",
                 "Vehicle Second Impact Location", "Number of Lanes", "Vehicle First Impact Location", "Lane Number"]
target2_feats = ["Collision Type", "Vehicle Make", "Vehicle Movement", "Incident Hit/Run", "NonTraffic", "Route Type",
                 "Vehicle Second Impact Location", "Vehicle First Impact Location", "Vehicle Body Type", "First Harmful Event",
                 "Driver At Fault", "Speed Limit", "Number of Lanes", "Parked Vehicle", "Lane Number", "Light"]
target3_feats = ["First Harmful Event", "Collision Type", "Driver At Fault", "NonTraffic", "Vehicle Movement",
                 "Number of Lanes", "Speed Limit", "Vehicle Second Impact Location", "Vehicle First Impact Location",
                 "Lane Number", "Vehicle Make", "Driver Substance Abuse", "Route Type", "Incident Hit/Run", "Light",
                 "Vehicle Body Type"]
```

In [28]:
```
feats_to_clean = set(target1_feats+target2_feats+target3_feats)
feats_to_clean
```

Out[28]:
```
{'Collision Type',
 'Driver At Fault',
 'Driver Substance Abuse',
 'First Harmful Event',
 'Incident Hit/Run',
 'Lane Number',
 'Light',
 'NonTraffic',
 'Number of Lanes',
 'Parked Vehicle',
 'Route Type',
 'Speed Limit',
 'Vehicle Body Type',
 'Vehicle First Impact Location',
 'Vehicle Make',
 'Vehicle Movement',
 'Vehicle Second Impact Location'}
```

In [29]:
```
clean_gdf = filtered_gdf.copy()
```

Convert to Date/Time object

In [30]:
```
clean_gdf['Crash Date/Time'] = pd.to_datetime(clean_gdf['Crash Date/Time'], format="%m/%d/%Y %I:%M:%S %p")
monthly_crashes = clean_gdf.resample('M', on='Crash Date/Time').size()
plt.figure(figsize=(12, 6))
monthly_crashes.plot()
plt.title('Crash Date/Time Distribution')
plt.xlabel('Month')
plt.ylabel('# of Driver Collisions')
plt.grid(True)
plt.show()
```

In [31]:
```
line_df = pd.DataFrame({
    'Date': clean_gdf['Crash Date/Time'],
    'Target': clean_gdf['Injury Severity Remapped']
})

monthly_counts = line_df.groupby([pd.Grouper(key='Date', freq='M'), 'Target']).size().unstack(fill_value=0)

plt.figure(figsize=(12, 6))
monthly_counts.drop(columns=['SERIOUS OR FATAL INJURY']).plot(kind='line', ax=plt.gca())
plt.title('Monthly Distribution of Injury Severity Remapped')
plt.xlabel('Month')
plt.ylabel('# of Occurrences')
plt.grid(True)
plt.legend(title='Target Categories')

plt.figure(figsize=(12, 6))
monthly_counts['SERIOUS OR FATAL INJURY'].plot(kind='line', color='red', ax=plt.gca())
plt.title('Monthly Distribution of Injury Severity Remapped (Serious or Fatal)')
plt.xlabel('Month')
plt.ylabel('# of Occurrences')
plt.grid(True)
plt.legend(['SERIOUS OR FATAL INJURY'], title='Target Categories')
plt.show()
```

In [32]:
```
line_df = pd.DataFrame({
    'Date': clean_gdf['Crash Date/Time'],
    'Target': clean_gdf['Vehicle Damage Extent Remapped']
})

monthly_counts = line_df.groupby([pd.Grouper(key='Date', freq='M'), 'Target']).size().unstack(fill_value=0)

plt.figure(figsize=(12, 6))
monthly_counts.drop(columns=['OTHER']).plot(kind='line', ax=plt.gca())
plt.title('Monthly Distribution of Vehicle Damage Extent Remapped')
plt.xlabel('Month')
plt.ylabel('# of Occurrences')
plt.grid(True)
plt.legend(title='Target Categories')
plt.show()
```

In [33]:
```
line_df = pd.DataFrame({
    'Date': clean_gdf['Crash Date/Time'],
    'Target': clean_gdf['Num Motorist Involved']
})

monthly_counts = line_df.groupby([pd.Grouper(key='Date', freq='M'), 'Target']).size().unstack(fill_value=0)

plt.figure(figsize=(12, 6))
monthly_counts.drop(columns=['2']).plot(kind='line', ax=plt.gca())
plt.title('Monthly Distribution of Num Motorist Involved')
plt.xlabel('Month')
plt.ylabel('# of Occurrences')
plt.grid(True)
plt.legend(title='Target Categories')

plt.figure(figsize=(12, 6))
monthly_counts['2'].plot(kind='line', color='red', ax=plt.gca())
plt.title('Monthly Distribution of Num Motorist Involved (2)')
plt.xlabel('Month')
plt.ylabel('# of Occurrences')
plt.grid(True)
plt.legend(['2'], title='Target Categories')
plt.show()
```

Standardize / Impute Booleans

In [34]:
```
bools = ['Driver At Fault', 'Incident Hit/Run', 'NonTraffic', 'Parked Vehicle']
clean_gdf[bools] = clean_gdf[bools].applymap({"Yes": True, "No":False}.get)
```

In [35]:
```
daf_distro = clean_gdf["Driver At Fault"].value_counts(normalize=True)
clean_gdf["Driver At Fault"].fillna(np.random.choice(daf_distro.index, p=daf_distro.values), inplace=True)
```

Clip numeric variables

In [36]:
```
num_var = ['Speed Limit', 'Vehicle Year', 'Lane Number', 'Number of Lanes', 'Num Non-Motorist Involved']
plt.figure(figsize=(20,10))

for i, var in enumerate(num_var):
    plt.subplot(2,3,1+i)
    plt.boxplot(clean_gdf[var].dropna())
    plt.xlabel(var,fontsize=15, weight='bold')
```

In [37]:
```
clean_gdf[['Number of Lanes', 'Lane Number']] = clean_gdf[['Number of Lanes', 'Lane Number']].clip(upper=8)
```

In [38]:
```
def clean_vehicle_yr(row, N=40):
    month = row['Crash Date/Time'].month
    max_yr = row['Crash Date/Time'].year
    vehicle_yr = int(row['Vehicle Year'])
    if month >= 10: # new model years happen in October
        max_yr += 1
    min_yr = max_yr - 40

    if (vehicle_yr > max_yr or vehicle_yr < min_yr):
        vehicle_yr = np.nan
    return vehicle_yr

condition = clean_gdf['Vehicle Year'].notna()
clean_gdf.loc[condition, 'Vehicle Year'] = clean_gdf.loc[condition].apply(clean_vehicle_yr, axis=1)
```

In [39]:
```
plt.figure(figsize=(20,10))
for i, var in enumerate(num_var):
    plt.subplot(2,3,1+i)
    plt.boxplot(clean_gdf[var].dropna())
    plt.xlabel(var,fontsize=15, weight='bold')
```

Standardize / Impute Categories

In [40]:
```
def mask(df, feature, old_cats, new_cat):
    df[feature] = df[feature].mask(df[feature].isin(old_cats), new_cat)
```

In [41]:
```
clean_gdf["Speed Limit"] = clean_gdf["Speed Limit"].astype(str)
```

In [42]:
```
recategorize = {"Driver Substance Abuse": {"ALCOHOL PRESENT OR CONTRIBUTED": ["ALCOHOL PRESENT", "ALCOHOL CONTRIBUTED"],
                                           "ILLEGAL DRUG PRESENT OR CONTRIBUTED": ["ILLEGAL DRUG PRESENT", "ILLEGAL DRUG CONTRIBUTED"],
                                           "MEDICATION PRESENT OR CONTRIBUTED": ["MEDICATION PRESENT", "MEDICATION CONTRIBUTED"],
                                           "COMBINED SUBSTANCE PRESENT OR CONTRIBUTED": ["COMBINED SUBSTANCE PRESENT", "COMBINATION CONTRIBUTED"],
                                           np.nan: ["OTHER"]},
                "First Harmful Event": {"NON-MOTORIST": ["PEDESTRIAN", "BICYCLE", "OTHER PEDALCYCLE", "ANIMAL"],
                                        "NON-FIXED OBJECT": ["OTHER OBJECT", "THROWN OR FALLING OBJECT", "SPILLED CARGO", "DOWNHILL RUNAWAY"],
                                        "VEHICLE MANEUVER/MALFUNCTION": ["OVERTURN", "BACKING", "U-TURN", "UNITS SEPARATED", "JACKKNIFE", "OTHER NON COLLISION"],
                                        "OTHER": ["OTHER CONVEYANCE", "FELL JUMPED FROM MOTOR VEHICLE", "RAILWAY TRAIN", "EXPLOSION OR FIRE", "IMMERSION"]},
                "Vehicle Movement": {"TURNING": ["MAKING LEFT TURN", "MAKING RIGHT TURN", "RIGHT TURN ON RED"],
                                     "PARKED/PARKING": ["PARKED", "PARKING", "STARTING FROM PARKED"],
                                     "MERGING": ["CHANGING LANES", "ENTERING TRAFFIC LANE", "LEAVING TRAFFIC LANE"],
                                     "OTHER": ["DRIVERLESS MOVING VEH."],
                                     "SKIDDING OR NEGOTIATING A CURVE": ["SKIDDING", "NEGOTIATING A CURVE"],
                                     "ACCELERATING/PASSING": ["ACCELERATING", "PASSING"]},
                "Vehicle Body Type": {"BUS": ["TRANSIT BUS", "SCHOOL BUS", "OTHER BUS", "CROSS COUNTRY BUS"],
                                      "PICKUP TRUCK": ["OTHER LIGHT TRUCKS (10,000LBS (4,536KG) OR LESS)"],
                                      "CARGO VAN": ["CARGO VAN/LIGHT TRUCK 2 AXLES (OVER 10,000LBS (4,536 KG))"],
                                      "SUV": ["(SPORT) UTILITY VEHICLE"],
                                      "MEDIUM/HEAVY TRUCKS": ["MEDIUM/HEAVY TRUCKS 3 AXLES (OVER 10,000LBS (4,536KG))", "TRUCK TRACTOR"],
                                      "PASSENGER CAR": ["STATION WAGON"],
                                      "OTHER": ["LIMOUSINE", "FARM VEHICLE", "LOW SPEED VEHICLE", "AUTOCYCLE", "ALL TERRAIN VEHICLE (ATV)", "SNOWMOBILE","RECREATIONAL VEHICLE"],
                                      "MOTORCYCLE": ["MOPED"],
                                      "POLICE VEHICLE": ["POLICE VEHICLE/EMERGENCY", "POLICE VEHICLE/NON EMERGENCY"],
                                      "AMBULANCE": ["AMBULANCE/EMERGENCY", "AMBULANCE/NON EMERGENCY"],
                                      "FIRE VEHICLE": ["FIRE VEHICLE/EMERGENCY", "FIRE VEHICLE/NON EMERGENCY"]},
                "Speed Limit": {"Below 25": ["0", "5", "10", "15", "20"],
                                "Above 55": ["60", "65", "70", "75"]}}
```

In [43]:
```
for feature, value in recategorize.items():
    for new_cat, old_cats in value.items():
        mask(clean_gdf, feature, old_cats, new_cat)
```

In [44]:
```
sub_distro = clean_gdf["Driver Substance Abuse"].value_counts(normalize=True)
clean_gdf["Driver Substance Abuse"].fillna(np.random.choice(sub_distro.index, p=sub_distro.values), inplace=True)
```

In [45]:
```
night = (clean_gdf["Crash Date/Time"].dt.hour >= 22) | (clean_gdf["Crash Date/Time"].dt.hour < 4)
day = (clean_gdf["Crash Date/Time"].dt.hour >= 9) | (clean_gdf["Crash Date/Time"].dt.hour < 16)
clean_gdf.loc[night & clean_gdf["Light"].isna(), "Light"] = "DARK -- UNKNOWN LIGHTING"
clean_gdf.loc[day & clean_gdf["Light"].isna(), "Light"] = "DAYLIGHT"
```

In [46]:
```
def fuzzy_compare(input_cat, cats, max_thr=50, min_thr=20):
    max_sim = 0
    match = None

    for cat in cats:
        sim = fuzz.ratio(input_cat, cat)
        if sim > max_sim:
            max_sim = sim
            match = cat
    if max_sim > max_thr:
        return match
    elif max_sim > min_thr:
        return "OTHER"
    else:
        return "UNKNOWN"
```

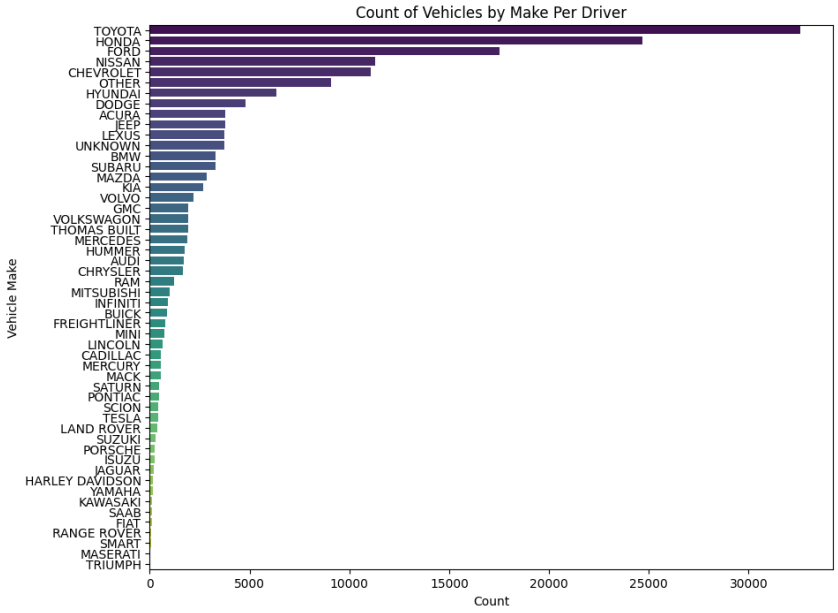
In [47]:
```
%%time

vehicle_makes = ["TOYOTA", "HONDA", "FORD", "NISSAN", "DODGE", "RAM", "HYUNDAI", "CHEVROLET", "JEEP", "BMW", "ACURA", "LEXUS",
        "KIA", "SUBARU", "MAZDA", "GMC", "MERCEDES", "AUDI", "CHRYSLER", "VOLVO", "VOLKSWAGON", "INFINITI", "BUICK",
        "MITSUBISHI", "MACK", "FREIGHTLINER", "LINCOLN", "SCION", "SATURN", "TESLA", "MINI", "PONTIAC", "MERCURY", "SUZUKI",
        "ISUZU", "LAND ROVER", "PORSCHE", "JAGUAR", "HARLEY DAVIDSON", "YAMAHA", "FIAT", "SAAB", "THOMAS BUILT", "KAWASAKI",
        "RANGE ROVER", "HUMMER", "MASERATI", "SMART", "TRIUMPH", "CADILLAC", "UNKNOWN"]


clean_gdf["Vehicle Make"] = clean_gdf["Vehicle Make"].apply(lambda x: fuzzy_compare(str(x), vehicle_makes, 50))
```

```
CPU times: total: 19.9 s
Wall time: 20.1 s
```

In [48]:
```
def remap_vbt_to_vmake(df, body_type_cat, possible_makes):
    mask = (df["Vehicle Body Type"] == body_type_cat) & (~df["Vehicle Make"].isin(possible_makes))
    make_distro = df[(df["Vehicle Body Type"] == body_type_cat) & (df["Vehicle Make"].isin(
                                                    possible_makes))]["Vehicle Make"].value_counts(normalize=True)
    df.loc[mask, "Vehicle Make"] = df.loc[mask, "Vehicle Make"].apply(lambda x: np.random.choice(make_distro.index,
                                                                                p=make_distro.values))
def remap_vmake_to_vbt(df, make_cat, possible_body_types):
    mask = (df["Vehicle Make"] == make_cat) & (df["Vehicle Body Type"].isin(possible_body_types))
    bt_distro = df[(df["Vehicle Make"] == make_cat) & (~df["Vehicle Body Type"].isin(
                                        possible_body_types))]["Vehicle Body Type"].value_counts(normalize=True)
    df.loc[mask, "Vehicle Body Type"] = df.loc[mask, "Vehicle Body Type"].apply(lambda x: np.random.choice(bt_distro.index,
                                                                                p=bt_distro.values))
```

In [49]:
```
clean_gdf.loc[clean_gdf["Vehicle Body Type"] == "PICKUP TRUCK", "Vehicle Make"] = (
clean_gdf.loc[clean_gdf["Vehicle Body Type"] == "PICKUP TRUCK", "Vehicle Make"].replace("DODGE", "RAM"))

vbt_to_vmake_map = {"AMBULANCE": ["OTHER", "FREIGHTLINER", "FORD"],
                    "BUS": ["OTHER", "THOMAS BUILT", "FREIGHTLINER"],
                    "CARGO VAN": ["FORD", "CHEVROLET", "OTHER", "GMC", "FREIGHTLINER", "DODGE", "RAM", "MERCEDES"],
                    "FIRE VEHICLE": ["OTHER"],
                    "MEDIUM/HEAVY TRUCKS": ["OTHER", "MACK", "FREIGHTLINER"],
                    "MOTORCYCLE": ["HONDA", "HARLEY DAVIDSON", "YAMAHA", "SUZUKI", "KAWASAKI", "OTHER", "BMW", "TRIUMPH"],
                    "POLICE VEHICLE": ["FORD", "DODGE", "CHEVROLET", "HONDA", "BMW", "HARLEY DAVIDSON", "OTHER"],
                    "VAN": ["FORD", "HONDA", "CHEVROLET", "TOYOTA", "DODGE", "CHRYSLER", "NISSAN", "GMC", "RAM", "KIA",
                            "OTHER", "MERCEDES", "MAZDA"],
                    "PICKUP TRUCK": ["FORD", "CHEVROLET", "TOYOTA", "RAM", "GMC", "NISSAN", "OTHER", "HONDA", "JEEP"],
                    "SUV": ["TOYOTA", "HONDA", "FORD", "JEEP", "NISSAN", "CHEVROLET", "LEXUS", "ACURA", "HYUNDAI", "GMC",
                            "SUBARU", "BMW", "KIA", "MAZDA", "HUMMER", "MERCEDES", "DODGE", "AUDI", "VOLVO", "LAND ROVER",
                            "MITSUBISHI", "CADILLAC", "INFINITI", "VOLKSWAGON", "BUICK", "LINCOLN", "PORSCHE", "SATURN",
                            "MERCURY", "ISUZU", "OTHER", "CHRYSLER", "RANGE ROVER", "SUZUKI", "TESLA", "SCION", "JAGUAR",
                            "MASERATI"]}

for key, value in vbt_to_vmake_map.items():
    remap_vbt_to_vmake(clean_gdf, key, value)

vmake_to_vbt_map = {"FREIGHTLINER": ["UNKNOWN", "PASSENGER CAR"],
                    "HARLEY DAVIDSON":["PASSENGER CAR"],
                    "KAWASAKI": ["PASSENGER CAR", "OTHER"],
                    "MACK": ["UNKNOWN", "PASSENGER CAR"],
                    "MINI": ["OTHER", "UNKNOWN"],
                    "SMART": ["OTHER"],
                    "THOMAS BUILT": ["OTHER", "PASSENGER CAR", "UNKNOWN"],
                    "TRIUMPH": ["OTHER"],
                    "YAMAHA": ["PASSENGER CAR", "OTHER"]}

for key, value in vmake_to_vbt_map.items():
    remap_vmake_to_vbt(clean_gdf, key, value)


all_known_makes = clean_gdf["Vehicle Make"].unique().tolist()
all_known_makes.remove("UNKNOWN")

for make in all_known_makes:
    remap_vmake_to_vbt(clean_gdf, make, ["UNKNOWN"])
```

In [50]:
```
vehicle_make_counts = clean_gdf['Vehicle Make'].value_counts()
vehicle_make_counts_df = vehicle_make_counts.reset_index()
vehicle_make_counts_df.columns = ['Vehicle Make', 'Count']

plt.figure(figsize=(10, 8))
sns.barplot(x='Count', y='Vehicle Make', data=vehicle_make_counts_df, palette="viridis")
plt.xlabel('Count')
plt.ylabel('Vehicle Make')
plt.title('Count of Vehicles by Make Per Driver')
plt.show()
```



In [51]:
```
"""https://data.imap.maryland.gov/datasets/montgomery-county-maintained-roads/explore"""
county_roads_gdf = gpd.read_file("../Datasets/Montgomery_County_Maintained_Roads.geojson")

"""https://data.imap.maryland.gov/datasets/maryland-routes-with-zip-code-interstates-md-state-us-state/explore?location=39.026218%2C-77.098585%2C12.88"""
interstate_roads_gdf = gpd.read_file("../Datasets/Maryland_Routes_with_ZIP_Code_Interstates.geojson")

"""https://data.imap.maryland.gov/datasets/5a3ca5b40d8340a990a6351ecfde709c_0/explore?location=39.118032%2C-76.740034%2C9.77"""
municipal_road_gdf = gpd.read_file("../Datasets/Municipal_Maintained_Roads.geojson")
```

In [52]:
```
def check_interesection(poly, gdf, gdf_sindex, route_type):
    for i in gdf_sindex.intersection(poly.bounds):
        if gdf.iloc[i]["geometry"].intersects(poly):
            return route_type
```

In [53]:
```
%%time
mask = clean_gdf["Route Type"].isna()

for idx, poly in clean_gdf.loc[mask, "geometry"].items():
    route_type = check_interesection(poly.buffer(0.0003), county_roads_gdf, county_roads_gdf.sindex, "County")
    if route_type is not None:
        clean_gdf.at[idx, "Route Type"] = route_type
        continue

    route_type = check_interesection(poly.buffer(0.0007), municipal_road_gdf, municipal_road_gdf.sindex, "Municipality")
    if route_type is not None:
        clean_gdf.at[idx, "Route Type"] = route_type
        continue

    route_type = check_interesection(poly.buffer(0.001), interstate_roads_gdf, interstate_roads_gdf.sindex, "Interstate (State)")
    if route_type is not None:
        clean_gdf.at[idx, "Route Type"] = route_type
```

```
CPU times: total: 14.5 s
Wall time: 14.6 s
```

Feature Engineering

In [54]:
```
"""https://data-mcplanning.hub.arcgis.com/datasets/3121d8a525904c75848197c51142d761_6/explore?"""
taz_gdf = gpd.read_file("../Datasets/Traffic_Analysis_Zones.geojson")
```

In [55]:
```
%%time
clean_gdf["Traffic Analysis Zone"] = np.nan
clean_gdf["Policy Name"] = np.nan
for i in range(taz_gdf.shape[0]):
    condition = clean_gdf.geometry.within(taz_gdf.iloc[i].geometry)
    clean_gdf["Traffic Analysis Zone"] = clean_gdf["Traffic Analysis Zone"].mask(condition, taz_gdf.iloc[i]["TAZ"])
    clean_gdf["Policy Name"] = clean_gdf["Policy Name"].mask(condition, taz_gdf.iloc[i]["POLICY_NAME"])
```

```
CPU times: total: 18.4 s
Wall time: 18.5 s
```

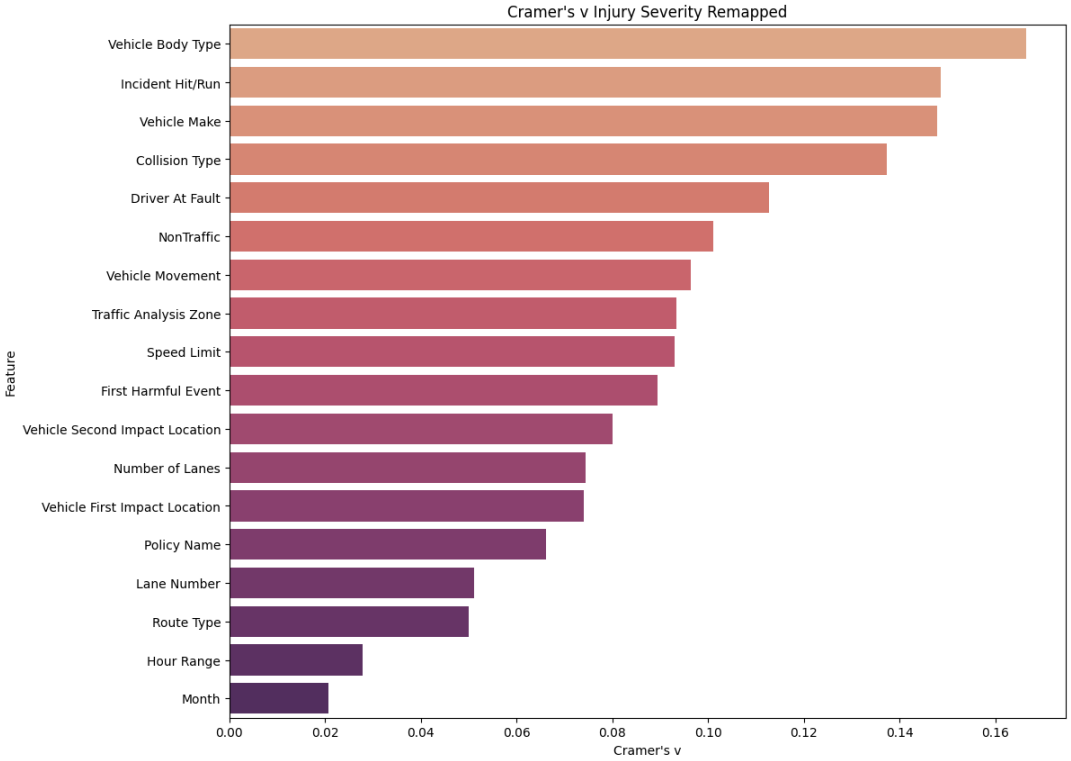
In [56]:
```
clean_gdf["Month"] = clean_gdf["Crash Date/Time"].dt.month_name()

hour_bins = pd.cut(clean_gdf["Crash Date/Time"].dt.hour, bins=range(0, 25), right=False, labels=False)
hour_ranges = [f"{i:02d}:00:00 - {i:02d}:59:59" for i in range(24)]
clean_gdf['Hour Range'] = pd.cut(clean_gdf["Crash Date/Time"].dt.hour, bins=range(0, 25), right=False, labels=hour_ranges)
```

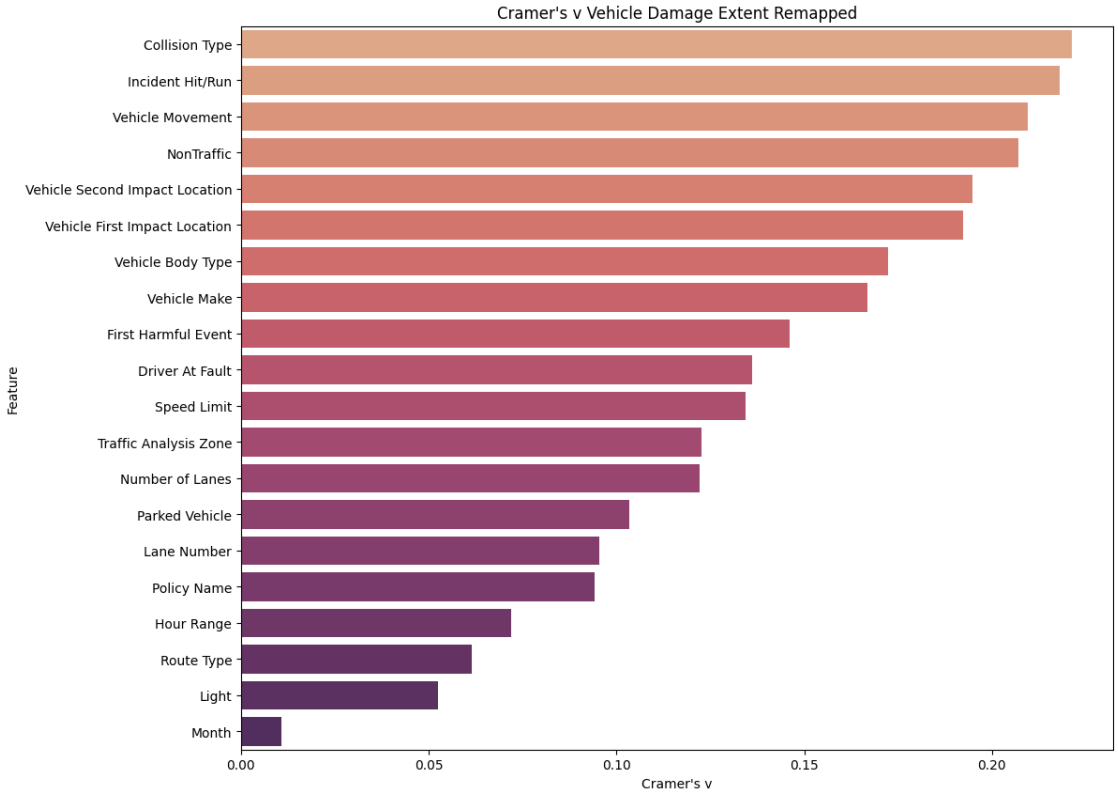
In [57]:
```
added_feats = ["Traffic Analysis Zone", "Policy Name", "Month", 'Hour Range']
plot_missing_feat(clean_gdf.loc[:, list(feats_to_clean)+added_feats],
                  "Missing Data Per Feature")
```

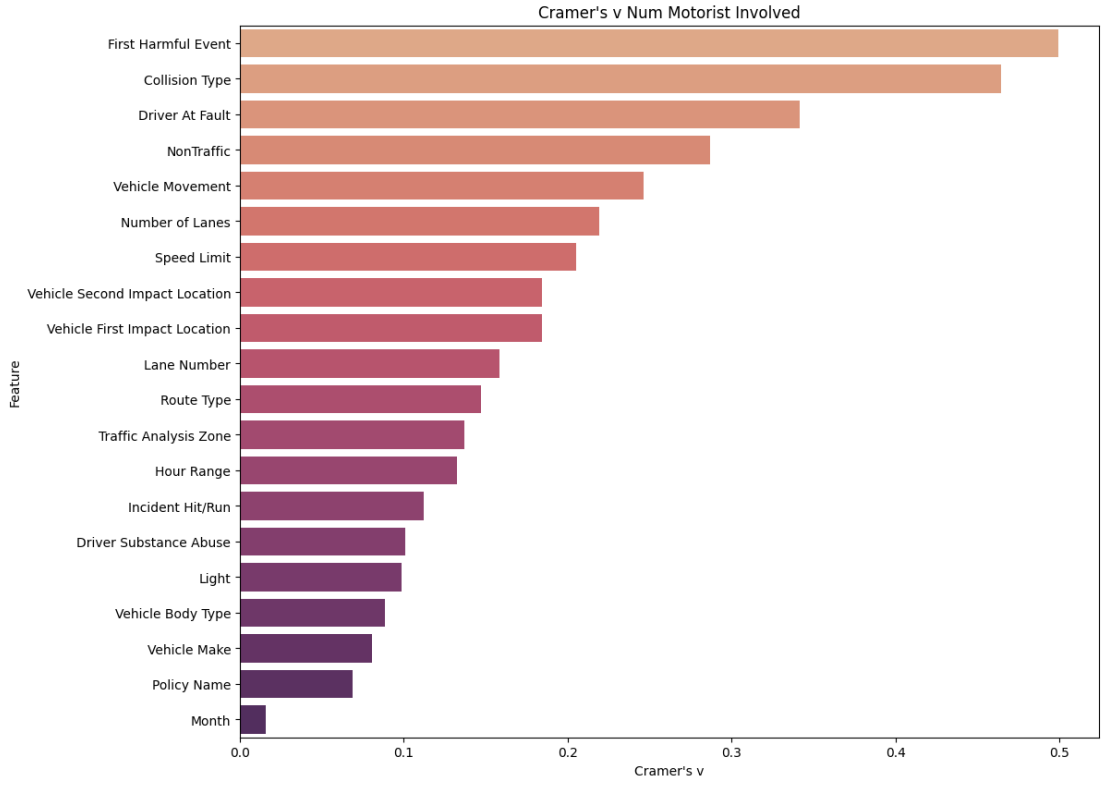
In [58]:
```
clean_gdf.to_csv("../Datasets/prepared_data.csv", index=False)
```

Final Cramer's v

In [59]:
```
tar1_cv_df_final = get_cramers_v(clean_gdf[target1_feats+added_feats+["Injury Severity Remapped"]], "Injury Severity Remapped")
plot_cramers_v(tar1_cv_df_final, "Injury Severity Remapped")

tar2_cv_df_final = get_cramers_v(clean_gdf[target2_feats+added_feats+['Vehicle Damage Extent Remapped']],
                                 "Vehicle Damage Extent Remapped")
plot_cramers_v(tar2_cv_df_final, "Vehicle Damage Extent Remapped")

tar3_cv_df_final = get_cramers_v(clean_gdf[target3_feats+added_feats+['Num Motorist Involved']], "Num Motorist Involved")
plot_cramers_v(tar3_cv_df_final, "Num Motorist Involved")
```




Cramer's v Vehicle Damage Extent Remapped
Collision Type
Incident Hit/Run
Vehicle Movement
NonTraffic
Vehicle Second Impact Location
Vehicle First Impact Location
Vehicle Body Type
Vehicle Make
First Harmful Event
Driver At Fault
Speed Limit
Traffic Analysis Zone
Number of Lanes
Parked Vehicle
Lane Number
Policy Name
Hour Range
Route Type
Light
Month
Feature
0.00
0.05
0.10
0.15
0.20
Cramer's v


Cramer's v Num Motorist Involved
First Harmful Event
Collision Type
Driver At Fault
NonTraffic
Vehicle Movement
Number of Lanes
Speed Limit
Vehicle Second Impact Location
Vehicle First Impact Location
Lane Number
Route Type
Traffic Analysis Zone
Hour Range
Incident Hit/Run
Driver Substance Abuse
Light
Vehicle Body Type
Vehicle Make
Policy Name
Month
Feature
0.0
0.1
0.2
0.3
0.4
0.5
Cramer's v

Other Visualizations

In [60]:
```
def plot_scatter_map(cat_colors, target, title):
    traces = []
    for category, color in cat_colors.items():
        trace = go.Scattermapbox(lat=clean_gdf[clean_gdf[target] == category]['Latitude'],
                                 lon=clean_gdf[clean_gdf[target] == category]['Longitude'], mode='markers',
                                 marker=dict(size=10, color=color, opacity=0.7), name=category)
        traces.append(trace)

    layout = go.Layout(title=title, mapbox=dict(style="open-street-map", zoom=9.5, center = dict(lat = 39.128, lon = -77.202)),
                       autosize=False, width=1000, height=800)

    fig = go.Figure(data=traces, layout=layout)
    fig.show()
```

In [61]:
```
injury_colors = {
    'NO APPARENT INJURY': 'green',
    'POSSIBLE OR MINOR INJURY': 'blue',
    'SERIOUS OR FATAL INJURY': 'red'
}
plot_scatter_map(injury_colors, 'Injury Severity Remapped', 'Injury Severity Scatter Plot')
```

In [62]:
```
damage_colors = {
    'DESTROYED OR DISABLING': 'red',
    'NO DAMAGE OR SUPERFICIAL': 'green',
    'FUNCTIONAL': 'blue'
}
plot_scatter_map(damage_colors, 'Vehicle Damage Extent Remapped', 'Vehicle Damage Geo-Scatter Plot')
```

In [63]:
```
damage_colors = {
    '1': 'green',
    '2': 'blue',
    '3': 'orange',
    '4+': 'red'
}
plot_scatter_map(damage_colors, 'Num Motorist Involved', 'Number of Motorists Per Accident Geo-Scatter Plot')
```

Target 1 – Injury Severity Notebook

Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from scipy.stats import chi2_contingency
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
```

Initializing Data

In [2]:
```
driver_dtypes = {'Local Case Number': "string", "Num Motorist Involved": "string"}
df = pd.read_csv("../Datasets/prepared_data.csv", dtype=driver_dtypes)

target1 = ["Injury Severity Remapped"]
target1_feats = ["Vehicle Body Type", "Incident Hit/Run", "Vehicle Make", "Collision Type", "Driver At Fault", "NonTraffic",
                 "Vehicle Movement", "Traffic Analysis Zone", "Speed Limit", "First Harmful Event", "Vehicle Second Impact Location",
                 "Number of Lanes", "Vehicle First Impact Location", "Policy Name", "Lane Number", "Route Type", "Hour Range",
                 "Month"]

target1_df = df[target1+target1_feats]
```

Checking Distribution

In [3]:
```
target1_df[target1].value_counts(normalize=True)
```

Out[3]:
```
Injury Severity Remapped
NO APPARENT INJURY          0.820143
POSSIBLE OR MINOR INJURY    0.170756
SERIOUS OR FATAL INJURY     0.009101
dtype: float64
```

In [4]:
```
target1_df.shape
```

Out[4]:
```
(171414, 19)
```

In [5]:
```
target1_df = target1_df.dropna()
target1_df[target1].value_counts(normalize=True)
```

Out[5]:
```
Injury Severity Remapped
NO APPARENT INJURY          0.810285
POSSIBLE OR MINOR INJURY    0.180473
SERIOUS OR FATAL INJURY     0.009242
dtype: float64
```

In [6]:
```
target1_df.reset_index(inplace=True, drop=True)
target1_df.shape
```

Out[6]:
```
(155813, 19)
```

In [7]:
```
def test_normality(distro):
    stat, p = stats.shapiro(distro)
    alpha = 0.05

    if p > alpha:
        print("p =",p)
        print("Normal Distribution (Fail to Reject Null Hypothesis)")
    else:
        print("Not Normal Distribution (Reject Null Hypothesis)")
```

In [8]:
```
test_normality(target1_df[target1].value_counts(normalize=True).tolist())
```

```
p = 0.3903374969959259
Normal Distribution (Fail to Reject Null Hypothesis)
```

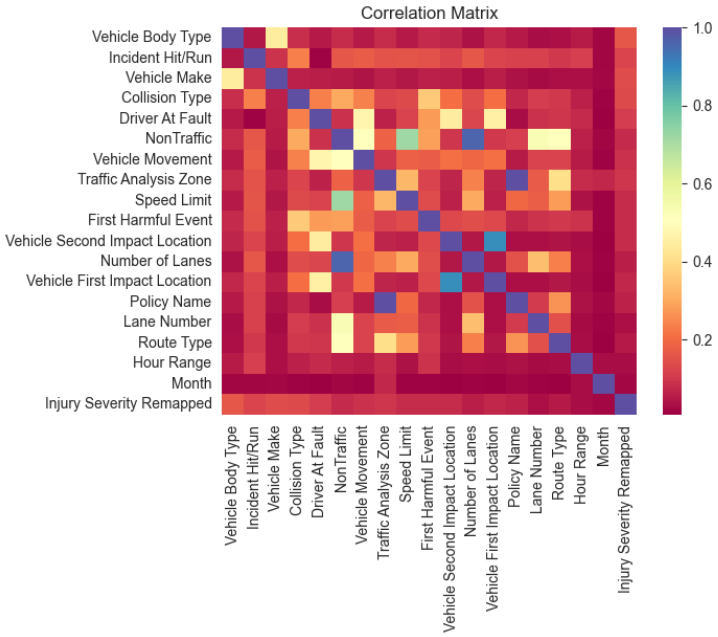
Test for Multicolinearity

In [9]:
```
corr = pd.DataFrame(index=target1_feats+target1, columns=target1_feats+target1)
```

In [10]:
```
def get_cramers_v(idx, col, n):
    contingency_table = pd.crosstab(idx, col)
    chi2, _, _, _ = chi2_contingency(contingency_table)
    feature_shape = contingency_table.shape[0]
    target_shape = contingency_table.shape[1]
    cramers_v = np.sqrt(chi2 / (n * min(feature_shape-1, target_shape-1)))
    return cramers_v
```

In [11]:
```
for idx in corr.index:
    for col in corr.columns:
        corr.at[idx, col] = get_cramers_v(target1_df[idx], target1_df[col], target1_df.shape[0])
```

In [67]:
```
sns.heatmap(corr.astype(float), cmap=sns.color_palette("Spectral", as_cmap=True))
plt.title("Correlation Matrix")
plt.show()
```



Variable Encoding

In [13]:
```
target1_df = target1_df.drop(columns=["Traffic Analysis Zone", "Number of Lanes", "Vehicle Second Impact Location"])
```

In [14]:
```
binary_feats = ["Incident Hit/Run", "Driver At Fault", "NonTraffic"]
ordinal_feats = ["Speed Limit", "Lane Number", "Vehicle First Impact Location", "Hour Range", "Month"]
nominal_feats = ["Vehicle Body Type", "Vehicle Make", "Collision Type", "Vehicle Movement", "First Harmful Event",
                 "Policy Name", "Route Type"]
```

In [15]:
```
model_df = pd.get_dummies(target1_df, columns=nominal_feats)
```

In [16]:
```
for col in binary_feats + ordinal_feats:
    model_df[col] = model_df[col].astype('category').cat.codes
```

Classifiers

Parameter Selections

In [17]:
```
def gridSearch(clf, params, df, n_rows, score='f1_weighted', is_y_np=False):
    df = df[: n_rows]

    X_n = df.drop(columns=["Injury Severity Remapped"])
    y_n = df["Injury Severity Remapped"]
    y_np = df["Injury Severity Remapped"].values
    y_np = LabelEncoder().fit_transform(y_np)

    grid_search = GridSearchCV(clf, params, cv=10, scoring=score)

    if is_y_np:
        grid_search.fit(X_n, y_np)
    else:
        grid_search.fit(X_n, y_n)

    return grid_search
```

In [18]:
```
%%time
rf_params = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_depth': [None, 5, 10, 15, 20, 25],
    'min_samples_split': [2, 5, 10],
    'class_weight': ['balanced']
}

gs_rf = gridSearch(RandomForestClassifier(), rf_params, model_df, int(model_df.shape[0]*.10))
print("Best Random Forest parameters found: ", gs_rf.best_params_)
print("Best Random Forest f1_weighted found: ", gs_rf.best_score_)
```

```
Best Random Forest parameters found:  {'class_weight': 'balanced', 'max_depth': None, 'min_samples_split': 10, 'n_estimators': 100}
Best Random Forest f1_weighted found:  0.7575096788214324
CPU times: total: 43min 14s
Wall time: 48min 11s
```

In [19]:
```
%%time
bernNB_params = {'fit_prior': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_bernNB = gridSearch(BernoulliNB(), bernNB_params, model_df.loc[:, ~model_df.columns.isin(ordinal_feats)], model_df.shape[0])
print("Best BernoulliNB parameters found: ", gs_bernNB.best_params_)
print("Best BernoulliNB f1_weighted found: ", gs_bernNB.best_score_)
```

```
Best BernoulliNB parameters found:  {'alpha': 25, 'fit_prior': True}
Best BernoulliNB f1_weighted found:  0.7492952999909017
CPU times: total: 4min 8s
Wall time: 4min 45s
```

In [20]:
```
%%time
compNB_params = {'norm': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_compNB = gridSearch(ComplementNB(), compNB_params, model_df, model_df.shape[0])
print("Best ComplementNB parameters found: ", gs_compNB.best_params_)
print("Best ComplementNB f1_weighted found: ", gs_compNB.best_score_)
```

```
Best ComplementNB parameters found:  {'alpha': 50, 'norm': True}
Best ComplementNB f1_weighted found:  0.7489156872382164
CPU times: total: 3min 43s
Wall time: 4min 12s
```

In [21]:
```
%%time
xgb_params = {'booster': ['gbtree'],
              'eta': [0.01, 0.05, 0.1],
              'max_depth': [3, 6, 9],
              'min_child_weight': [0.25, 0.5, 1, 5],
              'lambda': [0.1, 1.0, 10.0],
              'alpha': [0.0, 0.1, 0.5],
              'objective': ['multi:softmax'],
              'num_class': [4],
              'tree_method':['hist']}

gs_xgb = gridSearch(xgb.XGBClassifier(), xgb_params, model_df, int(model_df.shape[0]*.10), is_y_np=True)
print("Best XGBoost parameters found: ", gs_xgb.best_params_)
print("Best XGBoost f1_weighted found: ", gs_xgb.best_score_)
```

```
Best XGBoost parameters found:  {'alpha': 0.5, 'booster': 'gbtree', 'eta': 0.1, 'lambda': 0.1, 'max_depth': 9, 'min_child_weight': 1, 'num_class': 4, 'o
bjective': 'multi:softmax', 'tree_method': 'hist'}
Best XGBoost f1_weighted found:  0.7438776182670981
CPU times: total: 10h 58min 4s
Wall time: 58min 57s
```

Model Evaluations with Selected Parameters

In [50]:
```
classifiers = ["Random Forest", "BernoulliNB", "ComplementNB", "XGBoost"]
accs = []
f1s = []
```

In [23]:
```
def plot_cm(df, title):
    plt.figure(figsize=(10, 6))
    sns.heatmap(df, cmap = 'viridis', annot=True, fmt="d", square=True, linewidths=.5)
    plt.xlabel("Predicted Values", fontsize=10)
    plt.ylabel("Actual Values", fontsize=10)
    plt.xticks(wrap=True, fontsize=7, rotation=0)
    plt.yticks(wrap=True, fontsize=7)
    plt.title(title)
    plt.show()
```

In [24]:
```
def kfold_eval(clf, X, y):
    acc = []
    f1_weighted = []
    cm = []
    kf = StratifiedKFold(n_splits=10, shuffle=False)
    for train, test in kf.split(X, y):
        clf.fit(X.iloc[train], y[train])
        y_pred = clf.predict(X.iloc[test])
        class_names = sorted(set(y[test]) | set(y_pred))
        acc += [accuracy_score(y[test], y_pred)]
        f1_weighted += [f1_score(y[test], y_pred, average='weighted')]
        cm += [pd.DataFrame(confusion_matrix(y[test], y_pred, labels=class_names), index=class_names, columns=class_names)]
    return acc, f1_weighted, cm
```

In [25]:
```
X = model_df.drop(columns=["Injury Severity Remapped"])
y = model_df["Injury Severity Remapped"]

X_np = model_df.drop(columns=["Injury Severity Remapped"]).values
y_np = model_df["Injury Severity Remapped"].values
y_np = LabelEncoder().fit_transform(y_np)
```

In [51]:
```
%%time
rf = RandomForestClassifier(**gs_rf.best_params_)

acc, f1_weighted, cm = kfold_eval(rf, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'Random Forest CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'Random Forest CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
Random Forest CV accuracy=0.771 0.006
Random Forest CV f1_weighted=0.755 0.004
CPU times: total: 4min 46s
Wall time: 5min 18s
```

In [52]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'Random Forest CV Confusion Matrix for Fold with Best f1_weighted')
```

In [53]:
```
%%time
bernNB = BernoulliNB(**gs_bernNB.best_params_)

acc, f1_weighted, cm = kfold_eval(bernNB, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'BernoulliNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'BernoulliNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
BernoulliNB CV accuracy=0.793 0.003
BernoulliNB CV f1_weighted=0.750 0.003
CPU times: total: 10 s
Wall time: 11 s
```

In [54]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'BernoulliNB CV Confusion Matrix for Fold with Best F1_Weighted')
```

In [55]:
```
%%time
compNB = ComplementNB(**gs_compNB.best_params_)

acc, f1_weighted, cm = kfold_eval(compNB, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'ComplementNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'ComplementNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
ComplementNB CV accuracy=0.759 0.006
ComplementNB CV f1_weighted=0.749 0.004
CPU times: total: 8.78 s
Wall time: 9.77 s
```

In [56]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'ComplementNB CV Confusion Matrix for Fold with Best f1_weighted')
```

In [57]:
```
%%time
xgb_clf = xgb.XGBClassifier(**gs_xgb.best_params_)

acc, f1_weighted, cm = kfold_eval(xgb_clf, X, y_np)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'XGBoost CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'XGBoost CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
XGBoost CV accuracy=0.812 0.001
XGBoost CV f1_weighted=0.736 0.002
CPU times: total: 11min 3s
Wall time: 1min 1s
```

In [66]:
```
best_cm = cm[f1_weighted.index(max(f1_weighted))]
best_cm.index = target1_df['Injury Severity Remapped'].unique().tolist()
best_cm.columns = target1_df['Injury Severity Remapped'].unique().tolist()
plot_cm(best_cm, 'XGBoost CV Confusion Matrix for Fold with Best f1_weighted')
```

In [60]:
```
sns.set_style("whitegrid")
sns.set_palette("pastel")

data = {
    'Classifier': classifiers,
    'Accuracy': accs,
    'F1_weighted': f1s
}

results = pd.DataFrame(data)
results = pd.melt(results, id_vars='Classifier', var_name='Metric', value_name='Value')

plt.figure(figsize=(10, 6))
sns.barplot(data=results, x='Classifier', y='Value', hue='Metric')
plt.title('Classifier Performance Comparison - Injury Severity Remapped')
plt.ylabel('Score')
plt.xlabel('Classifier')
plt.ylim(0, 1)
plt.legend(title='Metric')
plt.show()
```

Target 2 – Vehicle Damage Notebook

Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.metrics import f1_score
from scipy.stats import chi2_contingency
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
```

Initializing Data

In [2]:
```
driver_dtypes = {'Local Case Number': "string", "Num Motorist Involved": "string"}
df = pd.read_csv("../Datasets/prepared_data.csv", dtype=driver_dtypes)

target2 = ["Vehicle Damage Extent Remapped"]
target2_feats = ["Collision Type", "Vehicle Make", "Vehicle Movement", "Incident Hit/Run", "NonTraffic", "Route Type",
                 "Vehicle Second Impact Location", "Vehicle First Impact Location", "Vehicle Body Type", "First Harmful Event",
                 "Driver At Fault", "Speed Limit", "Number of Lanes", "Parked Vehicle", "Lane Number", "Light",
                 "Traffic Analysis Zone", "Policy Name", "Hour Range"]

target2_df = df[target2+target2_feats]
```

Checking Distribution

In [3]:
```
target2_df = target2_df[target2_df[target2[0]] != "OTHER"]
target2_df[target2].value_counts(normalize=True)
```

Out[3]:
```
Vehicle Damage Extent Remapped
DESTROYED OR DISABLING            0.424315
NO DAMAGE OR SUPERFICIAL          0.303185
FUNCTIONAL                        0.272501
dtype: float64
```

In [4]:
```
target2_df.shape
```

Out[4]:
```
(171313, 20)
```

In [5]:
```
target2_df = target2_df.dropna()
target2_df[target2].value_counts(normalize=True)
```

Out[5]:
```
Vehicle Damage Extent Remapped
DESTROYED OR DISABLING            0.437819
NO DAMAGE OR SUPERFICIAL          0.289768
FUNCTIONAL                        0.272413
dtype: float64
```

In [6]:
```
target2_df.reset_index(inplace=True, drop=True)
target2_df.shape
```

Out[6]:
```
(153271, 20)
```

In [7]:
```
def test_normality(distro):
    stat, p = stats.shapiro(distro)
    alpha = 0.05

    if p > alpha:
        print("p =",p)
        print("Normal Distribution (Fail to Reject Null Hypothesis)")
    else:
        print("Not Normal Distribution (Reject Null Hypothesis)")
```

In [8]:
```
test_normality(target2_df[target2].value_counts(normalize=True).tolist())
```

```
p = 0.1825898140668869
Normal Distribution (Fail to Reject Null Hypothesis)
```

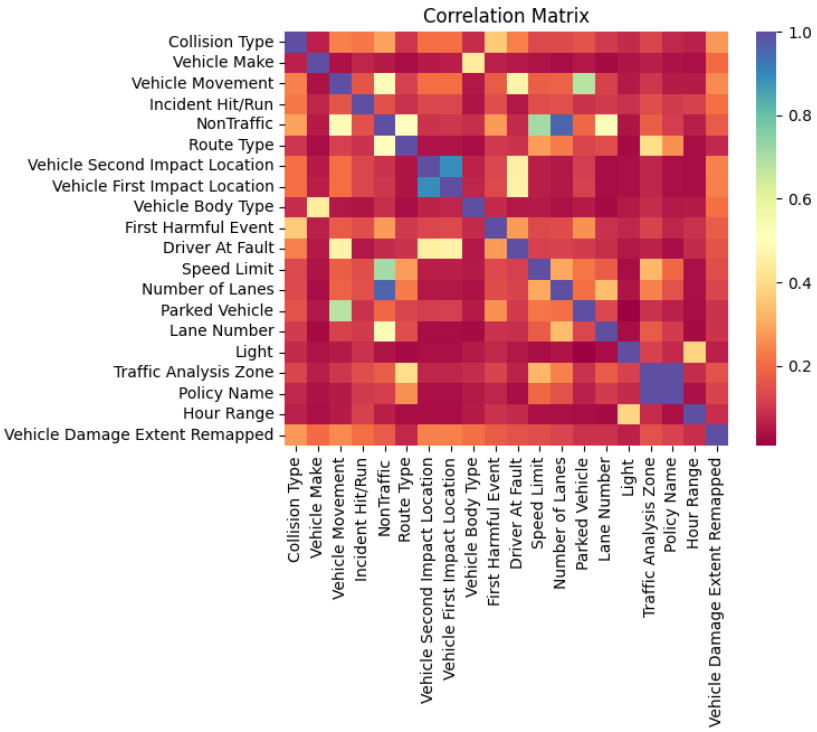
Test For Multicolinearity

In [9]:
```
corr = pd.DataFrame(index=target2_feats+target2, columns=target2_feats+target2)
```

In [10]:
```
def get_cramers_v(idx, col, n):
    contingency_table = pd.crosstab(idx, col)
    chi2, _, _, _ = chi2_contingency(contingency_table)
    feature_shape = contingency_table.shape[0]
    target_shape = contingency_table.shape[1]
    cramers_v = np.sqrt(chi2 / (n * min(feature_shape-1, target_shape-1)))
    return cramers_v
```

In [11]:
```
for idx in corr.index:
    for col in corr.columns:
        corr.at[idx, col] = get_cramers_v(target2_df[idx], target2_df[col], target2_df.shape[0])
```

In [12]:
```
sns.heatmap(corr.astype(float), cmap=sns.color_palette("Spectral", as_cmap=True))
plt.title("Correlation Matrix")
plt.show()
```



Variable Encoding

In [13]:
```
target2_df.drop(columns=["Traffic Analysis Zone", "Number of Lanes", "Vehicle Second Impact Location"], inplace=True)
```

In [14]:
```
binary_feats = ["Incident Hit/Run", "NonTraffic", "Driver At Fault", "Parked Vehicle"]
ordinal_feats = ["Vehicle First Impact Location", "Speed Limit", "Lane Number", "Hour Range"]
nominal_feats = ["Collision Type", "Vehicle Make", "Vehicle Movement", "Route Type", "Vehicle Body Type", "First Harmful Event",
                 "Light", "Policy Name"]
```

In [15]:
```
model_df = pd.get_dummies(target2_df, columns=nominal_feats)
```

In [16]:
```
for col in binary_feats + ordinal_feats:
    model_df[col] = model_df[col].astype('category').cat.codes
```

Classifiers

Parameter Selections

In [17]:
```
def gridSearch(clf, params, df, n_rows, score='f1_weighted', is_y_np=False, _target="Vehicle Damage Extent Remapped"):
    df = df[: n_rows]

    X_n = df.drop(columns=[_target])
    y_n = df[_target]
    y_np = df[_target].values
    y_np = LabelEncoder().fit_transform(y_np)

    grid_search = GridSearchCV(clf, params, cv=10, scoring=score)

    if is_y_np:
        grid_search.fit(X_n, y_np)
    else:
        grid_search.fit(X_n, y_n)

    return grid_search
```

In [18]:
```
%%time
rf_params = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_depth': [None, 5, 10, 15, 20, 25],
    'min_samples_split': [2, 5, 10],
    'class_weight': ['balanced']
}

gs_rf = gridSearch(RandomForestClassifier(), rf_params, model_df, int(model_df.shape[0]*.10))
print("Best Random Forest parameters found: ", gs_rf.best_params_)
print("Best Random Forest f1_weighted found: ", gs_rf.best_score_)
```

```
Best Random Forest parameters found:  {'class_weight': 'balanced', 'max_depth': 15, 'min_samples_split': 10, 'n_estimators': 500}
Best Random Forest f1_weighted found:  0.5961213701654454
CPU times: total: 46min 41s
Wall time: 49min 53s
```

In [19]:
```
%%time
bernNB_params = {'fit_prior': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_bernNB = gridSearch(BernoulliNB(), bernNB_params, model_df.loc[:, ~model_df.columns.isin(ordinal_feats)], model_df.shape[0])
print("Best BernoulliNB parameters found: ", gs_bernNB.best_params_)
print("Best BernoulliNB f1_weighted found: ", gs_bernNB.best_score_)
```

```
Best BernoulliNB parameters found:  {'alpha': 5, 'fit_prior': True}
Best BernoulliNB f1_weighted found:  0.5680809066700119
CPU times: total: 4min 42s
Wall time: 5min 6s
```

In [20]:
```
%%time
compNB_params = {'norm': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_compNB = gridSearch(ComplementNB(), compNB_params, model_df, model_df.shape[0])
print("Best ComplementNB parameters found: ", gs_compNB.best_params_)
print("Best ComplementNB f1_weighted found: ", gs_compNB.best_score_)
```

```
Best ComplementNB parameters found:  {'alpha': 20, 'norm': False}
Best ComplementNB f1_weighted found:  0.5641987029464881
CPU times: total: 4min 15s
Wall time: 4min 40s
```

In [21]:
```
%%time
xgb_params = {'booster': ['gbtree'],
              'eta': [0.01, 0.05, 0.1],
              'max_depth': [3, 6, 9],
              'min_child_weight': [0.25, 0.5, 1, 5],
              'lambda': [0.1, 1.0, 10.0],
              'alpha': [0.0, 0.1, 0.5],
              'objective': ['multi:softmax'],
              'num_class': [4],
              'tree_method':['hist']}

gs_xgb = gridSearch(xgb.XGBClassifier(), xgb_params, model_df, int(model_df.shape[0]*.10), is_y_np=True)
print("Best XGBoost parameters found: ", gs_xgb.best_params_)
print("Best XGBoost f1_weighted found: ", gs_xgb.best_score_)
```

```
Best XGBoost parameters found:  {'alpha': 0.1, 'booster': 'gbtree', 'eta': 0.1, 'lambda': 1.0, 'max_depth': 6, 'min_child_weight': 0.5, 'num_class': 4,
'objective': 'multi:softmax', 'tree_method': 'hist'}
Best XGBoost f1_weighted found:  0.5939561936645722
CPU times: total: 9h 52min 19s
Wall time: 1h 11min 51s
```

Model Evaluations with Selected Parameters

In [22]:
```
classifiers = ["Random Forest", "BernoulliNB", "ComplementNB", "XGBoost"]
accs = []
f1s = []
```

In [23]:
```
def plot_cm(df, title):
    plt.figure(figsize=(10, 6))
    sns.heatmap(df, cmap = 'viridis', annot=True, fmt="d", square=True, linewidths=.5)
    plt.xlabel("Predicted Values", fontsize=10)
    plt.ylabel("Actual Values", fontsize=10)
    plt.xticks(wrap=True, fontsize=7, rotation=0)
    plt.yticks(wrap=True, fontsize=7)
    plt.title(title)
    plt.show()
```

In [24]:
```
def kfold_eval(clf, X, y):
    acc = []
    f1_weighted = []
    cm = []
    kf = StratifiedKFold(n_splits=10, shuffle=False)
    for train, test in kf.split(X, y):
        clf.fit(X.iloc[train], y[train])
        y_pred = clf.predict(X.iloc[test])
        class_names = sorted(set(y[test]) | set(y_pred))
        acc += [accuracy_score(y[test], y_pred)]
        f1_weighted += [f1_score(y[test], y_pred, average='weighted')]
        cm += [pd.DataFrame(confusion_matrix(y[test], y_pred, labels=class_names), index=class_names, columns=class_names)]
    return acc, f1_weighted, cm
```

In [25]:
```
X = model_df.drop(columns=["Vehicle Damage Extent Remapped"])
y = model_df["Vehicle Damage Extent Remapped"]

X_np = model_df.drop(columns=["Vehicle Damage Extent Remapped"]).values
y_np = model_df["Vehicle Damage Extent Remapped"].values
y_np = LabelEncoder().fit_transform(y_np)
```

In [26]:
```
%%time
rf = RandomForestClassifier(**gs_rf.best_params_)

acc, f1_weighted, cm = kfold_eval(rf, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'Random Forest CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'Random Forest CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
Random Forest CV accuracy=0.605 0.014
Random Forest CV f1_weighted=0.595 0.008
CPU times: total: 12min 54s
Wall time: 13min 22s
```

In [27]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'Random Forest CV Confusion Matrix for Fold with Best f1_weighted')
```

In [28]:
```
%%time
bernNB = BernoulliNB(**gs_bernNB.best_params_)

acc, f1_weighted, cm = kfold_eval(bernNB, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'BernoulliNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'BernoulliNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
BernoulliNB CV accuracy=0.576 0.015
BernoulliNB CV f1_weighted=0.569 0.010
CPU times: total: 11.3 s
Wall time: 12.2 s
```

In [29]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'BernoulliNB CV Confusion Matrix for Fold with Best F1_Weighted')
```

In [30]:
```
%%time
compNB = ComplementNB(**gs_compNB.best_params_)

acc, f1_weighted, cm = kfold_eval(compNB, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'ComplementNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'ComplementNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
ComplementNB CV accuracy=0.577 0.014
ComplementNB CV f1_weighted=0.564 0.007
CPU times: total: 10.2 s
Wall time: 10.9 s
```

In [31]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'ComplementNB CV Confusion Matrix for Fold with Best f1_weighted')
```

In [32]:
```
%%time
xgb_clf = xgb.XGBClassifier(**gs_xgb.best_params_)

acc, f1_weighted, cm = kfold_eval(xgb_clf, X, y_np)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'XGBoost CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'XGBoost CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
XGBoost CV accuracy=0.611 0.013
XGBoost CV f1_weighted=0.586 0.008
CPU times: total: 7min 14s
Wall time: 47.4 s
```

In [33]:
```
best_cm = cm[f1_weighted.index(max(f1_weighted))]
best_cm.index = target2_df["Vehicle Damage Extent Remapped"].unique().tolist()
best_cm.columns = target2_df["Vehicle Damage Extent Remapped"].unique().tolist()
plot_cm(best_cm, 'XGBoost CV Confusion Matrix for Fold with Best f1_weighted')
```

In [34]:
```
sns.set_style("whitegrid")
sns.set_palette("pastel")

data = {
    'Classifier': classifiers,
    'Accuracy': accs,
    'F1_weighted': f1s
}

results = pd.DataFrame(data)
results = pd.melt(results, id_vars='Classifier', var_name='Metric', value_name='Value')

plt.figure(figsize=(10, 6))
sns.barplot(data=results, x='Classifier', y='Value', hue='Metric')
plt.title('Classifier Performance Comparison - Vehicle Damage Extent Remapped')
plt.ylabel('Score')
plt.xlabel('Classifier')
plt.ylim(0, 1)
plt.legend(title='Metric')
plt.show()
```

Creating Binary Target Variable

In [35]:
```
bin_map = {"NO DAMAGE OR SUPERFICIAL": "DRIVABLE", "FUNCTIONAL": "DRIVABLE",
           "DESTROYED OR DISABLING": "NOT DRIVABLE"}
model_df["Vehicle Damage Extent Binary"] = model_df["Vehicle Damage Extent Remapped"].replace(bin_map)
```

In [36]:
```
model_df["Vehicle Damage Extent Binary"].value_counts(normalize=True)
```

Out[36]:
```
DRIVABLE        0.562181
NOT DRIVABLE    0.437819
Name: Vehicle Damage Extent Binary, dtype: float64
```

In [37]:
```
model_df_2 = model_df.drop(columns=["Vehicle Damage Extent Remapped"])
```

Parameter Selections

In [38]:
```
%%time
rf_params = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_depth': [None, 5, 10, 15, 20, 25],
    'min_samples_split': [2, 5, 10],
    'class_weight': ['balanced']
}

gs_rf_2 = gridSearch(RandomForestClassifier(), rf_params, model_df_2, int(model_df_2.shape[0]*.10),
                     _target="Vehicle Damage Extent Binary")
print("Best Random Forest parameters found: ", gs_rf_2.best_params_)
print("Best Random Forest f1_weighted found: ", gs_rf_2.best_score_)
```

```
Best Random Forest parameters found:  {'class_weight': 'balanced', 'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 500}
Best Random Forest f1_weighted found:  0.7545305852372759
CPU times: total: 46min 7s
Wall time: 47min 31s
```

In [39]:
```
%%time
bernNB_params = {'fit_prior': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_bernNB_2 = gridSearch(BernoulliNB(), bernNB_params, model_df_2.loc[:, ~model_df_2.columns.isin(ordinal_feats)],
                         model_df_2.shape[0], _target="Vehicle Damage Extent Binary")
print("Best BernoulliNB parameters found: ", gs_bernNB_2.best_params_)
print("Best BernoulliNB f1_weighted found: ", gs_bernNB_2.best_score_)
```

```
Best BernoulliNB parameters found:  {'alpha': 0.75, 'fit_prior': True}
Best BernoulliNB f1_weighted found:  0.7365490804760209
CPU times: total: 3min 55s
Wall time: 4min 7s
```

In [40]:
```
%%time
compNB_params = {'norm': [True, False],
                 'alpha': [ 0.001, 0.025, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20]}
gs_compNB_2 = gridSearch(ComplementNB(), compNB_params, model_df_2, model_df_2.shape[0], _target="Vehicle Damage Extent Binary")
print("Best ComplementNB parameters found: ", gs_compNB_2.best_params_)
print("Best ComplementNB f1_weighted found: ", gs_compNB_2.best_score_)
```

```
Best ComplementNB parameters found:  {'alpha': 0.5, 'norm': False}
Best ComplementNB f1_weighted found:  0.7321640250472671
CPU times: total: 3min 21s
Wall time: 3min 31s
```

In [41]:
```
%%time
xgb_params = {'booster': ['gbtree'],
              'eta': [0.01, 0.05, 0.1],
              'max_depth': [3, 6, 9],
              'min_child_weight': [0.25, 0.5, 1, 5],
              'lambda': [0.1, 1.0, 10.0],
              'alpha': [0.0, 0.1, 0.5],
              'objective': ['multi:softmax'],
              'num_class': [4],
              'tree_method':['hist']}

gs_xgb_2 = gridSearch(xgb.XGBClassifier(), xgb_params, model_df_2, int(model_df_2.shape[0]*.10), is_y_np=True,
                      _target="Vehicle Damage Extent Binary")
print("Best XGBoost parameters found: ", gs_xgb_2.best_params_)
print("Best XGBoost f1_weighted found: ", gs_xgb_2.best_score_)
```

```
Best XGBoost parameters found:  {'alpha': 0.5, 'booster': 'gbtree', 'eta': 0.05, 'lambda': 10.0, 'max_depth': 9, 'min_child_weight': 1, 'num_class': 4,
'objective': 'multi:softmax', 'tree_method': 'hist'}
Best XGBoost f1_weighted found:  0.7553454423906056
CPU times: total: 6h 33min 50s
Wall time: 1h 5min 49s
```

In [42]:
```
%%time
logReg_params = {'clf__n_jobs': [-1],
                 'clf__penalty': ['l1', 'l2'],
                 'clf__solver': ['saga'],
                 'clf__C': [0.1, 0.25, 0.5, 1],
                 'clf__class_weight': ['balanced'],
                 'clf__tol': [0.00001, 0.0001, 0.001, 0.01],
                 'clf__max_iter': [50, 100, 250]}

pipeline = Pipeline([('scaler', StandardScaler()), ('clf', LogisticRegression())]) #mostly to standardize ordinal variables
gs_logReg = gridSearch(pipeline, logReg_params, model_df_2, int(model_df_2.shape[0]*.10),
                       _target="Vehicle Damage Extent Binary")
print("Best LogisticRegression parameters found: ", gs_logReg.best_params_)
print("Best LogisticRegression f1_weighted found: ", gs_logReg.best_score_)
```

```
Best LogisticRegression parameters found:  {'clf__C': 0.1, 'clf__class_weight': 'balanced', 'clf__max_iter': 100, 'clf__n_jobs': -1, 'clf__penalty': 'l
2', 'clf__solver': 'saga', 'clf__tol': 0.01}
Best LogisticRegression f1_weighted found:  0.7406994231121913
CPU times: total: 1min 15s
Wall time: 44min 3s
```

Model Evaluations with Selected Parameters

In [43]:
```
classifiers2 = ["Random Forest", "BernoulliNB", "ComplementNB", "XGBoost", "Logistic Regression"]
accs2 = []
f1s2 = []
```

In [44]:
```
X = model_df_2.drop(columns=["Vehicle Damage Extent Binary"])
y = model_df_2["Vehicle Damage Extent Binary"]

X_np = model_df_2.drop(columns=["Vehicle Damage Extent Binary"]).values
y_np = model_df_2["Vehicle Damage Extent Binary"].values
y_np = LabelEncoder().fit_transform(y_np)
```

In [45]:
```
%%time
rf = RandomForestClassifier(**gs_rf_2.best_params_)

acc, f1_weighted, cm = kfold_eval(rf, X, y)
accs2.append(np.mean(acc)); f1s2.append(np.mean(f1_weighted))
print(f'Random Forest CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'Random Forest CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

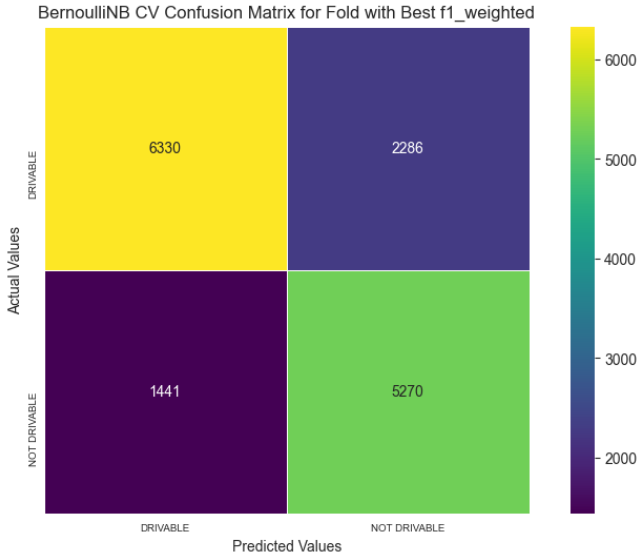
```
Random Forest CV accuracy=0.754 0.012
Random Forest CV f1_weighted=0.755 0.011
CPU times: total: 16min 11s
Wall time: 16min 46s
```

In [46]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'Random Forest CV Confusion Matrix for Fold with Best f1_weighted')
```

In [47]:
```
%%time
bernNB = BernoulliNB(**gs_bernNB_2.best_params_)

acc, f1_weighted, cm = kfold_eval(bernNB, X, y)
accs2.append(np.mean(acc)); f1s2.append(np.mean(f1_weighted))
print(f'BernoulliNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'BernoulliNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
BernoulliNB CV accuracy=0.736 0.009
BernoulliNB CV f1_weighted=0.736 0.009
CPU times: total: 9 s
Wall time: 9.64 s
```

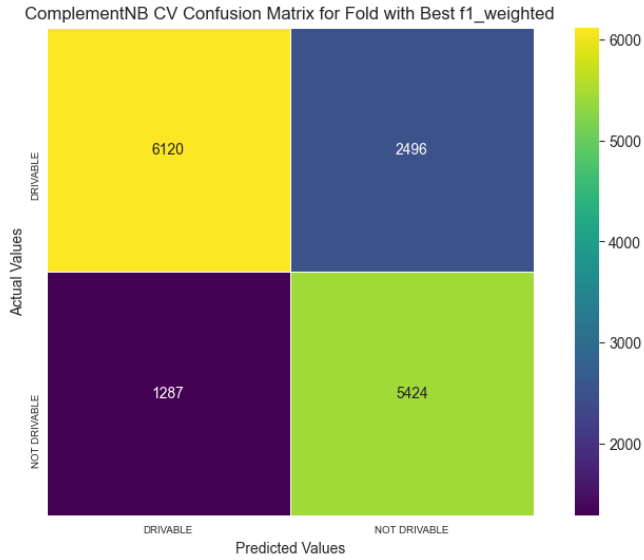
In [48]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'BernoulliNB CV Confusion Matrix for Fold with Best f1_weighted')
```



In [49]:
```
%%time
compNB = ComplementNB(**gs_compNB_2.best_params_)

acc, f1_weighted, cm = kfold_eval(compNB, X, y)
accs2.append(np.mean(acc)); f1s2.append(np.mean(f1_weighted))
print(f'ComplementNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'ComplementNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
ComplementNB CV accuracy=0.731 0.009
ComplementNB CV f1_weighted=0.732 0.009
CPU times: total: 7.89 s
Wall time: 8.55 s
```

In [50]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'ComplementNB CV Confusion Matrix for Fold with Best f1_weighted')
```



In [51]:
```
%%time
xgb_clf = xgb.XGBClassifier(**gs_xgb_2.best_params_)

acc, f1_weighted, cm = kfold_eval(xgb_clf, X, y_np)
accs2.append(np.mean(acc)); f1s2.append(np.mean(f1_weighted))
print(f'XGBoost CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'XGBoost CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

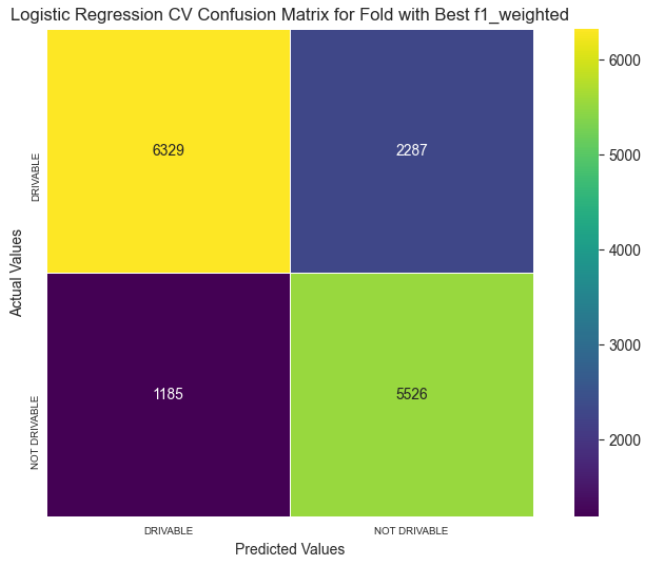
```
XGBoost CV accuracy=0.757 0.012
XGBoost CV f1_weighted=0.758 0.012
CPU times: total: 8min 46s
Wall time: 1min 5s
```

In [52]:
```
best_cm = cm[f1_weighted.index(max(f1_weighted))]
best_cm.index = model_df_2["Vehicle Damage Extent Binary"].unique().tolist()
best_cm.columns = model_df_2["Vehicle Damage Extent Binary"].unique().tolist()
plot_cm(best_cm, 'XGBoost CV Confusion Matrix for Fold with Best f1_weighted')
```

In [53]:
```
%%time
logReg_params = {param.replace('clf__', ''): value for param, value in gs_logReg.best_params_.items()}
logReg_clf = Pipeline([('scaler', StandardScaler()), ('clf', LogisticRegression(**logReg_params))])

acc, f1_weighted, cm = kfold_eval(logReg_clf, X, y)
accs2.append(np.mean(acc)); f1s2.append(np.mean(f1_weighted))
print(f'Logistic Regression CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'Logistic Regression CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
Logistic Regression CV accuracy=0.744 0.012
Logistic Regression CV f1_weighted=0.744 0.012
CPU times: total: 25.9 s
Wall time: 28.7 s
```

In [54]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'Logistic Regression CV Confusion Matrix for Fold with Best f1_weighted')
```



In [55]:
```
sns.set_style("whitegrid")
sns.set_palette("pastel")

data = {
    'Classifier': classifiers2,
    'Accuracy': accs2,
    'F1_weighted': f1s2
}

results = pd.DataFrame(data)
results = pd.melt(results, id_vars='Classifier', var_name='Metric', value_name='Value')

plt.figure(figsize=(10, 6))
sns.barplot(data=results, x='Classifier', y='Value', hue='Metric')
plt.title('Classifier Performance Comparison - Vehicle Damage Extent Binary')
plt.ylabel('Score')
plt.xlabel('Classifier')
plt.ylim(0, 1)
plt.legend(title='Metric')
plt.show()
```

Target 3 – Number of Motorist Involved Notebook

Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from scipy.stats import chi2_contingency
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
```

Initializing Data

In [2]:
```
driver_dtypes = {'Local Case Number': "string", "Num Motorist Involved": "string"}
df = pd.read_csv("../Datasets/prepared_data.csv", dtype=driver_dtypes)

target3 = ["Num Motorist Involved"]
target3_feats = ["First Harmful Event", "Collision Type", "Driver At Fault", "NonTraffic", "Vehicle Movement",
                 "Number of Lanes", "Speed Limit", "Vehicle Second Impact Location", "Vehicle First Impact Location",
                 "Lane Number", "Vehicle Make", "Driver Substance Abuse", "Route Type", "Incident Hit/Run", "Light",
                 "Vehicle Body Type", "Traffic Analysis Zone", "Policy Name", "Hour Range"]

target3_df = df[target3+target3_feats]
```

Checking Distribution

In [3]:
```
target3_df[target3].value_counts(normalize=True)
```

Out[3]:
```
Num Motorist Involved
2                       0.674402
1                       0.180026
3                       0.113760
4+                      0.031812
dtype: float64
```

In [4]:
```
target3_df.shape
```

Out[4]:
```
(171414, 20)
```

In [5]:
```
target3_df = target3_df.dropna()
target3_df[target3].value_counts(normalize=True)
```

Out[5]:
```
Num Motorist Involved
2                       0.691733
1                       0.152689
3                       0.121800
4+                      0.033778
dtype: float64
```

In [6]:
```
target3_df.reset_index(inplace=True, drop=True)
target3_df.shape
```

Out[6]:
```
(155813, 20)
```

In [7]:
```
def test_normality(distro):
    stat, p = stats.shapiro(distro)
    alpha = 0.05

    if p > alpha:
        print("p =",p)
        print("Normal Distribution (Fail to Reject Null Hypothesis)")
    else:
        print("Not Normal Distribution (Reject Null Hypothesis)")
```

In [8]:
```
test_normality(target3_df[target3].value_counts(normalize=True).tolist())
```

```
p = 0.07232625037431717
Normal Distribution (Fail to Reject Null Hypothesis)
```

Test for Multicolinearity

In [9]:
```
corr = pd.DataFrame(index=target3_feats+target3, columns=target3_feats+target3)
```

In [10]:
```
def get_cramers_v(idx, col, n):
    contingency_table = pd.crosstab(idx, col)
    chi2, _, _, _ = chi2_contingency(contingency_table)
    feature_shape = contingency_table.shape[0]
    target_shape = contingency_table.shape[1]
    cramers_v = np.sqrt(chi2 / (n * min(feature_shape-1, target_shape-1)))
    return cramers_v
```

In [11]:
```
for idx in corr.index:
    for col in corr.columns:
        corr.at[idx, col] = get_cramers_v(target3_df[idx], target3_df[col], target3_df.shape[0])
```

In [12]:
```
sns.heatmap(corr.astype(float), cmap=sns.color_palette("Spectral", as_cmap=True))
plt.title("Correlation Matrix")
plt.show()
```

Variable Encoding

In [13]:
```
target3_df.drop(columns=["Traffic Analysis Zone", "Number of Lanes", "Vehicle Second Impact Location"], inplace=True)
```

In [14]:
```
binary_feats = ["Incident Hit/Run", "NonTraffic", "Driver At Fault"]
ordinal_feats = ["Speed Limit", "Lane Number", "Vehicle First Impact Location", "Hour Range"]
nominal_feats = ["First Harmful Event", "Collision Type", "Vehicle Movement", "Vehicle Make", "Driver Substance Abuse",
                 "Route Type", "Light", "Vehicle Body Type", "Policy Name"]
```

In [15]:
```
model_df = pd.get_dummies(target3_df, columns=nominal_feats)
```

In [16]:
```
for col in binary_feats + ordinal_feats:
    model_df[col] = model_df[col].astype('category').cat.codes
```

Classifiers

Parameter Selections

In [17]:
```
def gridSearch(clf, params, df, n_rows, score='f1_weighted', is_y_np=False):
    df = df[: n_rows]

    X_n = df.drop(columns=['Num Motorist Involved'])
    y_n = df['Num Motorist Involved']
    y_np = df['Num Motorist Involved'].astype('category').cat.codes

    grid_search = GridSearchCV(clf, params, cv=10, scoring=score)

    if is_y_np:
        grid_search.fit(X_n, y_np)
    else:
        grid_search.fit(X_n, y_n)

    return grid_search
```

In [18]:
```
%%time
rf_params = {
    'n_estimators': [50, 100, 200, 300, 500],
    'max_depth': [None, 5, 10, 15, 20, 25],
    'min_samples_split': [2, 5, 10],
    'class_weight': ['balanced']
}

gs_rf = gridSearch(RandomForestClassifier(), rf_params, model_df, int(model_df.shape[0]*.10))
print("Best Random Forest parameters found: ", gs_rf.best_params_)
print("Best Random Forest f1_weighted found: ", gs_rf.best_score_)
```

```
Best Random Forest parameters found:  {'class_weight': 'balanced', 'max_depth': 25, 'min_samples_split': 2, 'n_estimators': 100}
Best Random Forest f1_weighted found:  0.7404115044291106
CPU times: total: 44min 39s
Wall time: 46min 14s
```

In [19]:
```
%%time
bernNB_params = {'fit_prior': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_bernNB = gridSearch(BernoulliNB(), bernNB_params, model_df.loc[:, ~model_df.columns.isin(ordinal_feats)], model_df.shape[0])
print("Best BernoulliNB parameters found: ", gs_bernNB.best_params_)
print("Best BernoulliNB f1_weighted found: ", gs_bernNB.best_score_)
```

```
Best BernoulliNB parameters found:  {'alpha': 100, 'fit_prior': True}
Best BernoulliNB f1_weighted found:  0.7658196085769923
CPU times: total: 5min 8s
Wall time: 5min 22s
```

In [20]:
```
%%time
compNB_params = {'norm': [True, False],
                 'alpha': [0.1, 0.25, 0.5, 0.75, 1, 1.5, 2, 5, 10, 15, 20, 25, 50, 100]}
gs_compNB = gridSearch(ComplementNB(), compNB_params, model_df, model_df.shape[0])
print("Best ComplementNB parameters found: ", gs_compNB.best_params_)
print("Best ComplementNB f1_weighted found: ", gs_compNB.best_score_)
```

```
Best ComplementNB parameters found:  {'alpha': 20, 'norm': True}
Best ComplementNB f1_weighted found:  0.7382601244250238
CPU times: total: 4min 31s
Wall time: 4min 44s
```

In [23]:
```
%%time
xgb_params = {'booster': ['gbtree'],
              'eta': [0.01, 0.05, 0.1],
              'max_depth': [3, 6, 9],
              'min_child_weight': [0.25, 0.5, 1, 5],
              'lambda': [0.1, 1.0, 10.0],
              'alpha': [0.0, 0.1, 0.5],
              'objective': ['multi:softmax'],
              'num_class': [4],
              'tree_method':['hist']}

gs_xgb = gridSearch(xgb.XGBClassifier(), xgb_params, model_df, int(model_df.shape[0]*.10), is_y_np=True)
print("Best XGBoost parameters found: ", gs_xgb.best_params_)
print("Best XGBoost f1_weighted found: ", gs_xgb.best_score_)
```

```
Best XGBoost parameters found:  {'alpha': 0.0, 'booster': 'gbtree', 'eta': 0.1, 'lambda': 0.1, 'max_depth': 9, 'min_child_weight': 5, 'num_class': 4, 'o
bjective': 'multi:softmax', 'tree_method': 'hist'}
Best XGBoost f1_weighted found:  0.7304844031177266
CPU times: total: 12h 14min 48s
Wall time: 1h 21min 34s
```

Model Evaluations with Selected Parameters

In [24]:
```
classifiers = ["Random Forest", "BernoulliNB", "ComplementNB", "XGBoost"]
accs = []
f1s = []
```

In [25]:
```
def plot_cm(df, title):
    plt.figure(figsize=(10, 6))
    sns.heatmap(df, cmap = 'viridis', annot=True, fmt="d", square=True, linewidths=.5)
    plt.xlabel("Predicted Values", fontsize=10)
    plt.ylabel("Actual Values", fontsize=10)
    plt.xticks(wrap=True, fontsize=7, rotation=0)
    plt.yticks(wrap=True, fontsize=7)
    plt.title(title)
    plt.show()
```

In [26]:
```
def kfold_eval(clf, X, y, nparray=False):
    acc = []
    f1_weighted = []
    cm = []
    kf = StratifiedKFold(n_splits=10, shuffle=False)
    if not nparray:
        for train, test in kf.split(X, y):
            clf.fit(X.iloc[train], y[train])
            y_pred = clf.predict(X.iloc[test])
            class_names = sorted(set(y[test]) | set(y_pred))
            acc += [accuracy_score(y[test], y_pred)]
            f1_weighted += [f1_score(y[test], y_pred, average='weighted')]
            cm += [pd.DataFrame(confusion_matrix(y[test], y_pred, labels=class_names), index=class_names, columns=class_names)]
    else:
        for train, test in kf.split(X, y):
            clf.fit(X[train], y[train])
            y_pred = clf.predict(X[test])
            class_names = sorted(set(y[test]) | set(y_pred))
            acc += [accuracy_score(y[test], y_pred)]
            f1_weighted += [f1_score(y[test], y_pred, average='weighted')]
            cm += [pd.DataFrame(confusion_matrix(y[test], y_pred, labels=class_names), index=class_names, columns=class_names)]
    return acc, f1_weighted, cm
```

In [27]:
```
X = model_df.drop(columns=['Num Motorist Involved'])
y = model_df['Num Motorist Involved']

X_np = model_df.drop(columns=['Num Motorist Involved']).values
y_np = model_df['Num Motorist Involved'].values
y_np = LabelEncoder().fit_transform(y_np)
```

In [28]:
```
%%time
rf = RandomForestClassifier(**gs_rf.best_params_)

acc, f1_weighted, cm = kfold_eval(rf, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'Random Forest CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'Random Forest CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
Random Forest CV accuracy=0.729 0.003
Random Forest CV f1_weighted=0.755 0.003
CPU times: total: 3min 55s
Wall time: 4min 11s
```

In [29]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'Random Forest CV Confusion Matrix for Fold with Best f1_weighted')
```

In [30]:
```
%%time
bernNB = BernoulliNB(**gs_bernNB.best_params_)

acc, f1_weighted, cm = kfold_eval(bernNB, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'BernoulliNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'BernoulliNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
BernoulliNB CV accuracy=0.777 0.003
BernoulliNB CV f1_weighted=0.765 0.004
CPU times: total: 12.1 s
Wall time: 12.9 s
```

In [31]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'BernoulliNB CV Confusion Matrix for Fold with Best F1_Weighted')
```

In [32]:
```
%%time
compNB = ComplementNB(**gs_compNB.best_params_)

acc, f1_weighted, cm = kfold_eval(compNB, X, y)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'ComplementNB CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'ComplementNB CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
ComplementNB CV accuracy=0.764 0.005
ComplementNB CV f1_weighted=0.738 0.004
CPU times: total: 11 s
Wall time: 11.8 s
```

In [33]:
```
plot_cm(cm[f1_weighted.index(max(f1_weighted))], 'ComplementNB CV Confusion Matrix for Fold with Best f1_weighted')
```

In [34]:
```
%%time
xgb_clf = xgb.XGBClassifier(**gs_xgb.best_params_)

acc, f1_weighted, cm = kfold_eval(xgb_clf, X, y_np)
accs.append(np.mean(acc)); f1s.append(np.mean(f1_weighted))
print(f'XGBoost CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'XGBoost CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
XGBoost CV accuracy=0.828 0.002
XGBoost CV f1_weighted=0.770 0.002
CPU times: total: 13min 58s
Wall time: 1min 24s
```

In [35]:
```
best_cm = cm[f1_weighted.index(max(f1_weighted))]
best_cm.index = target3_df['Num Motorist Involved'].unique().tolist()
best_cm.columns = target3_df['Num Motorist Involved'].unique().tolist()
plot_cm(best_cm, 'XGBoost CV Confusion Matrix for Fold with Best f1_weighted')
```

In [36]:
```
sns.set_style("whitegrid")
sns.set_palette("pastel")

data = {
    'Classifier': classifiers,
    'Accuracy': accs,
    'F1_weighted': f1s
}

results = pd.DataFrame(data)
results = pd.melt(results, id_vars='Classifier', var_name='Metric', value_name='Value')

plt.figure(figsize=(10, 6))
sns.barplot(data=results, x='Classifier', y='Value', hue='Metric')
plt.title('Classifier Performance Comparison - Num Motorist Involved')
plt.ylabel('Score')
plt.xlabel('Classifier')
plt.ylim(0, 1)
plt.legend(title='Metric')
plt.show()
```

Other Code

In [18]:
```
class PyTorchMLP(torch.nn.Module):  # One hidden layer
    def __init__(self, n_hidden=10, epochs=100, eta=0.001, minibatch_size=50, seed=0):
        super(PyTorchMLP, self).__init__()
        self.random = np.random.RandomState(seed)  # shuffle mini batches
        self.n_hidden = n_hidden  # size of the hidden layer
        self.epochs = epochs  # number of iterations
        self.eta = eta  # learning rate
        self.minibatch_size = minibatch_size  # size of training batch - 1 would not work
        self.optimizer = None
        self.loss_func = torch.nn.CrossEntropyLoss()
        self.model = None

    def init_layers(self, _M:int, _K:int) -> None:
        # data structure
        self.model = torch.nn.Sequential(
            torch.nn.Linear(_M, self.n_hidden),
            torch.nn.Sigmoid(),
            torch.nn.Linear(self.n_hidden, self.n_hidden),
            torch.nn.Sigmoid(),
            torch.nn.Linear(self.n_hidden, _K),
        )

    def predict(self, _X):
        _X = torch.flatten(torch.FloatTensor(_X), start_dim=1)
        assert self.model is not None
        self.model.eval()
        with torch.no_grad():
            y_pred = np.argmax(self.model(_X), axis=1)
        self.model.train()
        return y_pred.numpy()

    def fit(self, _X_train, _y_train, info=False):
        import sys
        _X_train = torch.flatten(torch.FloatTensor(_X_train), start_dim=1)
        _y_train = torch.LongTensor(_y_train)
        n_features= _X_train.shape[1]
        n_output= np.unique(_y_train).shape[0]  # number of class labels

        self.init_layers(n_features, n_output)
        self.optimizer = torch.optim.Rprop(self.model.parameters(), lr=self.eta)  # connect model to optimizer

        for i in range(self.epochs):
            indices = np.arange(_X_train.shape[0])
            self.random.shuffle(indices)  # shuffle the data each epoch

            for start_idx in range(0, indices.shape[0] - self.minibatch_size + 1, self.minibatch_size):
                batch_idx = indices[start_idx:start_idx + self.minibatch_size]
                self.optimizer.zero_grad()

                net_out = self.model(_X_train[batch_idx])

                loss = self.loss_func(net_out, _y_train[batch_idx])
                loss.backward()
                self.optimizer.step()

                if info:
                    sys.stderr.write(f"\r{i+1:03d} Loss: {loss.item():6.5f}")
                    sys.stderr.flush()
        return self
```

In [19]:
```
def MLPGridSearch(params, df, n_rows):
    X_n = model_df.drop(columns=['Num Motorist Involved']).iloc[:n_rows].values
    y_n = model_df['Num Motorist Involved'].iloc[:n_rows].values
    y_n = LabelEncoder().fit_transform(y_n)

    param_combos = ParameterGrid(params)
    best_score = 0.0
    best_combo = None
    for combo in param_combos:
        clf = PyTorchMLP(**combo)
        kf = StratifiedKFold(n_splits=10, shuffle=False)
        scores = []
        for train, test in kf.split(X_n, y_n):
            clf.fit(X_n[train], y_n[train])
            y_pred = clf.predict(X_n[test])
            scores += [f1_score(y_n[test], y_pred, average='weighted')]
        mean_score = np.mean(scores)
        if mean_score > best_score:
            best_score = mean_score
            best_combo = combo
    return best_combo, best_score
```

In [20]:
```
%%time
MLPNN_params = {'minibatch_size': [32, 64, 128],
                'n_hidden': [50, 100],
                'eta': [0.001, 0.01],
                'epochs': [100, 200, 400]}
gs_mlp_best_params, gs_mlp_best_score = MLPGridSearch(MLPNN_params, target3_df, int(model_df.shape[0]*.10))
print("Best MLP parameters found: ", gs_mlp_best_params)
print("Best MLP f1_weighted found: ", gs_mlp_best_score)
```

```
Best MLP parameters found:  {'epochs': 200, 'eta': 0.001, 'minibatch_size': 128, 'n_hidden': 50}
Best MLP f1_weighted found:  0.7045213748561249
CPU times: total: 3d 11h 41min 3s
Wall time: 16h 15min 11s
```

In [29]:
```
%%time
MLPNN = PyTorchMLP(**gs_mlp_best_params)
X_np = model_df.drop(columns=['Num Motorist Involved']).values
y_np = model_df['Num Motorist Involved'].values
y_np = LabelEncoder().fit_transform(y_np)

acc, f1_weighted, cm = kfold_eval(MLPNN, X_np, y_np, nparray=True)
print(f'MLPNN CV accuracy={np.mean(acc):.3f} {np.std(acc):.3f}')
print(f'MLPNN CV f1_weighted={np.mean(f1_weighted):.3f} {np.std(f1_weighted):.3f}')
```

```
MLPNN CV accuracy=0.819 0.004
MLPNN CV f1_weighted=0.750 0.004
CPU times: total: 8h 2min 31s
Wall time: 1h 33min 40s
```

In [30]:
```
print('MLPNN CV Confusion Matrix for Fold with Best f1_weighted:')
cm[f1_weighted.index(max(f1_weighted))]
```

```
MLPNN CV Confusion Matrix for Fold with Best f1_weighted:
```

Out[30]:

	0	1	2	3
0	2165	214	0	0
1	94	10684	0	0
2	6	1892	0	0
3	2	524	0	0