Alaska Car Crash Analysis

CSCE A470 Final Report

Moro Bamber, Youji Seto, Nemed Aleman

12/12/2024

Abstract

The goal of this project was to analyze factors associated with automobile crashes across different regions of Alaska. Alaska's unique characteristics, ranging from its remote wilderness to the densely traveled roads of urban areas, present distinct challenges. Our analysis aimed to identify contributing factors of crashes that could indicate the region where they occurred and analyze the severity of resulting injuries. Additionally, we developed a web application to store and filter crash data. This platform allows users to apply the machine learning techniques we utilized in our analysis and uncover new insights themselves.

Introduction

Alaska offers a unique opportunity to explore the factors contributing to car crashes across the state. Urban areas are highly car-dependent, with driving conditions often deteriorating even on the busiest roads. Meanwhile, rural regions range from suburban landscapes to some of the most remote areas imaginable.

Throughout our project, we used the term “crashes” instead of “accidents.” This distinction was intentional; traffic engineers define "accidents" as unavoidable events beyond the control of the parties involved, such as earthquakes or lightning strikes. Our goal was to contribute to research aimed at making Alaska’s roads safer by identifying the factors that lead to “crashes.”

The majority of the research centered around finding differences between crashes that occurred in urban, rural, and suburban areas using advanced Machine Learning (ML) models. We also created a web application that is designed to handle large crash data sets and allow users to extract insights.

Clients and Advisors

The crash data was provided by the Alaska Department of Transportation (DoT) via Dr. Vinod Vasudevan of the UAA Civil Engineering department. We were encouraged to share any important findings with the DoT. Dr. Vasudevan was an important advisor throughout the project and helped guide us towards meaningful results and a functional web app. We also met with Anna Bosin from the Alaska DoT near the end of the semester to share our findings. We were thrilled to learn our results were similar to the DoT’s internal research. She gave us suggestions on how to move forward with the project and suggested other sources of data to analyze. Additionally, Dr. Frank Witmer of the UAA Computer Science & Engineering department also acted as an advisor.

Project Requirements

Analysis Requirements

- Develop machine learning models that identify risk factors contributing to car crashes in rural versus urban Alaska.
- The model should be able to accurately classify these factors based on the data provided, drawing meaningful insights from the comparison.

Web app requirements

- The web app will allow users to interact with the ML model, showcasing the findings through an easy-to-navigate UI.
- Provide functionality for users to upload new car crash data, ensuring specific fields are present for model processing.
- Display data in a user-friendly format, making the insights clear and easy to interpret.
- Implement data security measures, such as password protection, for accessing sensitive data and model results.
- Ensure the web app supports scalability for potential future expansions or additional data integrations.
- Flexible with the data it handles, allowing for different formats and different attributes and datatypes.

Data Pre-Processing

Data Cleaning

To properly work with ML models, the dataset must be cleaned. We were given two sets of data which were split by time periods with different notation of crash attributes. The first set of data is from 2009-2012, the second is from 2013-2017. Both data periods are split into three levels, the first level is the Crash level, pertaining to characteristics with the crash and the causal vehicle; it has the least amount of rows as there is just one per crash. The Driver-Vehicle level contains attributes associated with the drivers of the units (vehicles) involved as well as the vehicles they were driving/riding; it has the second largest number of rows. The third level is the person level, which has attributes that pertain to all people involved in the crash. For example, if there were 10 people involved, then there would be 10 rows for a single crash. It goes more in depth into who was injured and how. This level has the largest number of rows.

We organized the data into a SQL database to allow us to write queries that would help us understand and build our datasets for analysis with machine learning. By using SQL in conjunction with python, we were able to start filtering for what we thought would give us the most accurate and meaningful results when put through a ML model.

The first step in this process was eliminating columns that had an unacceptably large amount of unknown values. We set the threshold for this to be 7% as greater than that would start yielding inaccurate results. For the remaining columns in the data, we wanted to justify each column's inclusion for our ML training data. We wrote a script to find the data set each column belongs to; a column can belong to one or more levels per period of time. We used this script to show us the minimum amount of ‘unknown’ values for that column across all levels to use as the benchmark for that column. After finding the correct numbers of ‘unknown’ values by normalizing values

like ‘null’, ‘undetermined’, ‘not reported’, etc. to ‘unknown’, we found more accurate results of columns missing the most information. We decided to keep any column under 7% unknown, and then we looked through these columns to see what is worthy of keeping. We ended up with 80-100 columns across all levels for a single period. We then went through this list of columns, and using our prior knowledge of what these columns contain as well as aggregate SQL queries to explore the column more, we fine tuned our column list to keep to 34 for both 2013-2017 and 2009-2012. See appendix for data set description.

Additional Data

One of primary goals in this project was to differentiate crashes based on where they occurred, urban or rural. For this we needed a classification system. We started by using the TIGER2010 census blocks dataset which is a dataset of vectors with population information from the 2010 census. The idea was to use the population of a given census tract to determine how populated the area was. This worked to some extent, and is included in our final dataset as the pop10 column, but one issue was that it would give some areas in an urban environment a population of 0, because the vector was only covering roads, and no one lives on roads, so this would be misleading if we tried to classify areas solely on this presumption. To combat this, we used the GHSL: Degree of Urbanization raster dataset. Based on this dataset, and satellite imagery, we created vectors of urban and suburban areas of Alaska. We then wrote a python script that calculated if a crash’s longitude and latitude intersected an urban or suburban area, classified it as such, and if not, then we know it was a rural crash.

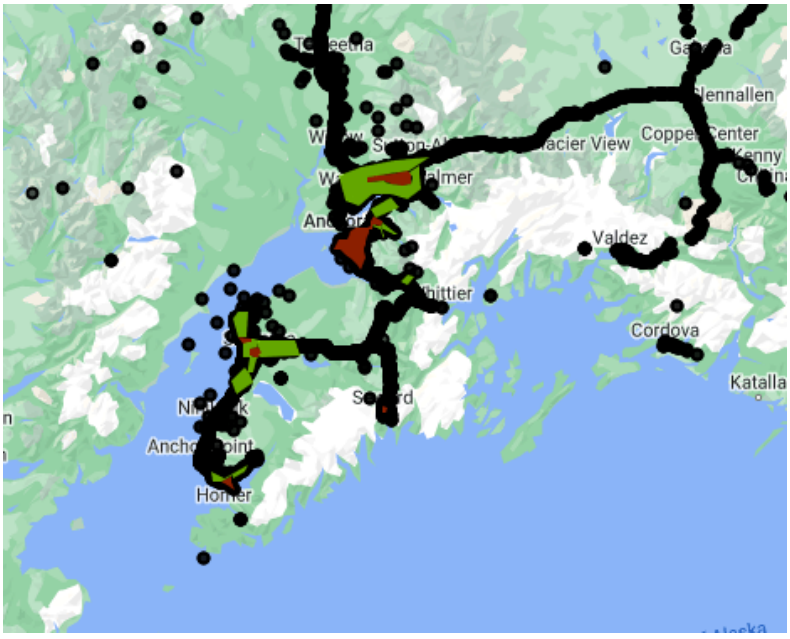


Figure 1: Sample Image of Google Earth Engine Process. Black dots are crashes, lime green areas are suburban, and red areas are urban.We were able to find if a crash’s coordinates intersect the red and green shapes to classify them. If not, the crash was rural. Appendix [1,2]

Finding Insights into Data with Machine Learning

Machine Learning Models

To identify which features contribute the highest to a crash severity, we used classification type models. We used the Crash Severity as our target variable, which ranges from no injury, to a fatal injury occurring in a crash. We first grouped the severity types into three categories: severe injury, moderate injury, and no injury type crashes. This grouping ensured adequate distinction between the categories and improved model performance. As part of this process, we mapped existing crash severity types into broader categories or excluded less relevant ones like ‘died before crash’.

Original Severity Type	Mapped Severity Score	Category
No Apparent Injury	0	No Injury
Possible Injury	1	Moderate
Suspected Minor Injury	1	Moderate
Suspected Serious Injury	2	Severe
Fatal Injury (killed)	2	Severe

We tested several known and frequently used classification models for the following reasons:

Decision Tree

- Effective for multi-class classification problems. Decision Trees are valuable for their interpretability and ability to handle both numerical and categorical data.
- The Decision Tree model provided valuable insights into the hierarchical relationships between different factors contributing to crash severity.
- We observed that the Decision Tree was prone to overfitting, especially when the tree depth was not properly controlled. This highlighted the need for careful hyperparameter tuning and the potential benefits of ensemble methods like Random Forests.

Random Forest

- To address the limitations of individual Decision Trees, we employed the Random Forest model, an ensemble method that combines multiple trees and reduces overfitting. By aggregating the predictions of a diverse set of trees, Random Forests often achieve higher accuracy and better generalization.

- This model provided a more robust and stable prediction compared to the Decision Tree. It also allowed us to analyze feature importance in how severe injuries in a crash were more reliably.
- The ensemble nature of Random Forests helped mitigate the overfitting issue observed with the Decision Tree, leading to improved performance on unseen data.

XGBoost

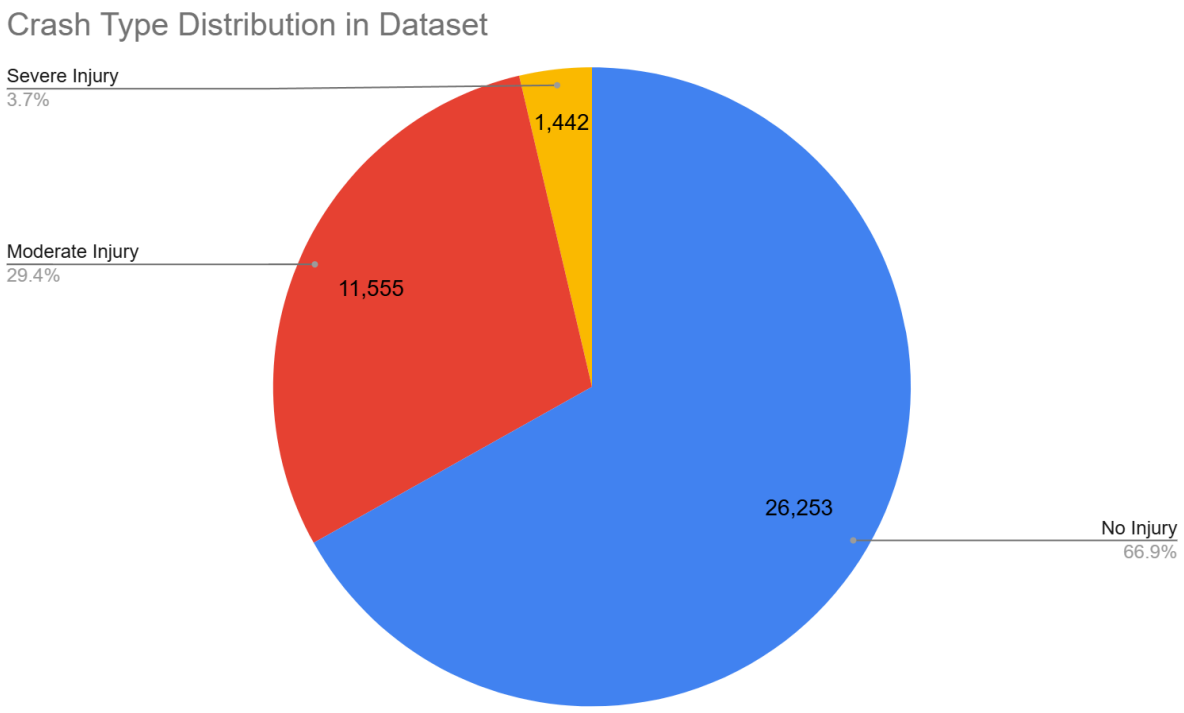
- This model is known for its performance in tabular data and high accuracy with structured datasets.
- It utilizes an ensemble learning method that builds a series of decision trees sequentially. Each tree corrects the errors of its predecessor by focusing more on the instances that were misclassified or had high prediction errors.

Multilevel Random Forest Model

- This model handles nested data well. It helped us to find which contributing factor correlates most with a specific crash type based on area (urban, rural, suburban).
- It trains a global random forest model to predict crash severity for all data.
- It also builds local random forest models for specific areas (e.g., rural, urban) to capture area-specific crash patterns.
- The predictions combine results from global and local models, giving more weight to the local model where applicable

Data Balancing

For this project, balancing the data was a crucial step to address the imbalance between different classes in the dataset. Without balancing, the model became biased towards the majority class leading to poor performance in the minority classes.



No Injury crash type makes up a large portion of the data. To mitigate this problem, several techniques were tested and implemented:

Synthetic Minority Oversampling Technique (SMOTE)

- Generates synthetic samples for the minority class by interpolating between existing data points in the minority class.

Downsampling

- Reduces the number of samples in the majority class to match the size of the minority class.

Cost-Sensitive Learning

- This technique penalizes misclassifications of the minority classes more during training by assigning higher weights to the minority classes, encouraging the model to focus on them.

Grouping Severity Levels

- This can help address imbalance by combining smaller classes into larger ones, making the distribution more even.

Adaptive Synthetic Sampling Approach (ADASYN)

- Similar to SMOTE but generates more synthetic samples for minority class instances that make it harder for the model to learn.

Random Oversampling

- Randomly duplicates instances from the minority class to increase its size. This is a simple technique but can lead to overfitting if not done carefully.

Results

XGBoost

Class	Precision	Recall	F1-Score	Support
0 (No Injury)	0.74	0.88	0.81	6,154
1 (Moderate)	0.52	0.32	0.40	2,768
2 (Severe)	0.42	0.31	0.36	349

Accuracy	0.69

- The model performed well in predicting No/Minor Injuries (class 0) but struggled with Moderate (class 1) and Severe/Fatal (class 2) injuries due to the complexity of distinguishing between these categories:
    - Class 0: Precision = 74%, Recall = 88%
    - Class 1: Precision = 52%, Recall = 32%
    - Class 2: Precision = 42%, Recall = 31%

- Although the model uses SMOTE to handle imbalances in the data, we still see signs that the model is biased towards Class 0 (No Injury) type crashes in how the accuracies are favored towards Class 0 and not to Class 1 or Class 2.

Decision Tree

The Decision Tree model achieved an overall accuracy of 88.75%. However, we have to keep in mind that the dataset was heavily imbalanced, potentially leading to the model being biased towards the low severity.

It performed well in predicting 'High Severity' crashes (precision: 1.00, recall: 1.00) but struggled with 'Moderate' crashes (precision: 0.63, recall: 0.83). This suggests that the model might need further refinement to improve its ability to identify moderate crashes.
Since we have an unbalanced data set, this “Overall accuracy” shouldn’t be regarded as final, instead, it was used as a reference for the other models to improve upon it.

Decision Tree

Accuracy	88%

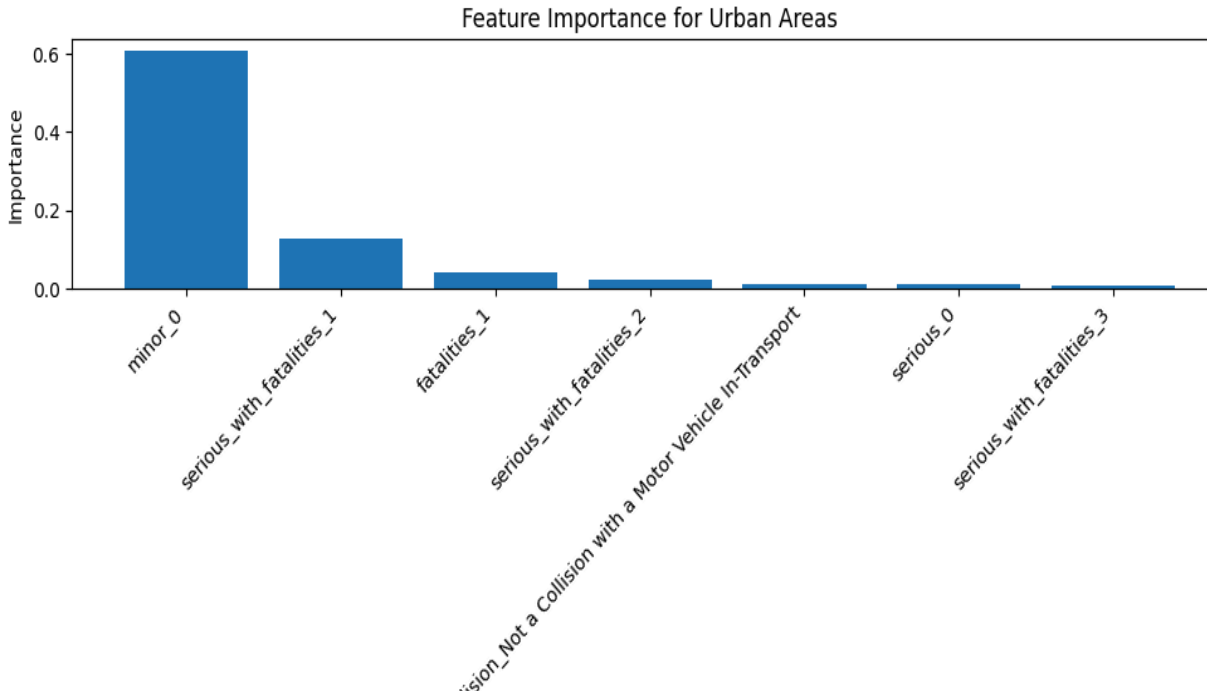
Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Low)	0.96	0.90	0.93	13,675
1 (Moderate)	0.63	0.83	0.71	2,820
2 (Severe)	1.00	1.00	1.00	112

Next we’ll see the Decision Tree’s results for each area; Urban, Rural, and Suburban. WIth their respective feature importance charts.

Urban

Accuracy	87%

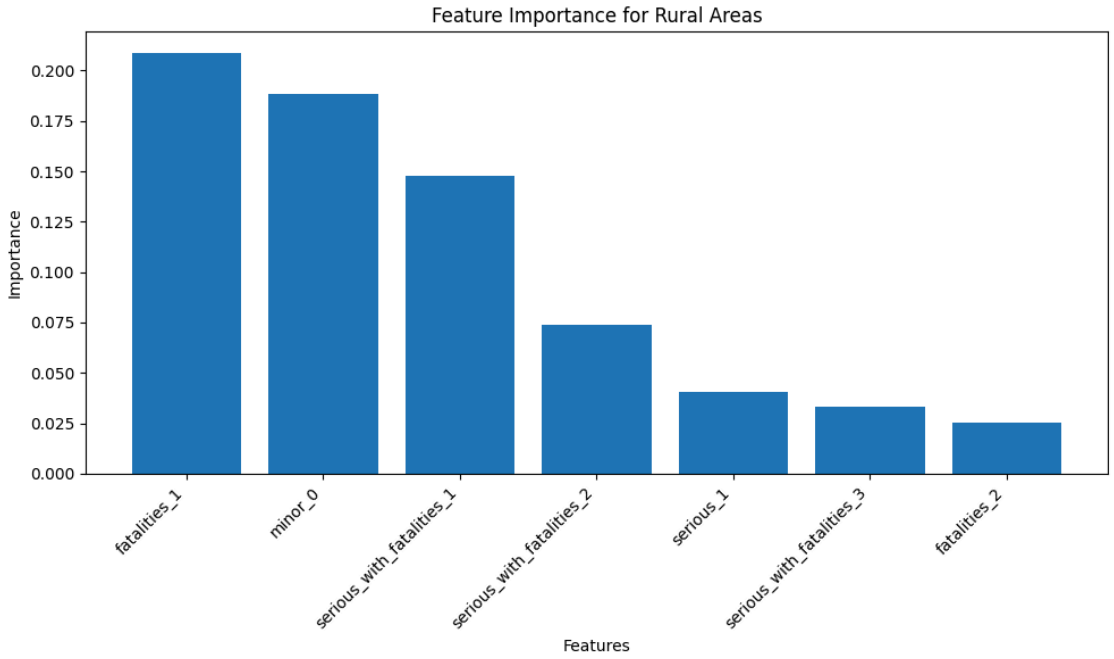
Class	Precision	Recall	F1-Score	Support
0 (Low)	0.97	0.87	0.92	8,527
1 (Moderate)	0.59	0.89	0.92	1,819
2 (Severe)	0.97	1.0	0.98	31



Rural

Accuracy	90%

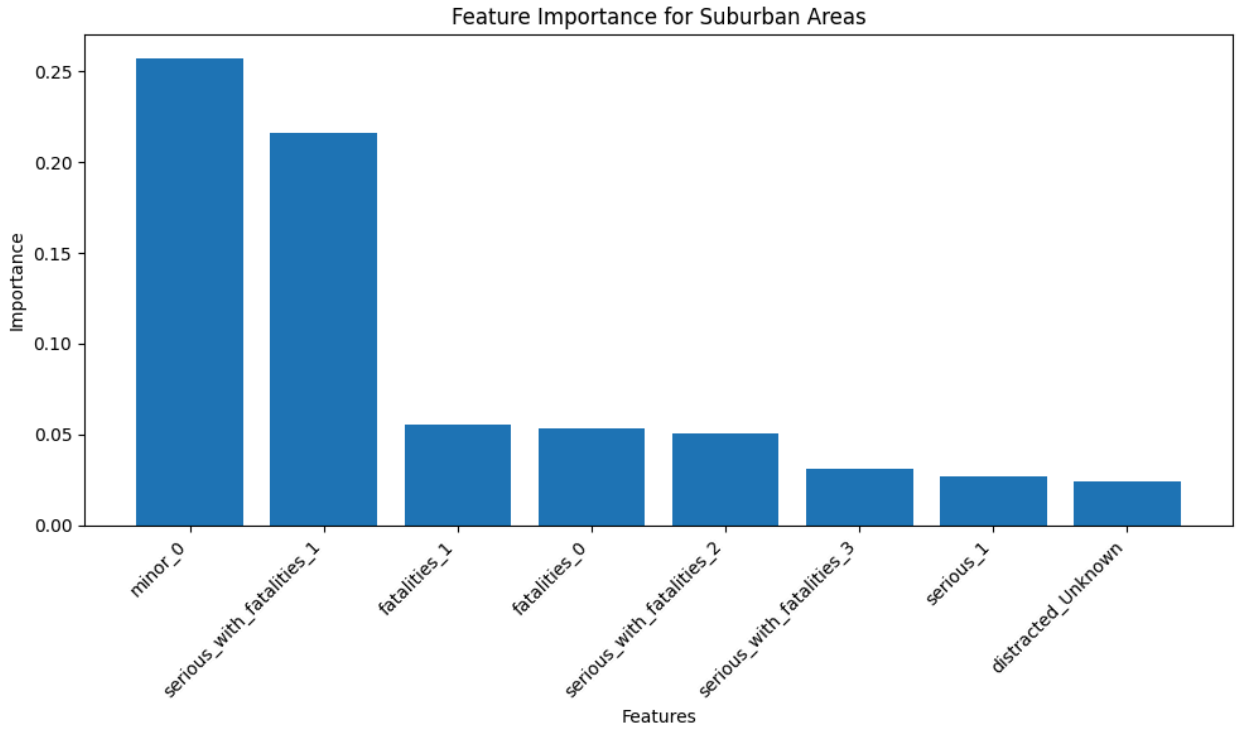
Class	Precision	Recall	F1-Score	Support
0 (Low)	0.92	0.97	0.95	1,552
1 (Moderate)	0.80	0.55	0.65	286
2 (Severe)	0.95	0.90	0.92	39



Suburban

Accuracy	85%

Class	Precision	Recall	F1-Score	Support
0 (Low)	0.96	0.86	0.91	1,389
1 (Moderate)	0.51	0.82	0.63	249
2 (Severe)	0.94	0.84	0.89	19



Random Forest

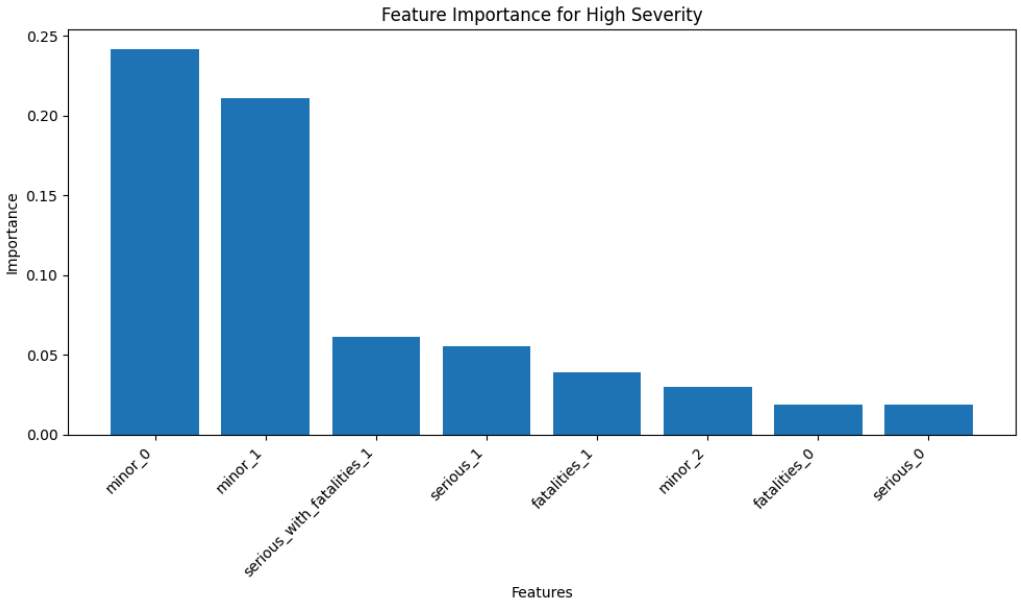
The Random Forest model achieved an overall accuracy of 87.61%, which was slightly lower than the Decision Tree. However, this doesn't necessarily mean that it performed worse, since the dataset is imbalanced we actually have almost exactly the same results if not better. The Random Forest model makes a more 'better' prediction which in turn slightly "lowers" the accuracy, but it does not mean that it's less accurate. We believe it predicted severity well. This demonstrates the benefits of using an ensemble method to improve prediction accuracy.

Random Forest Classification Report

Accuracy	87%

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0 (Low)	0.89	0.97	0.93	11,418
1 (Moderate)	0.75	0.43	0.55	2,404
2 (Severe)	1.00	0.85	0.92	96

Random Forest High Crash Severity Feature Importances



The feature minor_0 has the highest importance score by far. This suggests that the absence of minor injuries (no minor injuries reported) is a very strong predictor of a crash. Crashes without minor injuries are more likely to be classified as low severity.

The features minor_1 (minor injuries) and serious_with_fatalities_1 (serious injuries with fatalities) also have relatively high importance. This suggests that the presence of injuries, especially serious ones with fatalities, plays a significant role in predicting higher severity crashes

The features related to different injury types (minor, serious, fatalities) show a gradient of importance, with the absence of minor injuries being the strongest predictor, followed by the absence of serious injuries with fatalities, and then the presence or absence of other injury types. This reflects the logical progression of crash severity based on the types of injuries involved.

Multilevel Random Forest

The Multilevel Random Forest showed our most promising results. Not only does the model help answer the primary question of our project (which contributing factor correlates most with a specific crash type based on area (urban, rural, suburban)?) but it also shows reasonable accuracy that also reduces bias shown in other models.

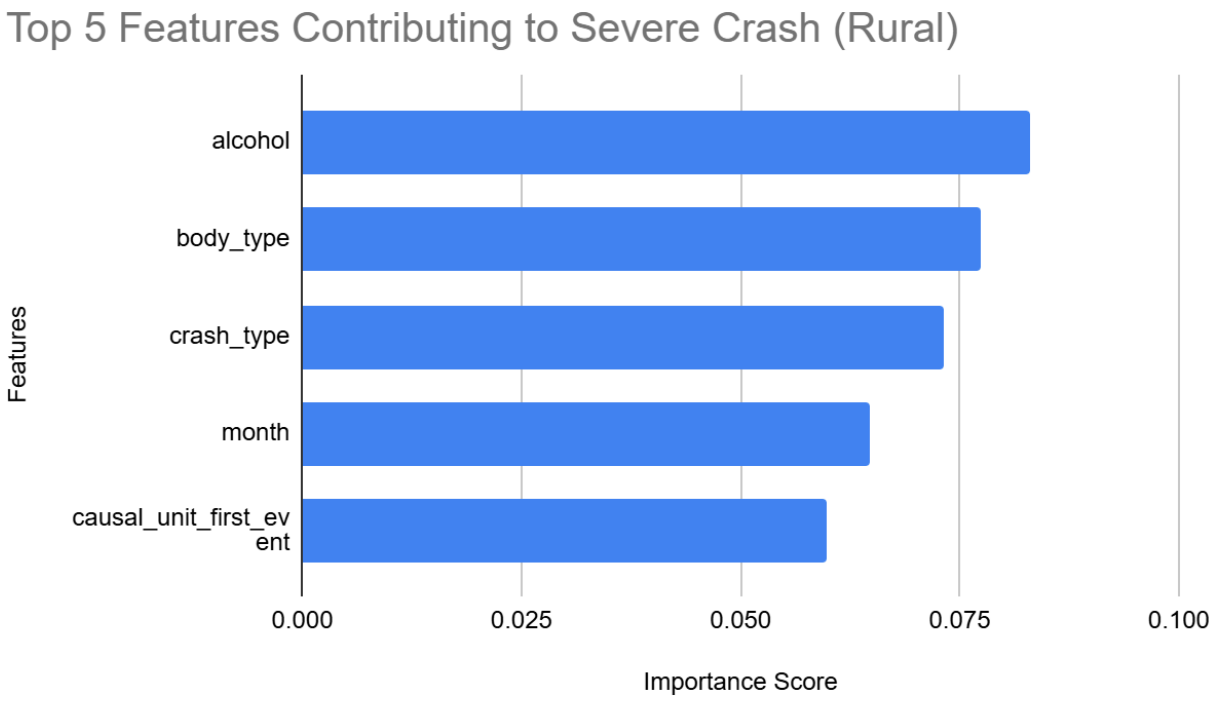
Our process involved downsampling the majority classes within each area to ensure equal representation between the classes being compared. The model handles balancing independently for each area - with urban moderate crashes, it first isolates urban crashes, splits them into "moderate" vs "not moderate," and then downsampled the larger group to match the smaller group. This process happens separately for each area, which is why we still see imbalances between areas - urban areas have 20,652 moderate crashes compared to about 3,300 in rural and suburban areas. While this area-level imbalance could be addressed with an additional balancing step across areas, the current approach ensures each area's model learns from balanced data.

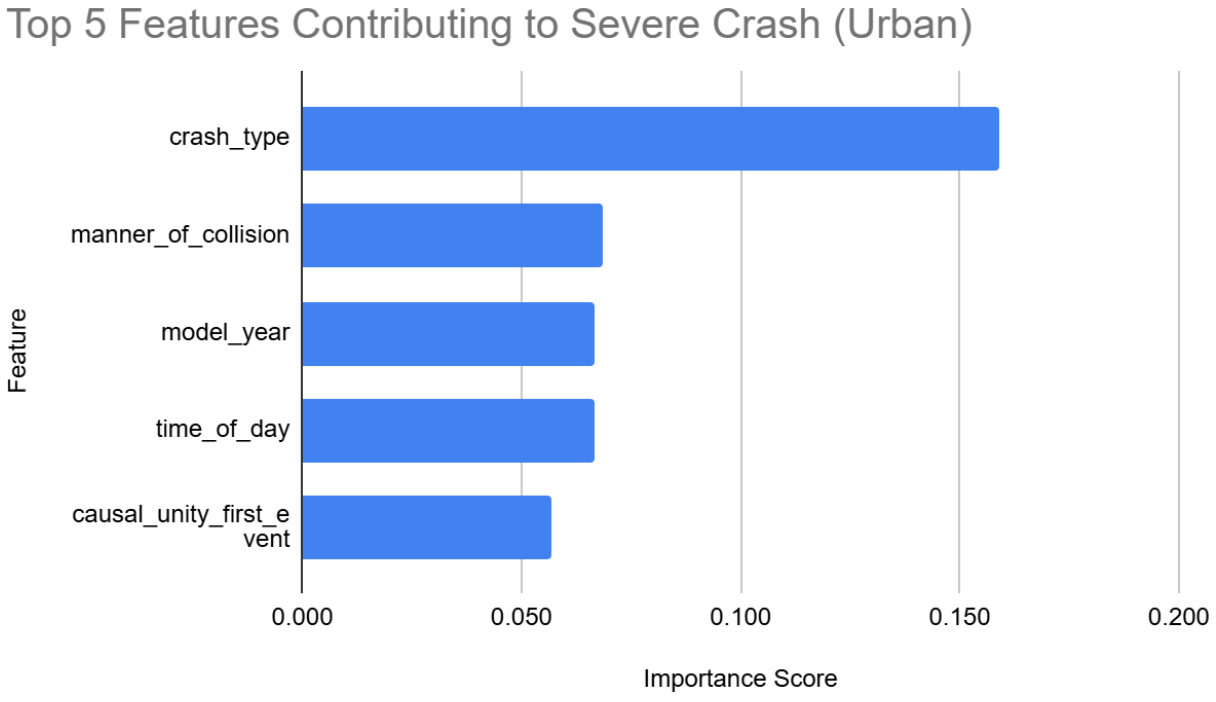
At first glance the accuracy for our Decision Tree and Random Forest appears to have a higher accuracy, these scores were misleading because they did not properly address the data imbalance problem and showed biased predictions. The multilevel model's results showed more reliable patterns, most notably achieving 77.2% accuracy and 84.3% ROC-AUC for severe crashes in rural areas despite having relatively few cases (1,190). This strong performance with limited data suggests that severe car crashes in rural areas have the most distinct and predictable patterns. In contrast, urban areas showed lower performance (73.4% accuracy) despite having more data, indicating less predictable crash patterns in urban environments.

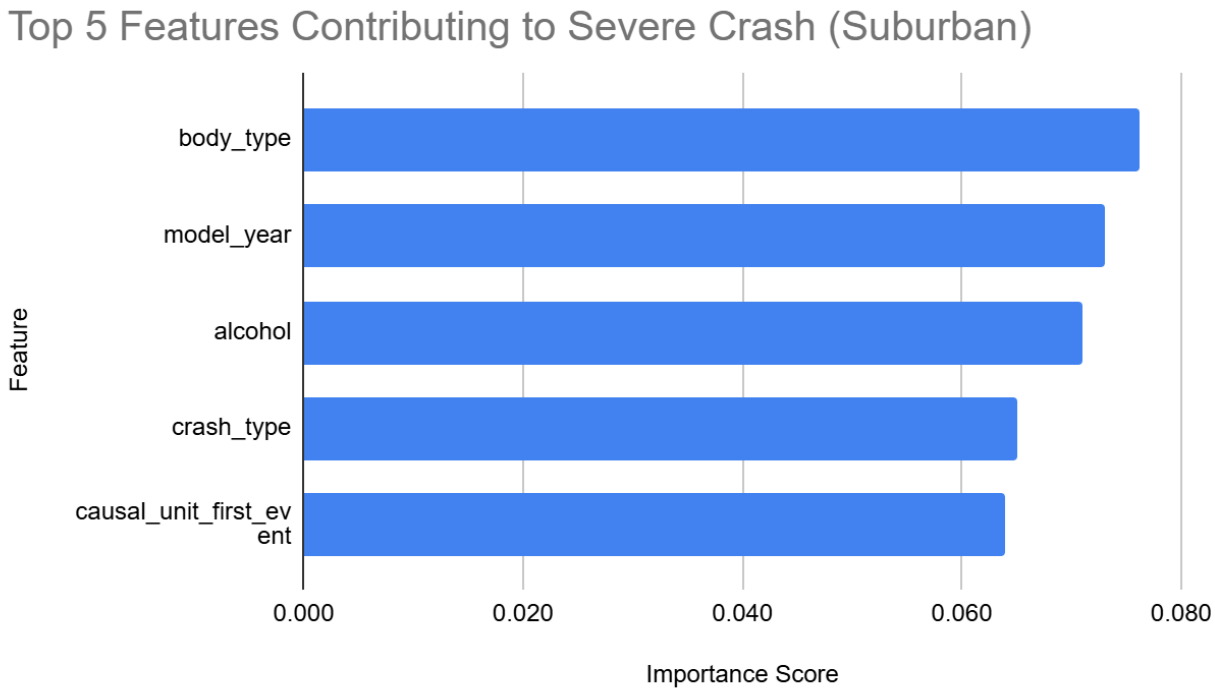
No Injury			
	Rural	**Urban**	**Suburban**
Accuracy	0.677176	0.621314	0.657425
Precision	0.665501	0.622782	0.645201
Recall	0.708802	0.61599	0.699416
F1	0.686278	0.619257	0.671076
ROC_AUC	0.746404	0.672016	0.709973
Support	5170	26386	4434

Moderate			
	Rural	**Urban**	**Suburban**
Accuracy	0.63	0.63	0.63
Precision	0.62	0.62	0.62
Recall	0.68	0.64	0.66
F1	0.65	0.63	0.64
ROC_AUC	0.69	0.68	0.68
Support	3392	20652	3318

Severe			
	Rural	**Urban**	**Suburban**
Accuracy	0.77	0.73	0.75
Precision	0.79	0.75	0.76
Recall	0.75	0.71	0.73
F1	0.77	0.73	0.75
ROC_AUC	0.84	0.81	0.81
	1190	1574	664


Top 5 Features Contributing to Severe Crash (Rural)
alcohol
body_type
crash_type
month
causal_unit_first_ev
ent
Features
0.000
0.025
0.050
0.075
0.100
Importance Score


Top 5 Features Contributing to Severe Crash (Urban)
crash_type
manner_of_collision
model_year
time_of_day
causal_unity_first_e
vent
Feature
0.000
0.050
0.100
0.150
0.200
Importance Score



Based on the feature importance charts for severe crashes in rural, urban, and suburban Alaska, gathered from the multiple models, the results tell us the following:

Rural

- Vehicle characteristics play a crucial role in rural severe crashes.
- Seasonal patterns are more important in rural areas.
- Alcohol is a more significant factor compared to other areas.

Urban

- Specific crash types dominate urban severe accidents.
- Traffic patterns and vehicle interactions are crucial.
- Time of day is more important than in rural areas.

Suburban

- Shows a hybrid pattern between urban and rural
- More balanced distribution of important factors
- Vehicle characteristics remain important

Future Research

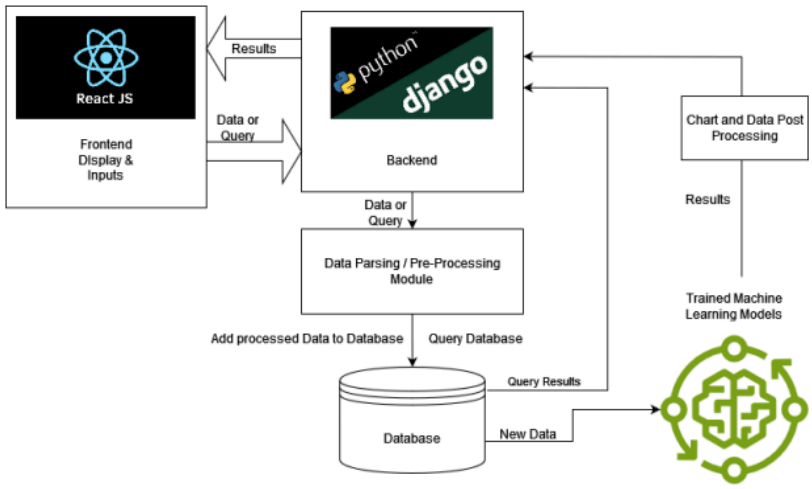
Our research expanded on work done by Lydia Stark and Cale Cornichuck in 2022 for their capstone project [1]. Our research focused mainly on the differences in crash characteristics between urban, suburban, and rural areas. When discussing our research with Anna Bosin from the DoT she suggested looking into the characteristics of the vehicles involved in crashes. Also, during our poster board session, the most common questions raised was about the vehicles involved in crashes. We realized that there is great interest in what types of vehicles get in certain types of crashes. We see future research in the arena of crash characteristics pertaining to different types of vehicles. The current data contains a few of these elements that could be analyzed, those being vehicle body type, model year, and model of vehicle. Combining these attributes of a crash with vehicle data from the DMV could lead to interesting insights into car crashes in Alaska.

Crash Analysis Tool

Design

We built the Crash Analysis Tool with the Django python web application framework. This framework was chosen due to its ease of use, scalability, and our prior experience with the python programming language. Python is also an excellent language to work with ML models because of the numerous libraries that aid in ML, like scikit learn. Our design went through a few iterations. At first, we wanted to use a Django backend and a ReactJS frontend that used API calls to interface with said backend. As time went on, we realized that Django supported all the frontend requirements we desired and decided to scrap the ReactJS part of the project.

Backend Architecture Mockup



Original Backend Design

Development Process

The development of the web app commenced after we had completed data pre-processing and creating documentation for why the columns we are using for our dataset are included. We used Django for our web app framework due to its scalability and our familiarity with python. Before we started writing any code, we designed our backend data pipeline as we knew this would be a challenge to work with if not done properly. We then implemented this pipeline as the first feature in our site; we tested it heavily in development to make sure data retrieval was robust enough to handle more complicated operations in the future. Once we were able to retrieve data, we started working on how to filter it. By using POST requests sent to backend python code, we could - in the front end - customize our data, and have these customizations be sent to the backend to be implemented. Once we were satisfied with filtering, we moved on to the Machine Learning module. We implemented the models created for our analysis and tweaked our backend to allow these models to work with the data on the site. After much of the functionality of the website was established, we worked with css and html to make it look better and improve the user experience. A lot of the refactoring done in the late stages of the project involved reducing the number of POST requests sent to the backend to improve the flow of the site. Also, in the later stages of the site, we implemented security features. A user cannot access the data, or any other page than the home page if they do not have credentials given to them by us. We stayed on schedule and were happy to be in a solid place coming into the final weeks of the semester.

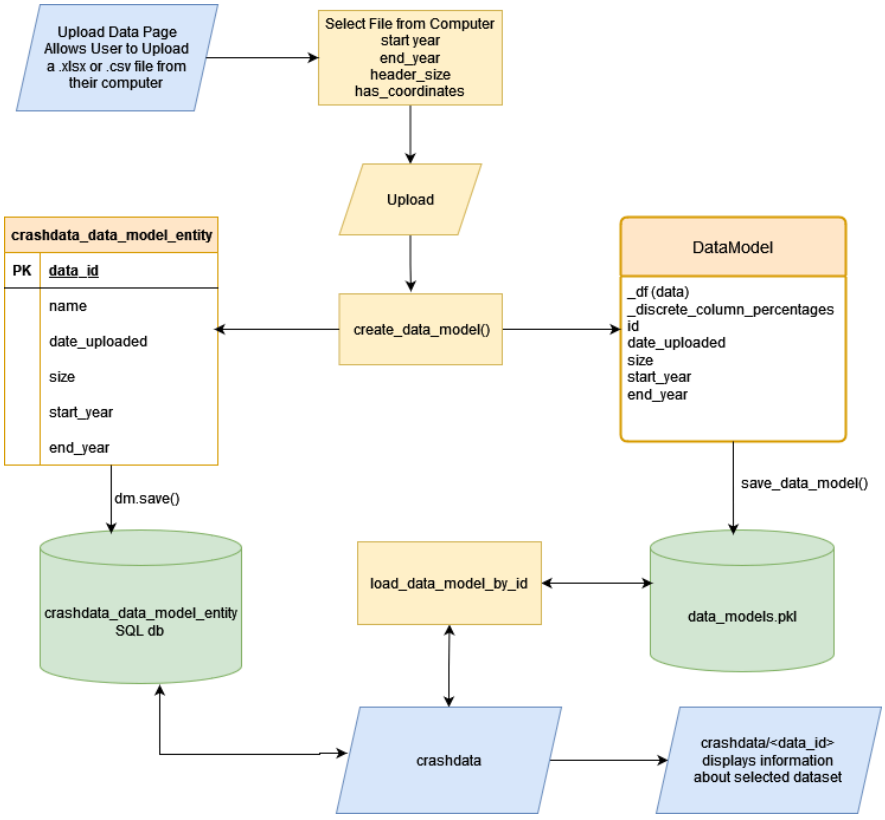
Project Schedule

Sprint 1 | 9/16 to 9/30	● Git Remote Repo for Project established ● Roles Established ● Establish Guiding questions ● Gather new data ● Cleaning data for use in ML models ● Establishing what models to use and how to run them ● UI/UX flow diagram ● UI/UX Wireframe ● Column Explorations and justifications for keeping them for research
Sprint 2 | 10/1 to 10/15	● Train ML models on specific data and factors and validating results ● Starting to integrate data with Django app ● Allow uploading and storage of data on web app

Sprint 3 | 10/16 to 10/30	● Integrate ML Models in DJango web app ● Share results of models to front end ● Enhance UI ● Advanced Data Filtering Feature
Sprint 4 | 10/31 to 11/14	● Code refactoring and enhancement ● Ensure complete security ● User testing ● Better flow - reduce requests sent to server ● Enhance ML Results output
Sprint 5 | 11/15 to 11/29	● Deployment ● Work on final report and poster ● Code Clean up/ Debugging ● Heavy user testing ● Enhance UI/UX
Sprint 6 | 11/30 to 12/10	● Deploy App live ● Bug Fixing ● Security Testing ● Finish Report and Poster

Data Storage

The first challenge we faced was how to store, retrieve, and use the data we uploaded to the site. We used code we wrote for cleaning the datasets. The idea of a data model class was used as a catch all for cleaning and querying data we were working with. This data model class housed a few key features for each set of data, chiefly the data itself, which would be stored as a pandas DataFrame. The idea of using pandas on the Django app made sense; it would allow us to write simple code to filter the data and retrieve specific aspects of the data for use in our ML models. To store the data on the site, we used the python pickle module that allows one to store custom python objects in one place, and retrieve that data based on a certain characteristic of the object. We gave each data model a unique ID that would allow us to call upon the data whenever we needed it on the front end. To store the unique IDs we used the built in SQLite database that comes with Django. Each time a dataset is uploaded to the site, we create a data model to store in the pickle file, as well as a data_model_entity that stores the information needed to retrieve the data. So, when looking at a list of the datasets that have been uploaded to the site, the user is seeing the data_model_entites. When one is clicked, we look through the pickle file for the object with the corresponding data ID, we then retrieve that object, and load the DataFrame attached to it.



Web app data pipeline

Current Functionality

The web application in its current state has all the functional requirements we wanted. A user is able to upload data, access this data, filter it, and run a ML model on it. The landing page includes information about the site and our project. One can then go to the Datasets page to see what data is available, or they can upload their own data to be included in the list of data sets.

Data Filtering

The data filtering tool allows the user to perform advanced filtering to a dataset. One can filter the dataset by any combination of row values, delete columns, and drop columns with an unwanted amount of unknown values. The function for clustering is also an important feature. The clustering function allows the user to select a column, and cluster values in that column. So if you have 5 discrete values, one can group some together to form 2 groups - one of 2 values, and one of 3. This allows a user to aggregate values when they are similar or if there are not many instances of a certain value. An example of this is grouping crash severity together; the value ‘Died Prior To Crash’ shows up very few times, so we are able to group this together with ‘Fatal’. One can then download a filtered dataset for whatever needs they may have.

Machine Learning
We included a feature for a user to run a variety of ML models on the site. The models have parameters and methods developed during our ML research. One can simply upload data, choose a model, select what attribute to predict for, and run. The output is a page that tells the user a few things. First is the classification report, which says how accurate the model was at predicting and giving it different scores based on different formulas, like F1, recall, and precision. It outputs a normalized confusion matrix - which is just an accurate label vs. predicted label table. And we also included a graph displaying the most important features in predicting a label. A user is able to download a folder including the confusion matrix, feature importances, and classification report.

Deployment
Deployment of the site proved to be challenging. We settled on using pythonanywhere as our hosting service as it is great for django web app support. The site works as it does in our local development environment except for the smoothness of using the ML models. That page in particular has some issues with persisting the data on the page. However, we are happy to have our app be live and accessible to those with credentials which we shared with our client.

Challenges and Future Work

Data Collection
One of the things we set out to do in this project was to augment the existing data with new attributes that could lead to more insight into the data. The most notable feature we added was an area classification. This came about after analyzing vector and shapefiles in Google Earth Engine and determining if an area was urban or rural. While this process worked fine, the ideal way to have done this would have been to vectorize a raster file. We had a raster file that was perfect for this, as it distinguished between urban and rural areas. We were able to vectorize some parts of it, but the combination of bad coordinate transformation and sheer number of vectors created made the process convoluted and inaccurate. Another issue we faced with new data collection was trying to gather accurate weather data. While the dataset does have an attribute for weather, we were attempting to use a historical weather database to query the date and location of the crash to have weather be as accurate as possible. When researching for this task, we found that the number of historical API calls to a database would cost a lot of money and without the promise they would even have data from remote Alaskan locations. So, we looked to NOAA. Their historical API is free, but to query it, you cannot use latitude and longitude, you must use the station ID. We downloaded a file of all the stations in Alaska, and used python to determine the closest station to a crash site. We then used the Date Time data from the crash to query the API. We found this process to be flawed, and the API was returning a lot of null data. This process

needed more refinement, but we ran out of time to perfect the data collection process. In the future, gathering the best weather data possible for crashes could be of great importance.

Machine Learning

One challenge we faced when implementing the machine learning model was determining which features were most significant during the training process. The dataset's scale included over 50,000 crashes and more than 200 features which required significant feature reduction. Although we successfully narrowed it down to approximately 35 features, we encountered issues with redundancy and biased results within this subset. Dr. Visudevan's civil engineering expertise was invaluable in identifying these challenges and guiding us toward more meaningful feature selection and interpretation of results.

Another challenge we encountered was the highly imbalanced dataset. As shown in the diagram above, non-injury crashes were significantly overrepresented compared to severe crashes. This imbalance became even more pronounced when separating crashes by area (urban, rural, suburban). Addressing this imbalance required us to experiment with various techniques, such as applying class weights during training and using SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic samples. However, many of our models still exhibited biased results and signs of overfitting, especially when predicting minority classes. While we achieved more balanced outcomes through undersampling, this approach reduced the amount of data available for training, which is not ideal.

Web Application

For the most part, the development of the web application was smooth. We executed a planned data pipeline architecture and it proved to be a robust way to render and manipulate data. We had challenges in developing the interactive ML models. The issues we faced involved allowing customization of parameters to send to the model. This is something that could be added in the future. We were unable to implement advanced parameter control unfortunately. We also feel as though some features involving data manipulation were left on the table. In the future, we would like to add functionality to allow a user to query and build a custom dataset using SQL. This feature would allow a user to have even greater control and understanding of the data. This feature could also have built in aggregating functions to see more advanced analytics for the data. Another thing we could have added but ran out of time for was error messages to the user. As of now, if there is an error with running a ML model, the site does not tell the user, it just does not run. We would like to have the user understand the error and give them the power to fix it.

One of the largest challenges in the latter stages of the project was deployment. We tried using the hosting site pythonanywhere. When we decided on a final version of the app, we updated the

code and fixed the dependencies, but the site was having issues rendering some of the css and JavaScript updates. It was also having trouble with running the Machine Learning models. We then tried using on Render but this was an even worse result. So we went back to pythonanywhere. After tweaking the code and debugging, the site became operational. There is an issue with persisting the data uploaded to run on a ML model. This feature does work, but it's very finicky and it is hard to deduce the cause for the choppy data persistence.

Conclusion

In our Capstone project, we focused on two key objectives. The first one was investigating the factors contributing most to car crashes in Alaska categorized by area (urban, rural, suburban). This involved implementing and testing a variety of machine learning classification models. After experimenting with several approaches, we found that the multilevel random forest model provided the best accuracy while also allowing us to group results by area. When presenting our findings to our client, Professor Vinod Vasudevan, and Anna Bosin, Traffic Safety Engineer at the Alaska Department of Transportation, we were pleased to learn that our model's results closely aligned with those of the Department of Transportation.

The second part of our project centered on developing a web application that enables users to upload their own data and run different machine learning models. This platform was designed to make machine learning accessible to individuals without any technical background, allowing them to analyze their data using the models we created.

In the future we see several opportunities to enhance the project. The web application could be expanded to incorporate additional data sources or machine learning models and provide more comprehensive insights. Our original design of the website included using a JavaScript framework for the frontend. Although this was cut during our project, we believe it would still be ideal to use a JavaScrip framework to enhance performance and usability. Refining the accuracy and interpretability of our models could contribute to a deeper understanding of traffic safety and inform public policy decisions. Overall, we are proud of the impact our project has had and the potential it holds for improving traffic safety in Alaska and beyond.

References

[1] https://coeng.uaa.alaska.edu/innovation/?p=2527

Appendix

Datasets for adding area classification

[1]2010 TIGER Census Data
https://developers.google.com/earth-engine/datasets/catalog/TIGER_2010_Blocks

[2] Degree of Urbanization Dataset
https://developers.google.com/earth-engine/datasets/catalog/JRC_GHSL_P2023A_GHS_SMOD_V2-0

GitHub Repo for Web Application: https://github.com/UnderYourSpell/akCrashData
GitHub Repo for ML Research: https://github.com/yjseto/capstone