Analysis of Vehicle Accidents and Injury Prediction Using Machine Learning Algorithms

Andrew Kehl, Dhruv Jain, Edward Montoya, Saumya Sinha

Department of Applied Data Sciences, San Jose State University

DATA270: Data Analytical Processes

Professor Eduardo Chan

2022, December 9

Contents

Abstract

Vehicle accidents and associated injury severity are a genuine concern for the city of San Jose, California. Based on a 2019 report from the Statewide Integrated Traffic Records System (SWITRS), there were 10,498 persons injured in traffic accidents in the San Jose area, of which 131 were fatally injured. This project aims to implement a variety of machine-learning and deep-learning techniques such as Logistic Regression, Random Forest, Extreme Gradient Boostings and Multi-Layer Perceptron to understand the types of behavior or situations that increase risk by accurately predicting injury severity for vehicle accidents in the San Jose area; it can help shape the decisions of city drivers, and city planners, making the city safer and even saving lives. Prior research shaped the understanding of the necessity of balance shaping classes in this domain. Predicting the severity of injuries in vehicle accidents using machine learning is not a novel approach; what distinguishes this paper from many others is the inclusion of environmental data such as weather and crash location. The models were evaluated with classification metrics such as Receiver Operating Characteristic, Area Under Curve, and F1-score. It was concluded that features such as age, if a police report was filed, distance to the intersection, and gender were key. Finally, the strongest model was Random Forest, which demonstrated the best performance with an F1-macro score of 0.4795 and an accuracy of 70.54% for its ability to predict injury severity.

1. Introduction

1.1 Project Background and Executive Summary

Driving is a vital part of the United States economy and everyday life. It allows supply chains to function and people to commute to work. Based on information from the Department Of Transportation (2021), the number of miles driven on highways in 2020 was estimated to be around 2.9 trillion miles. In the same year, the Insurance Institute of Highway Safety (2022) estimated that 38,824 people died from vehicle accidents, which resulted in the loss of lives and an estimated $242 billion in economic cost.

Traffic safety and injuries are especially a problem for the state of California. Based on a 2021 annual report from the California Office of Traffic Safety (2021), in the last 12 years, more than 30,000 people have lost their lives, over 100,000 people were seriously injured due to accidents, and California accounted for 3,847 fatalities, which is a 3.4 percent increase over the previous year. These numbers have steadily increased over the past 16 years based on data from the National Highway Traffic Safety Administration (2022) and perhaps require a closer look at what is causing the accidents and injuries.

This paper analyzes vehicle crashes, road conditions, and weather to find features that strongly correlate to traffic accidents in San Jose, California. In addition, the paper is focused on utilizing machine learning and deep learning models to predict injury severity, which has not been done for the San Jose area.

The data for this research was collected from a single source, the City of San Jose, and includes details such as vehicles, crashes, weather, and road surroundings. The data is collected and stored in CSV format, though the city of San Jose does give the option of several different data formats. Python libraries such as pandas were used to clean the datasets individually and

then merged into a single data frame for further analysis. After this, data preprocessing methods such as imputation and handling outliers occured. The exploratory data analysis was performed using libraries such as Matplotlib and seaborn. This allowed the understanding of how road and weather features correlate. Finally, the target variable - crash severity was extrapolated using feature engineering techniques such as categorical encoding.

The prediction models were created using the library Sci-kit learn and TensorFlow/Keras. The creation took place in increasing order of complexity. Logistic regression was used as a baseline model. Decision Tree (DT), Random Forest (RF), XgBoost (XgB), and a Multi-Layer Perceptron model were also implemented. The main evaluation metrics used are F1-macro Score and Area Under the Receiver Operating Characteristic Curve (ROC-AUC). The models were compared using these metrics to compare and evaluate the predicting capability

The models were also be used for identifying which features have the highest correlation to injury severity; this could allow for insightful recommendations on safety measures that could be implemented to mitigate the cause of such injuries. Theoretically, this could potentially equate to saving lives and lowering the economic cost of crashes.

The goal of this paper is to utilize machine learning and deep learning techniques to identify feature selection and then utilize those features to build models to predict future crash severity. These models have the potential to positively influence decision-makers in their ability to adapt novel policies for traffic safety measures.

In the future, the models could potentially be deployed to allow for immediate predictions of crash severity. This could benefit emergency response systems to allocate necessary resources properly.

1.2 Project Requirements

No data privacy issues are present in the datasets; the data utilized is either publicly available and has been scrubbed of any personally identifiable information. Importantly the tools of choice are Python 3.11 programming language in both Jupyter Notebook and Google Collab environments. The entire codebase is available upon publication or submission of this paper. Another tool utilized was Google Drive, which is for writing and formatting the paper, storing notebooks, along with collaboration. All machine learning models were run on local machines and Google Collab; much of the work was written and validated locally, then merged into the Google Collab environment and run again.

(Sperandei, 2014) Logistic Regression (LR) is a classification-based statistical model based on the probability of an event occurring. LR is used as a baseline model as it is a relatively less complex algorithm. This model can later be compared to more complex algorithms to evaluate the performance. Also, LR was be used to bring out relationships between the features.

Decision Tree (DT) is a supervised learning algorithm that can be used for both classification and regression problems. It consists of nodes and leaves where a node represents a feature, and the leaf represents its expected value. DT was chosen as it helps understand essential features affecting the target variable using Decision Rules (DR). A DR is a set of rules based on which the expected values are obtained (Abellan, 2013).

(Carmona et al., 2022) XGboost is an ensemble supervised machine learning algorithm. It is based on the gradient boosting decision tree technique. XGboost was chosen for its scalability and flexibility during hyperparameter tunning. One significant advantage is that it has various regularization methods, which can be helpful in curbing overfitting.

Random Forest (RF) is a classification algorithm that uses bagging techniques along with multiple independent decision trees in order to create an uncorrelated forest of trees(Louppe, 2014). In this classifier, each tree provides the classification, which is then summarized using the majority vote or confidence vote strategy. It has a shallow risk of overfitting since the results are calculated based on multiple decision trees.

The last model implemented in this paper is a Multi Layer Perceptron (MLP) model. A MLP is the foundation of deep learning, it is the most basic neural network. The goal of an MLP is to emulate a brain. An MLP is a network, a mesh of these perceptrons connected to each other in a layered order. It is a method where the same data is passed through the model numerous times allowing the model to update the kind of importance it puts on each feature (weights) and in what situation those importances become relevant(bias) (Hammerstrom, 1993).

There is only one data source but two separate datasets. The choices in datasets was made are related specifically to the scope of this project. San Jose data specifically dissected and the first place looked at was the San Jose vehicular crash records. Table 1 below contains the relevant number of records and a selection of variables in all the datasets used.

San Jose reports all crashes with some of the exact circumstances surrounding those crashes. This dataset consists of two tables. One contains vehicle-specific crash data from 2011-2020. The other table contains the specific crash data independent of the vehicles involved, such as location and road conditions. All of the data is presented in .csv format. There are just over 110,000 records between the tables, though much of that data was joined, bringing the record count down significantly (*San Jose Vehicle Crash Data*, n.d.).

Once the data is collected, the datasets were be joined into a single table. Each record was one vehicle accident with as many variables from each table joined together, with the label being how severe the accident is.

Table 1

Datasets with Selection of Variables and Records

Source	Name	Number of Records	Relevant Variables/Attributes	Limited Selection of Included Variables
***San Jose Vehicle Crash Data*, n.d.)**	Crashes 2011-2020	25,500	22	minor injuries, moderate injuries, severe injuries, fatalities, roadway conditions, lighting conditions, collision factor, weather
(*San Jose Vehicle Crash Data*, n.d.)	Vehicles 2011-2020	115,302	13	sex, vehicle type, age of driver, speed, party type, vehicle count, sobriety

1.3 Project Deliverables

Our project deliverables are considered within the scope and help support the project's objective. As part of the deliverables, listed there are a few essential items to keep in mind to understand better the tools or resources needed for this project.

Our first deliverable is keeping all the project execution plans at one shared location. Trello and third party Trello plugin, Team Gannt, were used as online project management tools where one can access, create, assign, and resolve the tasks allocated to relevant members. With the help of Trello and Team Gannt, the tasks were distributed equally based on the group member’s availability and strengths. The management approach was the CRISP-DM model approach to ensure all aspects of the project are covered.

Secondly, there is a detailed assessment report of datasets in the form of a python file. It consists of tasks related to data cleaning, such as removing duplicates or irrelevant data, fixing structural errors, handling outliers and missing values, and visual representation through graphs, charts, and plots. Also in the report there is also an operational structure of each aforementioned machine learning and deep-learning models with a high level of precision, accuracy, and predictability.

Each task was assigned and monitored to ensure completion on time. Doing this allowed the group to keep track of the initial set of objectives and actual project accomplishments. The deliverables comply with the deadline for each assignment, as assignments contain the subtasks of the project deliverables. Table 2 shows the description of the deliverables and expected completion dates.

Table 2

Deliverable Schedule

Deliverables	Finish Date	Due Date
Project Proposal	September 17, 2022	September 19, 2022
Chapter 1 - Introduction	September 23, 2022	September 25, 2022
Project Management	September 23, 2022	September 23, 2022
PM Tools & WBS	September 29, 2022	September 30, 2022
Data Management (DMP)	October 7, 2022	October 7, 2022
Effort Estimates, PERT & Gantt Charts	October 14, 2022	October 14, 2022
Data Collection and Data Exploration	October 14, 2022	October 14, 2022
Data Collection and Data Exploration	October 21, 2022	October 21, 2022
Data Engineering	October 28, 2022	October 28, 2022

1.4 Technology and Solution Survey

Many studies have been conducted to understand factors affecting traffic accidents and to predict injury severity, but the methods and processes have been different in each of them. Understanding these studies helped navigate the problem statement and produce more accurate results.

In a paper by Lu et al. (2015), the authors state, "A dynamic equilibrium driving system is made up by the driver, the car, the road, and the environment. Any element's failure of this system will lead to a traffic accident, and the incentives could be subjective or objective or both" (Lu et al., 2015, p. 2). Hence, to provide early crash warnings and improve safety, the authors decided to investigate how objective factors such as the road, the area around the road, and the vehicle impact traffic accidents. Crash data related to Bejing's ten main roads were sourced, and logistic regression was used to find that relationship. The target variable was accident hotspots. After the preprocessing steps and model creation, the authors concluded that accidents occur in a complex and random pattern (Lu et al., 2015, p. 4). However, they obtained 86.67% accuracy on accident hotspot prediction. But they want to take several factors such as meteorological data and other traffic-related data in future research.

In another paper, Santos et al. (2021) conducted a project under "The University of Évora in partnership with the Territorial Command of the GNR (National Republican Guard) of Setúbal, Portugal, financed by the FCT (Foundation for Science and Technology)" (Santos et al., 2021, p. 2). The main objective behind this project was to build a predictive model to reduce and estimate risk-increasing factors and the number of accidents around Setúbal district, Portugal. The author focused on building "the rule generation model to support analysis on the accident dataset, addressing the responsible factors of severe traffic accidents. Furthermore, the predictive model aims at mapping accident hotspots, highlighting areas where accidents are in given circumstances, accidents are likely to happen"(Santos et al., 2021, p. 2). Table 3 contains the comparison of the performance of the different models. The result showed that the RF model was the most effective regarding the accuracy, AUC, precision, sensitivity, and specificity. This made it easier to distinguish the most influential features.

Table 3

Model Performance Based on Paper by Santos et al.

Model	Accuracy	AUC Score	Precision	Recall/Sensitivity	Recall/Specificity
Random Forest	0.73	0.68	0.44	0.08	0.97
Logistic Regression	0.73	0.66	0.27	0.00	1.00
Decision Tree	0.65	0.55	0.35	0.34	0.76

Choosing the most suitable machine learning algorithm for predicting accident severity can be a mixed bag; there is a ton of research in the area but not a true consensus on the best model for this type of prediction. Fiorentini & Losa (2020) found that the best model they could come up with was using a K-Nearest Neighbors algorithm. That is not to say there is one best model for this kind of prediction: there are many complicating factors, from the kind of input data to the specific output goal. (Fiorentini & Losa 2020) used and tested a variety of models in their paper beyond the KNN, such as RF , Multivariate Logistic Regression, and Random Tree. The importance of this paper cannot be understated; instead of trying to get the best accuracy scores, this paper was specifically written to determine the best model for the imbalanced datasets typically presented when predicting accident severity. The authors determined the best course of action was to use a Random Undersampling of the Majority Class (RUMC) before training the algorithms. The goal of the RUMC was to balance the classes in training leading to better prediction.

Vaiyapuri & Gupta (2021) also came up with another robust method of dealing with such an imbalanced dataset. Instead of implementing an RUMC method, they took the opposite approach. The RUMC method discussed previously undersampled the majority class while the Synthetic Minority Oversampling technique (SMOTE) does much the opposite; it oversamples the minority class, in this case, fatal car accidents. Both the RUMC and SMOTE have the same goal: to reduce the input data minority/majority bias. Much like the previous paper, this one takes a multi-pronged approach to model selection: try lots and decide which is best. Vaiyapuri & Gupta (2021) used MLR, RF, KNN, Support Vector Machine (SVM), and Multi-layer Perceptron (MLP) models. For their non-deep learning models, KNN returned powerful results; their reported accuracy score was .85 while the MLP model was .9, with RF coming in at .88. The rest of the models they tried were all below .60. The item that is becoming clear here is that a preprocess to balance the training set is imperative to building a strong model in this space.

Subasi (2018) researched a traffic accident detection system based on data collected from vehicular ad-hoc networks(VANETs). The Intelligent Transportation Systems(ITSs) d the collected data and integrated them with Incident Detection Systems(IDS) to detect incidents that may lead to traffic accidents. The drivers were sent traffic alerts based on the velocity and coordinates of vehicles around them. Using supervised machine learning algorithms, these data can be classified and utilized to identify the likelihood of an accident occurring. Additionally, the author referred to a traffic simulator whose purpose was to identify any noticeable incidents in traffic due to vehicles slowing down compared to the speed limit. Stimulator information was transferred to a trained model installed in the vehicle so the driver could distinguish between accident-prone and normal situations. As a result, it appears that Random Forest appeared to be the best method in terms of accuracy ratio and sensitivity ratio with 92% and 94%.

Moving further, Mamlook et al. (2020) worked on identifying the risk factors for predicting crash injury severity for elderly drivers. They approached this problem by building five predictive models such as Naïve Bayesian (NB), Decision Tree (DT), Logistic Regression (LR), Light Gradient Boosting Machine (LightGBM), and Random Forest (p. 105). The dataset used for the research was taken from crash records in Michigan Traffic Crash Facts spanning 2010 to 2017. The data consisted of 106,274 instances for drivers 60 and over. They applied Synthetic Minority Oversampling Technique (SMOTE) balancing to the dataset since there were fewer instances of severe injury to assist with overfitting (p. 107). The authors found that the most important features were "driver's age, traffic volume, and car's age" for making predictions (p.105). The paper concludes that LightGBM outperformed the other four models based on Precision, Recall/Sensitivity, F-Measure, ROC, and AUC (p. 109). The authors concluded that this algorithm performed well because it "... is a variation of the decision tree algorithm that utilizes best-first strategy instead of the commonly used depth-wise strategy." (p. 110).

Ensemble methods such as bagging and boosting have been highly popular in the supervised learning segment. Ghandour et al. (2020) researched to understand why it was so. In the research, the performance of an ensemble bagging algorithm was compared with several single learning classifiers. Those classifiers included machine learning techniques such as sequential minimal optimization (SMO), random forest, logistic regression, and naive Bayes. It also included deep learning techniques like Artificial neural networks (ANN). The data source used by the authors for conducting the research was sourced from the Lebanese Road Accidents Platform (LRAP). The authors first analyzed single learning classifiers and validated the performance using the metrics - F1-score, AUC-PR, and Kappa. The results can be found below in Table 4.

Table 4

Results from Single Learning Classifiers

Algorithm	F1-Score	AUC-PR	Kappa
SMO	0.493	0.276	0.4678
Random Forest	0.453	0.376	0.4258
ANN	0.385	0.291	0.3462
Logistic Regression	0.455	0.361	0.4309
Naïve Bayes	0.313	0.337	0.294

The results of the above analysis conclude that the SMO algorithm performed better than the other classifiers. As discussed by the authors, the major disadvantage of a random forest algorithm is that they are poor for hyperparameter tuning. Hence, they adopted an ensemble technique by combining a Vote SMO and a J48 decision tree using the bagging technique. The result of the following experiment showed a performance increase when compared to the single learning classifiers. The results can be seen in Table 5**.** Then Chi-Squared test, along with 10-fold cross-validation, was used to understand how much each of the features impacted the target variable. It was observed that the impact of crash type was the largest, followed by severity level. The authors concluded the paper by stating the need for additional data to improve the performance model and gain more insights into other features apart from the ones used to conduct this research.

Table 5

Results from Ensemble Technique

	Vote SMO with Bagging J48
F1-score	0.435
AUC-PR	0.368
Kappa	0.4067

Lastly, Rahim & Hassan (2021) implemented a deep learning model to predict the severity of a traffic accident. The model used a "generalized numeric to image transformation technique" to convert their crash data from text format into an image using a customized f1-loss function (p. 2). By converting the data into an image, it allowed for the use of an EfficientNet-B7 model for feature extraction, and minor changes to the last connected layer allowed for classification. The process of feature selection made use of a Classification and Regression Tree (CART) model and Multivariate Adaptive Regression Spline (MARS) (p. 2). This resulted in 10 significant variables for CART and 12 significant variables for MARS that have a high correlation to crash severity. The metric used to evaluate the effectiveness of their models was precision and recall. Overall, the deep learning approach outperformed the basic Support Vector Machine (SVM) model in being able to correctly classify the crash severity. Additionally, the model that used MARS resulted in better recall values.

1.5 Literature Survey of Existing Research

Parra et al. (2020) conducted a study to understand how machine learning models performed to predict road accidents. The authors wanted to understand how factors such as weather conditions and geolocations impacted crashes. To conduct this research, the dataset was

obtained from Moosavi et al. (2021), which included temporal, location, weather, and objects oriented data. The authors ran into a problem with the geolocation features as there was a lot of disparity between the values. Hence, they extrapolated location-based features using the Hexagonal hierarchical geospatial indexing system (H3) developed by Brodsky (2020). H3 was used because it gave them control over the accuracy and mapping of hexagons so that they do not require the exact latitude and longitude values. Web scrapping methods were used to obtain the weather-related features as desired for extending the dataset. After preprocessing was done, the authors ran models such as decision tree (DT), random forest (RF), and xgboost. The results of the experiments can be seen in Table 6. The conclusion of the research was that the gradient boosting technique of XGBoost was the best-performing algorithm. The authors stated that the accuracy of the models could be improved by increasing the accuracy of H3 identifiers, Ghandour obtaining more refined weather-related data, and considered events where accidents do not take place.

Table 6

Comparison of the Performance Related to the Models

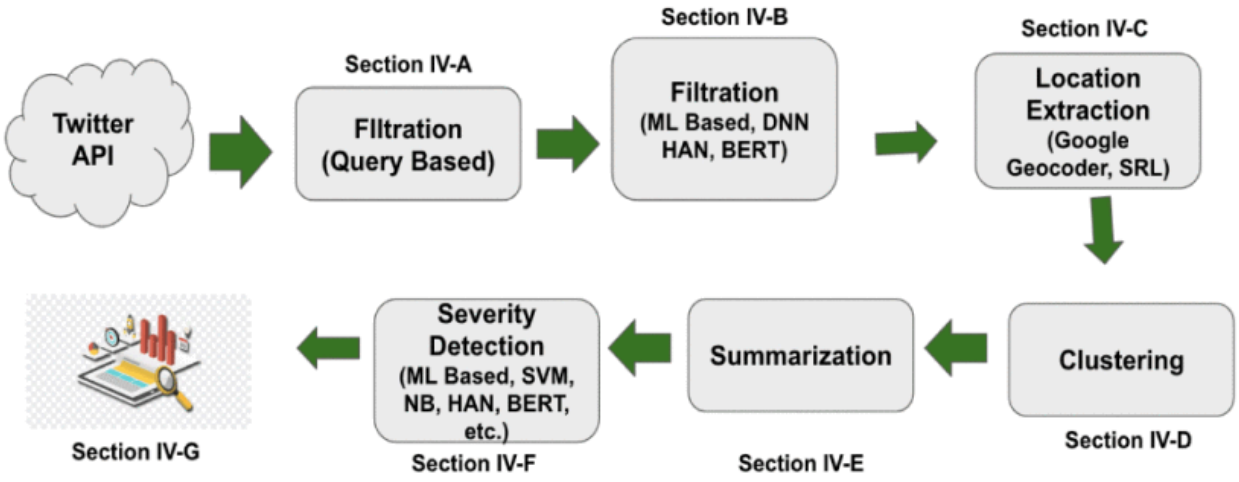
Algorithm	Accuracy	Precision	Recall	F1-Score
DT	74%	71%	69%	70%
RF	74%	71%	69%	70%
XGBoost	78%	79%	73%	74%

Salem et al. (2021) explored a method where they could obtain and integrate social media data to improve the prediction of road accident severity. The research included cities in the United States and Nigeria. The authors used Twitter API to obtain real-time accident-related

tweets continuously to perform analysis. After being collected, the tweets were classified into different clusters, and then severity-based analysis was implemented. Location-based features were obtained based on the tweets and added to the dataset. The authors used several machine learning techniques to filter out tweets based on accidents. They also used them for severity detection. The framework of the process they followed is illustrated in Figure 1**.** The algorithms used for severity prediction were Naive Bayes, Support Vector Machine (SVM), and Hierarchical Attention Network (HAN). The best performing model varied with different cities. More research needs to be done in this situation. The authors want to extend this study further by collecting more city-by-city data and annotating more tweets. A few challenges faced were the extraction of location from tweets and the dissemination of information. A framework for their model process is provided below by Salem et al. (2021).

Figure 1

Framework for tweet processing



Note. From Salem et al., "Exploring the roles of social media data to identify the locations and severity of road traffic accidents," 2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2021, pp. 62-71, doi: 10.1109/AIKE52691.2021.00016.

Ting et al. (2020) used Malaysian road crash data to predict injury severity for the goal of injury prevention in Malaysia. "Empirical study revealed that there were 26 important predictors for predicting accident fatality and the top five variables are month, speed limit, collision type, vehicle model and vehicle movemen." (Ting et al., 2020). The interesting variable here is the impact month plays on predicting outcomes. The other top variables are self-evident in their importance; of course, some vehicles are safer than others, speed increases injury risk, and type of crash matters, but the month is where this model may have found a niche. The kind of effects month can play two-fold. Some months contain more holidays, and holidays go hand in hand with increased impaired driving percentages (Rivara et al., 2012). The other effect month may play in weather; as some of the data included in this project is weather related, this was relevant.

Ting et al. (2020) found that their most accurate model between Naive Bayes with SVM, RF, XGBoost, Classification and Regression Tree, and Neural Net (NN) was actually just the Random Forest model with an accuracy of over 95%.

Labib et al. (2019) used a relatively novel approach to this problem; instead of simply looking at text-based input data, things such as the type of vehicle involved in an accident, the researchers instead used social media vehicle accident upload and CCTV footage and mapped the accident severity back to the image. They then ran a few ML and DL applications on the image and text-based data. The goal was to make a model that can predict injury and accident severity based on the image data alone. The application for this is plentiful. If, for instance, you run this model on all state-owned CCTV cameras, you can reliably call emergency services to an accident scene before a human makes the call. If an accident is severe enough and traffic is low enough, a human may not even be physically capable of making an emergency call after a vehicle collision. Frankly, this paper is unimpressive in its model application, the results of their models, and their summaries; what is impressive and noteworthy is the injection of the image-based data.

In the paper by Hamid, Y., & Habuza, T. (2020), they focus on building machine learning models for predicting traffic crash severity based on factors such as bad weather, road design problems, poor visibility, disregarding traffic signals, and stop signs. The author made a choice of building three models: Random Forest as it is more robust to noise and outliers, Gradient Boosted Tree as it uses ensemble techniques to improve accuracy and performance, and Decision Tree as it uses the divide and conquer technique to split the problem into subsets. The Decision tree-based model yielded 71.66% accuracy with a depth of 10, keeping information gain as splitting criteria. The Gradient Boosted model gave a result of 73.32% of accuracy with a depth

of 5. The Random Forest model yielded 72.98% accuracy with a depth of 10 using 100 decision tree classification combinations based on a confidence vote.

Yuan et al. (2021) mentioned in their paper to predict the severity of older pedestrian crashes. Older pedestrian traffic crashes are very common nowadays. To address this problem, the XGBoost machine learning model is applied. The author also focused on comparing Multinomial Logistic Regression (MNL) and XGboost. With the help of SHAP(Shapley additive explanations) interpretation, the author determined the significance, correlation, and impact of different features on different severity levels of older pedestrian crashes. In comparing the above models, the author concluded that multicollinearity among features was low in MNL, and it can lead to increased variance. The XGBoost model solved the multicollinearity problem by selecting only one correlated feature and yielding a high percent of accuracy.

Manzoor et al. (2021) authors worked on developing an ensemble machine learning method that combines machine learning and deep learning called a Random Forest and Convolutional Neural Network (RFCNN) to predict road accident severity (p. 128359). The authors focused on using machine learning methods as base learner models and combining them with deep learning. The base models include: "... RF, ADC, ETC, GBM and a Voting classifier (LR+SGD)." (p. 128362). The deep learning model consists of a Convolutional Neural Network (CNN), and the "CNN model consists of convolutional layers, pooling layers, activation layer, dropout layer, and flatten layer" (p. 128362). The best performing base model was Random Forest, which had an accuracy score of 0.744. This model was used to "... calculate the feature importance value of all features using the random forest classifier," which narrowed down the features from 43 to just the top 20 (p. 128365). The RF is then combined with the CNN by using a soft voting approach with the highest probability of the combined average of the two models

for classification. As a result, Manzoor et al. (2021) found that the RFCNN "... outperformed other models with 0.991 accuracy, 0.974 precision, 0.986 recall, and 0.980 F-score" (p. 128359).

In the paper by Jamal et al. (2021), they try to predict crash injury severity and provide insights that allow for strategies to mitigate the frequency of crashes. The dataset consists of 13,546 vehicle collisions across 15 rural highways from January 2017 to December 2019. The data was derived from the ministry of transport in Riyadh, Saudi Arabia. The paper discussed how traditional statistical methods had drawbacks in this research and how machine learning models would be ideal for modeling nonlinear relationships. For feature selection, the authors implemented gain ratio feature evaluation and used features like collision type, road surface conditions, vehicle type, weather, lighting, and damage type. The authors created four machine learning models. The models include XGBoost, random forest, logistic regression, and decision trees. The models had a target variable of three potential outcomes: fatal, injury, or property damage only. The models utilized 10-fold cross-validation for predictive modeling performance. The four models were compared using accuracy, precision, recall, and F-measure metrics. The results demonstrated XGBoost as the best-performing model with 93% accuracy on the test data. Decision trees and random forests performed at 88% and 84%, respectively. Logistic regression was the poorest model of the four, with an accuracy of just 63%. XGBoost was the best model in regards to accuracy and its predictive capabilities. In conclusion, it's important to note that these models were trained on limited data (13,546 rows), and could benefit from a larger dataset for greater accuracy. Perhaps future research could benefit from investigating specific crash types, which could help improve safety strategies overall.

Wang et al. (2019) the purpose of this paper is to predict the driving risk of future drivers and potentially implement safety interventions more effectively. The datasets used were derived

from the Kunshan Traffic Police Department in China on crashes, which were accumulated over the span of seven years from 2011 to 2017. This dataset's features included age, gender, driving time, road conditions, vehicle, and crash records. Additionally, the authors used the Kunshan Traffic Police Department traffic violation data from the same years as the crash dataset. These two datasets were joined by matching the IDs in both datasets, which equated to a dataset of 201,328 rows. The paper looked at two different technical aspects: driving risk and risky driving factors. For example, the authors looked to see if crashes occurred from individuals on a repeated basis in a two-year span and also identified drivers that were at fault for crashes. Based on these aspects, four models were constructed: random forest, gradient boosting decision tree, AdaBoost with a decision tree, and XGBoost. The authors implemented random bagging to select their samples from the dataset with the use of replacement for their ensemble models. The other models utilized boosting, which made use of a bootstrap method to select samples randomly. For parameter tuning, the authors utilized a grid search method. The results from the models found that seven of the top nine features strongly correlated with risky driving factors. This means risky driving features are strongly correlated with identifying a pattern for predicting risk. The authors concluded that driving risk was best defined by crash recurrence, severity, and fault commitment. A gradient-boosting decision tree was the model that performed the best with a precision of 0.68. The paper concluded interventions must be implemented to mitigate risk by targeting drivers with high crash risk. For future work, there could be more features that make use of detailed crash features to show a better in-depth history of the driver because this would help predict their future driving habits. For the purpose of safety interventions, the authors suggest that safety messages could be sent to high-risk drivers, and those found to be at high-risk could be required to attend additional traffic training.

Theofilatos, A., & Yannis, G (2014) this paper reviewed many characteristics related to traffic and weather to understand if there was any influence on crashes and accidents. Previous research on this topic only utilized traffic and weather characteristics, but this research also includes real-time data pertaining to this matter which may provide further insights. The traffic characteristics studied include flow, density, speed, and overall traffic considerations. The weather characteristics include precipitation, visibility, wind speed, air temperature, and other weather considerations. In comparison, the real-time data consisted of crash likelihood, crash severity, and crash frequency. The method of analysis used by the authors for crash frequency was the count regression technique. The authors, after completing thorough research, concluded that the effects of traffic parameters were varied. One of the limitations mentioned by the authors is that the studies were more focused on freeways and expressways rather than including rural and urban settings as well.

Lee et al. (2019) this paper discusses traffic accident prediction models and takes into consideration several major factors which could lead to traffic accidents. These models have been built to improve and control traffic safety. Three machine learning algorithms were implemented to address this problem: Random Forest, Artificial Neural Network, and Decision Tree. The datasets used are road geometry data, precipitation data, and traffic accident data from 2007 to 2015. Additionally, the paper also discusses the relationship between road friction coefficient and traffic accident risk. 75% out of 518 samples of accident reports were used for the training dataset and 25% for the test dataset. In the result, Random Forest turned out to be the most suited model for this forecasting based on MSE and RMSE evaluations. The authors concluded that the main factors that affected traffic injury severity were road wetness and geometry issues. One of the limitations of this paper is the unavailability of data related to

particular road surface conditions and recent road construction, which affected the prediction performance.

In the paper by Abellán et al. (2013), they aim to study Decision Trees as a method for examining traffic accident severity. The authors mentioned how good decision rules extracted from the structure could be. The main disadvantage of a single decision tree is that very few decision rules can be extracted from it. Hence, the authors proposed a method called the Information Root Node Variation (IRNV), where many decision trees were created by varying the root node. This would help in extracting way more decision rules from the same dataset. The authors implemented two different split criteria to extract decision rules - the Gini Index and Information Gain. The analysis was performed on traffic accidents on rural roads data from Spain. The authors obtained over 70 applicable rules using the proposed method on the same data. A single decision tree could have extracted only five relevant rules. The authors want to further extend this study by analyzing the environmental conditions.

Tao et al. (2022) concluded that the 800 plus pedestrian fatalities due to motor vehicle accidents over the past 30 years could be better understood through a machine or deep learning approach rather than basic statistical inference. Their idea is if you can accurately understand the factors or circumstances leading to pedestrian fatalities, you can help prevent some in the future. They believe the increase in accuracy over standard statistical models is necessary.

These researchers tried several methods, including a Neural Net model, a Random Forest model, and a standard Bayes model, before finally settling on a Bayesian Neural Net. Notably, the researchers used XGBoost for their model selection and 5-Fold cross-validation as part of their training process. The three most important variables they determined were the road speed limit, the type of crash, and the age of the pedestrian. The researchers were able to fairly

accurately predict the likelihood a pedestrian-involved car accident would be fatal. This model is very limited; they basically proved that higher speed limits among areas where pedestrians are present, and a pedestrian-related crash is imminent have a higher likelihood of said pedestrian dying. The authors suggested this model could be used to help authorities make decisions regarding speed limits in areas or times when pedestrians are more likely to be present.

Jahangiri et al. (2018), this group of researchers' goal was to use detailed accident-specific data to determine the more important factors involved in the severity of car accidents. They used a multi-faceted approach applying multinomial logit, mixed multinomial logit, and support vector machine (SVM) models. They found that the SVM model was narrowly the most accurate. The researchers used data from Highway Safety Information System (HSIS) database for only California-based records, time period 2007-2011. The group used an area under the curve approach to variable selection. The variables they ended up with were the age of the injured, sex, terrain level, weather, lighting condition, type of vehicle, number of lanes, and cause of the crash. The model determined that age was the most crucial factor in determining the level of injury for the crash. The researchers believe this kind of crash data is highly relevant for civil engineers in road design and even vehicle design and regulation. They limited their research to rear-end collisions but believe a great area to expand research is to apply the model to all types of collision instances.

The authors Ma et al. (2021) believe essential insights are being missed on vehicular accident injury severity data because the other machine learning approaches have ignored hyper-complex modeling techniques. Instead of evaluating several different modeling techniques, the researchers applied a mix of CatBoost and Shapely for model selection, then ran a K-means clustering algorithm to the data, finishing with a stacked sparse auto-encoder (SSAE)

model on the clustered groups to predict injury severity. They conclude that their model is the best around for predicting the injury severity of pedestrians. In the future, they would like to extend their research to increasing the tuning of their model for efficiency and accuracy improvements.

In the paper by Komol et al. (2021), they focus on implementing machine learning models which predict the crash severity based on different vulnerable road users (VRU)-pedestrians, bicyclists, and motorcyclists. Three modeling algorithms were implemented for prediction: K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Random Forest (RF). The dataset was collected from the Department of Transportation and main roads in Queensland, with a record of 21,158 VRU crashes from 2013 to 2019. The Random Forest technique was found to be the best classification model with a high percentage of accuracy. Model performance was measured based on accuracy, sensitivity, specificity, precision, F1 test, and ROC. Based on the comparative analysis, the authors concluded that motorcyclists are more likely to get exposed to higher crash severity. According to the authors, one of the limitations was not to include a dataset of collided vehicle records, such as private vehicles, cars, trucks, vans, etc., which was considered to be a primary reason for fatalities among VRU.

The paper by Rezapour et al. (2020) implements and assesses deep learning models to predict motorcycle crash severity. The models use a relatively minor dataset of 2,430 instances of motorcycle crashes that occurred in the United States over a 10-year period. The authors make use of basic single-layer deep learning methods and a recurrent neural network (RNN). The authors performed feature selection by minimizing the error rate. This allowed for features with the highest correlation to the target variable of predicting injury severity for motorcycle crashes to be selected for the purpose of training the models. In conclusion, the authors found that the

RNN outperformed the other basic deep learning methods. The authors also suggested that future research should implement a Long Short-Term Memory (LSTM) model, which should perform better where the RNN has issues.

2. Data and Project Management Plan

2.1 Data Management Plan

There are two distinct datasets both collected from the same place, the City of San Jose. The CSV files were downloaded directly from the City of San Jose website to local storage for the City of San Jose Vehicles involved in crashes dataset (Vehicles). The group then joined this dataset to the San Jose Crashed dataset (Crashes) After joining all the datasets, the unjoined data and the joined data are placed into a shared Google Drive folder, where data can easily be pulled this for local Jupyter notebooks or shared Google Collab environments. Every dataset was kept separately in a CSV file. Google Collab can access these datasets by establishing a connection with the Google Drive sources. The primary CSV file was also published to this folder when the datasets had been combined and cleaned of duplicate data. Three folders were be present: raw, which included only raw data; train, which contained only training data; and test, which had only testing data.The source of the dataset and the version, such as uncleaned, cleaned, etc., were included in the naming standard for the datasets. Vehicle crashes merged was the name of the finished dataset.

The datasets were distibuted to group members for further understanding, exploratory data analysis (EDA) and preprocessing. Using Google Drive cloud functionality, all group members were able to work on the same file at the same time and also monitor the changes instantly.

In the future, the results of predicted data can be further utilized to avoid upcoming crash severity in the San Jose area. In order to prevent such crashes, predicted data can be processed and sent to vehicle drivers driving near accident-prone areas through some systems/messengers. This will also help policymakers to make changes as per the result in the locality for better traffic safety.

2.2 Project Development Methodology

The System Development Life Cycle (SDLC) for this project that was implemented is the Cross Industry Standard Process for Data Mining (CRISP-DM) agile methodology. The CRISP-DM approach is not as rigid as the standard waterfall methodology. At first glance, this project does appear to represent a waterfall approach, but upon closer look, it demonstrates flexibility due to being an agile approach. Each phase of the CRISP-DM methodology is approached as if it were to be conducted in chronological order but offers the ability to have tasks going on simultaneously. Plus, in some instances, there's the potential for development to go back to a previous step based on feedback from the current step. This approach makes sense for the development of this project because its agile approach allowed for flexibility to achieve the desired results promptly and on time.

According to Chapman et al. (2000), in their brochure entitled *CRISP-DM 1.0*, the CRISP-DM model was developed and named for Cross Industry Standard Process for Data Mining. The brochure's model is broken down into six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. In each phase, there are a number of generic tasks which can be even further broken down into more specific tasks. Following, this paper will dissect the project tasks under the CRISP-DM framework.

For the business understanding portion of CRISP-DM, there are six broad tasks; the first is domain research related to vehicular traffic accidents. Next, there is more specific research related to vehicular traffic accidents in the San Jose area. Third, there existed a need to find potential data sources that would work for the goal of predicting vehicular accident injury severity. Following that, the need to define the project management tools and approaches to use. In this case, as evident in this section, this project went forward with a CRISP-DM approach to the project as a whole while using Trello for the day-to-day management of tasks. Both the project management goals and work structure are provided in Trello.

The next main phase of the CRISP-DM process is the Data Understanding section. In this section,there are three main subtopics. The first was collecting the datasets; in this case, it is the data from the City of San Jose Vehicular Accidents. The next step here is to get a good feel for the data, a strong understanding of its simple data exploration, and finish with a more strategic exploratory data analysis process using descriptive statistics.

Once there was a strong understanding of how the data works, the group moved on to the Data Preparation phase. The first process here was to clean the data, exclude unrelated data, and handle outliers, duplicates, null values, structural errors, and omitted data. For the models which needed it, convert the categorical data into numerical data. Next, the group ensured the datasets were compatible and that there was a common enough key between them. The next step, the group joined the datasets together into one even more comprehensive dataset. Again, for the models where it is required or helpful, the data was normalized. The last step here was to upload the data to Google Drive.

For the modeling section, there were a few steps repeated many times. Before the repeating steps, the data first had to be split into training, testing, and validation sets. The group

then ran the six models, taking intermediate steps for certain models like feature selection for regression and hyper-parameter tuning for some of the others. All the models finished with model testing, and certain models, with validation.

In the evaluation phase, it was started with cross-validating all the models then moved on to metrics comparisons. The next necessary step was to compare the accuracy, F1, C-matrixes, and ROC-AUC, along with the not as important to model accuracy metrics, such as time and resources costs. To finish the model evaluation phase, the group built visualizations for some of these metrics to help determine what the best model was.

The last major phase was the deployment step. For the purposes of this project, this was a matter of finalizing everything: building a dashboard, analyzing the model performances, writing up the findings, and reviewing the entire project. This project was finished by writing a paper and finally presenting it.

2.3 Project Organization Plan

Work breakdown structures are project management tools that help to maintain the oriented hierarchical structure of the work which are required to accomplish the objective of the project by the team members. Each level provides a detailed description of individual tasks and enables members with a perfect roadmap to work efficiently.

As a first step of the WBS process, the group determined the Business Understanding of this project by focusing on domain research for San Jose area vehicle accidents along with finding the potential data sources for accidents. Furthermore, the group defined the project goal and work structure based on the research on traffic safety measures.

In the next phase, the group showed the data understanding phase consisting of 3 major tasks: gathering datasets, understanding each dataset, and data exploration and analysis taking

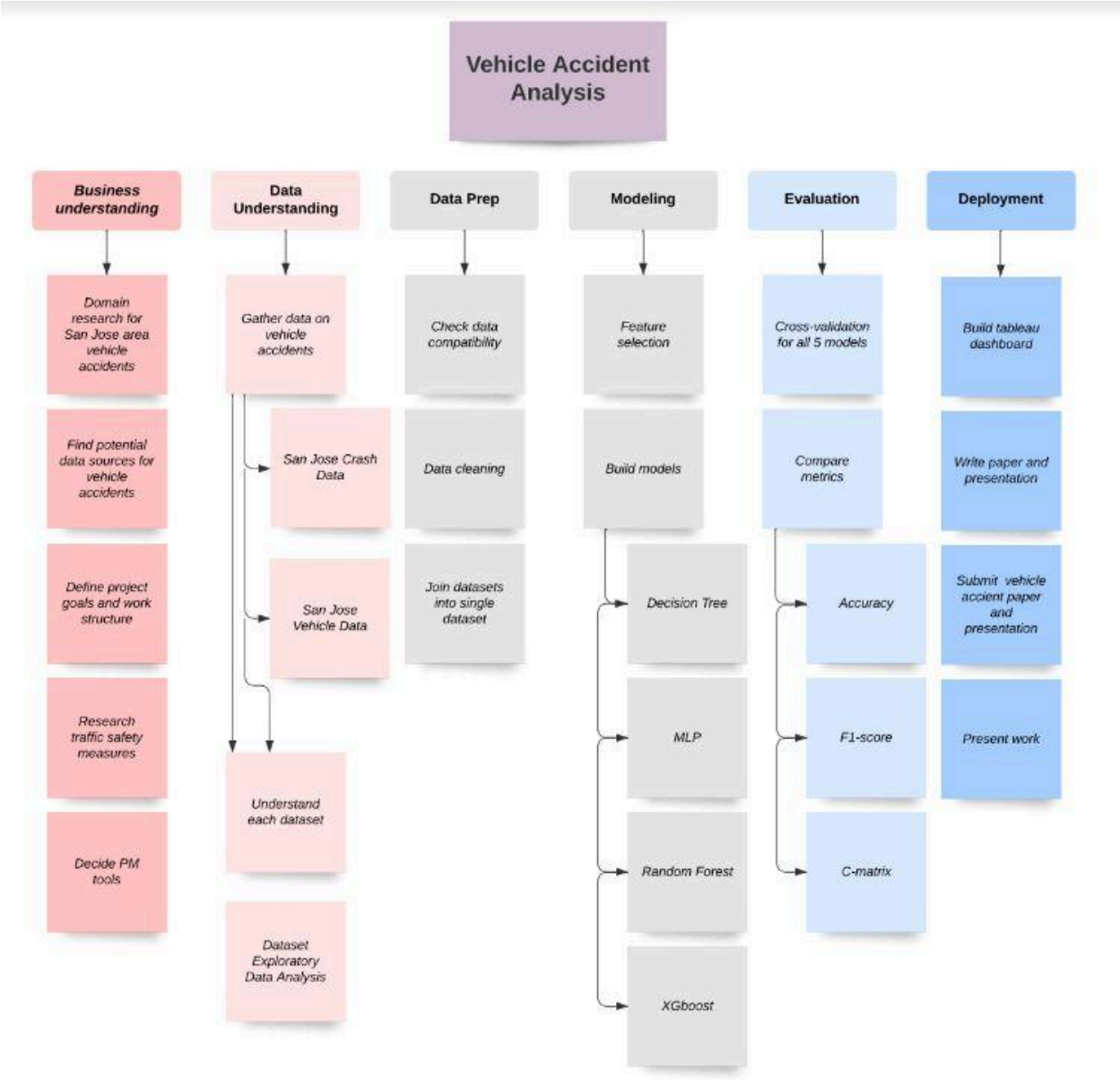
into account the two datasets - San Jose vehicle data and crash data. After data collection, the analysis of all the datasets in detail was done to understand which fields could be incorporated into the models. Last, for a deeper understanding of the datasets, the team performed data exploration individually.

After understand the next step was the data preparation phase, where the group checked data compatibility, cleaned each of the datasets, and made it ready for further use in modeling. Lastly, the datasets were combined into a final dataset. Moving ahead, the group performed feature selection for models where relevant. Finally having decided to build a mixture of five machine-learning and deep-learning models, which were be evaluated using cross-validation and accuracy and performance metrics such as accuracy, F1-macro score, and confusion matrices.

A tableau dashboard was built before finally deploying the model. The group also wrote a detailed report and prepared a presentation on the vehicle accident project. This step completes the WBS thoroughly and below Figure 2 shows the pictorial for the Work Breakdown Structure of this project.

Figure 2

Work Breakdown Structure for the Project



2.4 Project Resource Requirements and Plan

The necessary hardware for this project includes Google Colab, and individual laptops. Python, Trello, TeamGantt, Tableau, Google Drive, Github, and Zoom are among the software prerequisites.

The data was be stored in the cloud through Google Drive. Google Collab aiss the primary Python programming environment. Both of these services provide a free version, which this project used. Python is the choice language for writing scripts, and for this project, modules like Numpy, Pandas, Matplotlib, Seaborn, and Sci-kit Learn will be used to manipulate, clean, and model the data. Project management was done using TeamGantt and Trello. Data visualization and deployment were accomplished using Tableau. The team members communicated and shared files via Google Drive and Zoom. A summary of these requirements can be found in Table 7.

Table 7

Resource Requirements and Cost

Resource	Resource Type	Cost
Trello	Software	Free for base plan
TeamGantt	Software	Free for base plan
Python	Software	Free open source
Google Colab	Hardware	Free for base plan
Tableau	Software	$70 per month per license
Google Drive	Hardware	Free up to 15 Gb post that $1.99 per month
Zoom	Software	Free
Draw.io	Software	Free
Pandas	Python Package (Software)	Free
Numpy	Python Package (Software)	Free
scikit-learn	Python Package (Software)	Free
Matplotlib	Python Package (Software)	Free

Resource	Resource Type	Cost
Various Python libraries	Python Package (Software)	Free
Grammarly	Plagiarism Checking Tool(Software)	$149 per year

2.5 Project Schedule

Figure 3 shows the Gantt chart for this project, this shows the detailed structure of the timeline of the project.

Figure 3

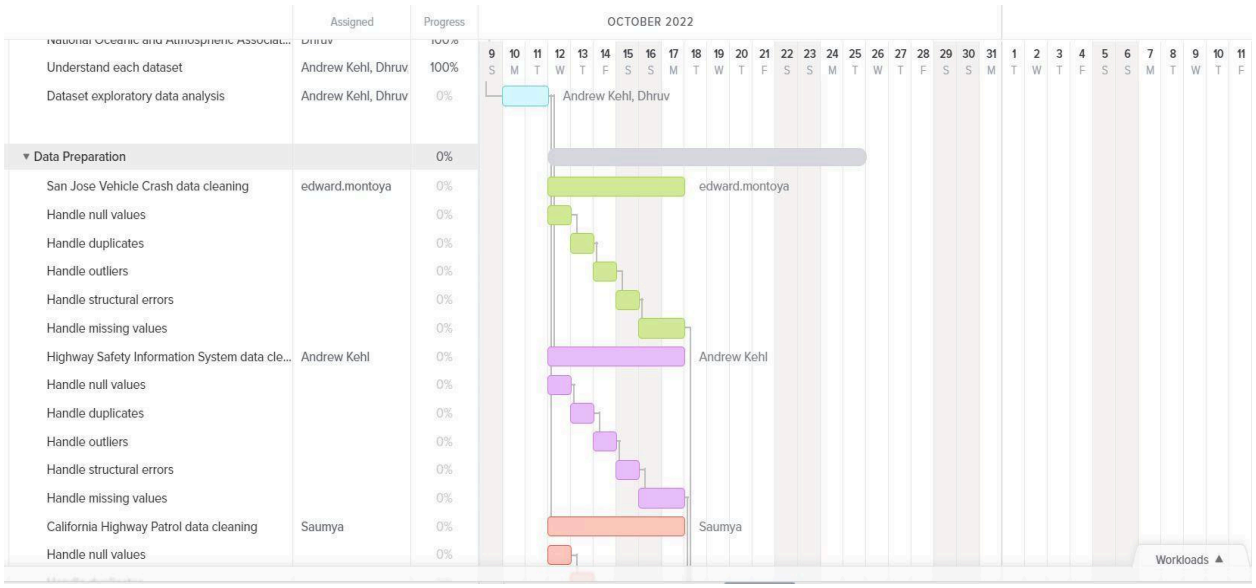
Timeline for Business Understanding and Data Understanding Tasks

The Business Understanding phase started on September 12 and ended on October 5. In the initial phase, the group allotted time for research and finding potential data from different sources and structuring the work as per the project goals. The Data Understanding phase started on September 29 and ended on October 11. The group distributed tasks equally among four members and completed these tasks by collecting four different datasets from different sources.

Further, was the allotment of days for data exploration and analysis. Figure 4 shows the timeline for preprocessing tasks.

Figure 4

Timeline for Data Preprocessing Tasks

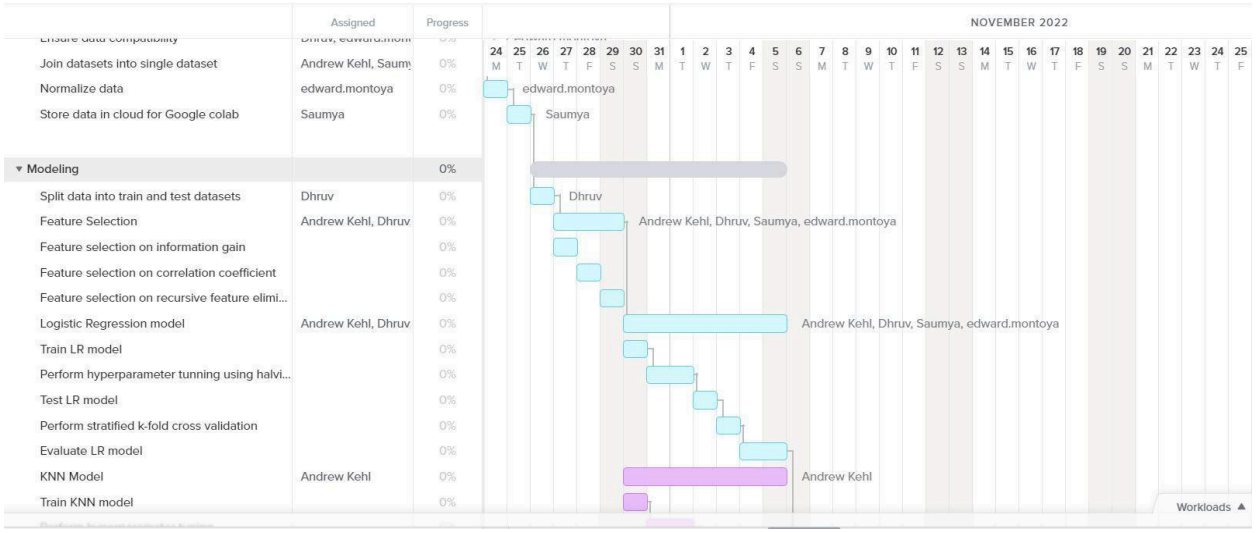


The data preparation phase started on October 12 and ended on October 25. Dataset tasks were divided equally among group members, and for preprocessing, the group performed null value imputation or removal of record, handling of duplicates, examination of outliers, and handling of structural errors. Additionally, the group analyzed data compatibility, joined datasets based on compatibility, and normalized the joined dataset from October 18 to October 25.

In the modeling phase, the group members worked individually on separate models. The start date of this task was October 26, and it ended on November 5. This included feature selection based on correlation, information gain, and recursive elimination. It finally ended with the testing of model accuracy metrics, hyper-parameter tuning, and evaluation. Figure 5 shows the timeline for data modeling.

Figure 5

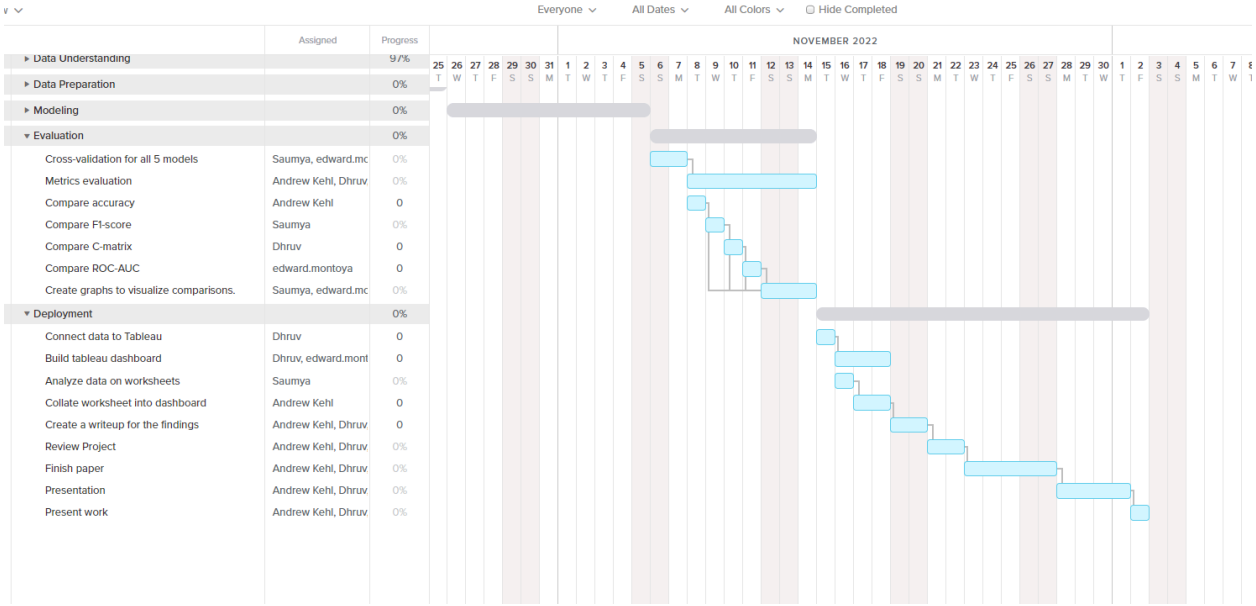
Timeline for Modeling Tasks



The evaluation phase started on November 6 and ended on November 14. As part of this phase, the group checked and evaluated the models to meet expected results as per project requirement criteria. The next task, deployment of the project, which started on November 15 and ended on November 18. This consisted of reviewing and evaluating the entire project process, writing a final report, and representing the final result in the presentation, which ended on December 2. Figure 6 shows the timeline for model evaluation.

Figure 6

Timeline for Evaluation and Deployment Tasks



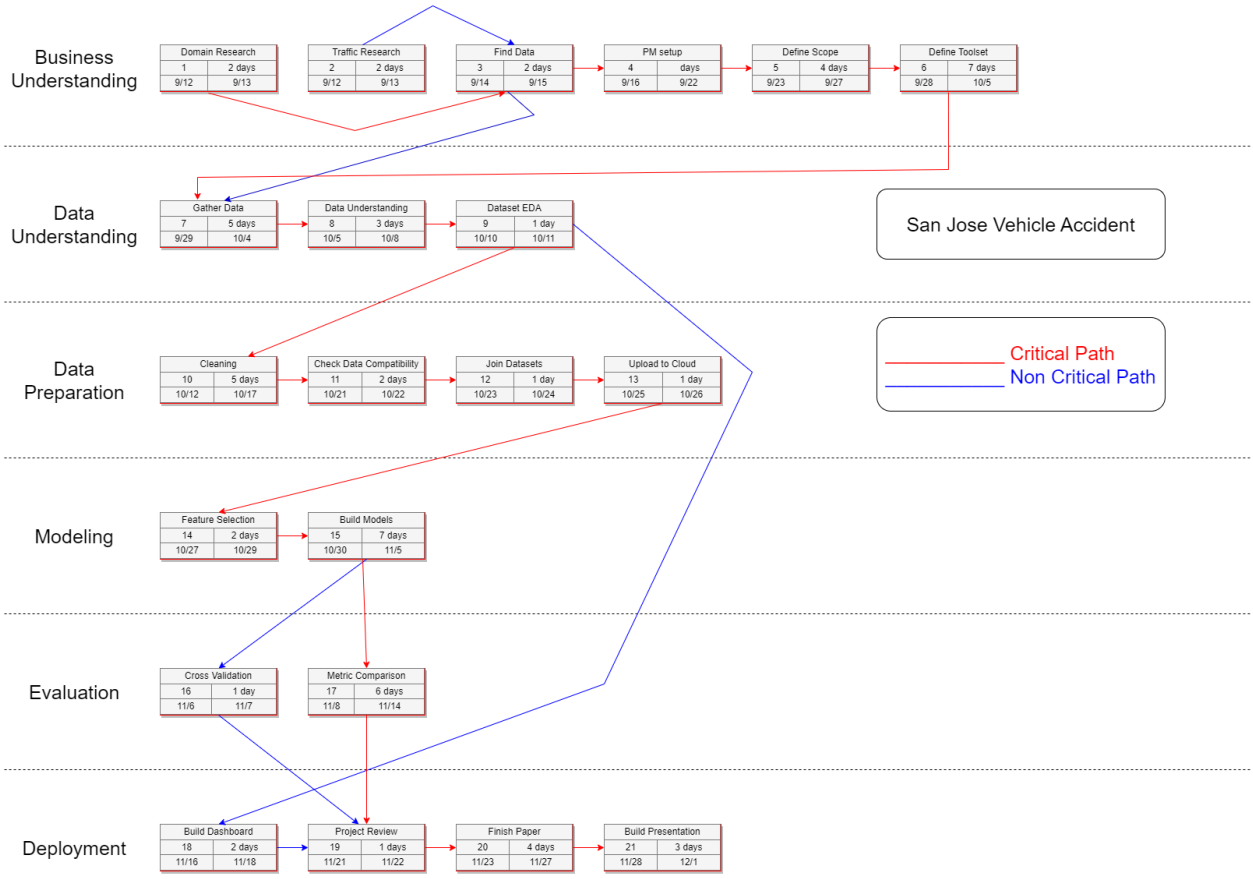
The evaluation phase started on November 6 and ended on November 14. As part of this phase, the group checked and evaluated the models to meet expected results as per project requirement criteria. The next task, deployment of the project, which started on November 15 and ended on November 18. This consisted of reviewing and evaluating the entire project process, writing a final report, and representing the final result in the presentation, which ended on December 2. Detailed Gantt chart can be found in Appendix A.

Below, in Figure 7 is the PERT chart. As Hudson and Marilyn (2015) describe it, a PERT chart or a Program Evaluation and Review Technique chart is essentially used to figure out the exact maximum amount of time a task must start and or finish by to stay on schedule. They go on to mention the usage and breakdown can vary wildly from incredibly intricate to very broad. For the purposes of this project, this was a task-based PERT. It is a medium-level chart showing a breakdown of most of the simplified yet critical tasks for this project. On the left-hand side, there

is a breakdown of which tasks belong to which section of the CRISP-DM process. In red is the critical path, while in blue is the pathway with slack.

Figure 7

CRISP-DM PERT Project Structure



3. Data Engineering

3.1 Data Process

For this project, the two different datasets were chosen: San Jose Crash dataset and San Jose vehicle dataset, which were collected from an open data source hosted by the City of San Jose. After analyzing the datasets, features of interest were identified with respect to the target problem. In the Crash dataset, it was found that there were a few attributes, such as minor

injuries, moderate injuries, severe injuries, and fatal injuries, which are really close to the target severity of accidents. Later on, these features were combined into one feature for the target as severity.

The vehicle dataset consisted of features such as Speed, VehicleDamage, PartyCategory, sobriety, Age, and vehicleCount. Later, feature selection techniques were performed, it was found that these are highly correlated with the target feature. The VehicleDamage feature gave the details related to vehicle damage after the crash i.e. minor, major, moderate, or none and found that most crashes lead to major vehicle damages which is an important aspect to the target. The PartyCategory variable describes the person's involvement in the accident with labels as Driver, Parked, Bicycle, Pedestrian, Unknown, and Other. Similarly, features such as sobriety have 10 different labels. Moving ahead, these categorical features were transformed using One-Hot Encoding technique. Among both datasets, CrashName feature was common which was used for merging the datasets.

For further cleaning and preprocessing, handled the missing data, outliers, inconsistencies, and label encoding, cleaned up null values, and checked for duplicate values. It was observed that few numerical features such as Age and distance had very large statistical values, so transformed the final merged dataset by performing min-max normalization except on one hot encoded features. The target feature was also transformed " Severity" into labels such as No Injury = 0, Minor Injury = 1, Moderate Injury = 2, Severe Injury = 3, Fatal Injury = 4.

For regularizing the final dataset, Lasso and Ridge regularization techniques were used. After combining the two datasets, 298 features were formed in total, which were reduced later by using the Principal Component Analysis (PCA) technique and this gave a result with 76 features among which 75 features were considered for input and one feature for output. Random

Forest feature selection was performed on the combined dataset. This gave a result of 297 features among which Age, distance, and sex showed with a high percentage of correlation with the target feature.

Later on, it was analyzed that the target feature severity had imbalance classes, so this was overcome by performing Synthetic Minority Oversampling Technique (SMOTE).

For validating the performance of the model, train-valid-test split was implemented along with KFold cross-validation. 10 Fold cross-validation where the dataset will be divided into 10 different non-overlapping folds.

3.2 Data Collection

3.2.1 San Jose Crash Dataset

The San Jose Crash dataset is hosted by the City of San Jose. The City of San Jose has an Open Data Policy and Open Data Community Architecture which allows for the acquisition of vehicle crash data. The dataset can be directly downloaded in CSV format from the City of San Jose‘s Open Data Portal. The actual historical dataset spans from January 1, 2011, to December 31, 2020. The dimensions of the dataset consist of 25 features and 56,044 rows. The data was collected from daily accident reports that were collected by the San Jose Police Department, and the Department of Transportation. Figure 8, Figure 9, Figure 10 and Figure 11 show the data collection plan. Whereas, Figure 12, Figure 13 and Figure 14 show the raw data samples.

Figure 8

Data Collection Plan Desciption for Crash Dataset

Data Collection Plan

Project number: | Project title: Analysis of Ve | Project leader: | Date: 10/20/2022

Description of the data collection	
Why are we collecting the dataset?	The Crash Dataset consisted the informations about the actual vehicle accidents. We are collecting the data to train a model which aims to accurately identify features that correlate to injury, and to implement a variety of machine-learning techniques to predict injury severity for vehicle accidents in the San Jose area
How are we collecting the dataset	The dataset can be directly downloaded in CSV format from the City of San Jose's Open Data Portal. The data was collected from daily accident reports that were collected by the San Jose Police Department, and the Department of Transportation. .
What will be done with the data once it has been collected?	After collecting the data, we analysed the dataset and prepared it by checking the compatibilit, performed cleaning, EDA, transformation and finally made it ready to be used in model training.

Figure 9

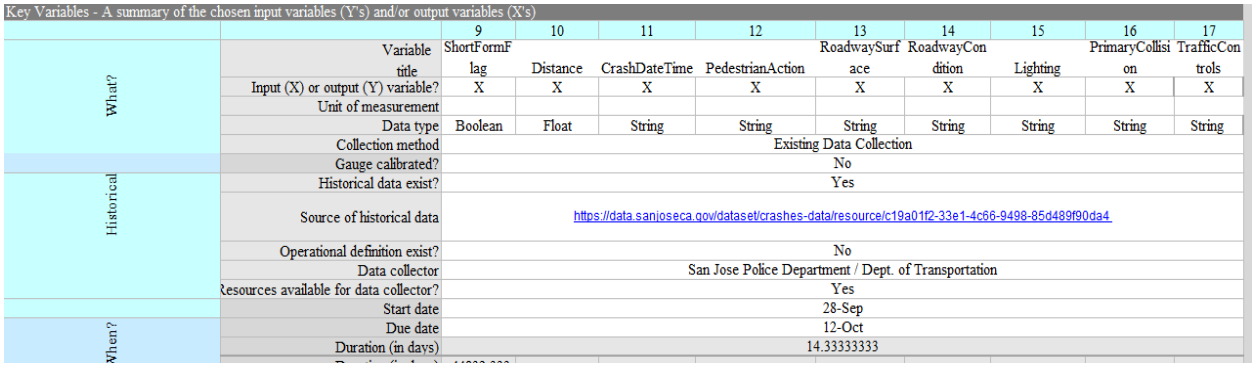
Data Collection Plan for Crash Dataset

Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)

		1	2	3	4	5	6	7	8
What?	Variable title	CrashFactId	Name	Minor Injuries	Moderate Injuries	Severe Injuries	Fatal Injuries	TcrNumber	CityDamageFla
	Input (X) or output (Y) variable?	X	X	Y	Y	Y	Y	X	X
	Unit of measurement								
	Data type	Integer	String	Integer	Integer	Integer	Integer	String	Boolean
	Collection method	Existing Data Collection							
	If manual								
	Gauge calibrated?	No							
Historical	Historical data exist?	Yes							
	Source of historical data	https://data.sanjoseca.gov/dataset/crashes-data/resource/c19a01f2-33e1-4c66-9498-85d489f90da4							
	Operational definition exist?	No							
	Data collector	San Jose Police Department / Dept. of Transportation							
	Resources available for data collector?	Yes							
When?	Start date	28-Sep							
	Due date	12-Oct							
	Duration (in days)	14.33333333							

Figure 10

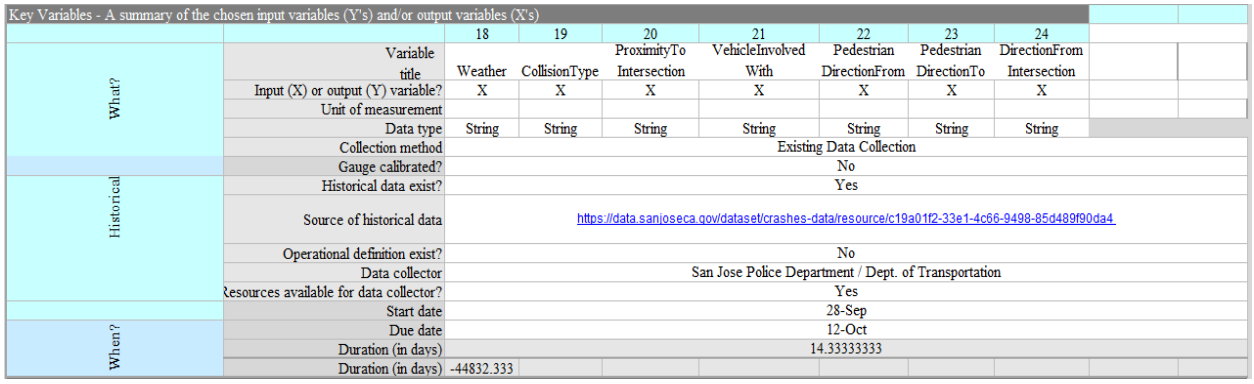
Data Collection Plan for Crash Dataset



Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)										
		9	10	11	12	13	14	15	16	17
What?	Variable title	ShortFormFlag	Distance	CrashDateTime	PedestrianAction	RoadwaySurface	RoadwayCondition	Lighting	PrimaryCollision	TrafficControls
	Input (X) or output (Y) variable?	X	X	X	X	X	X	X	X	X
	Unit of measurement									
	Data type	Boolean	Float	String	String	String	String	String	String	String
	Collection method	Existing Data Collection								
	Gauge calibrated?	No								
Historical	Historical data exist?	Yes								
	Source of historical data	https://data.sanjoseca.gov/dataset/crashes-data/resource/c19a01f2-33e1-4c66-9498-85d489f90da4								
	Operational definition exist?	No								
	Data collector	San Jose Police Department / Dept. of Transportation								
	Resources available for data collector?	Yes								
	Start date	28-Sep								
When?	Due date	12-Oct								
	Duration (in days)	14.33333333								

Figure 11

Data Collection Plan for Crash Dataset



Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)								
		18	19	20	21	22	23	24
What?	Variable title	Weather	CollisionType	ProximityToIntersection	VehicleInvolvedWith	PedestrianDirectionFrom	PedestrianDirectionTo	DirectionFromIntersection
	Input (X) or output (Y) variable?	X	X	X	X	X	X	X
	Unit of measurement							
	Data type	String	String	String	String	String	String	String
	Collection method	Existing Data Collection						
	Gauge calibrated?	No						
Historical	Historical data exist?	Yes						
	Source of historical data	https://data.sanjoseca.gov/dataset/crashes-data/resource/c19a01f2-33e1-4c66-9498-85d489f90da4						
	Operational definition exist?	No						
	Data collector	San Jose Police Department / Dept. of Transportation						
	Resources available for data collector?	Yes						
	Start date	28-Sep						
When?	Due date	12-Oct						
	Duration (in days)	14.33333333						
	Duration (in days)	-44832.333						

Figure 12

Raw Crash Dataset Samples For First Few Columns

_id	CrashFactId	Name	MinorInjuries	ModerateInjuries	SevereInjuries	FatalInjuries	TcrNumber	CityDamageFlag	ShortFormFlag	Distance
1	591079	CR-0000071607	0	0	0	0	18-073-0962	TRUE	FALSE	228
2	591080	CR-0000071780	0	0	0	0	18-060-0123	TRUE	FALSE	148
3	591081	CR-0000060418	0	0	0	0	16-033-0204	FALSE	FALSE	1583
4	591082	CR-0000060410	0	1	0	0	16-041-0882	FALSE	FALSE	295
5	591083	CR-0000060514	2	0	0	0	16-063-0761	FALSE	FALSE	0

Figure 13

Raw Crash Dataset Samples For Few Columns In the Middle

CrashDateTime	PedestrianAction	RoadwaySurface	RoadwayCondition	Lighting	PrimaryCollisionFactor	TrafficControl	Weather	CollisionType
2018-03-14T23:17:00	No Pedestrians Involved	Wet	No Unusual Conditions	Dark - Street Light	Violation Driver 1	No Controls Present/Factor	Clear	Hit Object
2018-03-01T07:30:00	No Pedestrians Involved	Wet	No Unusual Conditions	Daylight	Violation Driver 1	No Controls Present/Factor	Rain	Hit Object
2016-02-02T09:02:00	No Pedestrians Involved	Wet	No Unusual Conditions	Daylight	Violation Driver 1	No Controls Present/Factor	Rain	Overturned
2016-02-10T20:33:00	No Pedestrians Involved	Dry	No Unusual Conditions	Dark - Street Light	Violation Driver 1	No Controls Present/Factor	Clear	Head On
2016-03-03T19:04:00	Crossing - Not In Crosswalk	Dry	No Unusual Conditions	Dark - Street Light	Violation Driver 1	No Controls Present/Factor	Cloudy	Vehicle/Pedestrian

Figure 14

Raw Crash Dataset Samples For Last Few Columns

CollisionType	ProximityToIntersection	VehicleInvolvedWith	PedestrianDirectionFrom	PedestrianDirectionTo	DirectionFromIntersection
Hit Object	Non-Related	Fixed Object	Not Applicable	Not Applicable	East Of
Hit Object	Non-Related	Fixed Object	Not Applicable	Not Applicable	West Of
Overturned	Non-Related	Fixed Object	Not Applicable	Not Applicable	South Of
Head On	Non-Related	Fixed Object	Not Applicable	Not Applicable	East Of
Vehicle/Pedestrian	Intersection	Pedestrian	South	North	At

3.2.2 San Jose Vehicle Dataset

The City of San José provides excellent services to the local community in a periodic manner and a facility of Open Data Portal serves as means to implement the City's Open Data Policy and Open Data Community Architecture. The data was collected from San Jose Law enforcement accident reports. The data was acquired the data from the website of the City of San Jose under the Department of Transportation. It consists of crash events from January 1, 2011, to December 31, 2020. The dataset is comprised of approximately 115,302 rows and 15 columns.

Figure 15 , figure 16 and Figure 17 depict the data collection plan for the dataset. Whereas, Figure 18 and Figure 19 show the raw samples of the dataset.

Figure 15

Data Collection Plan Description for Vehicle Dataset

Data Collection Plan

Project number: Project title: Analysis of Ve Project leader: Date: 10/20/2022

Description of the data collection	
Why are we collecting the dataset?	The vehicles dataset contained information about occupants and the vehicle directions. We are collecting the data to train a model which aims to accurately identify features that correlate to injury, and to implement a variety of machine-learning techniques to predict injury severity for vehicle accidents in the San Jose area
How are we collecting the dataset	The dataset can be directly downloaded in CSV format from the City of San Jose's Open Data Portal. The data was collected from daily accident reports that were collected by the San Jose Police Department, and the Department of Transportation. .
What will be done with the data once it has been collected?	After collecting the data, we analysed the dataset and prepared it by checking the compatibilit, performed cleaning, EDA, transformation and finally made it ready to be used in model training.

Figure 16

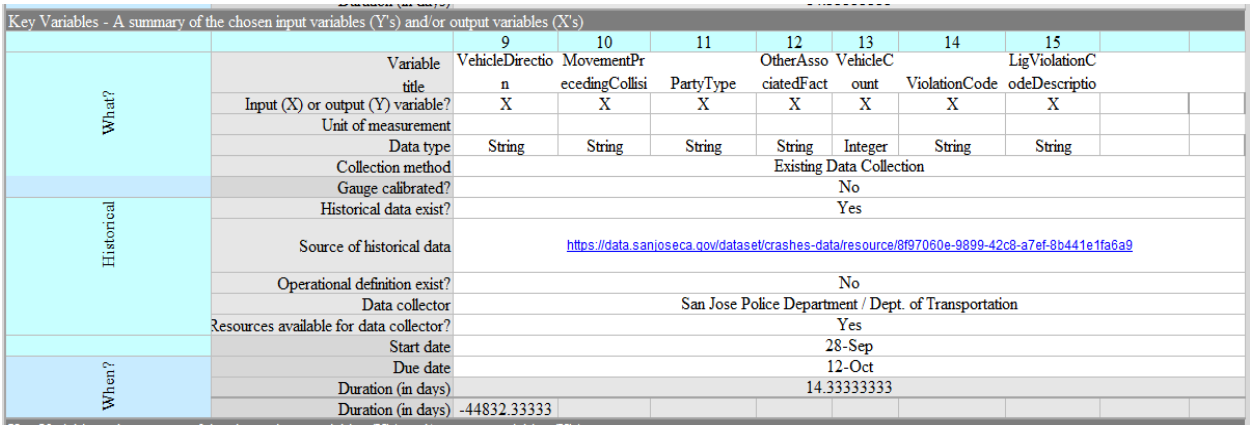
Data Collection Plan for Vehicle Dataset

Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)

		1	2	3	4	5	6	7	8
What?	Variable title	CrashName	Name	Sex	Age	Speed	VehicleDamage	PartyCategory	Sobriety
	Input (X) or output (Y) variable?	X	X	X	X	X	X	X	X
	Unit of measurement								
	Data type	String	String	String	Integer	Float	String	String	String
	Collection method	Existing Data Collection							
	If manual								
	Gauge calibrated?	No							
Historical	Historical data exist?	Yes							
	Source of historical data	https://data.sanjoseca.gov/dataset/crashes-data/resource/8f97060e-9899-42c8-a7ef-8b441e1fa6a9							
	Operational definition exist?	No							
	Data collector	San Jose Police Department / Dept. of Transportation							
	Resources available for data collector?	Yes							
When?	Start date	28-Sep							
	Due date	12-Oct							
	Duration (in days)	14.33333333							

Figure 17

Data Collection Plan Description for Vehicle Dataset



Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)									
		9	10	11	12	13	14	15	
What?	Variable title	VehicleDirection	MovementPrecedingCollisi	PartyType	OtherAssociatedFact	VehicleCount	ViolationCode	LigViolationCodeDescriptio	
	Input (X) or output (Y) variable?	X	X	X	X	X	X	X	
	Unit of measurement								
	Data type	String	String	String	String	Integer	String	String	
	Collection method	Existing Data Collection							
	Gauge calibrated?	No							
Historical	Historical data exist?	Yes							
	Source of historical data	https://data.sanjoseca.gov/dataset/crashes-data/resource/8f97060e-9899-42c8-a7ef-8b441e1fa6a9							
	Operational definition exist?	No							
	Data collector	San Jose Police Department / Dept. of Transportation							
	Resources available for data collector?	Yes							
When?	Start date	28-Sep							
	Due date	12-Oct							
	Duration (in days)	14.33333333							
	Duration (in days)	-44832.33333							

Figure 18

Raw Vehicle Dataset Samples

_id	CrashName	Name	Sex	Age	Speed	VehicleDamage	PartyCategory	Sobriety	VehicleDirection
1	CR-0000063652	ACV-0000000030	M	0		Minor	Driver	Impairment Not Known	East
2	CR-0000068628	ACV-0000000031		0		Unknown	Driver	Impairment Not Known	Unknown
3	CR-0000064498	ACV-0000000032	F	50		Minor	Driver	Had Not Been Drinking	South
4	CR-0000068721	ACV-0000000033	M	19		Minor	Driver	Had Not Been Drinking	North
5	CR-0000064227	ACV-0000000034	M	16		Unknown	Driver	Had Not Been Drinking	East

Figure 19

Raw Vehicle Dataset Samples

MovementPrecedingCollision	PartyType	OtherAssociatedFactor	VehicleCount	ViolationCode	ViolationCodeDescription
Proceeding Straight	Car	Unknown	1	00001	Other Improper Driving
Backing	Unknown	Inattention	1	00001	Other Improper Driving
Parking Maneuver	Car	Unknown	1	00001	Other Improper Driving
Proceeding Straight	Car	Inattention	1	00001	Other Improper Driving
Making Left Turn	Car	None Apparent	1	00001	Other Improper Driving

3.3 Data Pre-Processing

3.3.1 San Jose Crash Dataset

The Crash dataset consists of 25 columns distributed as data types: bool(2), float64(1), int64(5), and object(17). The total number of records was 56044. While checking for missing percentages, it was found that two features had missing values: The distance feature having 1.040254 and the Comment feature having 93.034045 percent.

In order to handle missing values, the column Comment was dropped as it has 93% of data missing. Figure 20 and Figure 21 show the list of columns before and after dropping Comment. For handling the Distance feature, the missing values were filled by the mean of the distance feature. Figure 22 shows the percentage of missing values before and after imputing for distance

Figure 20

Columns Before Dropping Comment Feature

```
'CrashFactId', 'Name', 'MinorInjuries', 'ModerateInjuries',
'SevereInjuries', 'FatalInjuries', 'TcrNumber', 'CityDamageFlag',
'ShortFormFlag', 'Distance', 'CrashDateTime', 'PedestrianAction',
'RoadwaySurface', 'RoadwayCondition', 'Lighting',
'PrimaryCollisionFactor', 'TrafficControl', 'Weather', 'CollisionType',
'ProximityToIntersection', 'VehicleInvolvedWith',
'PedestrianDirectionFrom', 'PedestrianDirectionTo',
'DirectionFromIntersection', 'Comment'],
```

Figure 21

Columns After Dropping Comment Feature

```
'CrashFactId', 'Name', 'MinorInjuries', 'ModerateInjuries',
'SevereInjuries', 'FatalInjuries', 'TcrNumber', 'CityDamageFlag',
'ShortFormFlag', 'Distance', 'PedestrianAction', 'RoadwaySurface',
'RoadwayCondition', 'Lighting', 'PrimaryCollisionFactor',
'TrafficControl', 'Weather', 'CollisionType', 'ProximityToIntersection',
'VehicleInvolvedWith', 'PedestrianDirectionFrom',
'PedestrianDirectionTo', 'DirectionFromIntersection', 'CrashDate',
'CrashTime'],
```

Figure 22

Missing Values in Crash Dataset

```
total missing values in Distance feature : 583

missing value in Distance feature after handling  : 0.0
```

It was found that the Distance feature has some outliers, which were handled by setting the range in between below Inter Quartile Range (IQR). Earlier, the data was spread around values 0.0 to 8448.0 (in ft). The distance a car moves after an accident cannot be 8448 feet. This distance is way too high therefor, outlier removal was performed. Even after removing outliers, some more data points were left from the maxima which were considered to be kept. Figure 23 and Figure 24 illustrate befor and after removing outliers using boxplot.

Figure 23

Boxplot Before Removing Outliers

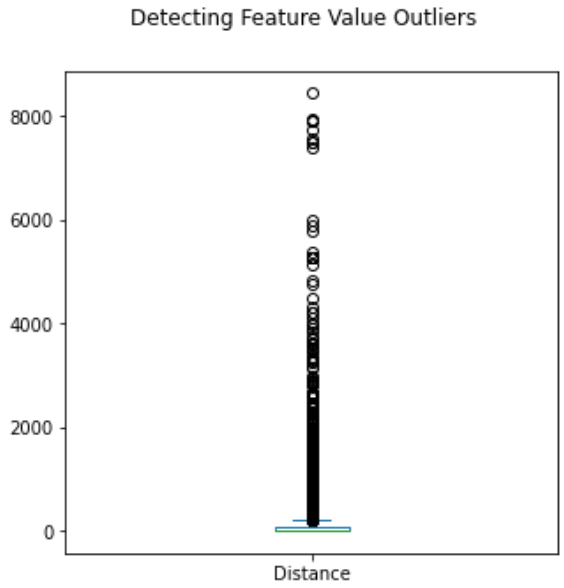
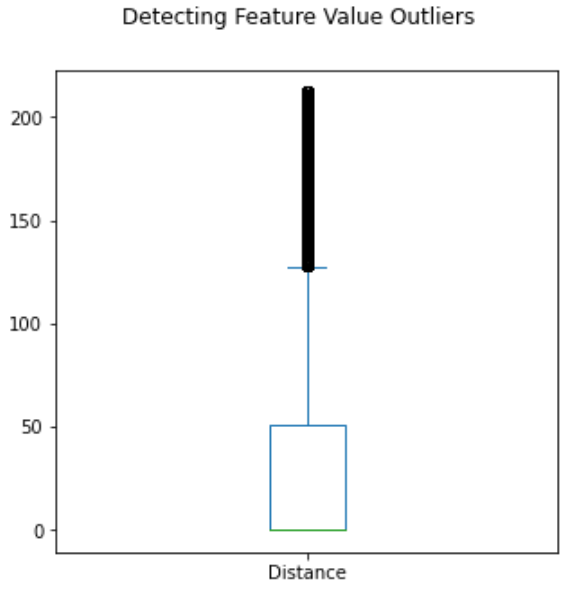


Figure 24

Boxplot After Removing Outliers

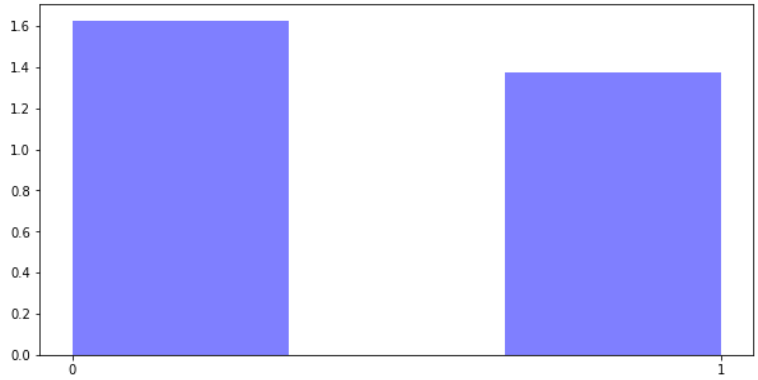


For further analysis, binning was performed on this feature and grouped the data into binary zero or one, such as all values that have zero values are kept under zero label and values

greater than zero are kept under label one. The lower boundary was set of the bin to 500 and the upper boundary to 1000. The below image is the histogram plot of the distance feature after performing binning.

Figure 25

Distribution of Distance After Binning



In order to check inconsistencies in the dataset, each column's unique values were checked and found that CrashDateTime feature had date and time combined together separated by space in MM/DD/YYYY HH:MM. New features named CrashDate and CrashTime were made and CrashDateTime feature was dropped. Figure 26 and Figure 27 show the above preprocessing.

Figure 26

Sample Data with CrashDateTime Feature

```
 3/14/2018 23:17
   3/1/2018 7:30
   2/2/2016 9:02
 2/10/2016 20:33
  3/3/2016 19:04
  5/6/2018 18:58
 5/25/2017 18:43
 2/26/2013 21:30
12/19/2014 21:50
 2/23/2013 18:59
```

Figure 27

Sample Data with CrashTime and CrashDate Features

```
0    0:09     12/26/2020
1    2:03     12/26/2020
2    2:08     12/26/2020
3    2:19     12/26/2020
4    2:33     12/26/2020
5    2:34     12/26/2020
6    3:33     12/26/2020
7    6:36     12/26/2020
8    6:50     12/27/2020
9    7:05     12/27/2020
```

Furthermore, performed grouping on all the injury types i.e. MinorInjuries, ModerateInjuries, SevereInjuries, and FatalInjuries. Each type was divided into binary groups such as zero for non-injuries and one for all injuries from level 1 to n e.g., fatal = {0: 0, 1: 1, 2: 1, 3: 1} and assigned back to the dataframe in order make that a binary feature**.**

Moving forward, the existing dataframe was copied into a new dataframe to perform One Hot Encoding for all categorical features and dropped all the other features which were not

required. This will help to achieve better results with the models. To perform One Hot encoding, a number of labels were checked that will be created after completing encoding on the selected features, and as a result, there are 101 features with 56044 records. For the naming scheme, the column names were used followed by the label. Then the main features were finally dropped upon which the encoding was performed. Finally, concatenation was performed on the main data frame which is df, and the final data frame which is Final_df. The total dimension of the final dataframe was 49607 rows, 126columns. At last, duplicated columns in the dataframe were checked, and it delivered the result of 0. Cross-validated the missing value after creating the final dataframe and got a result of 0 percent. Figure 28 shows sample data after encoding is done. Code implementation can be seen in Appendix C.

Figure 28

Sample Data After Performing One Hot Encoding

Distance	CrashDate	CrashTime	...	PedestrianDirectionTo_Southeast	PedestrianDirectionTo_Southwest	PedestrianDirectionTo_Unknown	PedestrianDirectionTo_West
0.0	1/1/2011	0:09	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	2:03	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	2:08	...	0.0	0.0	0.0	1.0
0.0	1/1/2011	2:19	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	2:33	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	3:33	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	6:36	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	6:50	...	0.0	0.0	0.0	0.0
0.0	1/1/2011	7:05	...	0.0	0.0	0.0	0.0

3.3.2 San Jose Vehicle Dataset

The first feature available in the dataset is CrashName. CrashName is a key between this dataset and the San Jose Crashes Dataset. This alphanumeric value is unique in that dataset but not unique in this one. This is the vehicles dataset, and because more than one vehicle can be

involved in a collision, the same value can show up multiple times here. The minimum number of vehicles involved in a collision is obviously one, and the maximum is 14. After some basic exploration, it was determined that only one vehicle is considered at fault in all of these accidents. While accidents can have 14 involved parties, because it was being tried to determine the severity of an accident. The first time each CrashName is listed in the dataset is the at-fault driver. The entire dataset was filtered down to only the first mention of each CrashName. Because this feature isn't being used in modeling, no other cleaning steps are required beyond having checked for nulls, of which there are none. The dataset dropped from the starting point of 115,302 records to 56,033 records after this filtering process.

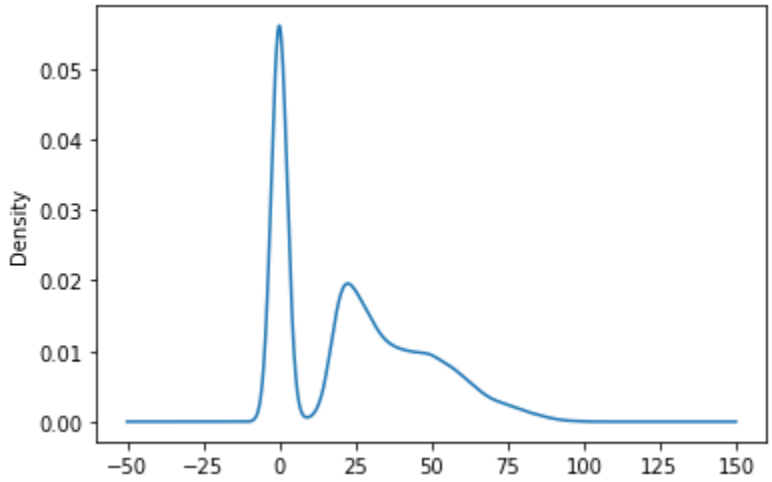
The next feature available in this dataset is Name. It is a non-null alphanumeric unique value for each record in this dataset. This is the key for this dataset. It may provide use for joining datasets, but it will not be used for modeling purposes, and so it was ignored for now.

Our first potentially useful feature is Sex, this feature is coded as M, F, or NaN. For this feature, the data was missing 22% of the values. Of the remaining 78%, 65% of those were male drivers, and 35% were female. Because Sex is not ordinal, and there are very few categories, it is the prime example of a time where One-Hot encoding shines; for those reasons,One-Hot encoding was implemented for this feature. The new column names are 'M,' 'F,' and 'SexUnknown.'

Moving on to Age, this should be a discrete variable in the 0-100 range, maybe higher if there are some really old people driving. The distribution of involved parties should be relatively normal, maybe left skewed slightly due to higher impulsivity and less experience in younger drivers. The distribution can be seen below in Figure 29.

Figure 29

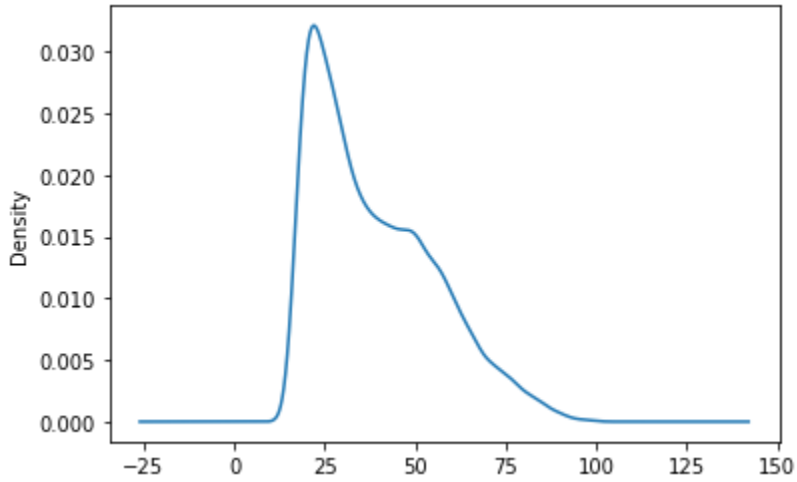
Distribution of Age Before Cleaning



Unfortunately, its was found that 36% of all records had an age of zero. There is an inconsistency. There are three things that can be done which have their pros and cons: replace the zero values with the mean or mode, throw out age as a feature, or throw out all records that contain age = 0. The decision was made to throw out all the records containing age = 0. While it is a hard decision to throw out 36% of your data, it was decided that the kind of records where the record-gatherer did not take the time to record age were likely going to be unreliable in other, not as easy-to-find aspects as well. Finally, ended up with 35,663 records left. After removing those records, the distribution looks far closer to what was expected. The results can be seen below in Figure 30.

Figure 30

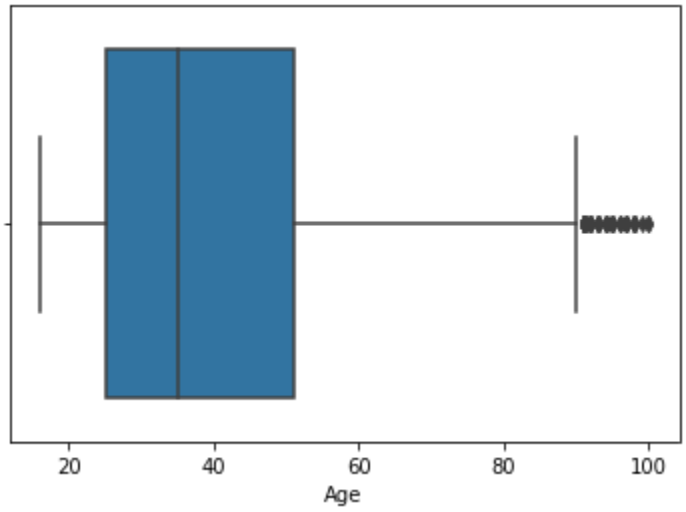
Distribution of Age After Removing Age=0 Values



Further, because the goal is to find the cause of accidents, and the legal driving age is 16 in San Jose, it was decided that it was pertinent to remove any values of Age under 16. This was because those under the age of 16 are likely not the cause of an accident. The record count is down to 35,158. Looking at a boxplot of the Age feature one can see that this is far more in line with expected results, still with some outliers. You can see the outliers in the boxplot in Figure 31.

Figure 31

Outliers in Age Feature



There are still calculated outliers here, but the group does not suspect these are measurement errors. All the detected outliers are toward Age=100, and as anybody with a grandparent knows, accidents and diminished driving ability are not exactly rare in that age group. It would be disingenuous to the results if these outliers were dropped, so they will stay.

Our last non-categorical feature in this dataset is Speed. It had three individual values here: 0, 30, and 80. One can expect that it is measured in miles-per-hour. In this dataset, over 99% of the values for speed are 0. It was assumed that nothing will be learned by including this feature, and as such, dropped it.

VehicleDamage was a feature expected to have a major impact on the results. This feature has zero nulls and is categorical, with the categories being Major, Moderate, Minor, Unknown, None, Not Applicable, and Totaled. This feature was One-Hot encoded to allow the models to have a bit more freedom. Theoretically, this is ordinal, which would give the option of label encoding but 17% of the records are unknown, so the group had to decide where in the order

those belong. Label encoding could damage the results because of such a large chunk of data missing; as such the group decided to proceed with one hot encoding.

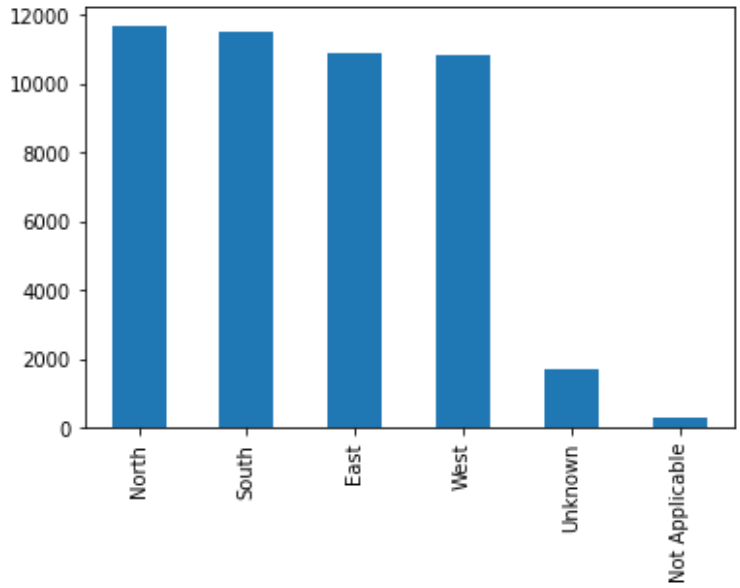
Our next feature is PartyCategory which is the type of party involved in this collision or the cause of the said collision. This is categorical with zero nulls but a few unknown values. The categories are Driver, Parked, Bicycle, Pedestrian, Other, and Unknown. Drivers make up 93% of the values here, but it was suspected that Pedestrians and Bicycles likely make up a more significant portion of those involved in more serious accidents. In this situation, there is no order to the categories, so the group chose to proceede with One-Hot encoding. The unknown values only made up less than .01% of the data, but they were One-Hot encoded to a new column anyway.

Another categorical feature the group believed would play a strong role in the findings is Sobriety. There are ten categories here, everything from Had Been Drinking to Sleep/Fatigued. There is no apparent order to these because it is hard to determine if being under the influence of drugs or alcohol is more serious in affecting one's driving abilities. It was decided to keep this data, so it was One-Hot encoded as well. This feature has zero nulls and zero unknowns but less than one percent of the values were written as Not Applicable; the group still included these in their own column.

Our next non-ordinal yet categorical variable is VehicleDirection. This feature uses the four compass directions to tell which way the vehicle is moving. There are two other categories of Unknown and Not Applicable, but combined, they only make up about 4% of the data. This is a feature where it is not immediately obvious why to keep it. By itself, this feature is likely useless. As this bar chart below in Figure 32 shows, accidents are very evenly distributed across the four directions.

Figure 32

Barchart of Vehicle Compass Direction



This dataset does not contain time information, but it will be joined to a dataset that does have time information. Theoretically, the group could have created a feature based on the sun's direction. For example, it is known the sun rises in the East and sets in the West, so one could create a binary feature based on time and direction data that is basically 'Driving in the Direction Facing the Sun'. Those drivers with increased glare on their windshields might be more likely to be involved in an accident, and those accidents might be more severe. For those reasons, this feature stays and will be One-Hot encoded as well.

Our next categorical and non-ordinal feature that again has zero null values is PartyType. This one contains the type or lack-of vehicle. There are 19 categories from non-motorized scooters to Ice Cream Trucks, notably, Train as well. This feature will be pertinent to the results and given the size of the dataset, the memory penalty with One-Hot encoding, a feature this size will be worth it. About 1% of the results are either Other or Unknown, but those got their individual columns as well.

There was yet another categorical feature to encode. MovementPrecedingCollision is things like parking, driving straight, etc. This was again a process of renamed the categories that are not self-evident before One-Hot encoding the categories. There are more than a dozen categories here but again; the memory penalty is worth it to keep this data. Unknown and Other make up an insignificant amount of this feature, and there were zero nulls. About half the values were for Proceeding Straight.

One of the final useful features is yet another zero null, non-ordinal, categorical feature. This one has 32 categories. The kinds of categories are things like brakes failing or improper signaling. While this has zero nulls, about 75% of records had answers like N/A, or Other. While there isn't a lot of useful data in this feature, the data that is here could be very relevant. Again One-Hot encoded this feature. The dataset is now up to 58 columns.

Our next feature is VehicleCount which is an integer between one and one. This feature has zero nulls, but every value is identical at one. This feature has been dropped as it will provide zero value.

Our next feature is ViolationCode. This is also categorical, with violations the party was cited for after the accident. The data had 80 unique values here. There are four that are problematic: N/A, Unknown, 00001, 00002. 00001 and 00002 are not California violation codes. Those two will be changed to unknown, and the whole thing will be One-Hot encoded. The numbers as column titles will stay. With there being 80 unique categories in this feature, One-Hot encoding will be more memory intensive than label encoding. With that being said, given the limited dataset and the potential model performance improvements, it is worth it.

Our final feature to clean is ViolationCode Description. These values relate directly to the violation code; they will have a nearly 100% correlation. It is improper to keep both. While these

labels are far more descriptive, the length of these descriptions makes it so the violation codes were rather kept as column names. This feature will be dropped. Code implementation can been in Appendix B.

3.4 Data Transformation

3.4.1 Merging datasets

The data transformation process begins by combining the San Jose Crash and San Jose Vehicle datasets inside of a Google Colab Jupyter notebook. This was done by using Pandas to join both datasets on the unique feature crash name "CR-00000XXXXX." The result of this equated to a combined dataset of 309 columns and 41,321 rows. It was then decided to drop unique identification type features (CrashName, CrashFactId, Unnamed: 0_x, Unnamed: 0_y, CrashName, TcrNumber). Lastly, for the purposes of this multi classification problem, dropped features related to date and time. The final combined dataset size equated to 301 columns and 41,321 rows. Figure 33, Figure 34 and Figure 35 show the sample of the combined dataset. Code implementation can been seen in Appendix D.

Figure 33

Sample of First Nine Columns in Combined Dataset

	MinorInjuries	ModerateInjuries	SevereInjuries	FatalInjuries	Distance	CityDamageFlag_False	CityDamageFlag_True	ShortFormFlag_False	ShortFormFlag_True
0	0.0	0.0	1.0	0.0	0.000000	1.0	0.0	1.0	0.0
1	1.0	1.0	0.0	0.0	0.000000	1.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.000000	0.0	1.0	1.0	0.0
3	0.0	1.0	0.0	0.0	0.000000	0.0	1.0	1.0	0.0
4	0.0	1.0	0.0	0.0	0.000000	0.0	1.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...
41413	1.0	0.0	0.0	0.0	45.000000	1.0	0.0	1.0	0.0
41414	0.0	0.0	0.0	0.0	51.000000	1.0	0.0	1.0	0.0
41415	0.0	0.0	0.0	0.0	111.000000	1.0	0.0	1.0	0.0
41416	1.0	0.0	0.0	0.0	30.000000	1.0	0.0	1.0	0.0
41417	1.0	0.0	0.0	0.0	31.760211	0.0	1.0	1.0	0.0

41316 rows × 301 columns

Figure 34

Sample of Middle Columns in Combined Dataset.

Weather_Clear	Weather_Cloudy	Weather_Fog	Weather_Other	Weather_Rain	Weather_Snow	Weather_Unknown	Weather_Wind	CollisionType_Broadside	CollisionType_Head On	CollisionType_Hit Object
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 35

Sample of Ending Columns in Combined Dataset

Sex	Age	VD_Major	VD_Minor	VD_Moderate	VD_N/A	VD_None	VD_Totaled	VD_Unknown	PC_Bicycle	PC_Driver	PC_Other	PC_Parked	PC_Pedestrian	PC_Unknown
1.0	21.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	21.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	33.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	28.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	21.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.0	78.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	26.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	33.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	23.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	34.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

3.4.2 Normalization

The next process for data transformation consisted of performing min-max normalization inside of Google Colab for features that were not already one-hot encoded. This resulted in the features ‘Age’ and ‘Distance’ being normalized from 0 to 1, which is consistent with the other

one-hot encoded features for the purposes of building machine learning models. Figure 36 shows the summary statistics.

Figure 36

Summary Statistics for Distance & Age

	Distance	Age
count	41321.000000	41321.000000
mean	0.133495	0.318424
std	0.237389	0.223980
min	0.000000	0.000000
25%	0.000000	0.123288
50%	0.000000	0.273973
75%	0.184332	0.479452
max	1.000000	1.000000

Figures 37 show samples of the combined dataset normalized.

Figure 37

Sample of Normalized Dataset

	MinorInjuries	ModerateInjuries	SevereInjuries	FatalInjuries	Distance	CityDamageFlag_False	CityDamageFlag_True	ShortFormFlag_False	ShortFormFlag_True
0	0.0	0.0	1.0	0.0	0.000000	1.0	0.0	1.0	0.0
1	1.0	1.0	0.0	0.0	0.000000	1.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.000000	0.0	1.0	1.0	0.0
3	0.0	1.0	0.0	0.0	0.000000	0.0	1.0	1.0	0.0
4	0.0	1.0	0.0	0.0	0.000000	0.0	1.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...
41413	1.0	0.0	0.0	0.0	0.207373	1.0	0.0	1.0	0.0
41414	0.0	0.0	0.0	0.0	0.235023	1.0	0.0	1.0	0.0
41415	0.0	0.0	0.0	0.0	0.511521	1.0	0.0	1.0	0.0
41416	1.0	0.0	0.0	0.0	0.138249	1.0	0.0	1.0	0.0
41417	1.0	0.0	0.0	0.0	0.146360	0.0	1.0	1.0	0.0

41316 rows × 301 columns

At this time, it was decided to adjust how the target feature was set up. There were four columns for the target: MinorInjuries, ModerateInjuries, SevereInjuries, FatalInjuries. It was

decided to combine these features into one column called Severity and label them as the following: No Injury = 0, Minor Injury = 1, Moderate Injury = 2, Severe Injury = 3, Fatal Injury = 4. This resulted in a single column as the target feature while still properly maintaining the target as multi classification. This resulted in the combined dataset reducing from 301 to 298 columns. Figure 38 shows the first five rows of the combined dataset with the new target feature.

Figure 38

Sample of Target Feature

Severity
3
1
0
2
2

3.4.3 Regularization

Regularization techniques were used to reduce the risk of overfitting the models. LASSO (least absolute shrinkage and selection operator) is a technique that performs variable selection as well as regularization to improve the statistical regression model's predictive performance and interpretability (Emmert-Streib & Dehmer, 2019). The Ridge technique was developed in response to the notion that by decreasing the values of regression coefficients or setting coefficients to zero, the accuracy of a prediction could be improved. To understand the performance differences between Lasso and Ridge regularization, the group implemented and compared both with the baseline model. There isn't much of a difference between the baseline, lasso, and ridge models. Table 8 and Figure 39 show that using ridge regularization results in a slight increase in evaluation metrics.

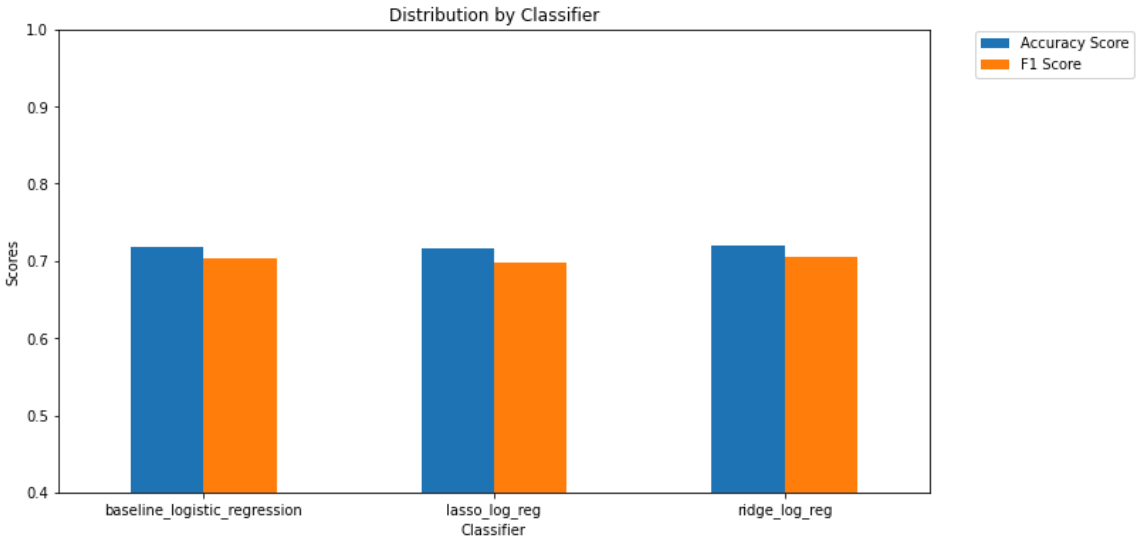
Table 8

Model Performance Based on Regularization

Classifier	Accuracy Score	F1 Score
baseline_logistic_regression	0.718400	0.703883
lasso_log_reg	0.715657	0.698578
ridge_log_reg	0.719529	0.704411

Figure 39

Distribution of Performance Metrics by Classifier



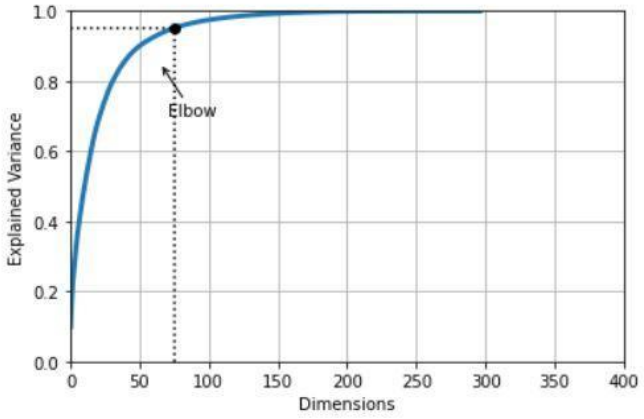
3.4.4 Reduction

For data reduction the group decided to use Principal Component Analysis (PCA). There are positives to going this route, and negatives. For the sake of analysis, PCA creates a black box type situation where one can not easily identify which features were used for reduction (Géron,

2019, p. 290). Although PCA provides the ability to have greater accuracy with its feature reduction capability than other feature reduction techniques, PCA works by not just lazily selecting features for a reduction but trying to maintain a threshold for variance with the fewest amount of features (Géron, 2019, p. 292). The parameter of variance was set to be maintained at 95% because the group felt this would allow for accuracy to be adequate for building and testing models. This equated to PCA reducing the combined dataset from 298 features to just 75 features. Figure 40 shows the PCA plot with elbow.

Figure 40

PCA Variance Plot with Elbow



PCA was tested with logistic regression to see how it performed against the baseline model which has all 293 features. This resulted in PCA with only 75 features performing nearly equal to the baseline model, which means that the data reduction technique was beneficial. Table 9 and Figure 41 shows the performance of PCA.

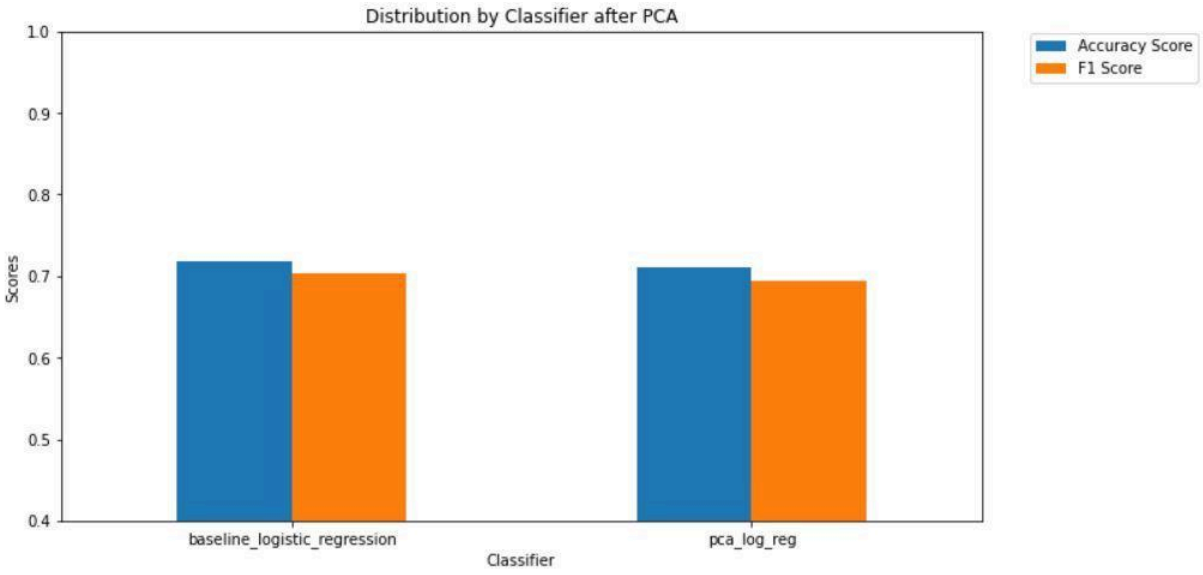
Table 9

Model Performance After Performing Smote

Classifier	Accuracy Score	F1 Score
baseline_logistic_regression	0.718400	0.703883
pca_log_reg	0.711462	0.694271

Figure 41

PCA Accuracy and F1 Score



3.4.5 Sampling

The target feature ‘severity’ consists of 5 classes and these classes are imbalanced. To overcome this imbalance, the group used Synthetic Minority Oversampling Technique (SMOTE). Here, the minority classes will be oversampled synthetically (Rančić et al., 2021, p.115). The distribution of classes is visible in Table 10. It can be noticed that there is a lot of

difference in the number of instances between No injury and Fatal injury. After implementing SMOTE, all the classes will consist of the same number of instances. This can be seen in Figure 42. Models were run to see if there was any improvement due to sampling. However, it was found out that there was a drop in performance. It can be seen in Table 11.

Table 10

Distribution of Severity Feature Before SMOTE

Class Name	Label	No. of Instances
No injury	0	20955
Minor injury	1	14486
Moderate injury	2	4637
Severe injury	3	987
Fatal injury	4	256

Figure 42

Distribution of Severity Feature After SMOTE

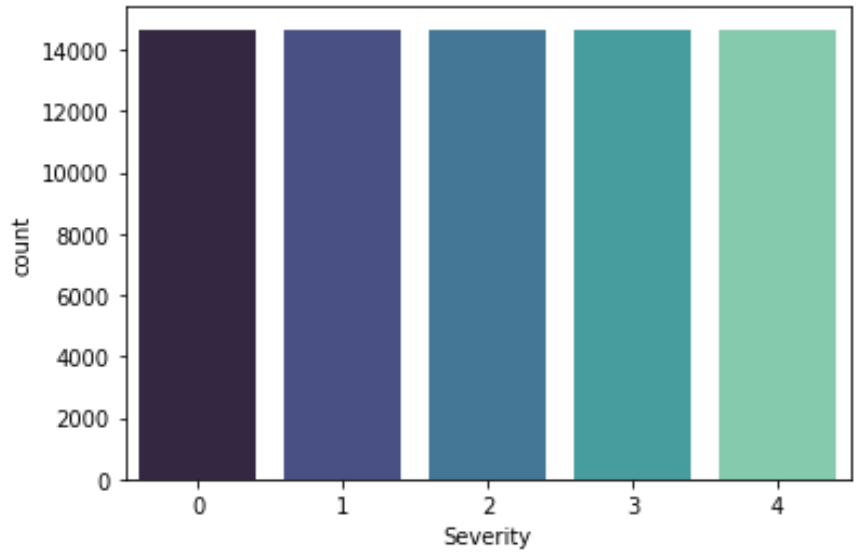


Table 11

Model Performance After SMOTE

Classifier	Accuracy Score	F1 Score
baseline_logistic_regression	0.650480	0.677839
lasso_log_reg	0.650722	0.677507
ridge_log_reg	0.649835	0.677305

Code implementation for transformations can be seen in Appendix E.

3.5 Data Preparation

Next step proceeded after the group completed the process of transforming the data using various strategies is data preparation. The total dataset contains 76 features and 41,423 instances. Firstly, Severity was assigned as the target feature and then it was used the Fast ML library's

train valid test split function to divide the dataset into training, validation, and testing datasets. The train-validate-test split is a method for assessing the performance of your machine learning model. The dataset will be divided into 70%/15%/15% sections, with 70% designated as a training dataset and 15% designated as testing and validation datasets. During the model development phase, K-fold cross-validation will be used to validate the performance. The shapes of the subsets can be seen in Figure 43.

Figure 43

Distribution of Training, Validation, and Testing Data

```
Training Data:
(28996, 75)
(28996,)
Validation Data:
(6213, 75)
(6213,)
Testing Data:
(6214, 75)
(6214,)
```

The training dataset after the split consists of 28996 rows and 75 columns which is 70% of the combined dataset. This dataset will be used to train the models. A sample of this dataset can be seen in Figure 44.

Figure 44

Sample of Training Dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0.431011	1.496665	0.874729	0.261071	-0.613081	-0.325199	-0.416613	-0.021119	0.448262	-0.941971	0.119415	-0.506185	0.004026	0.417524	-0.137096	1.183237	-0.217313	0.387868	-0.690408	-0.056115	-0.327721	0.331617
1	-0.924196	-0.489344	0.861949	1.276509	-0.544445	-0.054738	0.492802	-1.077580	-0.538482	0.958375	0.506749	0.327939	-0.754299	-0.796781	-0.103147	0.249189	-0.388781	0.563871	0.305227	-0.266727	0.713836	0.064986
2	0.863767	0.224296	-0.736938	0.683205	0.004380	0.733018	-0.619008	-0.691422	0.720822	0.547714	-0.579747	0.592261	0.041413	-0.330005	0.317236	0.713706	0.454323	1.426011	0.058433	0.317406	-0.169558	-0.360026
3	-0.599463	-0.961674	-0.555194	0.011145	-0.028033	0.145355	0.513477	0.097245	-0.106411	1.000309	0.462181	0.418784	0.638251	-0.491803	-0.590406	0.149619	-0.091356	0.524672	0.511448	-0.425801	0.254153	-0.045733
4	-0.241518	-0.922015	0.024374	-0.434244	0.848884	-0.442154	0.344167	-0.209467	-0.603053	0.072165	-0.287491	-0.328059	0.698174	0.208337	-0.646502	0.498220	-0.445050	0.151395	-0.741429	0.004927	-0.291974	-0.257373

The validation dataset consists of 6213 instances and 75 features. This will be used to check the performance of the model prior to testing on hidden data. The sample for this dataset is shown in Figure 45.

Figure 45

Sample of Validation Dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	-0.541900	0.524606	-1.160496	-0.155588	0.298291	1.146705	-0.769894	0.484120	-0.439378	0.271758	-0.475057	-0.435804	-0.845803	-0.040432	-0.388913	0.029000	-0.150012	0.067614	-0.288421	-0.907238	0.297106	-0.862902
1	-0.696515	1.102190	-1.103474	-0.822219	0.090599	-0.267785	-0.135192	0.743975	0.888928	-0.209786	-0.477928	1.004049	-0.115690	0.080394	-0.720149	0.361460	-0.461730	-0.096223	0.040553	-0.219652	0.254728	-0.044795
2	1.809382	-0.870321	1.156107	0.631660	-0.315384	-0.729008	0.379526	0.229005	0.487086	-0.125898	-0.159890	-1.083373	0.154014	0.486833	-0.339468	-0.266479	0.017747	-0.010112	0.310960	-0.490256	0.477749	0.253731
3	-1.053406	-0.836978	0.297964	-0.057193	0.204710	0.504952	0.805638	-1.316119	0.306991	1.005251	-0.721629	0.078316	0.070649	-0.558358	-0.306594	0.062840	-0.717308	0.675077	-0.622784	-0.102578	0.689566	0.723597
4	-0.339552	1.477388	0.278537	0.371622	-0.538640	-1.040594	0.097849	1.671175	-0.020411	0.614022	0.336637	0.419075	-0.927095	0.182549	-0.222137	-0.015336	-0.218387	-0.799568	0.087377	-0.378326	0.188288	-0.102038

Lastly, the testing dataset consists of 6214 instances and the same number of features as above. This dataset is hidden until the testing and evaluation phase begins. A sample of this dataset can be seen in Figure 46.

Figure 46

Sample of Test Dataset

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	-1.008822	1.732798	1.018269	0.515891	-0.354901	-0.676685	-0.806995	-0.468025	0.410404	-0.366029	-0.655220	0.147755	0.074162	0.253333	-0.144600	0.227394	0.546893	-0.515539	0.256608	-0.244083	-0.178990	-0.387770
1	0.338390	0.312624	1.493532	0.279143	1.835454	2.188155	-0.243868	0.023295	0.308668	-0.448188	1.057167	0.082582	0.186116	0.594714	0.334223	-0.486582	1.309104	0.107759	0.097118	0.292231	0.544711	0.264422
2	-0.651990	-1.244259	-0.235345	-0.431885	0.307883	-0.446208	0.595746	-0.255337	-0.631650	0.085866	-0.084158	-0.225064	0.795928	0.647512	0.230552	-0.566939	-0.138944	0.341848	0.030595	-0.300080	-0.451919	0.128680
3	0.178021	-0.799609	1.354455	-0.521103	1.411786	0.001693	-1.013078	0.137253	0.564811	-0.197956	0.191720	-0.728181	0.124849	-0.117110	0.097439	0.337493	-0.203269	0.671279	-0.182587	-0.063932	-0.285582	-0.076015
4	-1.041734	-0.304736	0.333221	-0.919910	-0.049584	0.463118	0.359906	-0.497084	0.336576	0.027434	0.014481	-0.859049	-0.029832	-0.034370	-0.886713	-0.033055	-0.874467	-0.022766	0.310306	0.399544	-0.476093	0.327388

3.6 Data Statistics

Explaining datasets in text is very helpful but nothing replaces hard numbers: the following table and figures give dataset statistics for all stages of the datasets from raw to their final pre-processed state. The first of those, Table 12 is the dissemination of information regarding the size and shape of the datasets. The following Figures, Figure 47 through Figure 52 show us, in histogram form, the distribution of datatypes in each stage of the data engineering method.

Table 12

Statistics of Datasets

Stage	Dataset	# of Rows	# of Columns	Storage Size
Raw	San Jose Vehicles	115302	15	13.2MB
	San Jose Crashes	56044	25	9.9MB
Pre-Processed	San Jose Vehicles	35054	183	8.1MB
	San Jose Crashes	49607	126	48.1MB
Combine	Combined	41423	298	94.2MB
Transformation	Feature Extraction and Dimensionality Reduction	41423	77	24.2MB
Preparation	Training	28996	76	16.8MB
	Validation	6213	76	3.6 MB
	Testing	6214	76	3.6 MB

Figure 47

Raw San Jose Vehicles Datatset Count of Data Type

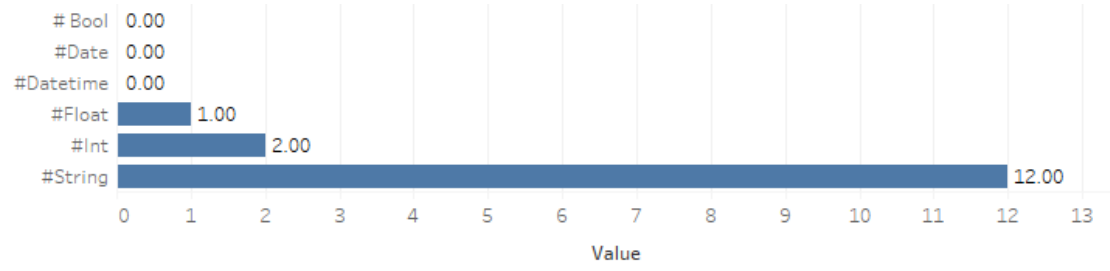


Figure 48

Raw San Jose Crashes Datatset Count of Data Type

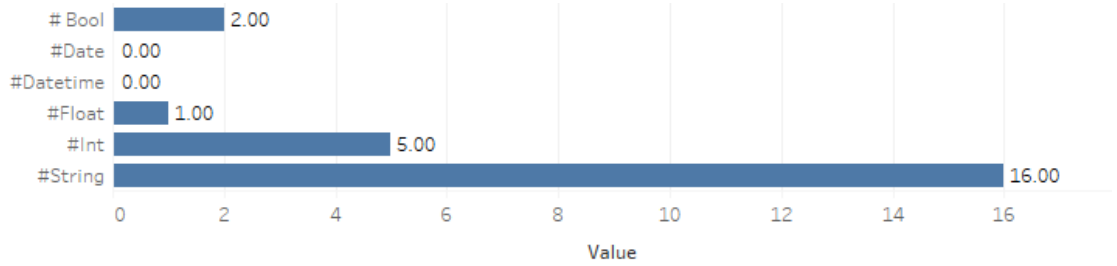


Figure 49

Pre-Processed San Jose Vehicles Datatset Count of Data Type

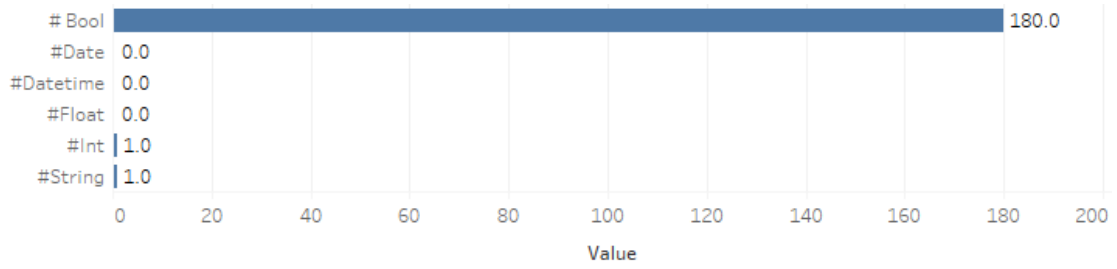


Figure 50

Pre-Processed San Jose Crashes Datatset Count of Data Type

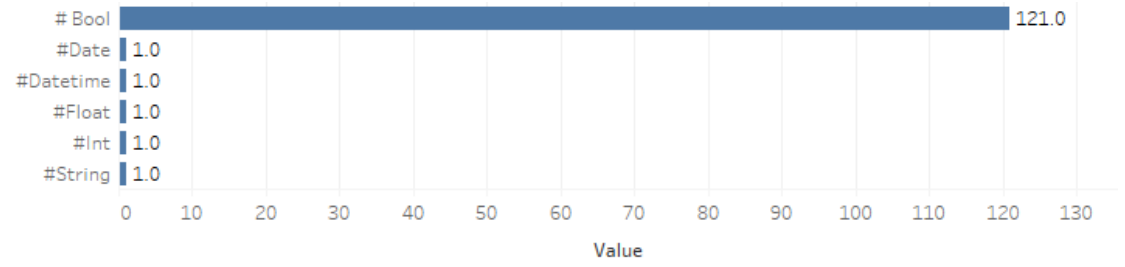


Figure 51

Combined Datatset Count of Data Type

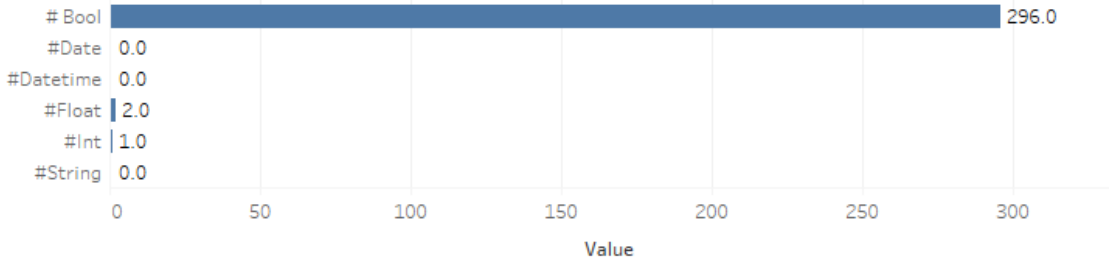
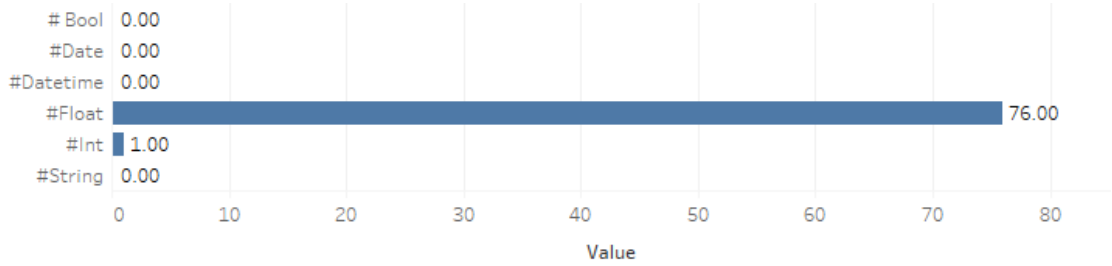


Figure 52

Prepared Datatset Count of Data Type



3.7 Data Analytics Results

GIS data from the City of San Jose were finally obtained that included longitude and latitude for the combined dataset. The GIS data was added to the combined dataset by using a join on the CrashName feature. This allowed the group to create geospatial visualizations of the target feature Severity, which has the following: no injury, minor injury, moderate injury, severe injury, and fatal injury. Figures 53, 54, 55, and 56 depict minor, moderate, severe, and fatal injuries respectively, for the San Jose area. For Figures 57 and 58, visualization of features were performed that have high importance for fatal injury based on Random Forest feature importance. Figure 57 depicts fatal injuries by the most important feature age, and Figure 58 depicts the next most important feature importance for fatal injuries, sex.

Figure 53

Geomap Depicting Minor Injuries

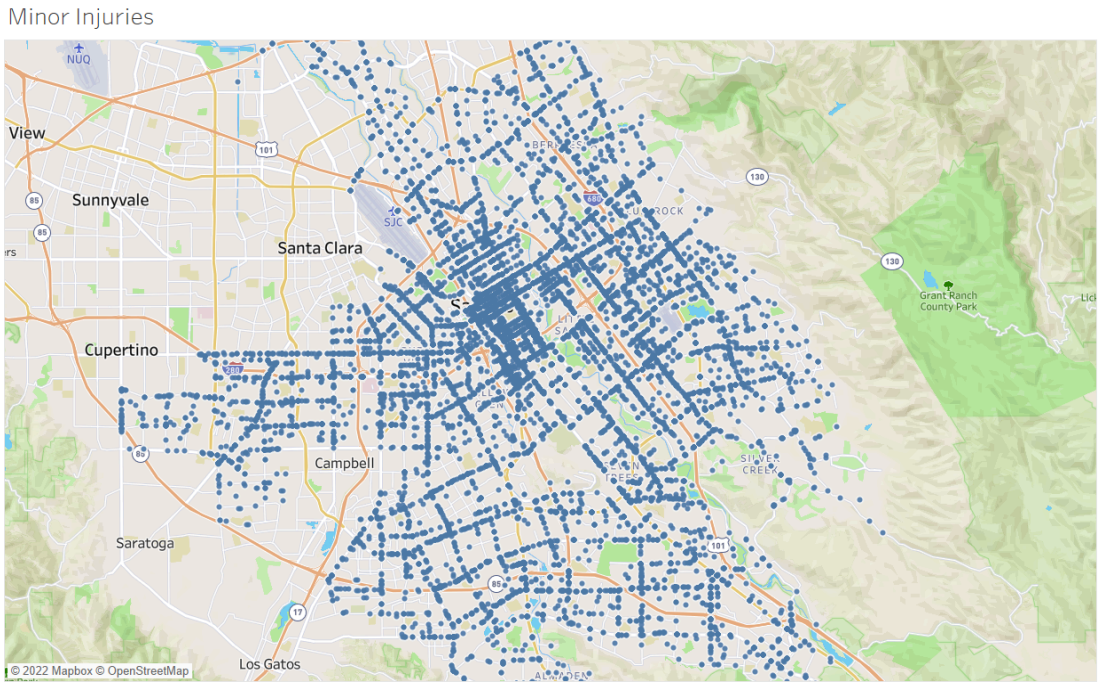


Figure 54

Geomap Depicting Moderate Injuries

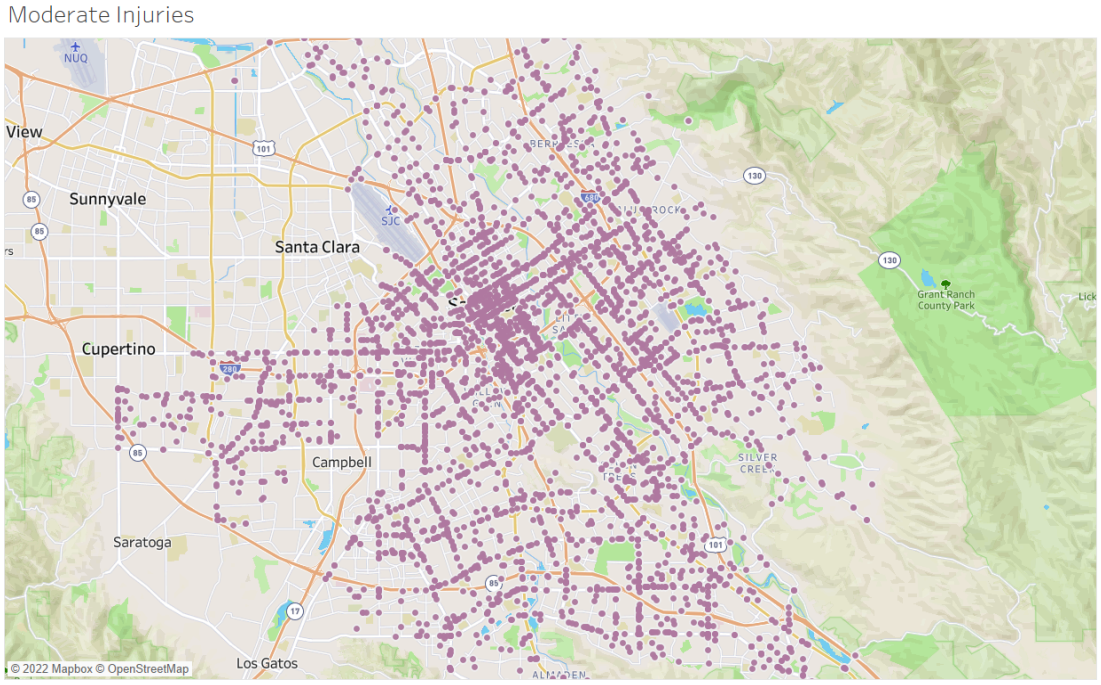


Figure 55

Geomap Depicting Severe Injuries

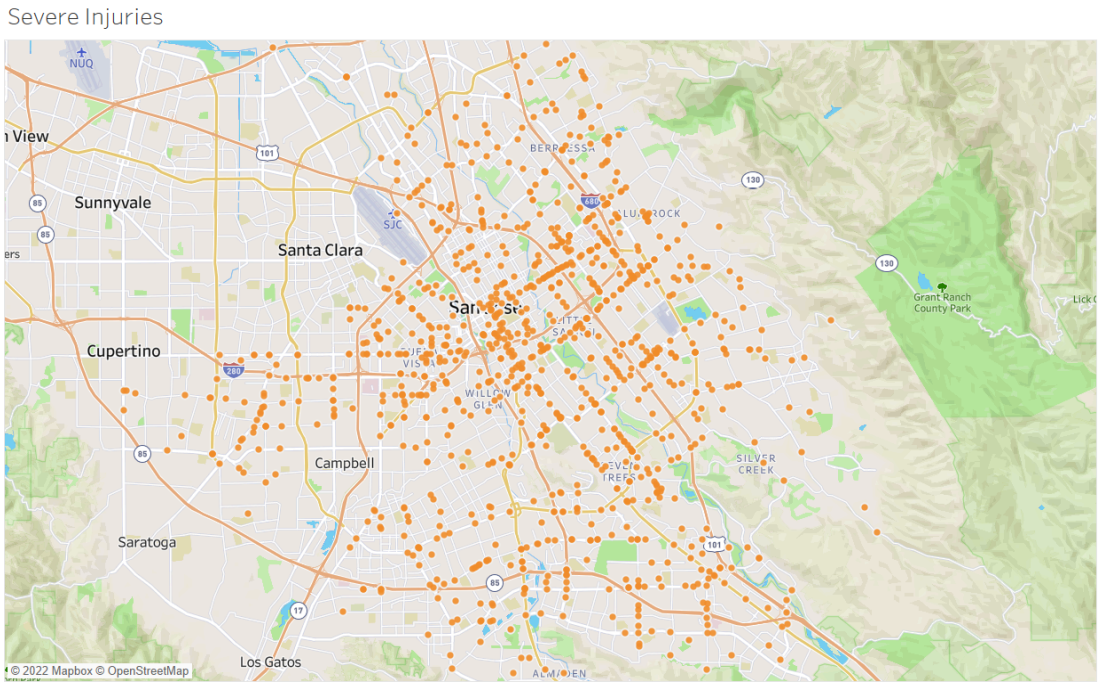


Figure 56

Geomap Depicting Fatal Injuries

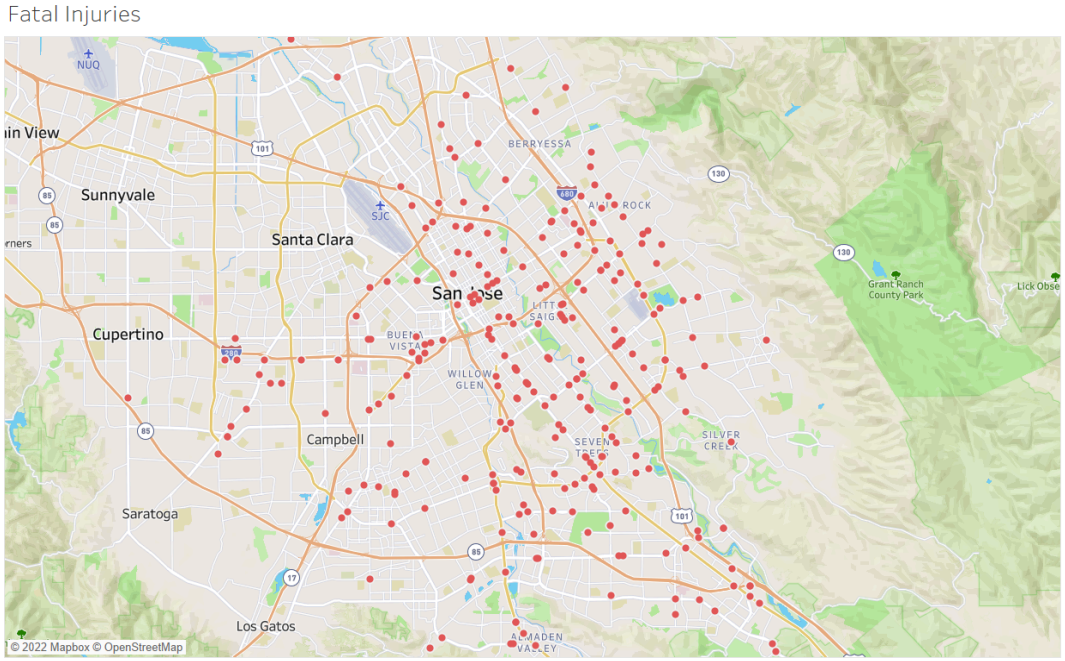


Figure 57

Geomap Depicting Fatal Injuries By Age

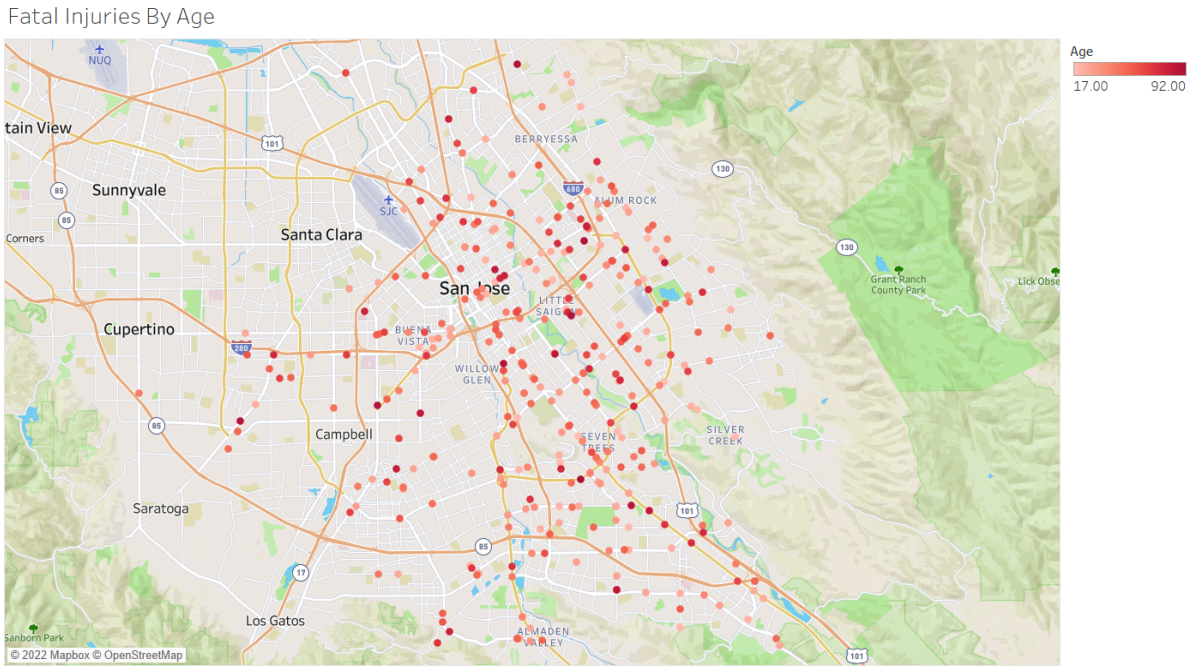
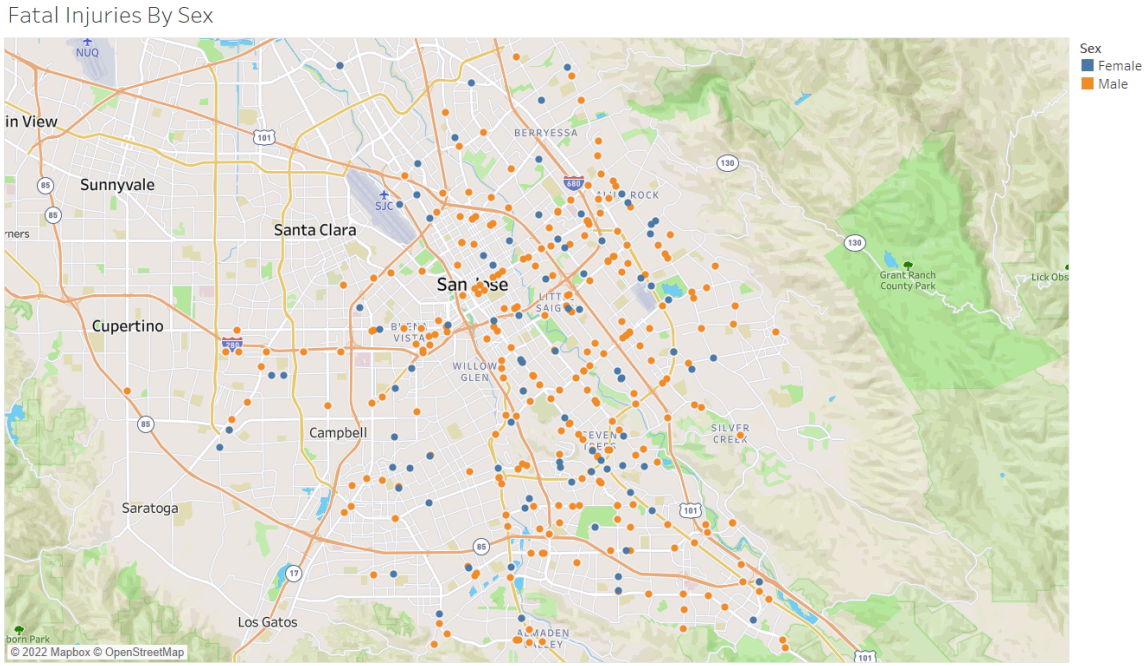


Figure 58

Geomap Depicting Fatal Injuries by Sex



4. Model Development

4.1 Model Proposals

The model proposal will be discussed in detail for Decision Tree, MLP, Random Forest, and XGBoost. All of the models discussed share inputs, outputs, features, training data, validation data, test data, and evaluation metrics.

All of the models listed above will be focused on the target problem of trying to predict the injury severity of a traffic accident in the San Jose, CA, area. The data used for all of the models consisted of a combined dataset that was created during the Data Engineering of this project. The finalized and joined dataset consisted of 41944 records and 309 columns.

4.1.1 Decision Tree

A decision tree is a supervised learning algorithm that is used for classification and regression modeling. Decision trees are top to bottom flowchart, starting at the root node with a specific question of data that leads to branches that hold potential answers followed until the data reaches the leaf node. The group chose DT as it helps understand essential features correlated with the target variable using Decision Rules. This also helps to handle the risk of overfitting in the model. Figure 59 shows the pseudocode for a Decision Tree (Batta, 2019).

Figure 59

Decision Tree Algorithm Pseudocode

```
def
decisionTreeLearning(examples, attributes,
parent_examples):
if len(examples) == 0:
return pluralityValue(parent_examples)
# return most probable answer as there is no training data
left
elif len(attributes) == 0:
return pluralityValue(examples)
elif (all examples classify the same):
return their classification
A = max(attributes, key(a)=importance(a, examples)
# choose the most promissing attribute to condition on
tree = new Tree(root=A)
for value in A.values():
exs = examples[e.A == value]
subtree = decisionTreeLearning(exs, attributes.remove(A),
examples)
# note implementation should probably wrap the trivial case
returns into trees for consistency
tree.addSubtreeAsBranch(subtree, label=(A, value)
return tree
```

Before building the model, the dataset was split into train and test sets in 70:30 ratio using train_test_split under sklearn.model_selection library. Then by keeping the k=5, Stratified k-fold was set up to test on training data and validate by importing StratifiedKFold using sklearn.model_selection library.

After presetting these parameters, the basic decision tree model was built using sklearn.tree DecisionTreeClassifier and keeping the random state as 42 and scoring as F1-macro. To select the model based on a balance between precision and recall, F1-macro is used for accessing the quality of multi-class problems. It aggregates the F1 score for all the classes and gives the accumulated result.

Moving further, with the motivation to improve the accuracy, the decision tree classifier was assigned a train and test set with reduced features using Principal component analysis(PCA)

where 76 features were selected as n_components along with stratified kFold and F1-macro as a parameter. As a result of stratified data, each fold of the dataset contains the same number of observations with a particular label.

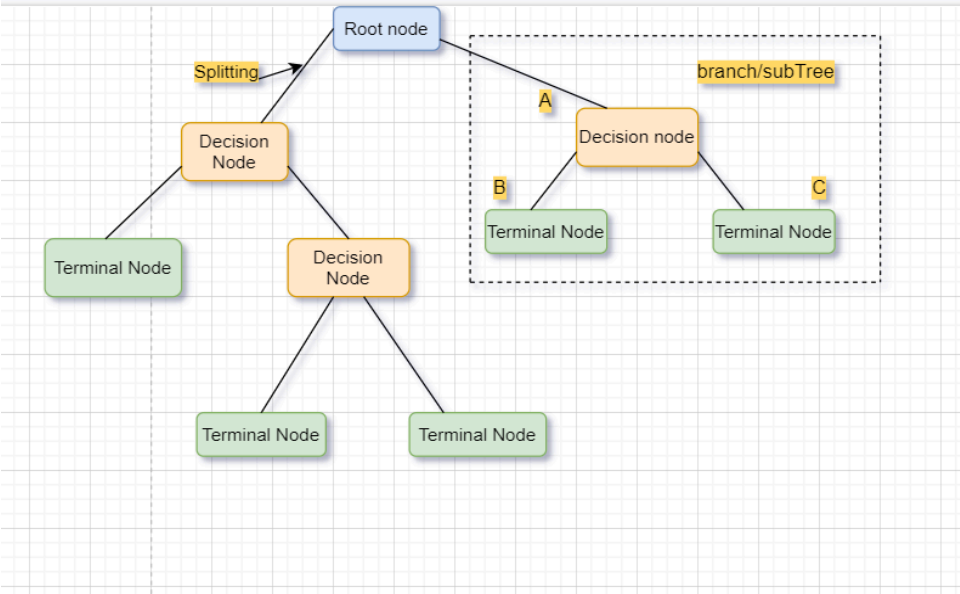
To further tune the decision tree, the Synthetic Minority Oversampling Technique (SMOTE) uses imblearn.over_sampling library was performed in order to assign balanced data in all the classes of the dataset. This is the oversampling technique mainly used for efficiently using the imbalanced classes. This is important because imbalanced data is sometimes confusing for the model while predicting. The Decision Tree classifier was assigned with parameter F1-macro and stratified cross-validation along with fit resampled train and test set dataset using smote.

Finally, in order to get the best-performing decision tree, a randomized grid search was performed with the hyper parameters max_leaf_nodes, min_sample_splits, StratifiedKFold keeping the split as five and scoring as F1-macro. A randomized grid search is a technique that utilizes random combinations of hyperparameters to find the best solution for the built model. This is similar to grid search but yet yields better results.

The way that the Decision tree classifies the data is by sorting them from the root node to the terminal nodes and then finally reaching the leaf node of the tree, which is the last node of each subtree. Tree nodes represent test cases for different attributes, and edges descending from them represent possible answers to those test cases. Each subtree rooted at a new node goes through this recursive process. Figure 60 describes the architecture of a DT.

Figure 60

Decision Tree Architecture



4.1.2 Multi Layer Perceptron

The last model are implemented is a Multi-Layer Perceptron. An MLP is a basic Neural Network, it is a web of connected neurons that learn data by passing it through the model repeatedly until the model learns everything it can. The first part to consider is the forward pass, this is where the data is introduced to the model, then it passes through every neuron, multiplying by the weights and adding the bias terms before passing through the activation function. The activation function changes the line space of the data, it essentially gives the model the ability to predict non-linear data. The next step is the loss calculations; the model makes predictions on the input data and then matches it against the label for the data; the further it is away, the more the loss function penalizes the choice. This brings one to the next step, the backward pass. The calculated loss, in combination with the optimization algorithm, is backpropagated to each set of weights and biases, updating them in a way that would minimize the loss the next time that same data is passed through the model. This is then repeated until human intervention or predefined

rules are based on unchanging loss, overfitting, time constraints, or monetary constraints (Hammerstrom, 1993).

There are an incredible number of hyperparameters that can be tuned in an MLP. To name a few of the most important ones: activation function(s), optimizer function, loss function, number of layers, number of neurons per layer, type of normalization, dropout, and learning rate. (Shams, 2022b) To begin, the optimizer function and the loss function work in conjunction to update the weights and biases of the model through backpropagation. The optimizer's goal is to help the model minimize the loss function. (Hammerstrom, 1993) The number of layers and the number of hidden units per layer are not the same things, but they can be summarized in a similar way; the bigger they are, the more complicated the network is. Having a complicated network can be both a good and bad thing depending on the situation. If the relationships or data are simple, then having a too complicated network will lead to overfitting issues, while if the data or relationships are complex, then a complex network is required. (Shams, 2022b) Normalization is one of the ways to combat overfitting, it adds some amount of generalization to the data. There are different types of normalization, but they all do sort of the same thing; they augment the data in a way to make it easier for the model to understand, but the goal of it is to retain relational information. (Shams, 2022b) Dropout is another one of the ways to combat overfitting in an MLP. Dropout is a technique that randomly turns off a percentage of the neurons in a layer. The idea with dropout is it implements an element of randomness in the model because the dropped perceptrons don’t have a pattern or reason for being dropped; it makes the loss optimization less specific to the data it is being trained on. This comes at the cost of learning; adding dropout takes more epochs to train the model but comes with the benefit of reducing overfitting through generalization. (Shams, 2022b) Finally, there is the learning rate;

this is an important hyperparameter and can be tuned endlessly. It is an incredibly important part of the MLP as it controls exactly how quickly the model is fitted to the data; too quickly, the model cannot optimize properly, and too slowly, the model will never learn enough for predictions. (Shams, 2022a) Modern deep learning packages, such as Keras and TensorFlow, which were used in this project, can dynamically tune the learning rate better than any human can, so this was left to the computer's devices on this model.

4.1.3 Random Forest

The following are equations to assist with the understanding of the Random Forest algorithm. Due to this paper's classifier being a classification problem, the equations will focus on the Random Forest aspect of classification. Figure 61 below is the classification rule for the algorithm conducted by Biau, G. (2016).

Figure 61

Classification Rule Equation for Random Forest

$$L(m_n) = \mathbb{P}[m_n(\mathbf{X}) \neq Y] \underset{n\to\infty}{\to} L^\star,$$

For classification, the algorithm must then conduct a majority vote to decide the classification. The equation for this can be seen in the below in Figure 62 once more performed by Biau, G. (2016).

Figure 62

Classification Majority Vote Equation for Random Forest

$$m_{M,n}(\mathbf{x}; \Theta_1, \ldots, \Theta_M, \mathcal{D}_n) = \begin{cases} 1 & \text{if } \frac{1}{M}\sum_{j=1}^{M} m_n(\mathbf{x}; \Theta_j, \mathcal{D}_n) > 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the algorithm resamples the data, and the equation for this process is shown below as performed by Biau, G. (2016) in Figure 63.

Figure 63

Random Forest Resampling Equation

$$L_{\text{class},n}(j,z) = p_{0,n}(A)p_{1,n}(A) - \frac{N_n(A_L)}{N_n(A)} \times p_{0,n}(A_L)p_{1,n}(A_L) - \frac{N_n(A_R)}{N_n(A)} \times p_{0,n}(A_R)p_{1,n}(A_R).$$

The actual pseudocode for the process depicted in the equations above can be found in the Figure 64, demonstrated by Biau, G. (2016). The figure walks through each process step for how the Random Forest classifier works.

Figure 64

Random Forest Pseudocode

Algorithm 1: Breiman's random forest predicted value at **x**.

Input: Training set $\mathcal{D}_n$, number of trees $M > 0$, $a_n \in \{1, \ldots, n\}$, `mtry` $\in \{1, \ldots, p\}$, `nodesize` $\in \{1, \ldots, a_n\}$, and $\mathbf{x} \in \mathcal{X}$.
Output: Prediction of the random forest at **x**.

for $j = 1, \ldots, M$ **do**
  Select a_n points, with (or without) replacement, uniformly in $\mathcal{D}_n$. In the following steps, only these a_n observations are used.
  Set $\mathcal{P} = (\mathcal{X})$ the list containing the cell associated with the root of the tree.
  Set $\mathcal{P}_{\text{final}} = \emptyset$ an empty list.
  while $\mathcal{P} \neq \emptyset$ **do**
    Let A be the first element of $\mathcal{P}$.
    if *A contains less than* `nodesize` *points or if all* $\mathbf{X}_i \in A$ *are equal* **then**
      Remove the cell A from the list $\mathcal{P}$.
      $\mathcal{P}_{\text{final}} \leftarrow Concatenate(\mathcal{P}_{\text{final}}, A)$.
    else
      Select uniformly, without replacement, a subset $\mathcal{M}_{\text{try}} \subset \{1, \ldots, p\}$ of cardinality `mtry`.
      Select the best split in A by optimizing the CART-split criterion along the coordinates in $\mathcal{M}_{\text{try}}$ *(see text for details)*.
      Cut the cell A according to the best split. Call A_L and A_R the two resulting cells.
      Remove the cell A from the list $\mathcal{P}$.
      $\mathcal{P} \leftarrow Concatenate(\mathcal{P}, A_L, A_R)$.
    end
  end
  Compute the predicted value $m_n(\mathbf{x}; \Theta_j, \mathcal{D}_n)$ at **x** equal to the average of the Y_i falling in the cell of **x** in partition $\mathcal{P}_{\text{final}}$.
end
Compute the random forest estimate $m_{M,n}(\mathbf{x}; \Theta_1, \ldots, \Theta_M, \mathcal{D}_n)$ at the query point **x** according to (1).

Random Forest can be optimized. In many ways, the optimization of a Random Forest resembles that of a Decision Tree because Random Forest is an ensemble of trees (Géron, 2019, p. 197). However, they differ because Random Forest utilizes a bagging-like technique. This means that Random Forest has all of the parameters that a decision tree has and can be tuned on the number of trees that it has (n_estimators), the maximum depth of a tree (max_depth), the minimum number of samples necessary to split a node (min_samples_split), the minimum number of sample to be a leaf node (min_samples_leaf), and the number of features to look to consider (max_features), (Géron, 2019, p. 197). In addition, random Forest is similar to bagging because it allows the ability to implement re-sampling of the dataset, otherwise known as bootstrapping. This combination of tuning like a decision tree and having bootstrap capabilities make for a highly tunable model because of its extra randomness when growing trees (Géron, 2019, p. 197). These optimization techniques equate to a model that provides higher bias and a lower variance, which typically results in a better-performing model (Géron, 2019, p. 198).

For the model discussed in this paper tuning the Random Forest significantly helped with overfitting issues and resulted in a better F1-macro score. The approach to tuning this model was initially made with a randomized search of provided parameters to help identify which parameters are best for the Random Forest. Then, with the result of the randomized search, a grid search was performed to fine-tune the Random Forest. This final tuning process resulted in the F1-macro score being improved by 0.08 and the AUC improvement of 0.05. Figure 65 shows the exact hyper-parameters used to obtain the performance gain.

Figure 65

Random Forest Hyper-Parameters

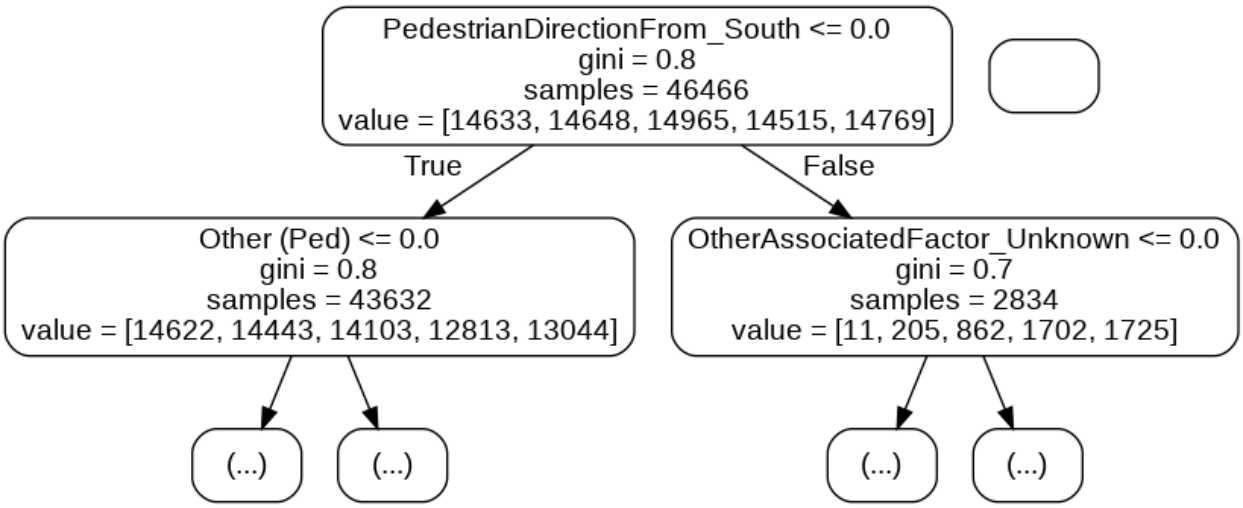
```
(n_estimators= 101, min_samples_split= 2, min_samples_leaf= 3, max_features= 'auto', max_depth= 80, bootstrap=True, random_state=42)
```

The tuning resulted in a model that could better classify each target class based on the metric F1, which is the harmonic mean of precision and recall (Vanderplas & VanderPlas, 2016, p. 476). In theory, hyperparameter tuning produces the best performance a model can yield, which is demonstrated by the performance gains for this Random Forest.

The model architecture of Random Forest performs similarly to a decision tree. Figure 66 depicts a root node for one of the trees used for the Random Forest model.

Figure 66

Random Forest Single Tree Node View



The Random Forest architecture makes decisions based on a Gini index and decides how to navigate the data (Géron, 2019, p. 195). Visualizing the Random Forest's total size would be nearly impossible due to the sheer size and complexity of the number of trees and tree depth. Figure 67 provides a higher-level visual representation of the complexity of a single tree. This visual is only showing a single tree with a depth of five.

Figure 67

Random Forest Single Tree with Depth of Five

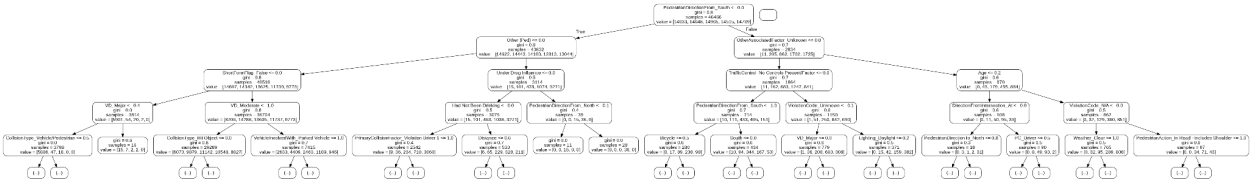


Figure 68

Random Forest Design Diagram

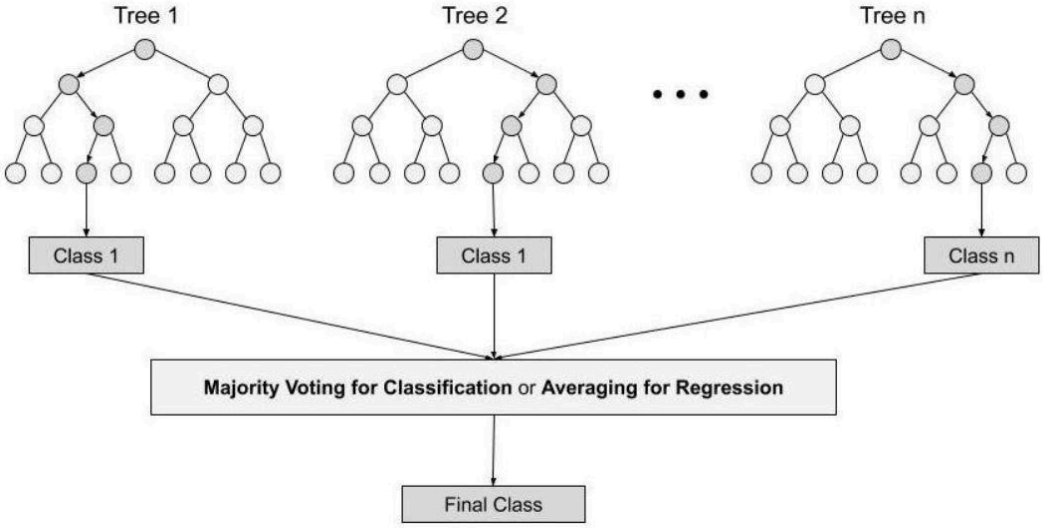


Figure 68 provided by R (2022) above does a great job of visualizing the architecture of a Random Forest. It shows the multiple decision trees in parallel and how they output a classification, and that classification is then voted on for a classification based on a majority vote.

4.1.4 XGBoost

XGBoost, also known as eXertme Gradient Boosting, is a popular supervised machine learning technique. It is a gradient boosting-based ensemble machine learning algorithm. Chen and Guestrin (2016) discussed how efficient and scalable a tree boosting technique is in their

article. They then discussed the factors that make XGBoost a highly scalable method. These include sparse data management, parallel and distributed computing, and out-of-core computation. Being an ensemble learning algorithm, the XGBoost model makes a prediction by combining the results of a number of models. Unlike Random Forest (RF), which uses bagging of decision trees, XGBoost uses boosting of decision trees to make predictions. As an extension of boosting, gradient boosting formalizes the process of additively creating weak learners or base learners as a gradient descent method over an objective function. Gradient boosting establishes desirable results for the following model to reduce errors. The gradient of the error with respect to the prediction determines the targeted outcomes for each case. The fundamental concept of this technique is to build the new base learners to have a maximum correlation with the ensemble's overall negative gradient of the loss function (Natekin & Knoll, 2013).

The XGBoost python package was initially needed to implement the XGBoost algorithm. Given that XGBoost can carry out both classification and regression, XGBClassifier() was initialized from this library. Some important parameters that were considered for this model are n_estimators, booster, max_depth, alpha, lambda, and learning_rate. One can see the hyperparameters initialized for the XGBoost model in Table 13. An effort was made to adjust the hyperparameters in a manner that would minimize overfitting.

Table 13

Hypermeters for XGBoost Model

Hyperparameter	Role	Hyperparameter Value
n_estimators	Count of tress that are boosted	100
booster	Type of booster to be used	gbtrees
max_depth	How many splits a tree can go through	6
alpha	Lasso regularization	0
lambda	Ridge regularization	1
learning_rate	Shrinkage of weights	0.3

4.2 Model Supports

4.2.1 Description of the Platform and Environment

The platform and environment utilized begins with the datasets that were saved as CSV files on a Google Drive. This was done so that the data could be easily accessed without having to repeat the process. The group decided to use Python as the preferred programming language for this project, Google Colaboratory (Colab) was the first pick for an environment in which to write scripts. perform Data analysis, data visualization, model construction, and evaluate all of the constructed models. Google Colab allows all of these steps to be done in a cloud-based Python environment. Python was picked because of its vast library of tools for the task needed.

4.2.2 Tools Used

The tools that were used can be found in Table 14.

Table 14
Model Support Tools Used

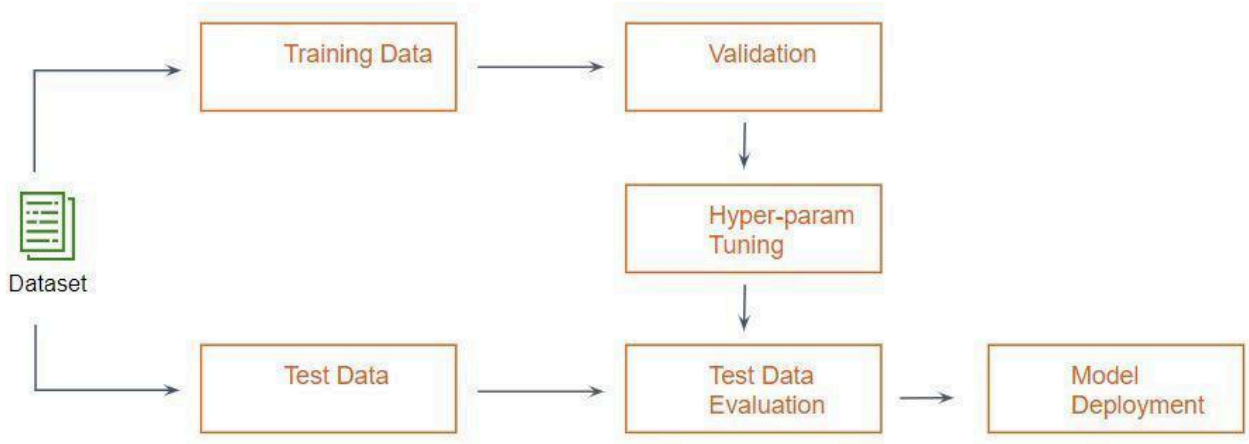
	Library	Method	Usage
Sci-kit Learn	sklearn.model_selection	train_test_split	Split the data into training and test data
	sklearn.model_selection	RandomizedSearchCV	Perform a random search of parameters for tuning
	sklearn.model_selection	GridSearchCV	Perform a grid search of parameters for fine-tuning
	sklearn.metrics	accuracy_score	Metric used for obtaining accuracy of classifier
	sklearn.metrics	f1_score	Metric used for obtaining precision-recall harmonic mean
	sklearn.metrics	confusion_matrix	Metric used for visually evaluating confusion matrix
	sklearn.metrics	classification_report	Metric used for obtaining precision, recall, F1-macro
	sklearn.metrics	roc_auc_score	Metric used for obtaining the AUC score
	sklearn.metrics	roc_curve	Metric used for visually evaluating the ROC curve
	sklearn.ensemble	RandomForestClassifier	Used for constructing Random Forest model
	sklearn.model_selection	StratifiedKFold	Performing Stratified Kfold for the validation process
	sklearn.model_selection	cross_val_score	Metric to evaluate the validation process
	sklearn.tree	DecisionTreeClassifier	Used for constructing Decision Tree model
	sklearn.decomposition	PCA	To perform PCA
XGBoost	xgboost	XGBClassifier	Used for constructing XGBoost model
Matplotlib	matplotlib.pyplot	-	Data Visualization
Seaborn	seaborn	-	Data Visualization

	Library	Method	Usage
Keras	keras	-	Deep learning model construction
	keras.callbacks	Callback	Perform early stopping
	keras.layers	BatchNormalization	Layer normalizes its inputs
	keras.utils.vis_utils	plot_model	Converts a Keras model to dot format
Imblearn	imblearn.over_sampling	SMOTE	Used to oversample the non-majority target classes
Pandas	pandas	-	Dataframe manipulations
Numpy	Numpy	-	Data calculations and manipulations

Figure 69 shows the data flow diagram for the process of this project.

Figure 69

Data Flow Diagram



4.3 Model Comparison and Justification

4.3.1 Decision Tree

Decision Trees are an older machine-learning approach upon which many new algorithms base their success. For example, Random Forest and XGboost both use a form of a decision tree (Chen & Guestrin, 2016, p. 782). The Decision Tree architecture is based on the Classification and Regression Tree (CART) algorithm, which splits the training data into subsets based on a feature and threshold (Géron, 2019, p. 244). The Decision Tree can then make a prediction for which class is best. Decision Trees can handle both classification and regression problems. Due to the algorithm used, the Decision Tree can handle large and complex data with ease (Géron, 2019, p. 244). However, issues of overfitting are known for Decision Trees. The model does not make assumptions about the data and can "memorize" the data, which results in overfitting (Géron, 2019, p. 246). Restricting the tree's depth is necessary to prevent overfitting. Overall, Decision Trees are still a competent machine-learning approach but have limitations, such as being susceptible to overfitting and suffering from training set rotations (Géron, 2019, p. 247).

4.3.2 Multi Layer Perceptron

MLP was chosen because of its robustness for performing various tasks. The kind of data used for training these models have no spatial or temporal information, nor are they images; they are just simple CSV samples. For those reasons, of any kind of Neural Network, an MLP makes some of the most sense. An MLP is like a Swiss army knife, it can handle most tasks, but it isn’t particularly the correct tool for any tasks; it is the jack of all trades and the master of none. (Shams, 2022b With that being said, when there aren’t any more specialized or niche networks for the style of data being used, then an MLP is the correct choice; that is the case here, so that is what is used. While other non-Neural Network-based models may produce good results very

efficiently, a Neural Network is all but guaranteed to produce at least decent results. (Shams, 2022b) An MLP was chosen because of the high likelihood of producing solid results based on how the network works and through the literature review of the model.

4.3.3 Random Forest

Random Forest functions similarly to a decision tree because it's an ensemble learner built upon decision trees (Vanderplas & VanderPlas, 2016, p.421). This architecture equates to a very intuitive understanding of how the model functions. The model asks a series of questions to identify a classification (Vanderplas & VanderPlas, 2016, 421). Random Forest can handle classification and regression problems. It performs exceptionally well on large data sets and sparse data. Sparse data can be challenging to many classifiers, but Random Forest handles it with minimal issues (Vanderplas & VanderPlas, 2016, 422). Unlike decision trees that have issues of "memorizing" the data with large depths of trees, Random Forest has the benefit of overcoming this problem by combining multiple overfitting trees to reduce the effect (Vanderplas & VanderPlas, 2016, 425). This provides Random Forest with its bagging-type technique because of its parallel estimators (Vanderplas & VanderPlas, 2016, 426). Random Forest does not suffer as much as other models from the need to have data preprocessing done exhaustively. Its ensemble approach overcomes missing data, is not as susceptible to outliers, and handles imbalanced data very well (Vanderplas & VanderPlas, 2016, 424). The training and prediction time is relatively speedy compared to the models tested for this project due to the simplicity of the underlying decision tree-type approach. Random Forest's parallelization allows for each tree to run simultaneously, which speeds up the process (Vanderplas & VanderPlas, 2016, 432).

Additionally, Random Forest handles imbalanced multi-class classification problems exceptionally well because of its sampling with the replacement of data and sampling of features

(Vanderplas & VanderPlas, 2016, 426). It's also very flexible and can outperform other models that might be underfitting (Vanderplas & VanderPlas, 2016, 432). The only limitation is that Random Forests can be challenging to interpret and understand if trying to analyze how the classifier came to its conclusion.

4.3.4 XGBoost

XGBoost is an architecture that utilizes "gradient boosted decision trees that tends to win a lot of Kaggle-style machine learning competitions" (Vanderplas & VanderPlas, 2016, 220). XGBoost works by implementing parallelization of gradient descent that produces a convex loss and measures prediction against the target with a penalty (Chen & Guestrin, 2016, p. 786). It utilizes a novel tree-learning algorithm that uses a sparsity-aware split-finding approach that allows for the exceptional handling of sparse data (Chen & Guestrin, 2016, p. 785). XGBoost implements a regularized model, directly reducing overfitting issues by tackling the problem similar to that of a greedy forest (Chen & Guestrin, 2016, p. 791). Like Random Forest, XGBoost also implements feature sampling to help with overfitting issues. Minimal pre-processing is needed since trees are better equipped to handle continuous features but are resistant to outliers and missing data. XGBoost excels at training time. It's touted as being "more than ten times faster than existing popular solutions on a single machine" (Chen & Guestrin, 2016, p. 785). This was true during the project's implementation, as it trained exceptionally fast compared to other models. The exact greedy algorithm for finding the best tree split is computationally expensive as it has to go through all of the possible splits, but this is still relatively low cost compared to other models (Chen & Guestrin, 2016, p. 787).

There is a reason that XGBoost wins so many machine learning competitions. It simply performs well and has many strengths. It is capable of handling large datasets with missing data

and outliers. Plus, its execution time is fast. Perhaps, a limitation for XGBoost is that it can be challenging to tune because it has many hyper-parameters.

4.4 Model Evaluation Methods

The primary metric used to evaluate the models on the validation data was F1-macro. The reason that F1-macro was chosen is that it is the average F1-score for each target class. The problem presented in this paper is a multi-class classification that contains five different target classes. The objective was to ensure that the models performed well on every target class, not just one. So to ensure that F1-macro was utilized. F1-macro works by adding up each target class F1-score and returning the average F1-score. F1-score is vital because it is a metric that displays the harmonic mean of precision and recall (Géron, 2019, p. 141). It would be incorrect to use accuracy as the primary metric for this problem because accuracy fails to identify if the model performs well on each target class. The equation for F1 is provided in Figure 70 by Géron (2019).

Figure 70

Equation for F1 Measure

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

The metrics used to evaluate the models on test data were more comprehensive. The test data was evaluated with the metrics F1-macro, accuracy, confusion matrix, receiver operating characteristic (ROC) curve, and the area under the curve (AUC) score. The equation is provided in Figure 71 by Kelleher et al. (2015, p.465).

Figure 71

Equation for Classification Accuracy

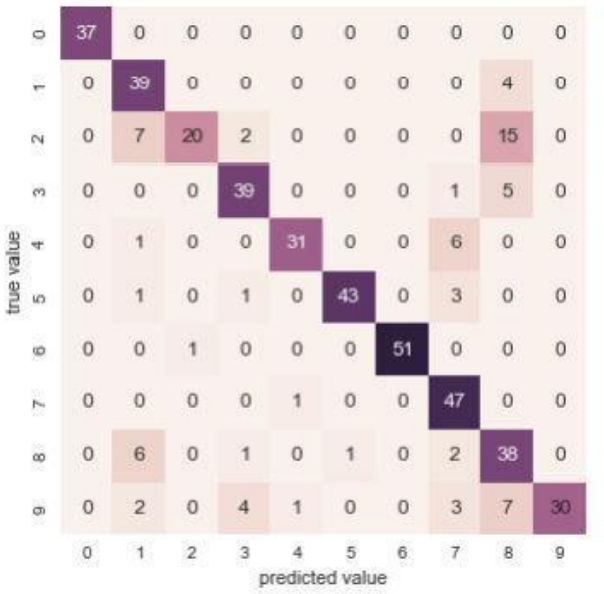
$$classification\ accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Accuracy is used for classification problems. It gives a general idea of how well the model performs in correctly classifying (Kelleher et al., 2015, p. 465).

The confusion matrix is a great way to evaluate a model's performance visually. The idea is to count the number of instances that the model incorrectly labeled a prediction (Géron, 2019, p. 138). When looking at a confusion matrix, the rows are actual classes, and the columns represent predicted classes (Géron, 2019, p. 139). The confusion matrix example in Figure 72 is provided by Vanderplas & VanderPlas (2016, p. 357).

Figure 72

Example of Confusion Matrix

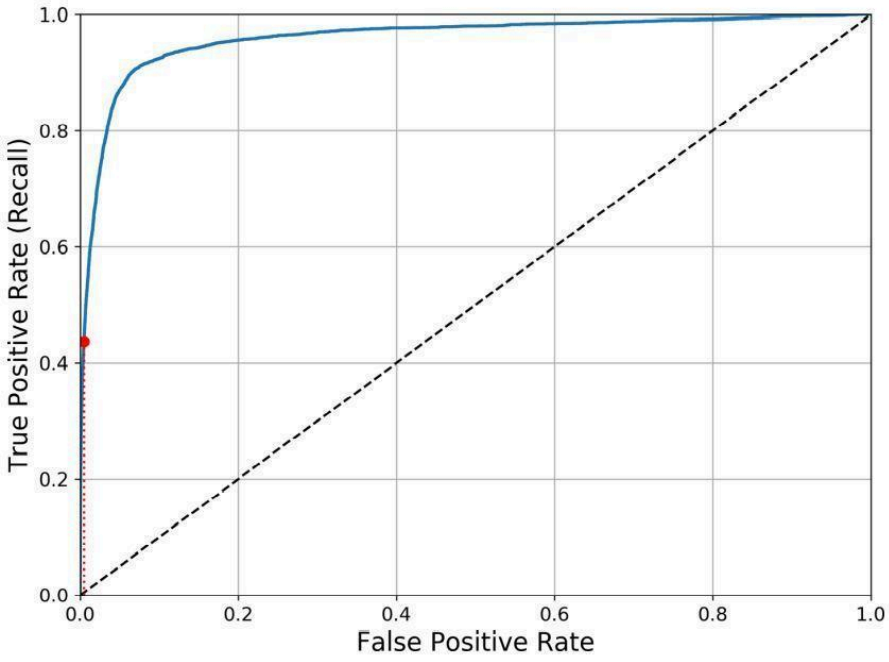


The figure above shows how each value is correctly predicted in the horizontal purple line. The values outside that horizontal line are incorrect predictions made by the model. This visual allows a quick way to identify which value or classes the model has issues with correctly predicting. For example, in the above example, if one were to look at the true value two, it can be seen that the model is having issues predicting that value with incorrect predictions of value 8 with 15 such instances.

The ROC curve is similar to F1, but it plots the true positive rate against the false positive rate (Géron, 2019, p. 146). Another way of stating it is that ROC plots the sensitivity vs. the model's specificity. The way to read a ROC curve is that the dotted line is the performance of a random classifier and the plots of the model as it performs better move closer to the top left corner (Géron, 2019, p. 147). The example of the ROC curve in Figure 73 is provided by Géron (2019).

Figure 73

Example of ROC Curve



The AUC score is computed by the area underneath the ROC curve. The equation can be found below in Figure 74. For example, in the figure above, the AUC score would be computed for all of the area that is below and to the right of the blue line. This metric allows for models to quickly be compared against one another for performance (Géron, 2019, p. 148). A perfect score for AUC would be 1.0, and 0 would be the worst (Géron, 2019, p. 148). The equation below for computing AUC is provided by Géron (2019).

Figure 74

Equation for Computing AUC

$$\sum_{i=2}^{|\mathbf{T}|} \frac{(FPR(\mathbf{T}[i]) - FPR(\mathbf{T}[i-1])) \times (TPR(\mathbf{T}[i]) + TPR(\mathbf{T}[i-1]))}{2}$$

4.5 Model Validation and Evaluation Results

All of the models discussed in this section were trained, validated, and tested on the same exact data. This process was done to ensure that the results were valid and consistent for all of the models built.

4.5.1 Decision Tree

Initially, the base model decision tree didn't perform well, and later having tuned yielded better results. For model building, the dataset was split into a Train set and a Test set at 70% and 30%. The Evaluation of the base decision tree, decision tree with Smote, and best hyper tuning parameters with Smote are shown in Figure 75, Figure 76, and Figure 77 below.

Figure 75

Classification Report for Base Decision Tree

	precision	recall	f1-score	support
0	0.77	0.75	0.76	6293
1	0.56	0.58	0.57	4315
2	0.25	0.26	0.26	1434
3	0.14	0.14	0.14	309
4	0.16	0.14	0.15	76
accuracy			0.61	12427
macro avg	0.38	0.37	0.38	12427
weighted avg	0.62	0.61	0.62	12427

Figure 76

Classification Report for Decision Tree Model with SMOTE

	precision	recall	f1-score	support
0	0.77	0.75	0.76	6293
1	0.58	0.57	0.57	4315
2	0.28	0.31	0.30	1434
3	0.15	0.17	0.16	309
4	0.21	0.21	0.21	76
accuracy			0.62	12427
macro avg	0.40	0.40	0.40	12427
weighted avg	0.63	0.62	0.62	12427

Figure 77

Classification Report for Decision Tree with Hyperparameter Tuning

	precision	recall	f1-score	support
0	0.82	0.78	0.80	6293
1	0.67	0.61	0.64	4315
2	0.36	0.28	0.31	1434
3	0.11	0.41	0.17	309
4	0.11	0.38	0.17	76
accuracy			0.65	12427
macro avg	0.42	0.49	0.42	12427
weighted avg	0.70	0.65	0.67	12427

The resulting figure shows that classification report with hyper tuning parameters yielded the best results for decision tree with hyper tuning parameters using randomized grid search with

max_leaf_node as 97, min_samples_split': [2, 3, 4], StratifiedKFold where k=5 and keeping the random state as 42.

As discussed above, the evaluation measures for Decision Tree gave the results for an accuracy score of 0.65, ROC- AUC resulted in 0.83, and F1-macro as 0.41 for the tuned model. The below Figure 78 illustrates the confusion matrix, and Figure 79 illustrates the ROC-AUC curve for the decision tree.

Figure 78

Decision Tree Confusion Matrix on Test Data

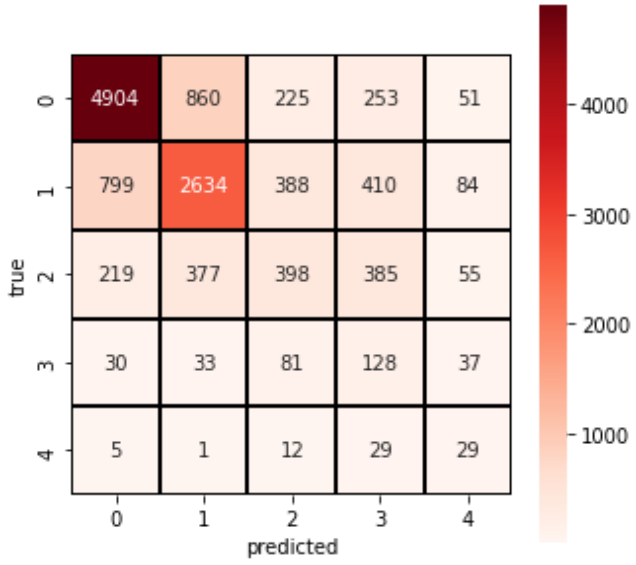
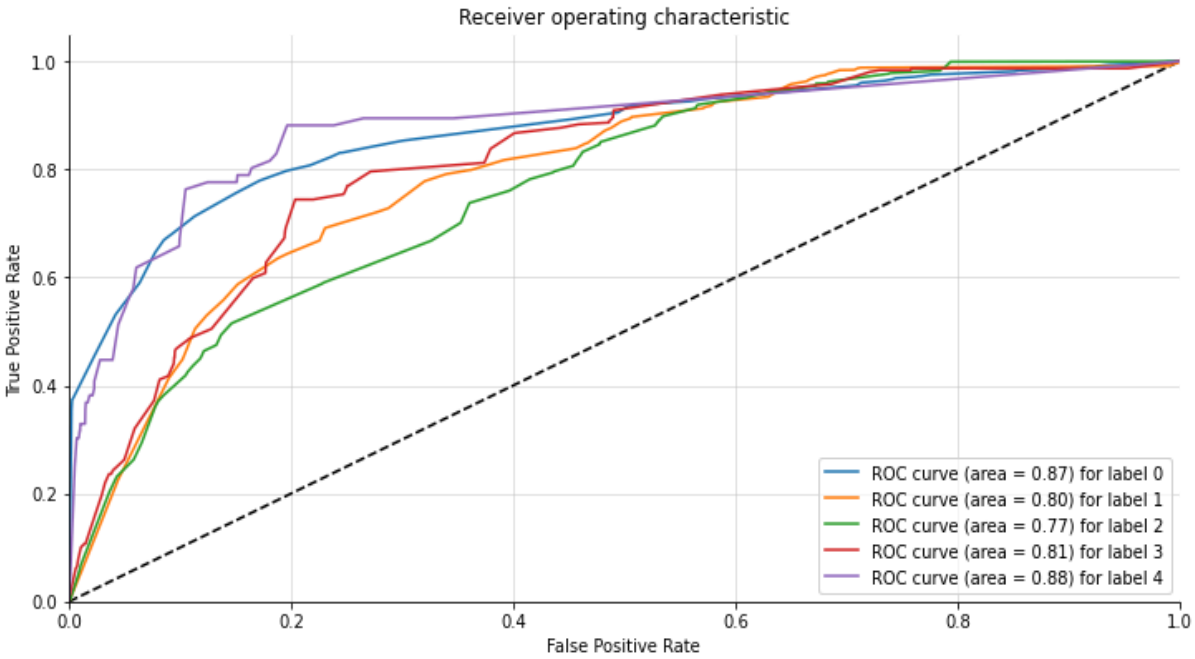


Figure 79

Decision Tree ROC Curve on Test Data



Finally, figure 80 shows the best-performing Decision Tree evaluated on test data.

Figure 80

Decision Tree F1-Macro, AUC Score, and Accuracy on Test Data

	Test Data		
Model	**F1-Macro**	**AUC**	**accuracy**
DT	0.4198	0.83	0.6512

In the decision tree, F1 score, precision, and recall for class label zero depicted the highest result for the No Injuries label that has been denoted as zero for the classification. The ROC curve for the decision tree showed the opposite result of the validation scores. The model got a better AUC for the label Fatal Injuries, which is denoted as four for the classifier, and the poorest performance was observed by the zero labels. This is concluded by looking into the area

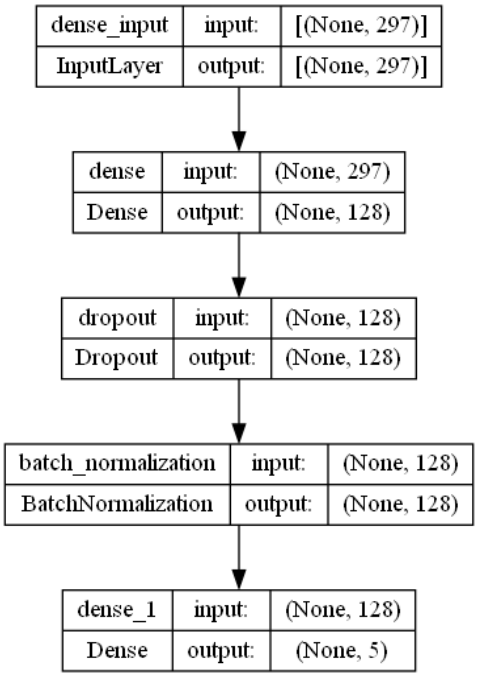
under the curve of ROC for the decision tree. This result was mainly obtained by handling the imbalanced target classes using Smote technique. Furthermore, the best results were also achieved by hyper tuning the parameters for the decision tree based on a randomized search.

4.5.2 Multi Layer Perceptron

The MLP model, as most models begin, didn't perform particularly well, it took a significant amount of time in tuning to get it performing as well as it did. This was performed by implementing the goal metric for this project the F1-macro score into the Keras model. For each epoch the data ran, it printed the F1-macro score. What this did, was determine how well the model was performing during training. Because the model didn't have to be fully trained to determine how well it was performing on corrections made, this cut a significant amount of time out of the tuning process. The model went through an enormous number of iterations before its final structure was determined and set. The final layer structure for the model can be seen below in Figure 81

Figure 81

MLP Model Layer Structure



The model had to be checked for overfitting, this was done by plotting the validation and test accuracy and loss. One can see in Figure 82 and Figure 83 there isn't a significant amount of overfitting likely due to the normalization steps and included dropout in the model. These were added as a reactionary measure because the model previously did have significant overfitting issues.

Figure 82

MLP Accuracy of Validation Versus Training Sets

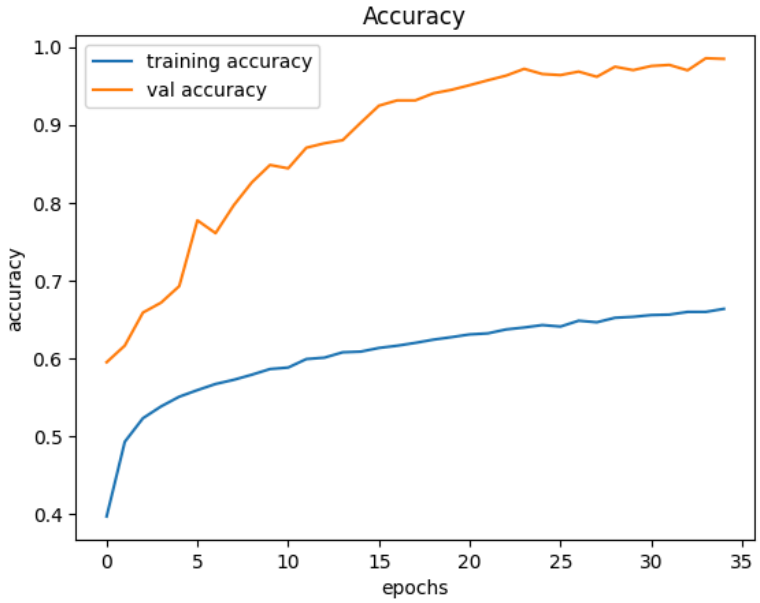


Figure 83

MLP Loss of Validation Versus Training Sets

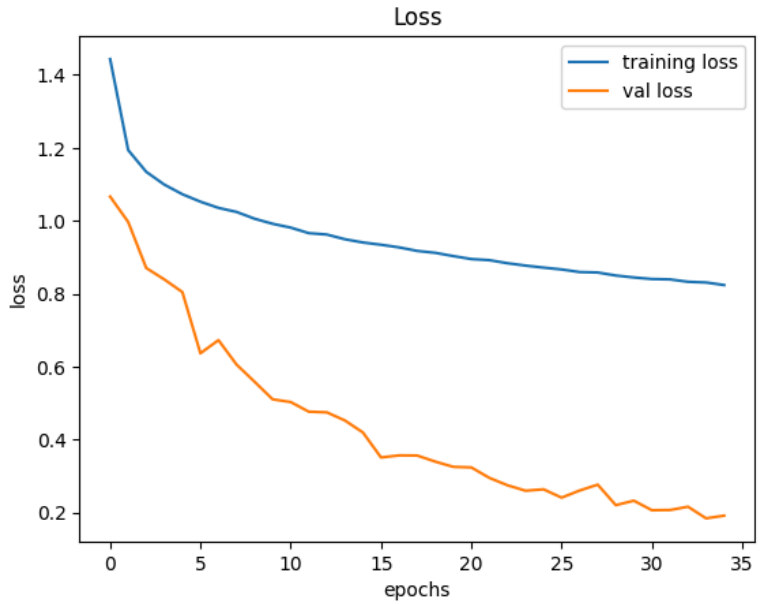


Figure 84

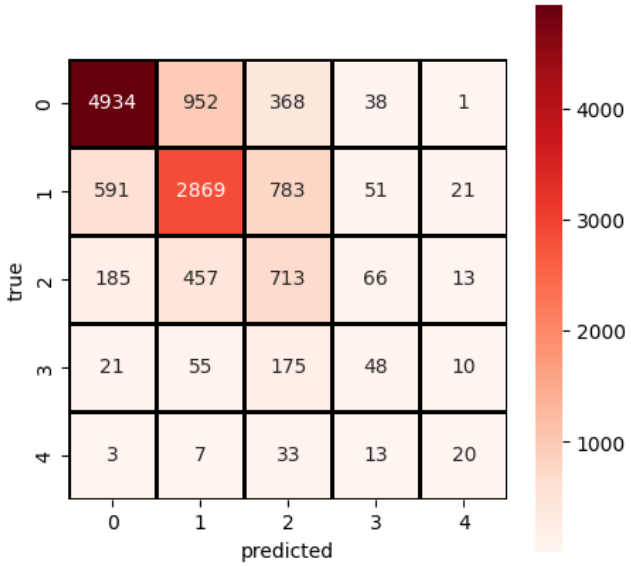
MLP F1-Macro, AUC Score, and Accuracy on Test Data

	Test Data		
Model	**F1-Macro**	**AUC**	**Accuracy**
MLP	0.4713	0.85	.6908

The results of the F1-macro score were promising. Given the number of classes and the incredible disparity between the number of samples for the major and minor classes, a .47 F1-macro was a strong results. It did not however turn out to be the strongest model in this paper. Figure 85 shows the confusion matrix for MLP.

Figure 85

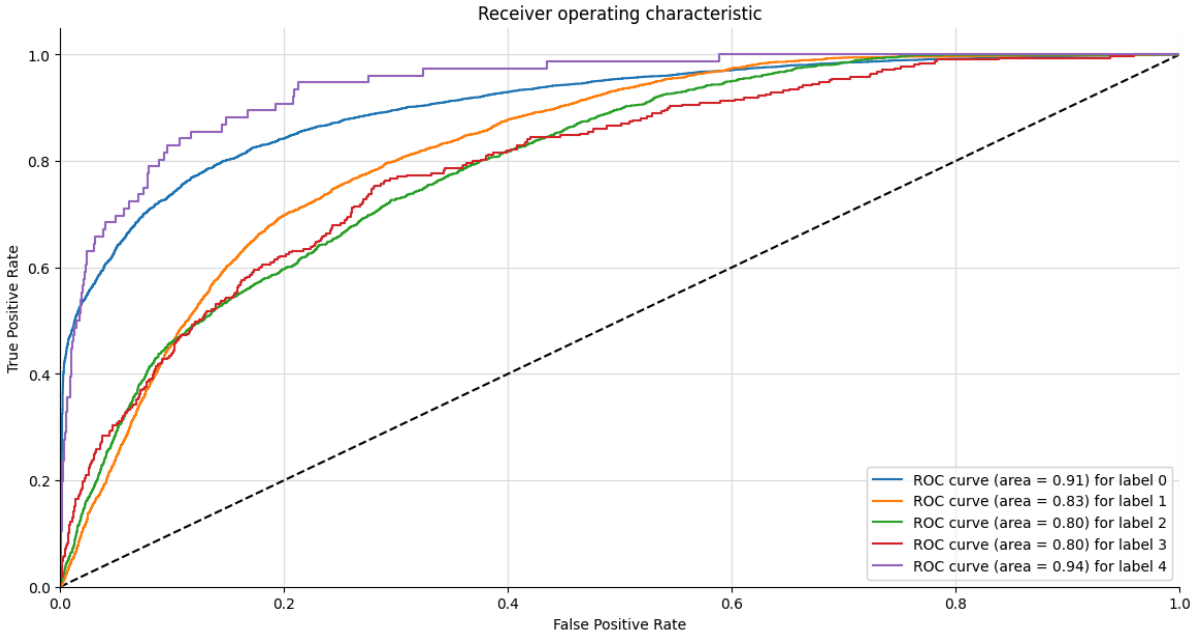
MLP Confusion Matrix on Test Data



Based on this confusion matrix, the model is still struggling to classify the minor classes 2,3,4 significantly, this was the best the model could perform, but better results in this area would have been much preferred.

Figure 86

MLP ROC Curve on Test Data



The ROC curves told the group much the same as the confusion matrix, the model, despite being tuned for the minor classes, still underperformed in those classes. This is very much expected with the significant disparity in the number of samples.

4.5.3 Random Forest

The model Random Forest was initially evaluated on validation data using F1-macro. Figure 87 below shows its performance on the validation data.

Figure 87

Random Forest Initial Performance on Validation Data

```
[0.80300184 0.8602846  0.86510792 0.87864972 0.87246045]
0.8559009057259608
```

The model generalizes fairly well along all five training folds but somewhat underperforms on the first fold. This could be due to an overfitting issue the model has with

generalizing. To remedy this, the model was hyper-parameter-tuned. Upon completion of tuning, the model was re-evaluated on the validation data. Figure 88 below shows the performance.

Figure 88

Random Forest Tuned Performance on Validation Data

```
[0.8324969  0.89107219 0.89600538 0.91832212 0.926132  ]
0.8928057188181647
```

At this point, the model is ready to be evaluated on test data. The model had 12,427 rows of test data to be evaluated upon. Figure 89 shows its F1-macro, AUC score, accuracy, and figure 90 shows its confusion matrix. Finally, Figure 91 shows the ROC curve performance on the test data.

Figure 89

Random Forest F1-Macro, AUC Score, and Accuracy on Test Data

	Test Data		
Model	**F1-Macro**	**AUC**	**Accuracy**
RF	0.4795	0.88	0.7054

Figure 90

Random Forest Confusion Matrix on Test Data

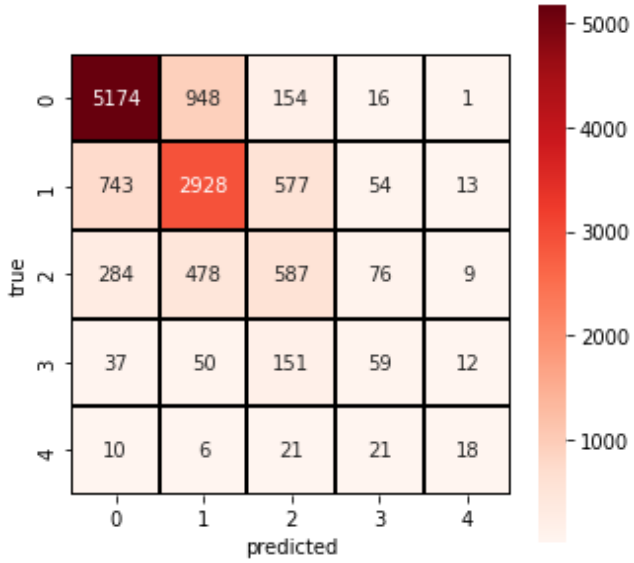
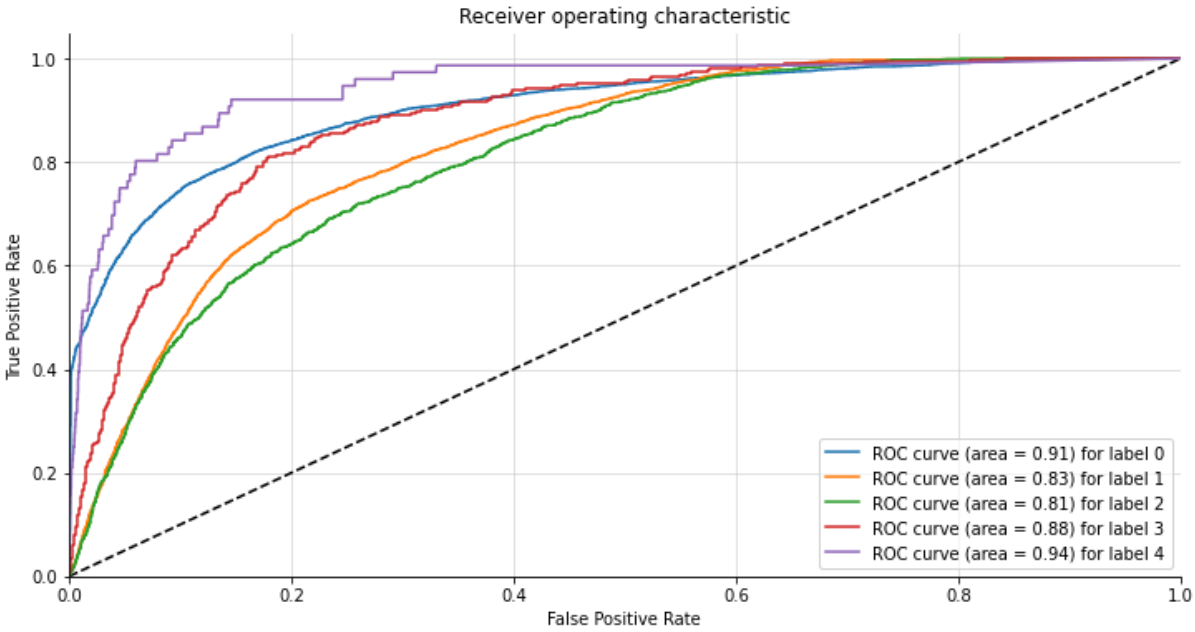


Figure 91

Random Forest ROC Curve on Test Data



Based on the confusion matrix, it is apparent that the model struggles to classify classes three and four correctly. This is most likely because the number of training examples that existed

before SMOTE was 681 and 185, respectively. This means that the classes were synthetically oversampled to 14,706. The model still struggled to classify these classes correctly.

4.5.4 XGBoost

Three models were produced using the XGBoost classifier. To prevent overfitting of the training dataset, these were assessed using the stratified kfold cross-validation method. All the features were present in the initial model. The data from PCA were fed to the following model, and the data from SMOTE were input to the final model. Initially, the training data were fed to the model and evaluated. Table 15 compares the results of the evaluation metrics, F1-macro, on the training dataset.

Table 15

Stratified Cross Validation Scores Based on F1-Macro

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
XGB - Preprocessed Features	0.411	0.364	0.394	0.388	0.407	0.393
XGB - PCA	0.349	0.340	0.347	0.348	0.351	0.347
XGB - SMOTE	0.644	0.686	0.687	0.692	0.681	0.678

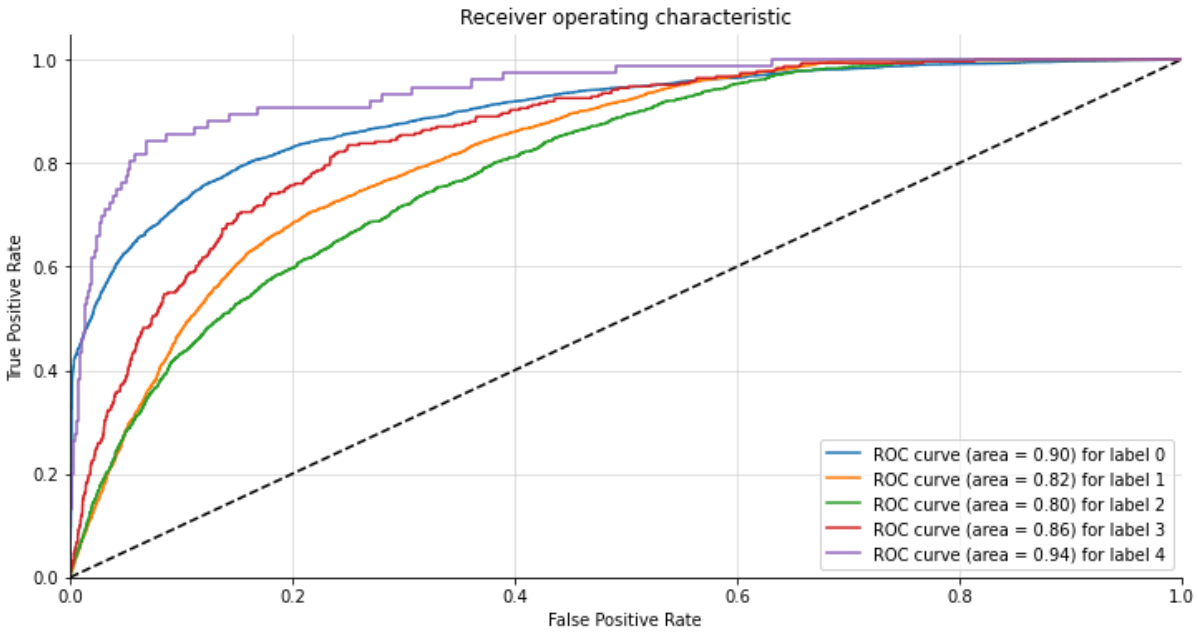
When trained on SMOTE data, the XGBoost model achieved the best F1-macro score. This demonstrates how the imbalance of the minority classes caused problems for the other

models. The balance between the minority classes has a positive impact on model performance in this case. The hindsight to these scores is that the SMOTE oversamples the minority classes; this means that the model may have gone through the same data repeatedly, which induces bias and hence the inflated score. To verify the test dataset and provide predictions, the XGBoost model developed on SMOTE was utilized. Instead, the XGBoost model trained on preprocessed data was used.

To better understand how the model was doing, the ROC curve was visualized. The outcomes were pleasing. The ROC curve, which was produced using the Python library matplotlib, is shown in Figure 92. Each of the classes of the target feature had a good ROC curve score which showed that all the classes were predicting somewhat accurately.

Figure 92

XGBoost ROC Curve on Test Data

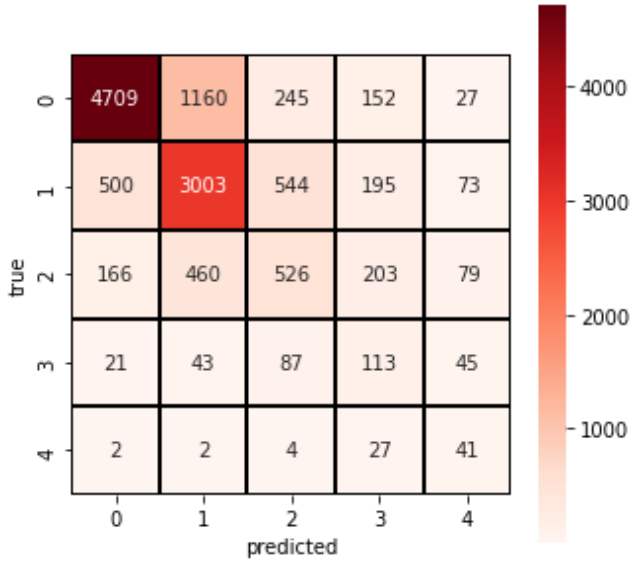


Confusion matrix was created to understand further how many misclassifications took place in each of the classes. It is valuable because they allow you to compare values such as True

Positives, False Positives, True Negatives, and False Negatives directly. Figure 93 shows the confusion matrix for the model chosen. From the confusion matrix, it can be noticed that there were quite a few misclassifications. These misclassifications might have been decreased by fine-tuning the XGBoost model's hyperparameters using Grid Search or Random Search. The fundamental disadvantage of using these strategies for hyperparameter adjustment is the high computational cost. These consume a lot of resources and take a lot of time. Hence, the hyperparameter tuning techniques were not used.

Figure 93

XGBoost Confusion Matrix on Test Data



Classification Report was used to summarize these metrics. They display a per-class representation of the main classification metrics. These provide a more in-depth understanding of the classifier's behavior rather than global accuracy, which might disguise functional deficiencies in one class of a multi-class problem. Figure 94 summarizes these metrics in the form of a table for the chosen tested XGBoost model. It can be seen that the F1-macro is poor compared to the accuracy for the classes. This was due to the class imbalance.

Figure 94

XGBoost F1-Macro, AUC score, And Accuracy On Test Data

	Test Data		
Model	**F1-Macro**	**AUC**	**Accuracy**
XGB	0.4623	0.86	0.6753

In conclusion, applying the hyperparameter tuning strategies would have improved the XGBoost. However, it would have cost money and time in terms of computation. The potential for XGBoost is significant, though, if more effort is spent manually fine-tuning the model.

4.5.5 Comparison Of All Models

Figure 95 shows the evaluation results on the test data for all models.

Figure 95

Comparing All Models on Test Data

	Evaluation on Test Data		
Model	**F1-Macro**	**AUC**	**Accuracy**
DT	0.4198	0.83	0.6512
MLP	0.4713	0.85	0.6908
RF	**0.4795**	0.88	0.7054
XGB	0.4623	0.86	0.6753

The models went through a similar process to produce the best-performing model. The application of SMOTE and hyper-parameter tuning produced some of the best results. Overall, the best-performing model was a hyper-parameter-tuned Random Forest, which slightly

outperformed the MLP model. Code implementation for model development and evaluation can be seen in Appendix E.

References

Abellán, J., López, G., & de Oña, J. (2013, November). Analysis of traffic accident severity using Decision Rules via Decision Trees. *Expert Systems With Applications*, 40(15), 6047–6054. https://doi.org/10.1016/j.eswa.2013.05.027

Biau, G. (2016, April 19). A random forest guided tour. SpringerLink. https://link.springer.com/article/10.1007/s11749-016-0481-7?error=cookies_not_supported&code=2d11c1a6-1178-4f4c-8ebd-495928c00343

Brodsky, I, "H3: Hexagonal hierarchical geospatial indexing system," *Uber Open Source*. 2020; https:// github.com/uber/h3

Brownlee, J. (2021, July 6). A gentle introduction to long short-term memory networks by the experts. *Machine Learning Mastery*. Retrieved September 20, 2022, from https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

C. Parra, C. Ponce and S. F. Rodrigo, Evaluating the Performance of Explainable Machine Learning Models in Traffic Accidents Prediction in California. *2020 39th International Conference of the Chilean Computer Science Society (SCCC*), 2020, pp. 1-8, doi: 10.1109/SCCC51225.2020.9281196.

California Office of Traffic Safety. (2021). *California Annual Report 2021* https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-05/CA_FY2021_AR.pdf

Carmona, P., Dwekat, A., & Mardawi, Z. (2022). No more black boxes! Explaining the predictions of a machine learning XGBoost classifier algorithm in business failure. *Research in International Business and Finance*, *61*, 101649. https://doi.org/10.1016/j.ribaf.2022.101649

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide. undefined. Retrieved October 14, 2022, from https://www.semanticscholar.org/paper/CRISP-DM-1.0%3A-Step-by-step-data-mining-guide-Chapman-Clinton/54bad20bbc7938991bf34f86dde0babfbd2d5a72

Chen, T., & Guestrin, C. (2016). XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/2939672.2939785

Emmert-Streib, F., & Dehmer, M. (2019). High-Dimensional LASSO-Based Computational Regression Models: Regularization, Shrinkage, and Selection. Machine Learning and Knowledge Extraction, 1(1), 359–383. https://doi.org/10.3390/make1010021

Fiorentini, N., & Losa, M. (2020). Handling Imbalanced Data in Road Crash Severity Prediction by Machine Learning Algorithms. *Infrastructures*, *5*(7), 61. https://doi.org/10.3390/infrastructures5070061

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.). O'Reilly Media.

Ghandour AJ, Hammoud H, Al-Hajj S. Analyzing Factors Associated with Fatal Road Crashes: A Machine Learning Approach. *International Journal of Environmental Research and Public Health*. 2020; 17(11):4111. https://doi.org/10.3390/ijerph17114111

Hamid, Y., & Habuza, T. (2020, June 5). Road Crashes Analysis and Prediction using Gradient Boosted and Random Forest Trees. *2020 6th IEEE Congress on Information Science and Technology (CiSt)*. https://doi.org/10.1109/cist49399.2021.9357298

Hammerstrom. (1993, July). Working with neural networks. Working With Neural Networks | IEEE Journals & Magazine | IEEE Xplore. Retrieved December 4, 2022, from https://ieeexplore.ieee.org/abstract/document/222230/authors#authors

Insurance Institute for Highway Safety (IIHS). (2022, May). *Fatality Facts 2020 Yearly snapshot*. https://www.iihs.org/topics/fatality-statistics/detail/yearly-snapshot

Jahangiri, A., Berardi, V., & Machiani, S. G. (2018, November 20). Crash severity analysis of rear-end crashes in California using statistical and machine learning classification methods. *Journal of Transportation Safety & Security*, 12(4), 522–546. https://doi.org/10.1080/19439962.2018.1505793

Jamal, A., Zahid, M., Tauhidur Rahman, M., Al-Ahmadi, H. M., Almoshaogeh, M., Farooq, D., & Ahmad, M. (2021, June 1). *Injury severity prediction of traffic crashes with ensemble machine learning techniques: a comparative study.* International Journal of Injury Control and Safety Promotion, 28(4), 408–427. https://doi.org/10.1080/17457300.2021.1928233

Kelleher, J. D., Namee, B. M., & D'Arcy, A. (2015). Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies. Amsterdam University Press.

Komol, M. M. R., Hasan, M. M., Elhenawy, M., Yasmin, S., Masoud, M., & Rakotonirainy, A. (2021, August 5). Crash severity analysis of vulnerable road users using machine learning. *PLOS ONE*, 16(8), e0255828. https://doi.org/10.1371/journal.pone.0255828

Labib, M. F., Rifat, A. S., Hossain, M. M., Das, A. K., & Nawrine, F. (2019). Road Accident Analysis and Prediction of Accident Severity by Using Machine Learning in Bangladesh. *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. https://doi.org/10.1109/icscc.2019.8843640

Lee, J., Yoon, T., Kwon, S., Lee, J., & Lee, Jongtae. (2019, December 23). Model Evaluation for Forecasting Traffic Accident Severity in Rainy Seasons Using Machine Learning Algorithms: Seoul City Study. *Applied Sciences,* 10(1), 129. https://doi.org/10.3390/app10010129

Louppe, G. (2014, July 28). *Understanding Random Forests: From Theory to Practice*. arXiv.Org. https://doi.org/10.48550/arXiv.1407.7502

Lu, T., Dunyao, Z., Lixin, Y., & Pan, Z. (2015). The traffic accident hotspot prediction: Based on the logistic regression method. *2015 International Conference on Transportation Information and Safety (ICTIS)*. Published. https://doi.org/10.1109/ictis.2015.7232194

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*, 106322. https://doi.org/10.1016/j.aap.2021.106322

Mamlook, R. E., Abdulhameed, T. Z., Hasan, R., Al-Shaikhli, H. I., Mohammed, I., & Tabatabai, S. (2020, July). Utilizing Machine Learning Models to Predict the Car Crash Injury Severity among Elderly Drivers. *2020 IEEE International Conference on Electro Information Technology (EIT)*. https://doi.org/10.1109/eit48999.2020.9208259

Manzoor, M., Umer, M., Sadiq, S., Ishaq, A., Ullah, S., Madni, H. A., & Bisogni, C. (2021). RFCNN: Traffic Accident Severity Prediction Based on Decision Level Fusion of Machine and Deep Learning Model. *IEEE Access*, *9*, 128359–128371. https://doi.org/10.1109/access.2021.3112546

Marilyn, L., & Hudson, S. (2015, July). Use of a PERT chart to improve efficiency of the Dissertation. LWW. Retrieved October 15, 2022, from https://journals.lww.com/neponline/Fulltext/2015/07000/Use_of_a_PERT_Chart_to_Improve_Efficiency_of_the.12.aspx?casa_token=eAkdqNPuGdgAAAAA%3A7YCOcDL7A0PMe60vG7Fcw4V_g5PFnhxA7pMMFcb3my1W0ZMr_w2KsgxvHATak7nwkTX7gfATHh8Y2D2aDVMrYw

Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. Frontiers in Neurorobotics, 7. https://doi.org/10.3389/fnbot.2013.00021

National Highway Traffic Safety Administration (NHSTA). (2022, May 17) Newly released Estimates Show Traffic Fatalities Reached a 16-Year High in 2021 https://www.nhtsa.gov/press-releases/early-estimate-2021-traffic-fatalities#:~:text=NHTSA%20projects%20that%20an%20estimated,Fatality%20Analysis%20Reporting%20System%27s%20history

Nujjetty, A., Mohamedshah, Y., & Council, F. (2014, June). Highway Safety Information System. Highway Safety Information System. https://www.hsisinfo.org/pdf/guidebook_CA.pdf

R, S. E. (2022, November 30). Understanding Random Forest. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Rahim, M. A., & Hassan, H. M. (2021, May). A deep learning based traffic crash severity prediction framework. *Accident Analysis &Amp; Prevention*, *154*, 106090. https://doi.org/10.1016/j.aap.2021.106090

Rančić, S., Radovanović, S., & Delibašić, B. (2021). Investigating Oversampling Techniques for Fair Machine Learning Models. Lecture Notes in Business Information Processing, 110–123. https://doi.org/10.1007/978-3-030-73976-8_9

Rezapour, M, Nazneen, S, Ksaibati, K. Application of deep learning techniques in predicting motorcycle crash severity. Engineering Reports. 2020; 2:e12175. https://doi.org/10.1002/eng2.12175

Rivara, F. P., Boisvert, D., Relyea-Chew, A., & Gomez, T. (2012). Last Call: decreasing drunk driving among 21–34-year-old bar patrons. *International Journal of Injury Control and Safety Promotion*, *19*(1), 53–61. https://doi.org/10.1080/17457300.2011.603150

Salem et al., "Exploring the roles of social media data to identify the locations and severity of road traffic accidents," 2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2021, pp. 62-71, doi: 10.1109/AIKE52691.2021.00016.

San Jose Vehicle Crash Data. (n.d.). Https://Data.Sanjoseca.Gov/Dataset/Crashes-Data.

Santos, D., Saias, J., Quaresma, P., & Nogueira, V. B. (2021, November 24). Machine Learning Approaches to Traffic Accident Analysis and Hotspot Prediction. *Computers*, *10*(12), 157. https://doi.org/10.3390/computers10120157

Shams, S. (2022, August). Neural Network Basics. Lecture, San Jose; San Jose State University.

Shams, S. (2022, September). Training Deep Neural Networks. Lecture, San Jose; San Jose State University.

Sperandei, S. (2014). Understanding logistic regression analysis. *Biochemia Medica*, 12–18. https://doi.org/10.11613/bm.2014.003

Subasi, A. (2018, February). Traffic accident detection using random forest classifier. *2018 15th Learning and Technology Conference (L&Amp;T)*. https://doi.org/10.1109/lt.2018.8368509

Tao, W., Aghaabbasi, M., Ali, M., Almaliki, A. H., Zainol, R., Almaliki, A. A., & Hussein, E. E. (2022, February 20). *An Advanced Machine Learning Approach to Predicting Pedestrian Fatality Caused by Road Crashes: A Step toward Sustainable Pedestrian Safety*. Sustainability, 14(4), 2436. https://doi.org/10.3390/su14042436

Theofilatos, A., & Yannis, G. (2014, November). *A review of the effect of traffic and weather characteristics on road safety.* Accident Analysis & Prevention, 72, 244–256. https://doi.org/10.1016/j.aap.2014.06.017

Ting, C. Y., Tan, N. Y. Z., Hashim, H. H., Ho, C. C., & Shabadin, A. (2020). Malaysian Road Accident Severity: Variables and Predictive Models. Lecture Notes in Electrical Engineering, 699–708. https://doi.org/10.1007/978-981-15-0058-9_67

U.S. Department of Transportation (DOT), Federal Highway Administration. (2021, Dec 20). Highway Statistics 2020. https://www.bts.gov/content/us-vehicle-miles

Vanderplas, J., & VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. Van Duuren Media.

Vaiyapuri, T., & Gupta, M. (2021). Traffic accident severity prediction and cognitive analysis using deep learning. *Soft Computing*. Published. https://doi.org/10.1007/s00500-021-06515-5

Wang, C., Liu, L., Xu, C., & Lv, W. (2019, January 25). *Predicting Future Driving Risk of Crash-Involved Drivers Based on a Systematic Machine Learning Framework*. International Journal of Environmental Research and Public Health, 16(3), 334. https://doi.org/10.3390/ijerph16030334

Yuan, Z., Janson, B., Peng, Y., Yang, Y., & Wang, W. (2021, January 18). Older Pedestrian Traffic Crashes Severity Analysis Based on an Emerging Machine Learning XGBoost. *Sustainability*, *13*(2), 926. https://doi.org/10.3390/su13020926

Appendix A

Project Management Plan

Figure A1. Gantt Chart of the project showing Business and Data Understanding

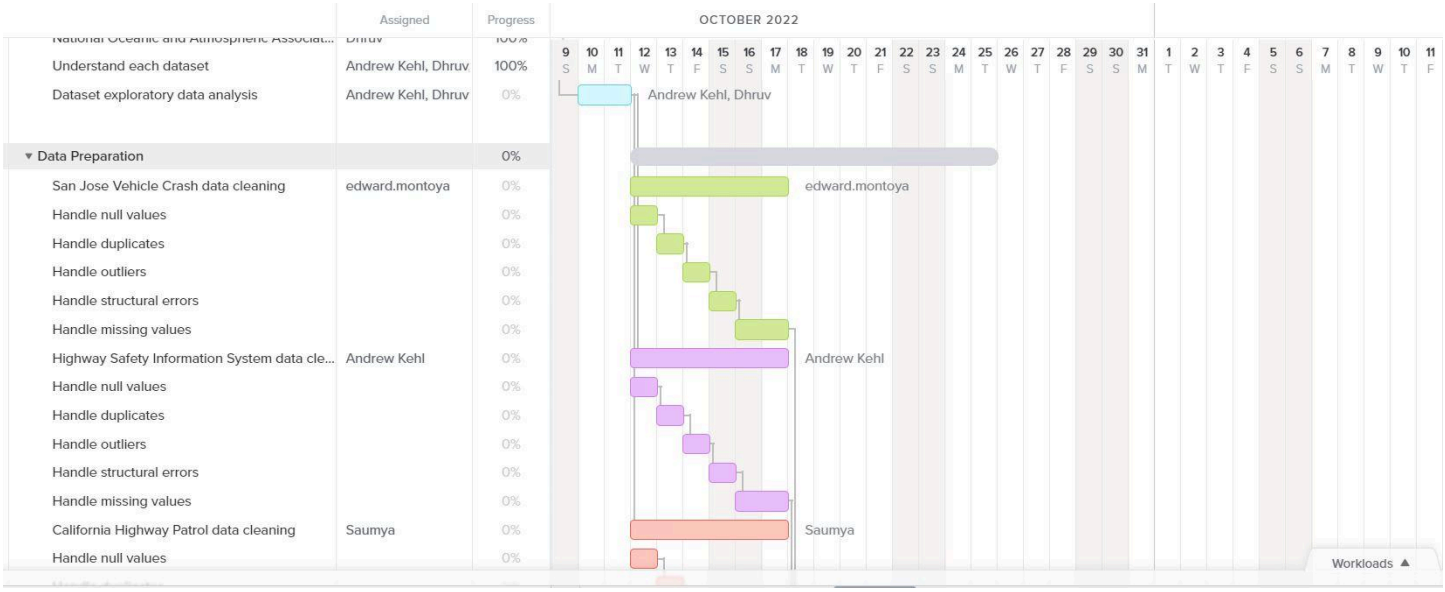


Figure A2. Gantt Chart of the project showing Data Preparation

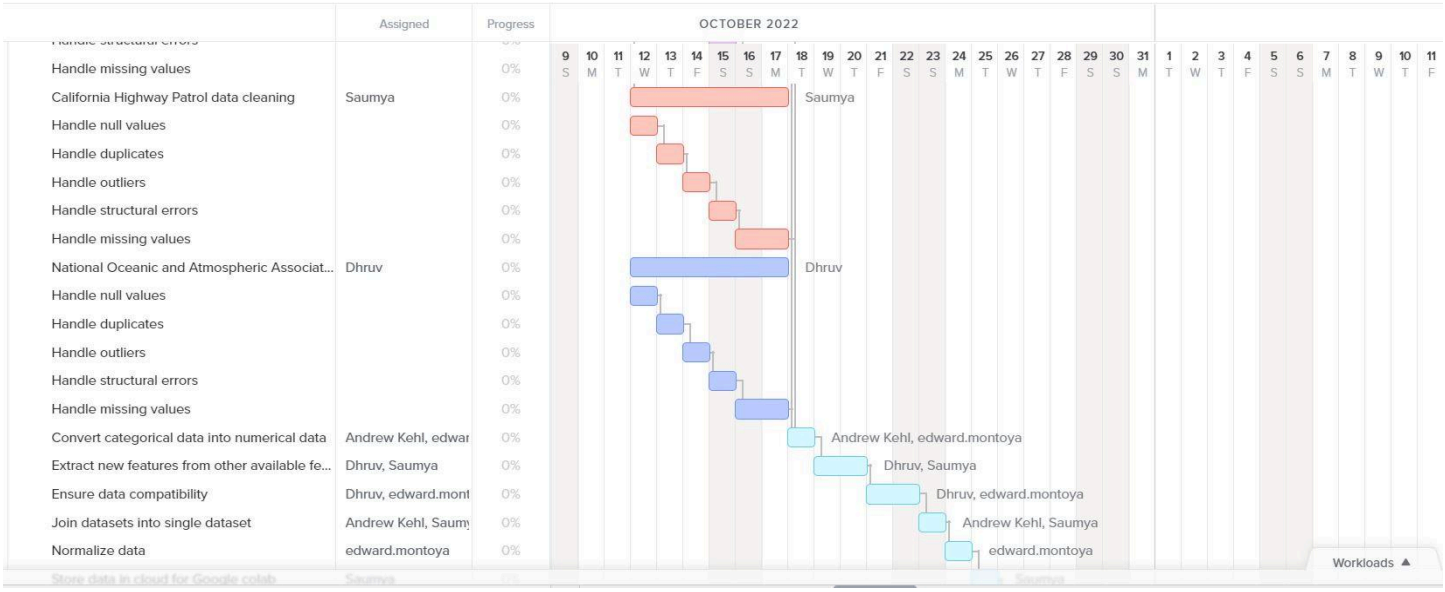


Figure A3. Gantt Chart of the project showing more Data Preparation tasks

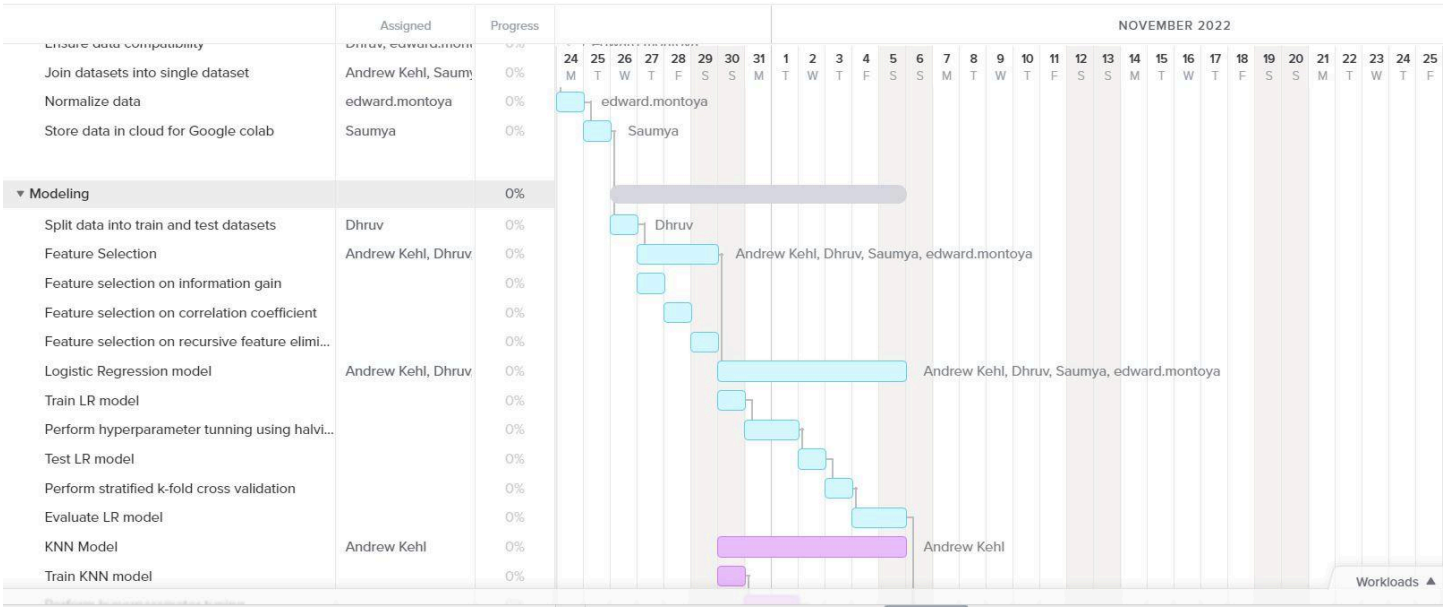


Figure A4. Gantt Chart of the project showing Data Modeling

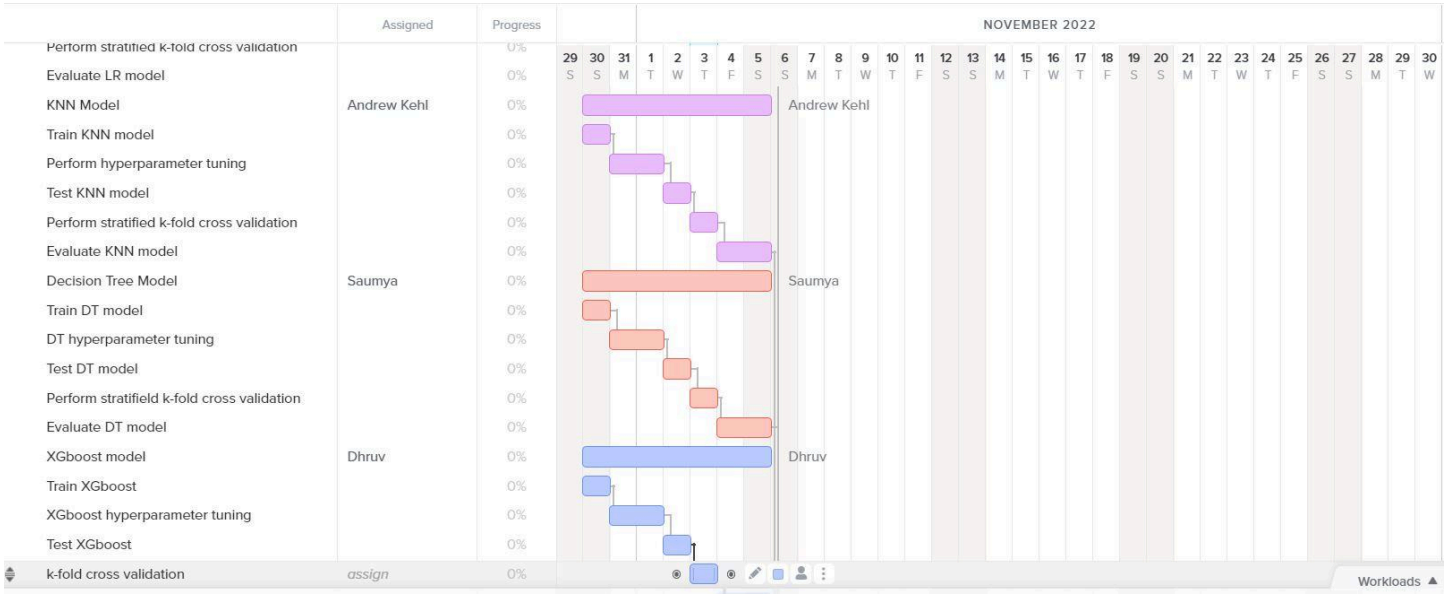


Figure A5. Gantt Chart of the project showing further Data Modeling tasks

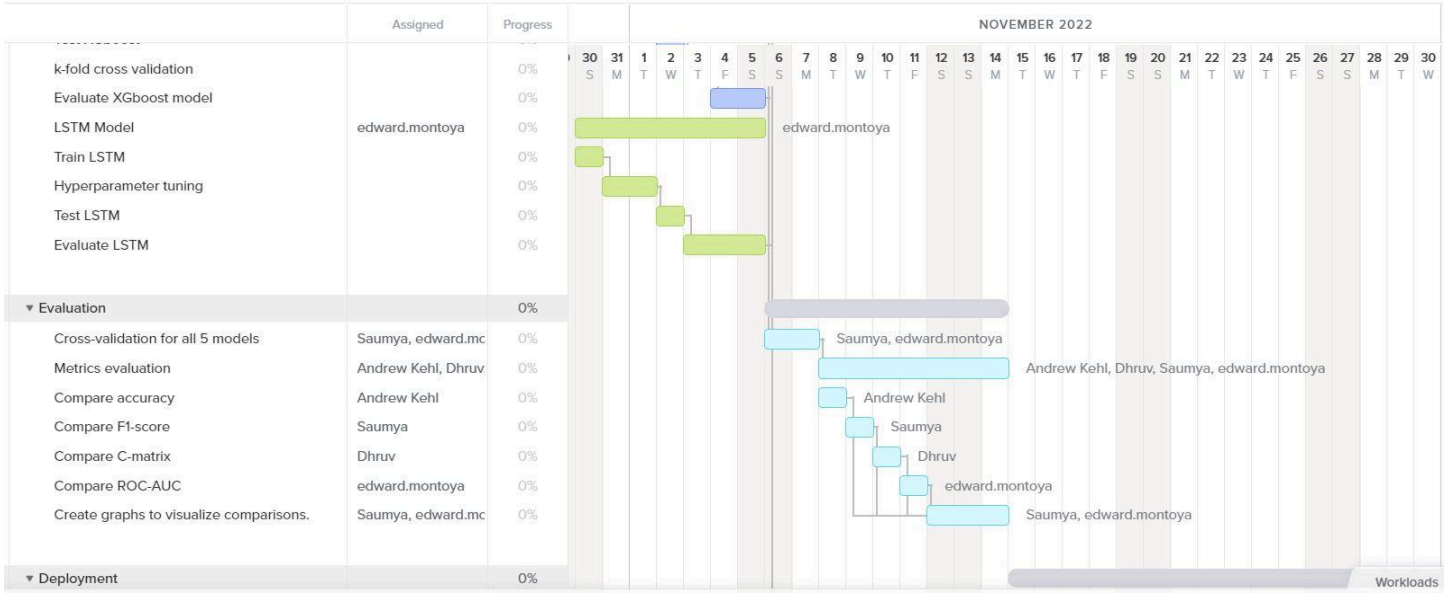


Figure A6. Gantt Chart of the project showing Model Evalutation

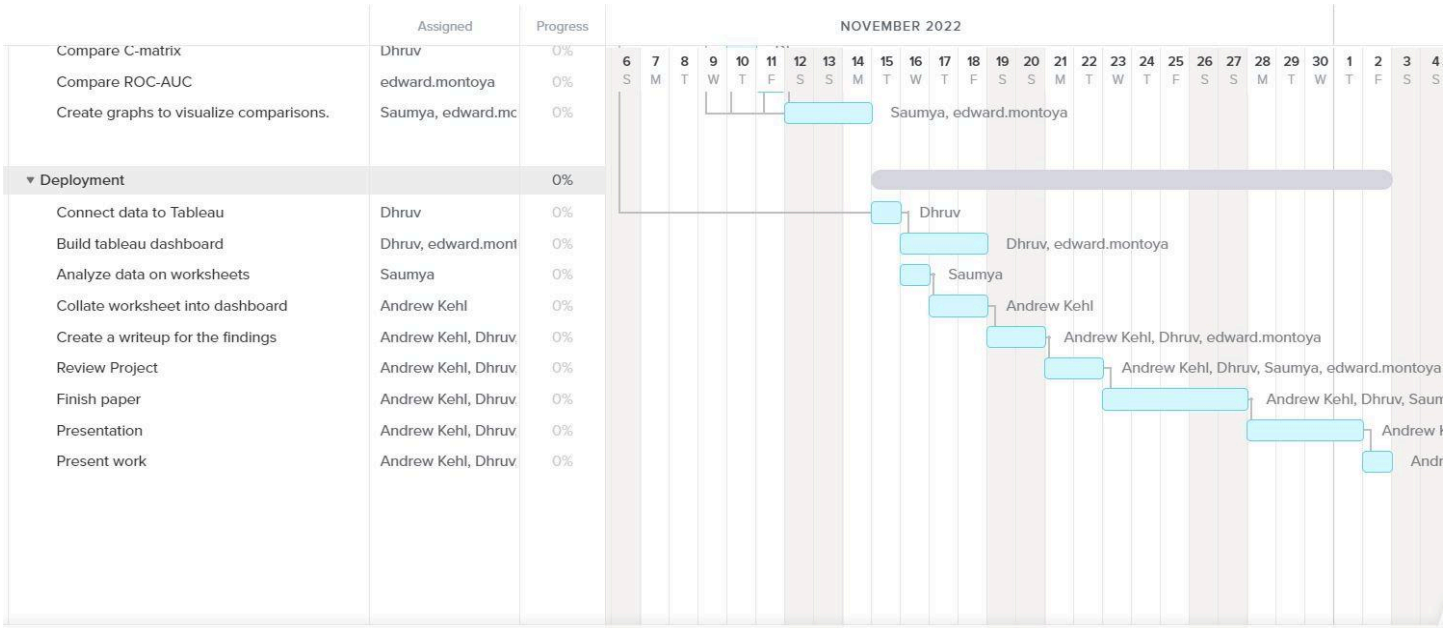


Figure A7. Gantt Chart of the project showing Deployment

Appendix B

Code Implementation for San Jose Vehicles Dataset Preprocessing

```
df = pd.read_csv("https://data.sanjoseca.gov/dataset/918fb7f0-60c0-484e-b31c-334d1ec74e92/resource/8f97060e-9899-42c8-a7ef-8b441e1fa6a9/download/vehiclecrashdata2011-2020.csv")
```

```
df.shape
```

```
(115302, 15)
```

B1. Loading data for San Jose Vehicles dataset and checking the shape

```
df.drop_duplicates(subset="CrashName",keep= 'first', inplace=True)  #need first instance of CrashName
```

```
df.reset_index(inplace = True) #to reset index since it changes when duplicates are drop
df.drop('index',axis = 1, inplace = True) #dropping index column
```

B2. Code for dropping duplicates and resetting the index

```
missing_vals = df.isna().sum()/len(df)*100 #Checking for missing values
missing_vals = missing_vals[missing_vals > 0] #Assigning variables with missig values > 0
print('Percentage of Missing Values:')
print(missing_vals)
```

B3. Code to check for Missing Values in the Vehicles Dataset

```
df["Sex"]=df["Sex"].replace(np.NaN, "Sex_Unknown") #replace missing values for sex
one_hot_pc = pd.get_dummies(df["Sex"]) #encoding categorical to numerical
df = df.drop('Sex',axis = 1)
df = df.join(one_hot_pc) #joining these cols to dataframe
```

B4. Treating missing values for Sex and then one hot encoding it

```
df = df[df.Age >15]
```

B5. Keeping data for legal-age drivers in the data frame

```
Q1 = np.percentile(df.Age, 25,
                   interpolation = 'midpoint') #calculating Quartile 1

Q3 = np.percentile(df.Age, 75,
                   interpolation = 'midpoint') #calculating Quartile 3
IQR = Q3 - Q1

print("Old Shape: ", df.Age.shape) #Shape before treatment

# Upper bound

upper = np.where(df.Age >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(df.Age <= (Q1-1.5*IQR))


''' Removing the Outliers '''
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)

print("New Shape: ", df.Age.shape) #Shape after treatment
```

```
Old Shape:  (35158,)
New Shape:  (35054,)
```

B6. Inter Quartile Range based treatment of outliers for Age feature

```
#Replacing values for VehicleDamage for One Hot encoding
df['VehicleDamage']=df['VehicleDamage'].replace(['Unknown'],"VD_Unknown")
df['VehicleDamage']=df['VehicleDamage'].replace(['Moderate'],"VD_Moderate")
df['VehicleDamage']=df['VehicleDamage'].replace(['Minor'],"VD_Minor")
df['VehicleDamage']=df['VehicleDamage'].replace(['Major'],"VD_Major")
df['VehicleDamage']=df['VehicleDamage'].replace(['None'],"VD_None")
df['VehicleDamage']=df['VehicleDamage'].replace(['Not Applicable'],"VD_N/A")
df['VehicleDamage']=df['VehicleDamage'].replace(['Totaled'],"VD_Totaled")
one_hot_vd = pd.get_dummies(df['VehicleDamage']) #Encoding the variable
df = df.drop('VehicleDamage',axis = 1)
df = df.join(one_hot_vd)
```

B7. Performing One-hot encoding for VehicleDamage feature

```
df.to_csv('cleaned_vehicles.csv')
```

B8. Code to save cleaned DataFrame into CSV

Appendix C

Code Implementation for San Jose Crashes Dataset Preprocessing

```
df = pd.read_csv("https://data.sanjoseca.gov/dataset/918fb7f0-60c0-484e-b31c-334d1ec74e92/resource/c19a01f2-33e1-4c66-9498-85d489f90da4/download/crashdata2011-2020.csv")
```

C1. Loading San Jose Crashes Dataset

```
df.shape
```

```
(56044, 25)
```

C2. Code to check shape of the Dataset

```
df.describe()
```

	CrashFactId	MinorInjuries	ModerateInjuries	SevereInjuries	FatalInjuries	Distance
count	56044.000000	56044.000000	56044.000000	56044.000000	56044.000000	55461.000000
mean	636551.058561	0.389355	0.142745	0.031368	0.008244	82.821676
std	25178.436452	0.696574	0.395495	0.189516	0.094094	237.568783
min	591079.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	614554.750000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	639772.500000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	658137.250000	1.000000	0.000000	0.000000	0.000000	87.000000
max	676829.000000	8.000000	10.000000	5.000000	3.000000	8448.000000

C3. Code to check the summary statistics of the Dataset

```
missing_vals = missing_vals[missing_vals > 0] #assign missing values > 0
print('Percentage of Missing Values:')
print(missing_vals) #print missing value features
```

```
Percentage of Missing Values:
Distance     1.040254
Comment     93.034045
dtype: float64
```

C4. Code to check for missing values in the Crashes Dataset

```
df['MinorInjuries'].value_counts()
```

```
0    39436
1    12668
2     2995
3      703
4      181
5       43
6       13
7        3
8        2
Name: MinorInjuries, dtype: int64
```

C5. Code to check for values of MinorInjuries

```
# This will group all minor injuries into one group, and non-minor injuries into one group
# MinorInjuries is basically binary now

minor = {0: 0, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1}

df.MinorInjuries = [minor[item] for item in df.MinorInjuries]
```

C6. Code to convert MinorInjuires to binary. Same was done for other injury types.

```
# split  CrashDateTime into date and time, add new columns to df
df[['CrashDate', 'CrashTime']] = df['CrashDateTime'].str.split(' ', expand=True)
```

C7. Code to split CrashDateTime to seperate Date and Time features

```
df=df.drop(columns=['CrashDateTime','Comment'])
```

C8. Code to drop unnecessary features

```
#count of labels in each feature
for col in df_One_Hot.columns:
  print(col, ':',len(df_One_Hot[col].unique()), 'Labels')
```

```
CityDamageFlag : 2 Labels
ShortFormFlag : 2 Labels
PedestrianAction : 11 Labels
RoadwaySurface : 5 Labels
RoadwayCondition : 9 Labels
Lighting : 6 Labels
PrimaryCollisionFactor : 9 Labels
TrafficControl : 5 Labels
Weather : 8 Labels
CollisionType : 9 Labels
ProximityToIntersection : 4 Labels
VehicleInvolvedWith : 18 Labels
PedestrianDirectionFrom : 11 Labels
PedestrianDirectionTo : 11 Labels
DirectionFromIntersection : 6 Labels
```

C9. Code to check for number of labels in each of the categorical features

```
ohe = OneHotEncoder(categories='auto') #initialize one hot encoder

feature_arr = ohe.fit_transform(df_One_Hot).toarray() #convert the columns to numerical

ohe_labels = ohe.get_feature_names(df_One_Hot.columns) #get column names

df_One_Hot = pd.DataFrame(feature_arr,columns=ohe_labels) #assign to dataframe
```

C10. Code to perform one hot encoding on categorical features

```
Final_df.to_csv('San_Jose_Crashes_Cleaned.csv')
```

C11. Code to save cleaned DataFrame into CSV

Appendix D

Code Implementation for Combined Dataset Preprocessing

```
# Check amount of rows that both datasets share on Name and CrashName (our "primary key")
vals =  set(df_crash['Name']).intersection(df_veh['CrashName'])
count = 0
for i in vals:
  count = count + 1

print (count)
```

```
41944
```

D1. Code to check the number of common CrashName in both datasets

```
# combine both datasets with an outer join on CrashName
combined_df = pd.merge(df_crash, df_veh, on= 'CrashName', how='outer')
```

D2. Code to merge both datasets

```
# Normalize Distance and Age
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
cols_to_norm = ['Age','Distance']
combined_df[cols_to_norm] = scaler.fit_transform(combined_df[cols_to_norm])
```

D3. Code to normalize numerical features

```
# Handle target feature by taking 4 columns and combining into one column
#
combined_df['Severity'] = 0
for i in range(len(combined_df)):
  if combined_df['MinorInjuries'][i] == 1:
    combined_df['Severity'][i] = 1
  elif combined_df['ModerateInjuries'][i] == 1:
    combined_df['Severity'][i] = 2
  elif combined_df['SevereInjuries'][i] == 1:
    combined_df['Severity'][i] = 3
  elif combined_df['FatalInjuries'][i] == 1:
    combined_df['Severity'][i] = 4
  else:
    combined_df['Severity'][i] = 0
```

```
# 0 = no injury
# 1 = minor
# 2 = moderate
# 3 = severe
# 4 = fatal
combined_df['Severity'].value_counts()
```

```
0    20999
1    14521
2     4652
3      990
4      261
Name: Severity, dtype: int64
```

D4. Code to create target feature - Severity from features MinorInjuries, ModerateInjuries, SevereInjuries and FatalInjuries

```
# Now we can drop MinorInjuries, ModeratInjuries, SevereInjuries, FatalInjuries
combined_df.drop(columns=['MinorInjuries','ModerateInjuries','SevereInjuries','FatalInjuries'], inplace=True)
```

D5. Code to drop not required features

```
combined_df.shape
```

```
(41423, 298)
```

D6. Code to check shape of preprocessed dataset

Appendix E

Code Implementation for Model Development and Evalutation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from pydrive.auth import GoogleAuth
from google.colab import drive
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

import warnings
warnings.filterwarnings("ignore")
```

E1. Loading important libraries

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

file_id = '1tq4NEEKTnchbTlXlzfgNNAxY2n0azvhp' #<-- You add in here the id from you google drive file, you can find it
download = drive.CreateFile({'id': file_id})

# Download the file to a local disc
download.GetContentFile('file.csv')
combined_df  = pd.read_csv("file.csv")
```

E2. Code to load the preprocessed data

```
# Split into 70% training and 30% for testing
# The 70% will be used with stratified k-fold to train and validate
y = combined_df['Severity'] #Features to be used for prediction
X = combined_df.drop('Severity',axis=1) #Target feature
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

E3. Code to create train and test subsets

```
# Setting up Stratified k-fold to test on training data and validate
# K=5, which means there's 5 folds
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

skfold = StratifiedKFold(n_splits=5)
```

E4. Code to setup Stratified Kfold

```
# Instead of arbitrarily choosing the number of dimensions to reduce down to, it is
# simpler to choose the number of dimensions that add up to 95% variance
from sklearn.decomposition import PCA

# Computes the minimum number of dimensions required to preserve 95% of the training
# set's variance
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1  # d equals 75
d
```

```
76
```

E5. Code to setup Principal Component Analysis (PCA)

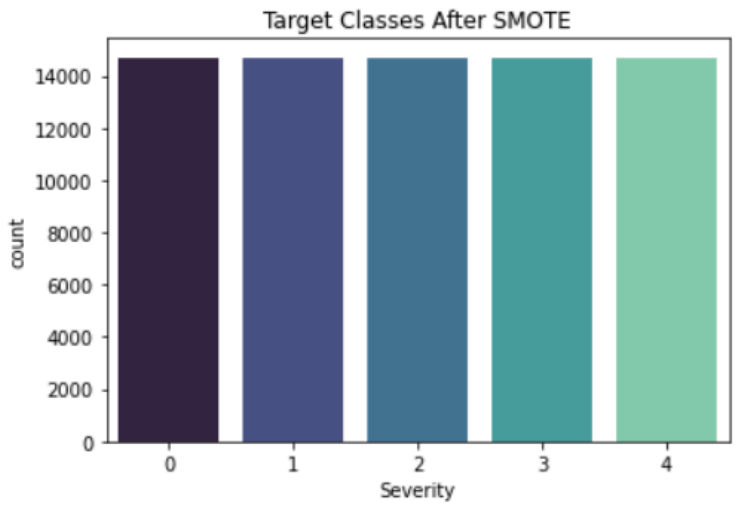
```
# n_components is the ratio of variance we wish to preserve
# We know that 75 features will preserve 95% variance
# So we will set 'n_components' = d
pca = PCA(n_components=d)

# Apply to x_train and x_test
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
```

E6. Code to implement PCA on train and test datasets

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

E7. Code to setup and implement Synthetic Minority Oversampling Technique (SMOTE) on train dataset

```
sns.countplot(y_res,palette = 'mako');
plt.title('Target Classes After SMOTE');
```



E8. Code to plot countplot of target feature after SMOTE

```
lgr_clf = LogisticRegression(random_state = 42)
```

```
# stratified k-fold (k=5) on training data w/ f1 metric
lgr_scores = cross_val_score(lgr_clf, X_train, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Baseline Logestic Regression")
print(lgr_scores)
print(np.mean(lgr_scores))
```

```
Stratified Cross-Val for Baseline Logestic Regression
[0.44332838 0.44790268 0.43830601 0.44061124 0.45743724]
0.44551710985045256
```

E9. Code to develop and validate Logistic Regression model

```
# check training scores
lgr_clf.fit(X_train, y_train)

y_pred_baseline = lgr_clf.predict(X_train)
print(classification_report(y_train, y_pred_baseline))
```

```
              precision    recall  f1-score   support

           0       0.82      0.85      0.83     14706
           1       0.65      0.74      0.69     10206
           2       0.47      0.27      0.34      3218
           3       0.45      0.10      0.16       681
           4       0.57      0.29      0.39       185

    accuracy                           0.73     28996
   macro avg       0.59      0.45      0.48     28996
weighted avg       0.71      0.73      0.71     28996
```

E10. Code to predict and evaluate test data using Logistic Regression

```
dtc_clf = DecisionTreeClassifier(random_state=42)
```

```
dtc_scores = cross_val_score(dtc_clf, X_train, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Decision Tree")
print(dtc_scores)
print(np.mean(dtc_scores))
```

```
Stratified Cross-Val for Decision Tree
[0.40488214 0.39486907 0.3744716  0.3723694  0.39017174]
0.3873527892505949
```

```
# check training scores
dtc_clf.fit(X_train, y_train)

y_pred_dtc = dtc_clf.predict(X_train)
print(classification_report(y_train, y_pred_dtc))
```

E11. Code to develop, evaluate and test Decision Tree on preprocessed data

```
dtc_pca_clf = DecisionTreeClassifier(random_state=42)
```

```
dtc_pca_scores = cross_val_score(dtc_pca_clf, X_train_reduced, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Decision Tree w/ PCA")
print(dtc_pca_scores)
print(np.mean(dtc_pca_scores))
```

```
Stratified Cross-Val for Decision Tree w/ PCA
[0.35404117 0.34395783 0.34929671 0.33068941 0.35112207]
0.3458214392016186
```

E12. Code to develop, evaluate and test Decision Tree on PCA data

```
dtc_smote_clf = DecisionTreeClassifier(random_state=42)
```

```
dtc_smote_scores = cross_val_score(dtc_smote_clf, X_res, y_res, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Decision Tree w/ smote")
print(dtc_smote_scores)
print(np.mean(dtc_smote_scores))
```

```
Stratified Cross-Val for Decision Tree w/ smote
[0.73570428 0.80338889 0.81134494 0.82699773 0.82923316]
0.8013338002567473
```

E13. Code to develop, evaluate and test Decision Tree on SMOTE data

```
# Decision Tree w/ SMOTE
from sklearn.model_selection import GridSearchCV

dtc_params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
```

```
dtc_grid_search_cv = GridSearchCV(DecisionTreeClassifier(random_state=42), dtc_params, verbose=1, cv= skfold, scoring='f1_macro')
# X_res and y_res is SMOTE training
dtc_grid_search_cv.fit(X_res, y_res)
```

```
Fitting 5 folds for each of 294 candidates, totalling 1470 fits
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                            13, 14, 15, 16, 17, 18, 19, 20, 21,
                                            22, 23, 24, 25, 26, 27, 28, 29, 30,
                                            31, ...],
                         'min_samples_split': [2, 3, 4]},
             scoring='f1_macro', verbose=1)
```

```
# print the best parameters
print ('Decision Tree Best Parameters: ', dtc_grid_search_cv.best_estimator_, ' \n')
```

```
Decision Tree Best Parameters:  DecisionTreeClassifier(max_leaf_nodes=97, random_state=42)
```

E14. Code to tune Decision Tree with SMOTE using GridSearchCV

```
xgb_clf = XGBClassifier(random_state=42)
```

```
xgb_scores = cross_val_score(xgb_clf, X_train, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for XGboost")
print(xgb_scores)
print(np.mean(xgb_scores))
```

```
Stratified Cross-Val for XGboost
[0.41178374 0.36400818 0.39456057 0.38886311 0.40724019]
0.3932911560012928
```

E15. Code to develop and evaluate XGBoost on preprocessed train data

```
xgb_pca_clf = XGBClassifier(random_state=42)
```

```
xgb_pca_scores = cross_val_score(rnd_pca_clf, X_train_reduced, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for XGboost w/ PCA")
print(xgb_pca_scores)
print(np.mean(xgb_pca_scores))
```

```
Stratified Cross-Val for XGboost w/ PCA
[0.34997725 0.34016815 0.34734303 0.34835025 0.35137842]
0.34744341873001455
```

E16. Code to develop and evaluate XGBoost on PCA train data

```
xgb_smote_clf = XGBClassifier(random_state=42)
```

```
xgb_smote_scores = cross_val_score(xgb_smote_clf, X_res, y_res, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for XGboost w/ smote")
print(xgb_smote_scores)
print(np.mean(xgb_smote_scores))
```

```
Stratified Cross-Val for XGboost w/ smote
[0.64403631 0.68663655 0.6873819  0.69235477 0.68144453]
0.6783708136654927
```

E17. Code to develop and evaluate XGBoost on SMOTE train data

```
rnd_clf = RandomForestClassifier(random_state=42)
```

```
rnd_scores = cross_val_score(rnd_clf, X_train, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Random Forest")
print(rnd_scores)
print(np.mean(rnd_scores))
```

```
Stratified Cross-Val for Random Forest
[0.39009787 0.36104813 0.37342332 0.4066577  0.40499488]
0.3872443812259757
```

E18. Code to develop and evaluate Random Forest on preprocessed train data

```
rnd_pca_clf = RandomForestClassifier(random_state=42)
```

```
rnd_pca_scores = cross_val_score(rnd_pca_clf, X_train_reduced, y_train, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Random Forest w/ PCA")
print(rnd_pca_scores)
print(np.mean(rnd_pca_scores))
```

```
Stratified Cross-Val for Random Forest w/ PCA
[0.34997725 0.34016815 0.34734303 0.34835025 0.35137842]
0.34744341873001455
```

E19. Code to develop and evaluate Random Forest on PCA train data

```
rnd_smote_clf = RandomForestClassifier(random_state=42)
```

```
rnd_smote_scores = cross_val_score(rnd_smote_clf, X_res, y_res, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Random Forest w/ smote")
print(rnd_smote_scores)
print(np.mean(rnd_smote_scores))
```

```
Stratified Cross-Val for Random Forest w/ smote
[0.8324969  0.89107219 0.89600538 0.91832212 0.926132  ]
0.8928057188181647
```

E20. Code to develop and evaluate Random Forest on SMOTE train data

```
n_estimators = [5,20,50,100] # number of trees in the random forest
max_features = ['auto', 'sqrt'] # number of features in consideration at every split
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)] # maximum number of levels allowed in each decision tree
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False] # method used to sample data points

rnd_params = {'n_estimators': n_estimators,
'max_features': max_features,
'max_depth': max_depth,
'min_samples_split': min_samples_split,
'min_samples_leaf': min_samples_leaf,
'bootstrap': bootstrap}
```

```
from sklearn.model_selection import RandomizedSearchCV

rnd_random_search = RandomizedSearchCV(RandomForestClassifier(random_state=42), param_distributions= rnd_params,
               n_iter = 100, cv= 5, verbose=1, n_jobs = -1, scoring='f1_macro')
```

```
rnd_random_search.fit(X_res_pca, y_res)
```

```
Fitting 5 folds for each of 500 candidates, totalling 2500 fits
```

E21. Code to perform hyperparameter tuning using RandomizedSearchCV

```
RandomForestClassifier(max_depth=80, min_samples_leaf=3, n_estimators=101,
                       random_state=42)
```

```
rnd_tuned_scores = cross_val_score(rnd_tuned, X_res, y_res, cv= skfold, scoring='f1_macro')
```

```
print("Stratified Cross-Val score based on F1-macro for Random Forest tuned")
print(rnd_tuned_scores)
print(np.mean(rnd_tuned_scores))
```

```
Stratified Cross-Val score based on F1-macro for Random Forest tuned
[0.80300184 0.8602846  0.86510792 0.87864972 0.87246045]
0.8559009057259608
```

E22. Code to develop and evaluateTuned Random Forest on SMOTE train data

```
import tensorflow as tf
from tensorflow.python.keras import backend as K
import keras
from imblearn.over_sampling import SMOTE
from keras.callbacks import Callback
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.layers import Dropout
from sklearn.metrics import roc_curve, auc
from keras.layers import BatchNormalization
import time
#from keras_visualizer import visualizer
from PIL import Image
import os
from keras.utils.vis_utils import plot_model
```

E23. Loading libraries for Multi Layer Perceptron (MLP)

```
model = tf.keras.Sequential()
#model.add(keras.layers.Dense(256, activation='relu'))
#model.add(Dropout(0.2))
#model.add(BatchNormalization())
model.add(keras.layers.Dense(128, activation='relu'))

model.add(Dropout(0.7))
model.add(BatchNormalization())
model.add(keras.layers.Dense(5, activation='softmax'))
model.compile(optimizer='SGD',loss='CategoricalCrossentropy', metrics=['accuracy'])
```

E24. Code to develop a MLP

```
start_time = time.time()
history = model.fit(x_res,y_res,batch_size=100, epochs=35,verbose=1,callbacks=[logs,metrics])
etime=(time.time() - start_time)
#40,200
```

```
Epoch 20/35
389/389 [==============================] - 1s 1ms/steposs: 0.8
 F1 macro : 0.40207584817774417
736/736 [==============================] - 3s 5ms/step - loss: 0.8243 - accuracy: 0.6625
Epoch 21/35
389/389 [==============================] - 1s 1ms/steposs: 0.8
 F1 macro : 0.40843733961679235
736/736 [==============================] - 3s 4ms/step - loss: 0.8172 - accuracy: 0.6666
Epoch 22/35
389/389 [==============================] - 1s 3ms/step
 F1 macro : 0.4098793283156986
```

E25. Code to Evaluate MLP on SMOTE train data

```
from sklearn import metrics
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17, 6)):
    y_score = clf.predict_proba(X_test)
    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for label %i' % (roc_auc[i], i))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()
```

E26. Function to plot ROC-AUC curve to test models

```
def evaluate(model):
    target_names = ['No Injury', 'Minor', 'Moderate','Severe','Fatal']
    y_test_pred = model.predict(X_test)
    auc_score= roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr')
    print()
    print("Test Data Results: \n==============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, target_names=target_names, output_dict=True))
    clf_report = clf_report.round(4)
    #print()
    print('Area Under Curve (AUC): {:0.2f}'.format(auc_score))
    print()
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

E27. Function to predict and create classification report on test data

```
# Visualize Confusion Matrix
def matrix_plot(model):
  y_test_pred = model.predict(X_test)
  cm = confusion_matrix(y_test, y_test_pred)
  f, ax = plt.subplots(figsize =(5,5))
  sns.heatmap(cm, annot= True, cmap='Reds', linewidths=1,
              linecolor='k', square=True, mask=False,
              fmt = ".0f",cbar=True,ax=ax)
  plt.xlabel('predicted')
  plt.ylabel("true")
  plt.show()
```

E28. Function to visualize confusion matrix

```
evaluate(dtc_tuned)
```

E29. Code to call function evaulate

```
matrix_plot(dtc_tuned)
```

E30. Code to call function matrix_plot

```
#====== Plot ROC ========
plot_multiclass_roc(dtc_tuned, X_test, y_test, n_classes=5, figsize=(12, 6))
```

E31. Code to call function plot_multiclass_roc