# Predictive Analytics

# Final Project Report

Predicting Severity of Illinois Road Accidents

## Introduction

### Problem Statement

The objective of this project is two-fold:

**(1) Predict the severity level of accidents.**

Severity level can range from 1-4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).

**(2) Identify the factors responsible for accidents of a more severe nature.**

Insights can be leveraged to mitigate risks, reduce the number of accidents, related deaths and inconvenience caused due to traffic delays, etc.

There can be various factors that impact accident severity such as weather conditions, location, time of day, road condition/layout (bump, turning loop) etc. Their relationship with severity of accident (response variable) could be highly complicated and would hence allow the opportunity to apply a wide range of algorithms, such as random forests, multinomial logistic regression, gradient boosting, neural networks etc., to train models for prediction as well as inference.

### Motivation

US accidents data has the potential to be used for numerous applications such as:

- Real-time accident prediction
- Studying accident hotspot locations
- Resource planning
- Casualty analysis
- Studying impact of precipitation or other environmental stimuli on accident occurrence.

We hope to find some meaningful and actionable insights from this project about accident prediction and thus avoidance. Also, the data is quite rich, with almost 3 million records and around 50 features, to provide a considerable challenge to be solved by applying machine learning algorithms. Additionally, the data is collected in real-time, from Feb 2016 to Dec 2019, which is very recent and could potentially make the results of our predictive analytics problem more relevant.

### Description of data

This is a country-wide traffic accident dataset covering 49 states of the United States. The data is collected from Feb 2016 to Dec 2019, using several data providers, including two APIs which provide streaming traffic event data.

The data (1.05GB) is provided in a single csv file, with 2,974,335 rows and 49 columns. Every row represents a unique accident record. Columns are attributes about accident descriptions, time, location, period of day, weather and road conditions. Source

- Kaggle data (https://www.kaggle.com/sobhanmoosavi/us-accident s)
- Population data (https://www.census.gov/)

| Sl No. | Attribute Name | Description | Relevant for modeling | Justification |
|---|---|---|---|---|
| 1 | ID | This is a unique identifier of the accident record. | No | |
| 2 | Source | Indicates source of the accident report (i.e. the API which reported the accident.). | No | Would not impact accident severity |
| 3 | TMC | A traffic accident may have a Traffic Message Channel (TMC) code which provides a more detailed description of the event. | No | Would require NLP |
| 4 | Severity | Shows severity of accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay). | Yes | Response |
| 5 | Start_Time | Shows start time of the accident in the local time zone. | Yes | Certain times of day may be more prone to severe accidents |
| 6 | End_Time | Shows end time of the accident in the local time zone. | No | Retrospective information |
| 7 | Start_Lat | Shows latitude in GPS coordinate of the start point. | Yes | Certain locations may be prone to more severe accidents |
| 8 | Start_Lng | Shows longitude in GPS coordinate of the start point. | Yes | |
| 9 | End_Lat | Shows latitude in GPS coordinate of the end point. | No | Retrospective information |
| 10 | End_Lng | Shows longitude in GPS coordinate of the end point. | No | Retrospective information |
| 11 | Distance(mi) | The length of the road extent affected by the accident. | No | Retrospective information |
| 12 | Description | Shows natural language description of the accident. | No | Would require NLP |
| 13 | Number | Shows the street number in the address field. | No | Too granular |
| 14 | Street | Shows the street name in the address field. | No | Too granular |
| 15 | Side | Shows the relative side of the street (Right/Left) | Yes | May impact severity |
| 16 | City | Shows the city in the address field. | No | Too many levels |
| 17 | County | Shows the county in the address field. | Yes | Used to merge external information |

| | | | | |
|---|---|---|---|---|
| 18 | State | Shows the state in the address field. | No | Only Illinois data is used |
| 19 | Zipcode | Shows the zip code in the address field. | No | Too granular |
| 20 | Country | Shows the country in the address field. | No | All in USA |
| 21 | Timezone | Shows timezone based on the location of the accident (eastern, central, etc.). | No | Would not impact accident severity |
| 22 | Airport_Code | Denotes an airport-based weather station which is the closest one to location of the accident. | No | Would not impact accident severity |
| 23 | Weather_Timestamp | Shows the time-stamp of the weather observation record (in local time). | No | Would not impact accident severity |
| 24 | Temperature | Shows the temperature (in Fahrenheit). | Yes | Certain weather conditions may impact accident severity |
| 25 | Wind_Chill | Shows the wind chill (in Fahrenheit). | Yes | Certain weather conditions may impact accident severity |
| 26 | Humidity | Shows the humidity (in percentage). | Yes | Certain weather conditions may impact accident severity |
| 27 | Pressure | Shows the air pressure (in inches). | Yes | Certain weather conditions may impact accident severity |
| 28 | Visibility | Shows visibility (in miles). | Yes | Certain weather conditions may impact accident severity |
| 29 | Wind_Direction | Shows wind direction. | No | Relative to vehicle's direction |
| 30 | Wind_Speed | Shows wind speed (in miles per hour). | Yes | Certain weather conditions may impact accident severity |
| 31 | Precipitation | Shows precipitation amount in inches, if there is any. | Yes | Certain weather conditions may impact accident severity |
| 32 | Weather_Condition | Shows the weather condition (rain, snow, etc.) | Yes | Certain weather conditions may impact accident severity |

| | | | | |
|---|---|---|---|---|
| 33 | Amenity | A POI annotation which indicates presence of amenity in a nearby location. | Yes | Presence of facilities may impact traffic and subsequently accident severity |
| 34 | Bump | A POI annotation which indicates presence of speed bump or hump in a nearby location. | Yes | May impact accident severity |
| 35 | Crossing | A POI annotation which indicates presence of crossing in a nearby location. | Yes | May impact accident severity |
| 36 | Give_Way | A POI annotation which indicates presence of give_way in a nearby location. | Yes | May impact accident severity |
| 37 | Junction | A POI annotation which indicates presence of junction in a nearby location. | Yes | May impact accident severity |
| 38 | No_Exit | A POI annotation which indicates presence of no_exit in a nearby location. | Yes | May impact accident severity |
| 39 | Railway | A POI annotation which indicates presence of railway in a nearby location. | Yes | May impact accident severity |
| 40 | Roundabout | A POI annotation which indicates presence of roundabout in a nearby location. | Yes | May impact accident severity |
| 41 | Station | A POI annotation which indicates presence of station in a nearby location. | Yes | May impact accident severity |
| 42 | Stop | A POI annotation which indicates presence of stop in a nearby location. | Yes | May impact accident severity |
| 43 | Traffic_ Calming | A POI annotation which indicates presence of traffic_calming in a nearby location. | Yes | May impact accident severity |
| 44 | Traffic_Signal | A POI annotation which indicates presence of traffic_signal in a nearby location. | Yes | May impact accident severity |
| 45 | Turning_Loop | A POI annotation which indicates presence of turning_loop in a nearby location. | Yes | May impact accident severity |
| 46 | Sunrise_ Sunset | Shows the period of day (i.e. day or night) based on sunrise/sunset. | Yes | Certain times of day may be more prone to severe accidents |
| 47 | Civil_Twilight | Shows the period of day based on civil twilight. | Yes | Certain times of day may be more prone to severe accidents |
| 48 | Nautical_ Twilight | Shows the period of day based on nautical twilight. | Yes | Certain times of day may be more prone to severe accidents |

| | | | | |
|---|---|---|---|---|
| 49 | Astronomical_ Twilight | Shows the period of day based on astronomical twilight. | Yes | Certain times of day may be more prone to severe accidents |
| 50 | Population | County level yearly population | Yes | Highly populated areas may be prone to severe accidents |

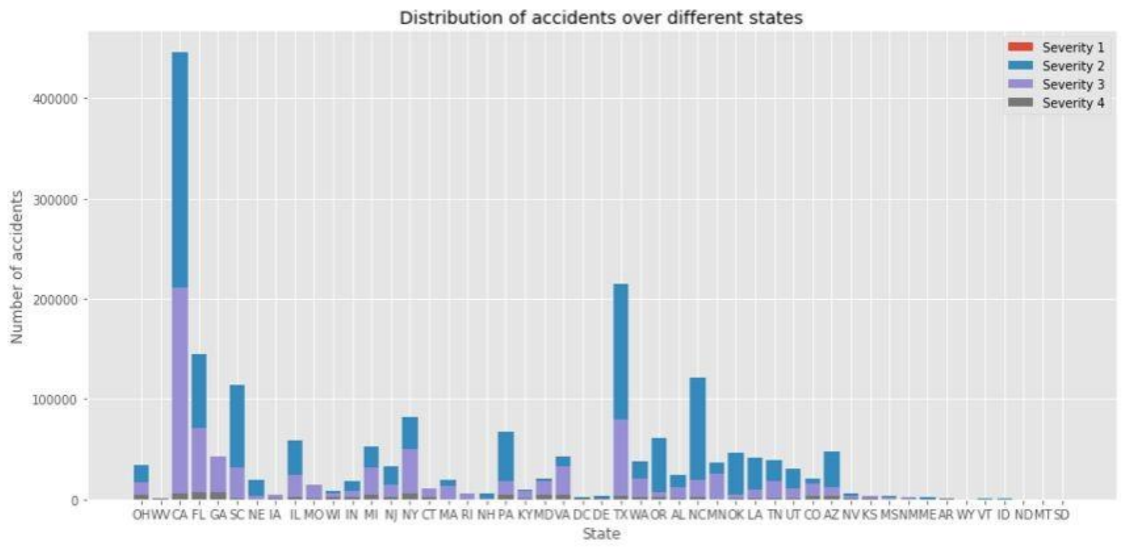
We have placed the codes for all the analyses in the report in the appendix. The appendix has been broken into sections, same as that in the main report. Hence, to find codes for any particular result placed in the report, the corresponding appendix section can be referred.

## Data Pre-processing

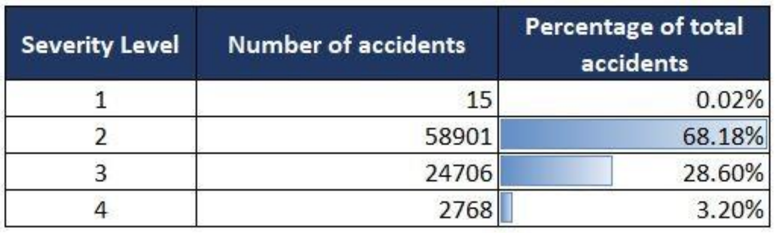
<u>Determining Scope</u>

As mentioned above the dataset initially consisted of around 3 million rows of accident related information across all 50 states in the US. Considering the compute intensive nature of the algorithms that we wanted to explore for this project, we decided to narrow down our scope to only Illinois, thereby reducing the size of the dataset significantly.

The graph below indicates the distribution of accident severity across all states in the US.



The distribution of accident severity for the state of Illinois is as follows:



| Severity Level | Number of accidents | Percentage of total accidents |
|---|---|---|
| 1 | 15 | 0.02% |
| 2 | 58901 | 68.18% |
| 3 | 24706 | 28.60% |
| 4 | 2768 | 3.20% |

As we can see above, the proportion of accidents of severity 1 and 4 are considerably lower than 2 and 3. To make the classification problem more meaningful, we decided to binarize the target variable, describing

Level 1 and 2 as 0, i.e., ‘Not Severe’
Level 3 and 4 as 1, i.e., ‘Severe’

After fixing the scope of the analysis, the dataset reduced to 86,390 rows.

Data Cleaning

Upon further exploration we discovered some discrepancies in the data which were cleaned as follows so as to not interfere with model performance and interpretation

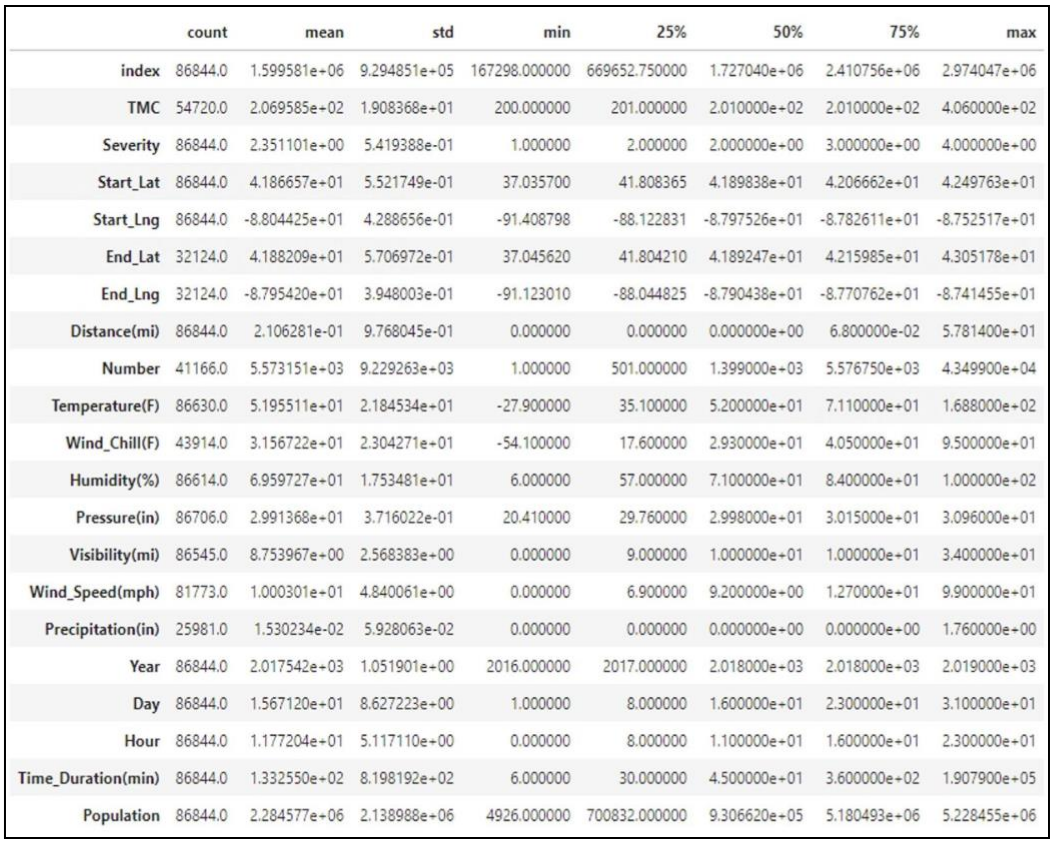
1. Data pertaining to erroneous zip codes were excluded - some zip codes were beginning with 4 or 5 which do not correspond to Illinoi
2. Inconsistency in recording County names - county names were recorded inconsistently, for e.g, Mchenry and McHenry, Saint Clair and St.Clair. These were corrected to have uniformity
3. Reduced levels of categorical variables - Weather_Condition had 59 distinct levels with very little variation between them. This was cleaned up to have 6 distinct levels - snow, rain, clear, thunder, fog, cloudy
4. Repetitive variables removed - variables like ‘Sunrise_Sunset’, ‘Civil_Twilight’, ‘Nautical_Twilight’ and ‘Astronomical_Twilight’ which impart the same information were assessed and only ‘Sunrise_Sunset’ was retained

## Exploratory Data Analysis

Once the data was cleaned, we started by looking at the variable distributions and inter variable relations

1. Derived variable creation - incremental predictors were created from the available fields such as
   a. Hour, day, weekday, month and year from ‘Start_time’
   b. Duration of accident from ‘Start_time’ and ‘End_time’
   c. ‘Time_period’ was created from ‘hour’ to split it into 4 categories: morning, afternoon, evening, night.

2. Initial data distribution was as follows

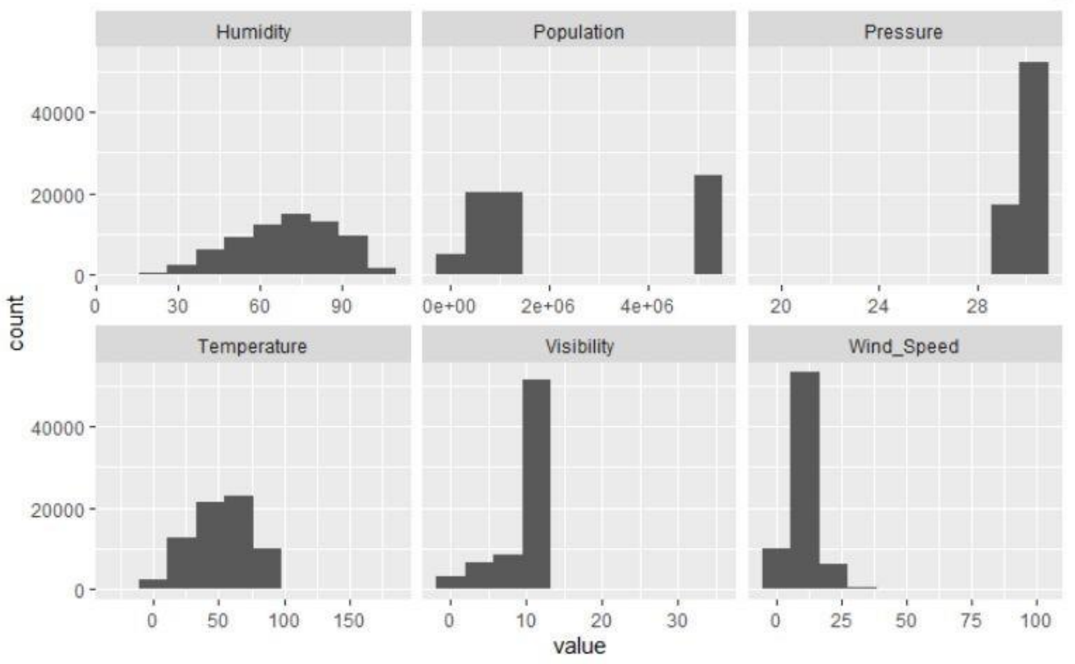


| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| index | 86844.0 | 1.599581e+06 | 9.294851e+05 | 167298.000000 | 669652.750000 | 1.727040e+06 | 2.410756e+06 | 2.974047e+06 |
| TMC | 54720.0 | 2.069585e+02 | 1.908368e+01 | 200.000000 | 201.000000 | 2.010000e+02 | 2.010000e+02 | 4.060000e+02 |
| Severity | 86844.0 | 2.351101e+00 | 5.419388e-01 | 1.000000 | 2.000000 | 2.000000e+00 | 3.000000e+00 | 4.000000e+00 |
| Start_Lat | 86844.0 | 4.186657e+01 | 5.521749e-01 | 37.035700 | 41.808365 | 4.189838e+01 | 4.206662e+01 | 4.249763e+01 |
| Start_Lng | 86844.0 | -8.804425e+01 | 4.288656e-01 | -91.408798 | -88.122831 | -8.797526e+01 | -8.782611e+01 | -8.752517e+01 |
| End_Lat | 32124.0 | 4.188209e+01 | 5.706972e-01 | 37.045620 | 41.804210 | 4.189247e+01 | 4.215985e+01 | 4.305178e+01 |
| End_Lng | 32124.0 | -8.795420e+01 | 3.948003e-01 | -91.123010 | -88.044825 | -8.790438e+01 | -8.770762e+01 | -8.741455e+01 |
| Distance(mi) | 86844.0 | 2.106281e-01 | 9.768045e-01 | 0.000000 | 0.000000 | 0.000000e+00 | 6.800000e-02 | 5.781400e+01 |
| Number | 41166.0 | 5.573151e+03 | 9.229263e+03 | 1.000000 | 501.000000 | 1.399000e+03 | 5.576750e+03 | 4.349900e+04 |
| Temperature(F) | 86630.0 | 5.195511e+01 | 2.184534e+01 | -27.900000 | 35.100000 | 5.200000e+01 | 7.110000e+01 | 1.688000e+02 |
| Wind_Chill(F) | 43914.0 | 3.156722e+01 | 2.304271e+01 | -54.100000 | 17.600000 | 2.930000e+01 | 4.050000e+01 | 9.500000e+01 |
| Humidity(%) | 86614.0 | 6.959727e+01 | 1.753481e+01 | 6.000000 | 57.000000 | 7.100000e+01 | 8.400000e+01 | 1.000000e+02 |
| Pressure(in) | 86706.0 | 2.991368e+01 | 3.716022e-01 | 20.410000 | 29.760000 | 2.998000e+01 | 3.015000e+01 | 3.096000e+01 |
| Visibility(mi) | 86545.0 | 8.753967e+00 | 2.568383e+00 | 0.000000 | 9.000000 | 1.000000e+01 | 1.000000e+01 | 3.400000e+01 |
| Wind_Speed(mph) | 81773.0 | 1.000301e+01 | 4.840061e+00 | 0.000000 | 6.900000 | 9.200000e+00 | 1.270000e+01 | 9.900000e+01 |
| Precipitation(in) | 25981.0 | 1.530234e-02 | 5.928063e-02 | 0.000000 | 0.000000 | 0.000000e+00 | 0.000000e+00 | 1.760000e+00 |
| Year | 86844.0 | 2.017542e+03 | 1.051901e+00 | 2016.000000 | 2017.000000 | 2.018000e+03 | 2.018000e+03 | 2.019000e+03 |
| Day | 86844.0 | 1.567120e+01 | 8.627223e+00 | 1.000000 | 8.000000 | 1.600000e+01 | 2.300000e+01 | 3.100000e+01 |
| Hour | 86844.0 | 1.177204e+01 | 5.117110e+00 | 0.000000 | 8.000000 | 1.100000e+01 | 1.600000e+01 | 2.300000e+01 |
| Time_Duration(min) | 86844.0 | 1.332550e+02 | 8.198192e+02 | 6.000000 | 30.000000 | 4.500000e+01 | 3.600000e+02 | 1.907900e+05 |
| Population | 86844.0 | 2.284577e+06 | 2.138988e+06 | 4926.000000 | 700832.000000 | 9.306620e+05 | 5.180493e+06 | 5.228455e+06 |

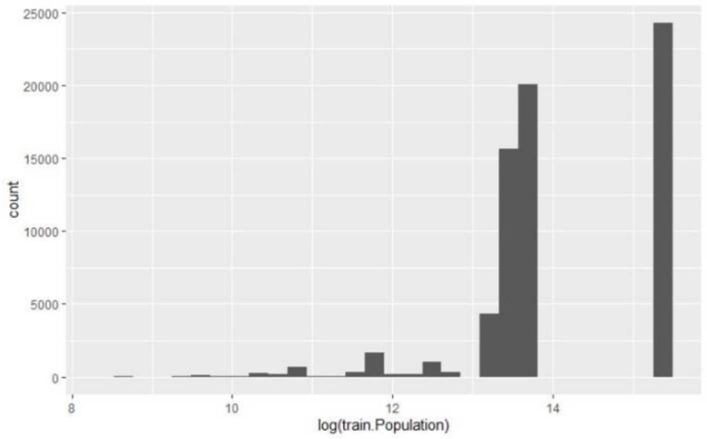
3. Outlier treatment - for each numerical predictor, values beyond $\mu \pm 3\sigma$ are considered as outliers and hence capped at $\mu \pm 3\sigma$ . Some variables that were treated are duration and distance. Weather related variables like temperature and wind_speed were not capped as they might represent meaningful extreme weather days
4. Missing value imputations - missing observations were noticed in the weather related variables, wind_chill, precipitation, temperature, pressure, wind_speed, visibility, humidity and weather_condition
    a. Wind_chill and precipitation had over 50% missing and hence were dropped
    b. Other variables were imputed as follows,
        i. Wherever possible with mean value (mode for categorical variable) of corresponding city
        ii. Where no other data point in corresponding city was available, it was imputed with overall mean/mode
5. Splitting into training and test samples - the dataset was then split into training and test samples in the ratio 80:20. Stratified sampling was carried out to ensure equal distribution of the response variable in both samples

| | 0 (Not Severe) | 1 (Severe) |
|---|---|---|
| Train data | 47,345 (68.15%) | 22,126 (31.85%) |
| Test data | 11,837 (68.15%) | 5,531(31.85%) |

6. Variable transformation and standardization - For all the numerical columns in the data, we observed their distributions as follows:

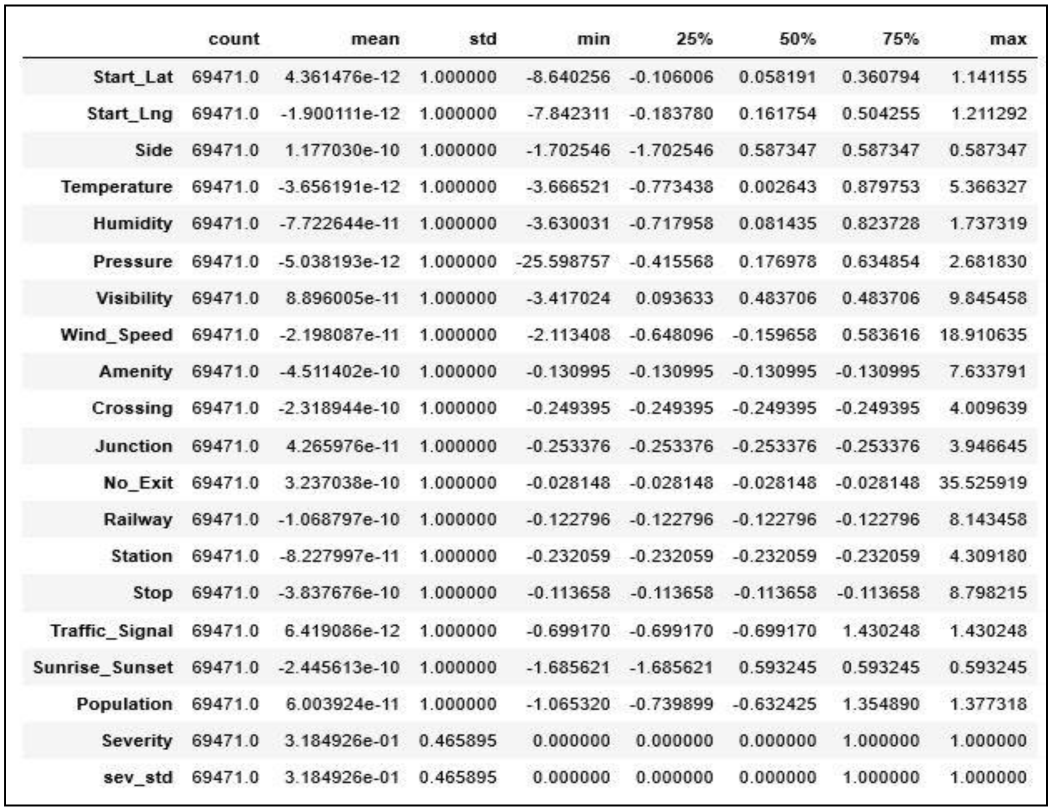


Since the population variable is right skewed, we decided to apply a log transformation to it. The image below shows the distribution of the variable after log transformation



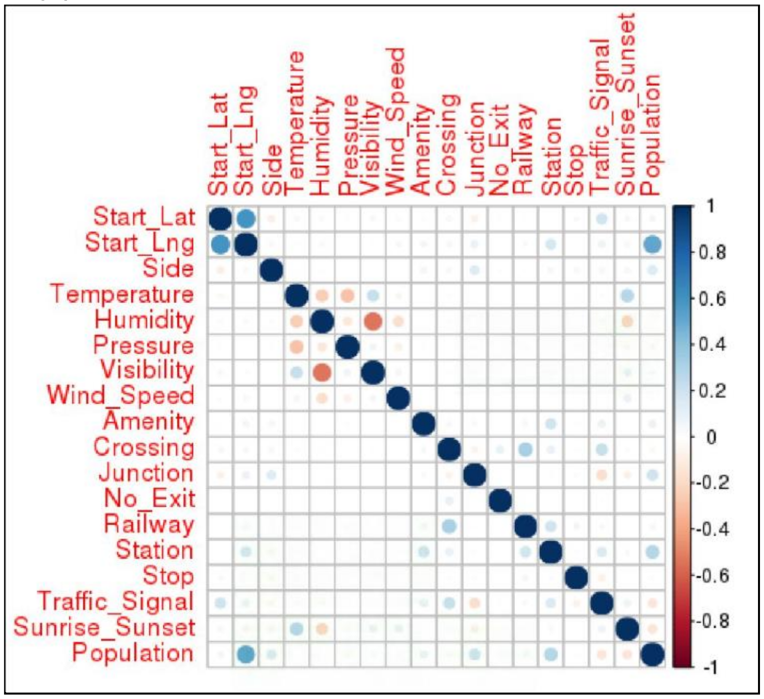
7. Standardization & Normalization - We standardized all numerical predictors in the dataset and created a copy of the normalized version of the response variable as sev_std. These transformations are necessary for models like neural networks. Based on the model type, we will select which version of the response variable to use for training.

   After data transformations, we now have the final dataset ready which will feed into the different models as inputs. Summary of dataset after data cleaning and appropriate transformations is as follows:

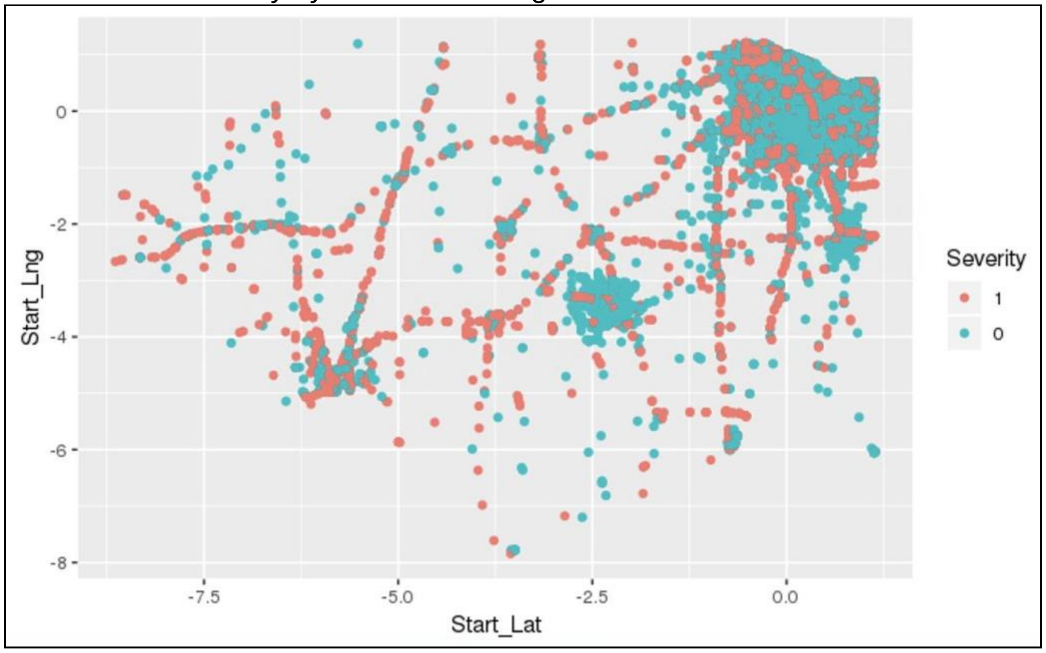


| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Start_Lat | 69471.0 | 4.361476e-12 | 1.000000 | -8.640256 | -0.106006 | 0.058191 | 0.360794 | 1.141155 |
| Start_Lng | 69471.0 | -1.900111e-12 | 1.000000 | -7.842311 | -0.183780 | 0.161754 | 0.504255 | 1.211292 |
| Side | 69471.0 | 1.177030e-10 | 1.000000 | -1.702546 | -1.702546 | 0.587347 | 0.587347 | 0.587347 |
| Temperature | 69471.0 | -3.656191e-12 | 1.000000 | -3.666521 | -0.773438 | 0.002643 | 0.879753 | 5.366327 |
| Humidity | 69471.0 | -7.722644e-11 | 1.000000 | -3.630031 | -0.717958 | 0.081435 | 0.823728 | 1.737319 |
| Pressure | 69471.0 | -5.038193e-12 | 1.000000 | -25.598757 | -0.415568 | 0.176978 | 0.634854 | 2.681830 |
| Visibility | 69471.0 | 8.896005e-11 | 1.000000 | -3.417024 | 0.093633 | 0.483706 | 0.483706 | 9.845458 |
| Wind_Speed | 69471.0 | -2.198087e-11 | 1.000000 | -2.113408 | -0.648096 | -0.159658 | 0.583616 | 18.910635 |
| Amenity | 69471.0 | -4.511402e-10 | 1.000000 | -0.130995 | -0.130995 | -0.130995 | -0.130995 | 7.633791 |
| Crossing | 69471.0 | -2.318944e-10 | 1.000000 | -0.249395 | -0.249395 | -0.249395 | -0.249395 | 4.009639 |
| Junction | 69471.0 | 4.265976e-11 | 1.000000 | -0.253376 | -0.253376 | -0.253376 | -0.253376 | 3.946645 |
| No_Exit | 69471.0 | 3.237038e-10 | 1.000000 | -0.028148 | -0.028148 | -0.028148 | -0.028148 | 35.525919 |
| Railway | 69471.0 | -1.068797e-10 | 1.000000 | -0.122796 | -0.122796 | -0.122796 | -0.122796 | 8.143458 |
| Station | 69471.0 | -8.227997e-11 | 1.000000 | -0.232059 | -0.232059 | -0.232059 | -0.232059 | 4.309180 |
| Stop | 69471.0 | -3.837676e-10 | 1.000000 | -0.113658 | -0.113658 | -0.113658 | -0.113658 | 8.798215 |
| Traffic_Signal | 69471.0 | 6.419086e-12 | 1.000000 | -0.699170 | -0.699170 | -0.699170 | 1.430248 | 1.430248 |
| Sunrise_Sunset | 69471.0 | -2.445613e-10 | 1.000000 | -1.685621 | -1.685621 | 0.593245 | 0.593245 | 0.593245 |
| Population | 69471.0 | 6.003924e-11 | 1.000000 | -1.065320 | -0.739899 | -0.632425 | 1.354890 | 1.377318 |
| Severity | 69471.0 | 3.184926e-01 | 0.465895 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |
| sev_std | 69471.0 | 3.184926e-01 | 0.465895 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |

8. Bi-variate correlation - Correlation matrix of numerical variables is plotted below. The highest correlation is 0.59 (between Start_Lng and Start_Lat), the second highest one is -0.53 (between Visibility and Humidity). We can thus conclude that there is no obvious multicollinearity problem.

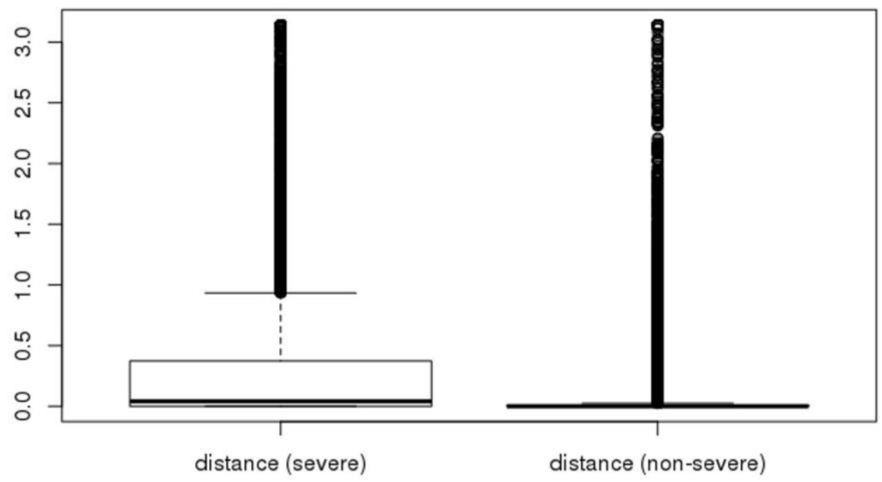


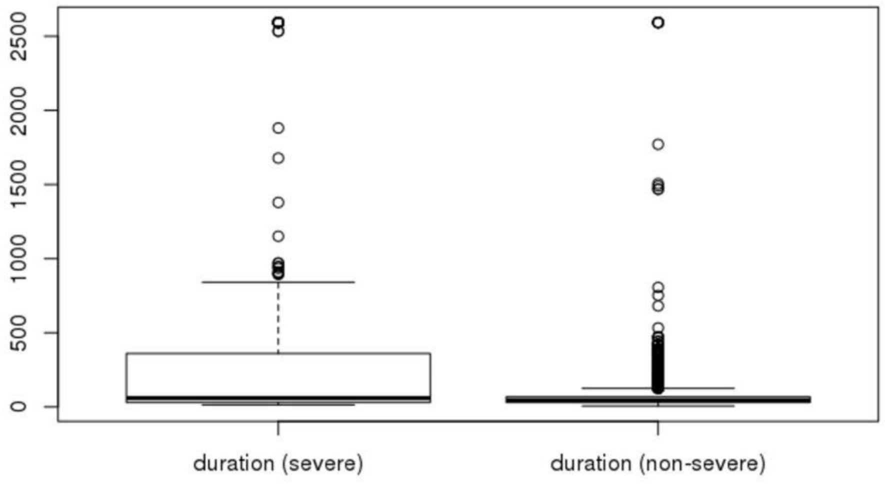
9. Distribution of severity by latitude and longitude



Latitude longitude information is to be used as proxy for categorical location, so as to reduce the dimensions in the model

10. Dropping 'Duration' and 'Distance' - we decided to drop the "Distance" and "Duration" variables. Intuitively, a more severe accident would lead to longer length of the road affected and longer duration. Boxplots of these two variables for each level of severity are presented below. They are separable for severe and non-severe accidents. In addition, we do not want our model to be able to predict the severity of an accident only when an accident has ended, at which point duration and distance could be obtained.





## Candidate Models and Cross Validation

Considering the nature of our problem and the majority of the predictor variables at our disposal, we decided to try the following 7 models:

- Logistic Regression
- Classification Tree
- Random Forest
- Gradient Boosting Machine
- Neural Network
- Projection Pursuit Regression
- Support Vector Machine
- Naive Bayes Classification

Our basic approach was to tune each model separately through cross validation to obtain the best parameter combination corresponding to that algorithm. Each of these tuned models were then compared among each other through 3 replicates of 10-fold cross validation. The models were assessed on a combination of metrics,

- Correct classification rate
- Precision
- Recall
- F1 score
- False Positive Rate
- Custom objective function*

*Considering the aim is to predict severe accidents, the cost of misclassifying a severe accident as non-severe is intuitively much higher than the other way around. Based on this, we decided to optimize the following objective function

$$\underline{\hspace{8cm}}(5 * FaTlsoet\ Nale\ Ngautmivbee + r\ \ o1f\ \ *A\ Fccaildseen\ Ptsositive)$$

where False Negative - severe accidents misclassified as non severe False
Positive - non severe cases misclassified as severe

This weighted average provides a composite measure for achieving our business goal - identifying more risky and severe accidents.

The model depicting the best performance in cross validation was then selected as the final model and was retrained on the complete training data.

<u>Justification for selecting the above models and CV tuning results</u>

1. Logistic Regression

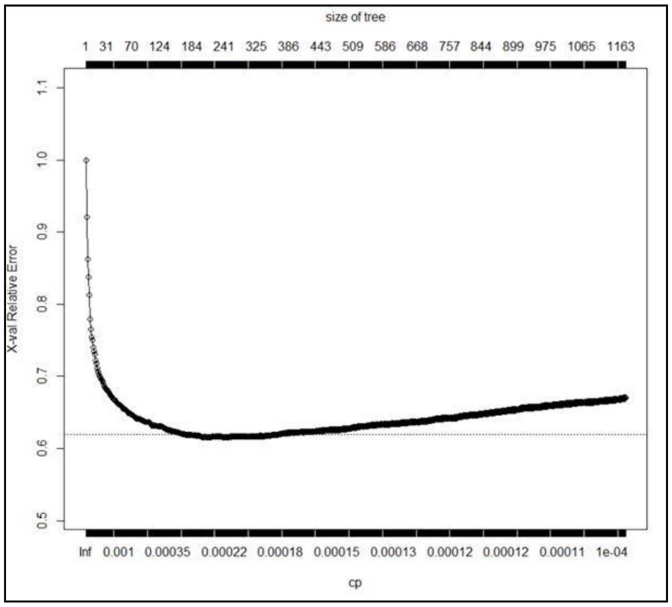
   The baseline method is selected as logistic regression, which models the probability of being a severe event as a linear combination of predictors with a log link function. As the most classic binary classification algorithm, logistic regression plays a major role in providing better model interpretability. The coefficients show the partial effect of each predictor on the log-odds of being a severe accident.

   | Logistic Regression CV Results | | | | | |
   |---|---|---|---|---|---|
   | **Correct Classification Rate** | **Precision** | **Recall** | **F1 Score** | **False Positive Rate** | **Custom Objective Function** |
   | 0.7639 | 0.6909 | 0.4680 | 0.5580 | 0.2161 | 0.5029 |

   However, logistic regression has the typical disadvantage that it does not always produce accurate prediction, since it assumes only a linear relationship between the predictors and the log-odds of response. It does not automatically perform shrinkage and variable selection, and hence may suffer from estimation variance when the predictors are highly correlated.

2. Classification Tree

   Tree-based methods are extremely relevant to classification tasks, and the key advantage of a classification tree is its simple format and interpretability. It returns a set of if-then rules, explaining how the logic flow leads to classification decisions. Moreover, decision trees can handle categorical variables extremely efficiently and considering that the bulk of our predictors are categorical, tree-based methods were an obvious choice. Initially, the built-in cross validation of the R rpart function was utilized to tune the complexity parameter. The graph below shows the results from the built-in rpart function.



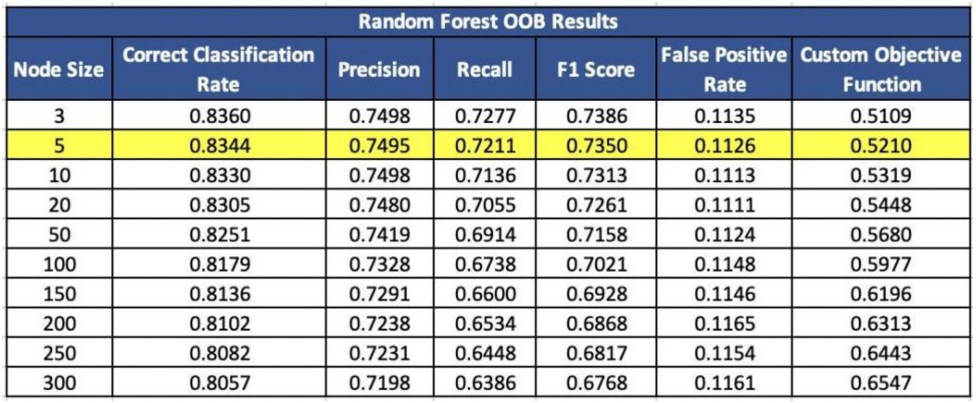
After selecting the optimum complexity parameter (0.000243), we re-ran the model with our predefined folds to ensure consistency while comparing models. The CV results for the best complexity parameter is as presented below.

| Decision Tree CV Results | | | | | |
|---|---|---|---|---|---|
| Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| 0.8048 | 0.6939 | 0.6926 | 0.6932 | 0.1428 | 0.5868 |

The key disadvantage of a Decision Tree is that it tends to not have very high predictive power, especially in case of numerical variables and estimating smooth surfaces. This is usually alleviated through boosting or bagging trees, i.e., Gradient Boosting and Random Forests.

3. Random Forest

Random Forest ensembles a large number of trees, each of which is built on a bootstrapped sample of the original training data and utilizes a randomly selected set of predictors. The final prediction is a combination of the individual tree predictions. Each tree has to be grown to the right depth to ensure high predictive power without overfitting to the training data. The overfitting problem in Random Forest arises more due to overgrowing the individual trees and less from using too many bootstrapped samples. Hence several values of nodesize were tried with constant number of trees as 1000 and out-of-bag(OOB)* results were noted to choose the best value as follows.



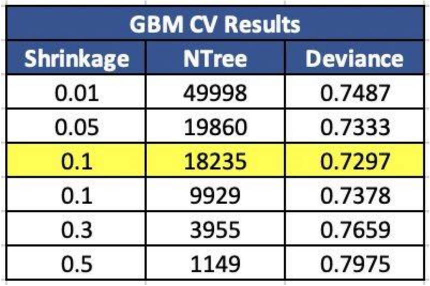
| Random Forest OOB Results | | | | | | |
|---|---|---|---|---|---|---|
| Node Size | Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| 3 | 0.8360 | 0.7498 | 0.7277 | 0.7386 | 0.1135 | 0.5109 |
| 5 | 0.8344 | 0.7495 | 0.7211 | 0.7350 | 0.1126 | 0.5210 |
| 10 | 0.8330 | 0.7498 | 0.7136 | 0.7313 | 0.1113 | 0.5319 |
| 20 | 0.8305 | 0.7480 | 0.7055 | 0.7261 | 0.1111 | 0.5448 |
| 50 | 0.8251 | 0.7419 | 0.6914 | 0.7158 | 0.1124 | 0.5680 |
| 100 | 0.8179 | 0.7328 | 0.6738 | 0.7021 | 0.1148 | 0.5977 |
| 150 | 0.8136 | 0.7291 | 0.6600 | 0.6928 | 0.1146 | 0.6196 |
| 200 | 0.8102 | 0.7238 | 0.6534 | 0.6868 | 0.1165 | 0.6313 |
| 250 | 0.8082 | 0.7231 | 0.6448 | 0.6817 | 0.1154 | 0.6443 |
| 300 | 0.8057 | 0.7198 | 0.6386 | 0.6768 | 0.1161 | 0.6547 |

*OOB - since RF is fit on bootstrapped samples from the training data, the data points that are not selected can be considered as sort of a hold-out sample referred to as out-of-bag, and hence the OOB results serve the same purpose as cross-validation results.

4. Gradient Boosting Machine

As mentioned previously, boosted trees is another advanced algorithm that greatly enhances the predictive power of classification trees, while retaining their other benefits. GBM sequentially fits a tree to the residual from the previous iteration. The key advantage is that each sequentially added tree will capture information that the previous trees did not focus on, thereby increasing prediction accuracy. However, this model tends to overfit, therefore we pay special attention to hyperparameters such as number of trees and the learning shrinkage. Learning shrinkage parameter plays a vital role in controlling the marginal contribution from each additional tree. It may not be desirable to have a large value of learning rate, however too small learning rate will lead to large computational cost.

The built-in cross validation of the R gbm package was utilized to tune different combinations of number of trees and shrinkage, and the best was chosen based on minimum deviance as shown below.



| GBM CV Results | | |
|---|---|---|
| Shrinkage | NTree | Deviance |
| 0.01 | 49998 | 0.7487 |
| 0.05 | 19860 | 0.7333 |
| 0.1 | 18235 | 0.7297 |
| 0.1 | 9929 | 0.7378 |
| 0.3 | 3955 | 0.7659 |
| 0.5 | 1149 | 0.7975 |

5. Neural Network

Neural Networks are an advanced algorithm that can approximate a complex function with high level of classification accuracy. However, like GBM, even neural networks tend to overfit the training data. To prevent this, there are two parameters that have to be tuned through cross validation - decay rate and number of nodes in the hidden layer. The CV results for a grid search of the best hyper parameter combination as presented below.

| Neural Network CV Results | | | | | | | |
|---|---|---|---|---|---|---|---|
| Num_Nodes | Decay | Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| 5 | 1 | 0.793439 | 0.7080851 | 0.5979541 | 0.6483762 | 0.115204 | 0.7187555 |
| 5 | 0.5 | 0.7882474 | 0.7006596 | 0.5851186 | 0.6376975 | 0.1168233 | 0.7402993 |
| 5 | 0.1 | 0.7957757 | 0.7019035 | 0.6236404 | 0.66046 | 0.1237794 | 0.6836954 |
| 5 | 0.05 | 0.7927529 | 0.7011938 | 0.6086655 | 0.6516608 | 0.1212166 | 0.7057957 |
| 5 | 0.01 | 0.7857427 | 0.6999386 | 0.5728705 | 0.6300595 | 0.1147745 | 0.7584076 |
| 10 | 1 | 0.7876092 | 0.7012037 | 0.5805086 | 0.6351673 | 0.1156053 | 0.7468104 |
| 10 | 0.5 | 0.7961595 | 0.7057277 | 0.6174486 | 0.6586405 | 0.1203225 | 0.6911997 |
| 10 | 0.1 | 0.7941587 | 0.7031217 | 0.6121908 | 0.6545112 | 0.1208012 | 0.6998988 |
| 10 | 0.05 | 0.7874941 | 0.7004632 | 0.5814276 | 0.6354022 | 0.1162038 | 0.7457548 |
| 10 | 0.01 | 0.7917836 | 0.6937789 | 0.6198289 | 0.6547212 | 0.1278558 | 0.6925432 |
| 20 | 1 | 0.7962219 | 0.7096056 | 0.60969 | 0.6558631 | 0.1166051 | 0.7010215 |
| 20 | 0.5 | 0.7941971 | 0.7069328 | 0.6043719 | 0.6516379 | 0.1170909 | 0.7098214 |
| 20 | 0.1 | 0.7876764 | 0.6992576 | 0.5849227 | 0.6369892 | 0.1175696 | 0.7411198 |
| 20 | 0.05 | 0.7940196 | 0.6982801 | 0.6220434 | 0.6579607 | 0.1256099 | 0.6874859 |
| 20 | 0.01 | 0.792393 | 0.7015607 | 0.6059387 | 0.6502439 | 0.1204703 | 0.7096294 |

6. Projection Pursuit Regression

   PPR model uses univariate regression functions instead of their multivariate form, thus effectively dealing with the curse of dimensionality. Also, it can ignore variables with low explanatory power. However, PPR models are difficult to interpret, which makes the model more useful for prediction than for understanding the data.

   One parameter needs to be tuned through cross validation - number of terms. The CV results are presented as below. We could see that nterms=5 and nterms=8 have very close performance in terms of our six metrics and their difference is negligible. We go for a simpler model and nterms=5 is chosen. The CV results for nterms=8 and nterms=10 are the same, we guess that it might be because 8 terms have already captured enough information and adding more terms further does not help.

| Projection Pursuit Regression CV Results | | | | | | |
|---|---|---|---|---|---|---|
| Num_Terms | Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| 1 | 0.7511576 | 0.663712 | 0.4432945 | 0.5315594 | 0.2252147 | 0.5349858 |
| 2 | 0.7567762 | 0.6623839 | 0.4820121 | 0.5579808 | 0.2147455 | 0.5562177 |
| 5 | 0.7587387 | 0.663594 | 0.4918196 | 0.5649346 | 0.2118611 | 0.5588999 |
| 8 | 0.7587483 | 0.6636884 | 0.4916689 | 0.5648686 | 0.2118944 | 0.55866 |
| 10 | 0.7587483 | 0.6636884 | 0.4916689 | 0.5648686 | 0.2118944 | 0.55866 |

7. Support Vector Machine

   SVM is one of the new models we fitted that was not covered explicitly in class. SVM classifier finds the best hyperplane which is the one with the largest separation of two classes, or in other words, the largest margin between two classes. We choose the hyperplane where the distance from it to the closest point on each side is maximized. Under the majority of applications, we consider soft margin classifiers, whose objective function involves maximizing the margin and penalizing the misclassification error. The higher the regularization term lambda, the larger penalty we impose on wrong classification results.

   The effectiveness of SVM largely depends on the selection of kernel function, and other hyperparameters. A kernel function will map the original features into high-dimensional features space. A common choice is the radial basis function, which gives higher weights to the observation near the center of distribution and lower weights to those distant observations. The tuning parameter gamma has the range from 0.04 to 6 with a step size 1, and the cost parameters range from 0.04 to 6 with a step size 1. We perform grid search and select the best hyperparameters' combination that maximizes the objective function and other metrics.

The major drawback of SVM is that it will only return a binary class prediction, rather a probability value. Also, the computation is expensive and it is difficult to train SVM models when the dataset is very large. Furthermore, SVM would not be a good choice when we want to interpret the feature importance and inferential results.

In the process of fitting SVM models, due to the magnitude of our dataset, we found that even a single kernel with parameter tuning had not stopped running after two days. Thus finally we tried radial basis kernel and sigmoid kernel, with 10-fold cross validation with default parameters. The results are presented as below. Radial kernel has much better results than sigmoid kernel.

| Support Vector Machine CV Results | | | | | | |
|---|---|---|---|---|---|---|
| Kernel | Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| Radial | 0.7768 | 0.7353 | 0.4677 | 0.5717 | 0.2126 | 0.4377 |
| Sigmoid | 0.6183 | 0.401 | 0.4022 | 0.4016 | 0.2798 | 1.1471 |

8. Naive Bayes Classification

We also selected Naive Bayes model as a supplementary model not covered in the lectures. The key advantage of Naive Bayes is simple and fast to compute the prediction result. The key model assumption is that the features are independently distributed conditional on a given class. This bayes classifier considers each of the features contributing independently to the probability of severity and non-severity in this case, regardless any correlation among predictors. Even though this conditional independence assumption is hard to satisfy in reality, Naive Bayes classifier is always a good choice for binary classification with its desirable Bayes probabilistic structure and computational efficiency for large dataset.

| Naïve Bayes Classifier CV Result | | | | | |
|---|---|---|---|---|---|
| Correct Classification rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| 0.6885 | 0.2910 | 0.9780 | 0.4486 | 0.9559 | 0.7137 |

## Final Model Selection

Once we had the best tuned hyperparameter combination for each of the modeling techniques discussed above, we compare them according to the performance metrics discussed above through 3 replicates of 10-fold cross validation. The final model is selected keeping in mind both prediction accuracy as well as computational expenditure.

The following table illustrates the cross validation results for each of the contender models,

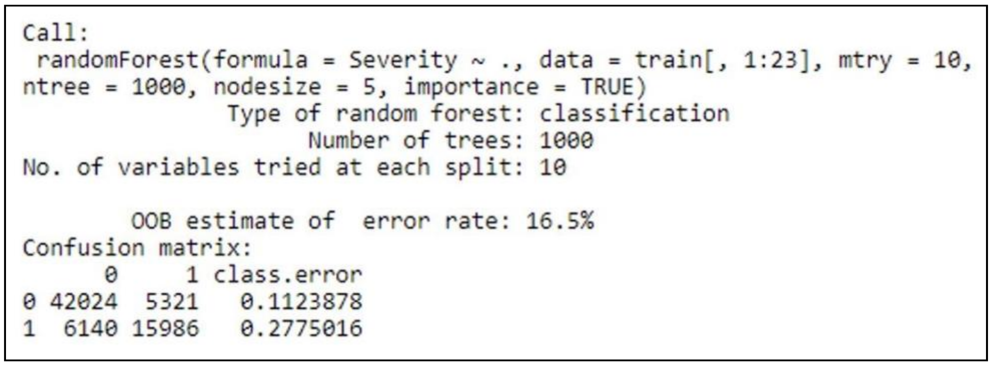
| Summary of Cross Validation Results | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Model | Parameter 1 | Parameter 2 | Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
| Logistic | - | - | 0.7639 | 0.6909 | 0.468 | 0.558 | 0.2161 | 0.5029 |
| Classification Tree | 0.000243 | - | 0.8048 | 0.6939 | 0.6926 | 0.6932 | 0.1428 | 0.5868 |
| Random Forest | 5 | - | 0.8344 | 0.7495 | 0.7211 | 0.7350 | 0.1126 | 0.5210 |
| GBM | 0.1 | 18235 | 0.8373 | 0.7603 | 0.7142 | 0.7365 | 0.1052 | 0.5269 |
| Neural Network | 5 | 0.01 | 0.7857 | 0.6999 | 0.5728 | 0.6301 | 0.1147 | 0.7584 |
| PPR | 5 | - | 0.7587 | 0.6636 | 0.4918 | 0.5649 | 0.2119 | 0.5589 |
| SVM | Radial | - | 0.7768 | 0.7353 | 0.4677 | 0.5717 | 0.2126 | 0.4377 |
| Naïve Bayes | - | - | 0.6885 | 0.291 | 0.978 | 0.4486 | 0.9559 | 0.7137 |

As can be seen here, tree based methods are performing the best with almost comparable accuracy metrics. Even though GBM shows slightly better metric values than the rest, it is building ~18000 trees to provide this marginal gain in predictive power. Hence, keeping in view the immense difference in the computation expense, we decide to choose Random Forest as our best model. Additionally, RF performs the best on our custom objective function.

## Final Model Results

### Training Final Model

As discussed above, we select Random Forest with node size 5 and number of trees 1000 as our best model. At this stage we go back to our original training set of 69,471 rows and refit the selected model. Please find below the results pertaining to the final model,

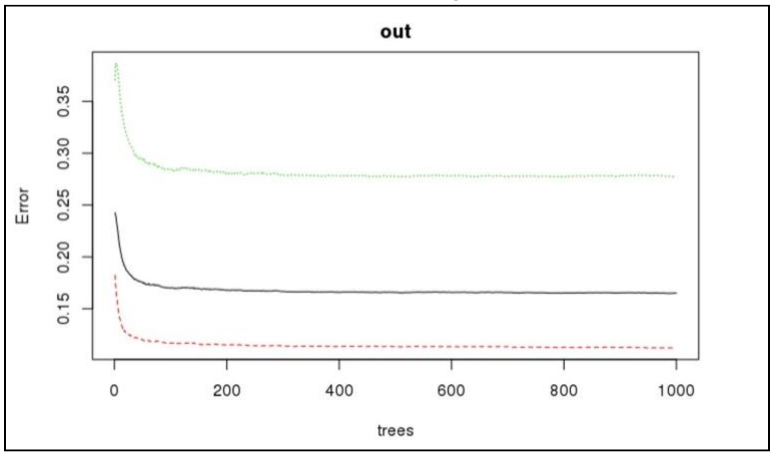


```
Call:
 randomForest(formula = Severity ~ ., data = train[, 1:23], mtry = 10,
ntree = 1000, nodesize = 5, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 10

        OOB estimate of  error rate: 16.5%
Confusion matrix:
      0     1 class.error
0 42024  5321   0.1123878
1  6140 15986   0.2775016
```

We can see from the above results that the Out of Bag mis-classification rate is 16.5%, which is synonymous to the cross validation mis-classification rate.

The plot below shows that the OOB error rate converges with 1000 trees,



### Scoring Test Sample

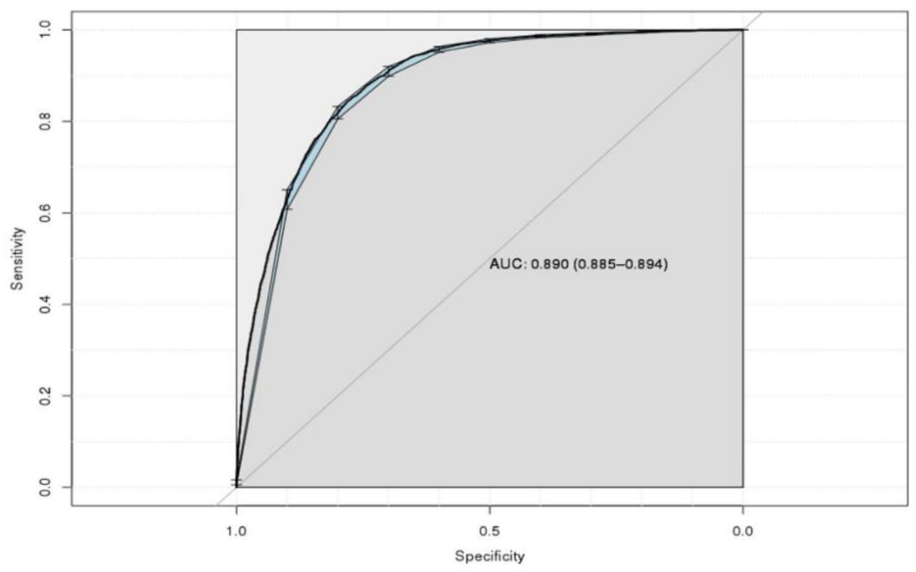
We had initially set aside 20% of the data to serve as a hold-out sample to validate our final model on. At this stage we score this test sample with the model trained above and obtain the following accuracy metrics.

| Model Performance on Test Data |
|---|

| Correct Classification Rate | Precision | Recall | F1 Score | False Positive Rate | Custom Objective Function |
|---|---|---|---|---|---|
| 0.8155 | 0.7414 | 0.6458 | 0.6903 | 0.1053 | 0.6357 |

As can be seen, even though the accuracy metrics are slightly worse than what we saw during training, our model does considerably well in correctly classifying 82% of the accidents as severe or non-severe.

To further examine the predictive power of the model, we plot the Receiver Operating Characteristic (ROC) curve on the test data as shown below,

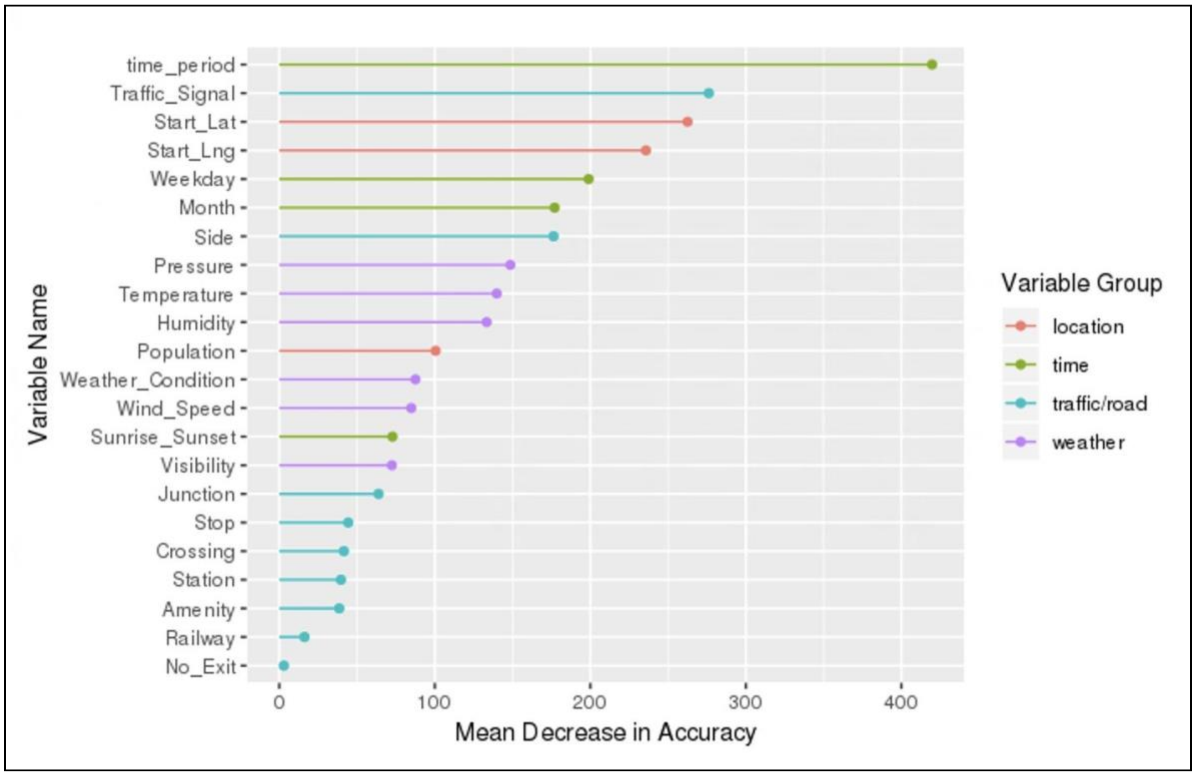


The ROC curve plots Sensitivity which is the True Positive Rate against 1-Specificity which is the False Positive Rate. This curve is used to assess the performance of the binary classifier for the entire range [0,1] of possible prediction cut-off probabilities. The steeper the ROC curve, the better the performance of the model as it means, the model can capture most of the events (severe accidents in our case) without incorrectly labeling too many of the non-events (i.e., non severe accidents).

The Area Under ROC curve (AUC) is often used as a metric to quantify the discriminating power of the binary classifier. For our model we get a AUC of 0.89 which is quite high. Also, AUC lies within the range [0.885 - 0.894] with 95% confidence, so it is stable as well.

<u>Model Interpretation & Insights</u>

The most significant predictors are as follows,

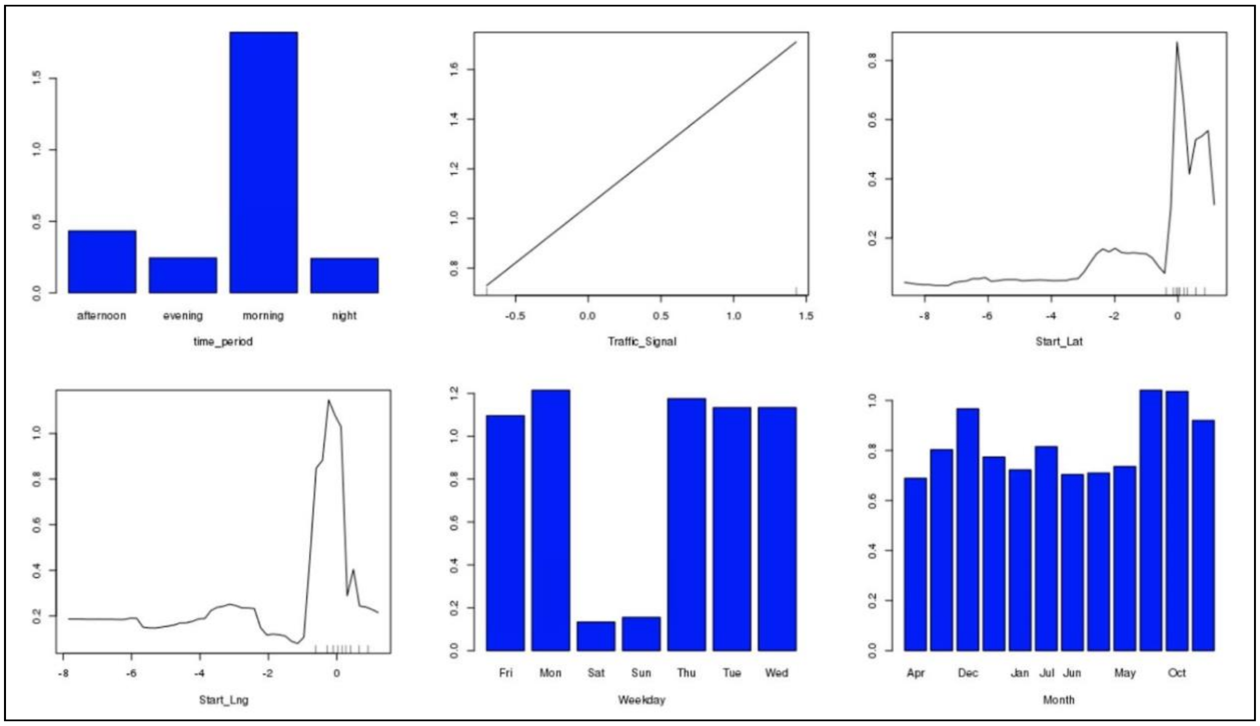


We grouped the variables into the following 4 categories of information to aid model interpretation,

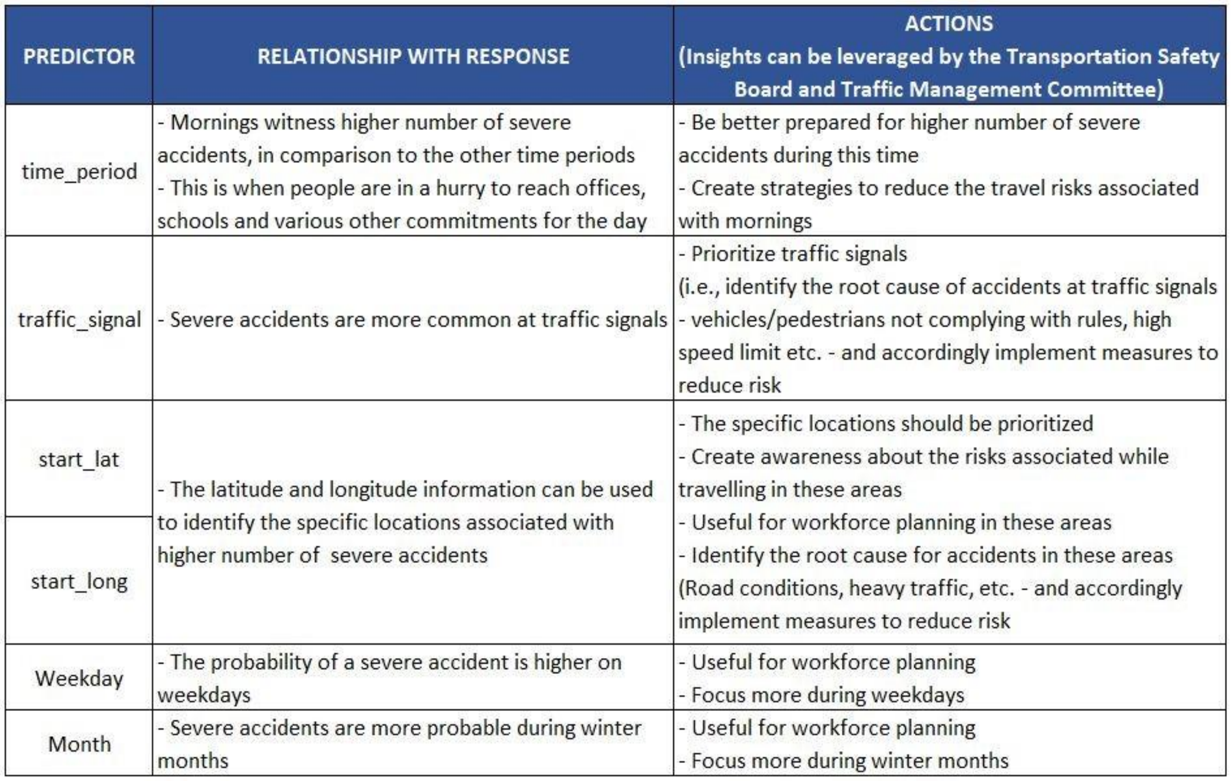
1. Location - start_lat, start_lng, population
2. Time - time of day, weekday, month, sunrise_sunset 3. Traffic/Road - traffic_signal, side of road, junction stop etc.

4. Weather - pressure, temperature, humidity, visibility etc.

Among the top 5 predictors we have time of day of the accident, traffic_signal, location (start_lat, start_long) and weekday. The next step is to create partial dependence plots, analyze their relationship with the response variable and accordingly provide actionable recommendations for a safer Illinois.



## Insights and Actions



| PREDICTOR | RELATIONSHIP WITH RESPONSE | ACTIONS (Insights can be leveraged by the Transportation Safety Board and Traffic Management Committee) |
|---|---|---|
| time_period | - Mornings witness higher number of severe accidents, in comparison to the other time periods<br>- This is when people are in a hurry to reach offices, schools and various other commitments for the day | - Be better prepared for higher number of severe accidents during this time<br>- Create strategies to reduce the travel risks associated with mornings |
| traffic_signal | - Severe accidents are more common at traffic signals | - Prioritize traffic signals<br>(i.e., identify the root cause of accidents at traffic signals - vehicles/pedestrians not complying with rules, high speed limit etc. - and accordingly implement measures to reduce risk |
| start_lat | - The latitude and longitude information can be used to identify the specific locations associated with higher number of severe accidents | - The specific locations should be prioritized<br>- Create awareness about the risks associated while travelling in these areas<br>- Useful for workforce planning in these areas<br>- Identify the root cause for accidents in these areas (Road conditions, heavy traffic, etc. - and accordingly implement measures to reduce risk |
| start_long | | |
| Weekday | - The probability of a severe accident is higher on weekdays | - Useful for workforce planning<br>- Focus more during weekdays |
| Month | - Severe accidents are more probable during winter months | - Useful for workforce planning<br>- Focus more during winter months |

**APPENDIX**

**Section 1**: **Codes for Data Pre-processing & Cleaning**

```python cd"/team/PA2/Shreyashi"
```

```python
## Import numpy, pandas, matplotlib.pyplot
import numpy as np import pandas as pd
import matplotlib.pyplot as plt
```

*## Reading in data* ```python acc =
pd.read_csv("US_Accidents_Dec19.csv")

```python acc.head()
```

```python acc.columns
```

*## Filtering out in scope observations - Illinois*
```python
## Filtering out IL - in scope test =
acc[acc['State']=='IL'].reset_index()
test.head() Test.shape
#rows - 86390
#columns - 50 acc
= test
```

```python print(acc['Start_Time'].min(axis=0))
print(acc['Start_Time'].max(axis=0))
```

*## Creating features -      1.*
*time duration of accidents*
   *2. Month, day and weekday of accident*
```python
## extract year, month day, hour, weekday, and time to clear accident acc["Start_Time"]
= pd.to_datetime(acc["Start_Time"],errors="coerce") acc["End_Time"] =
pd.to_datetime(acc["End_Time"],errors="coerce")
## extract year, month day, hour  acc["Year"] = acc["Start_Time"].dt.year
acc['Month']=acc['Start_Time'].dt.strftime('%b')
acc['Day']=acc['Start_Time'].dt.day acc['Hour']=acc['Start_Time'].dt.hour
acc['Weekday']=acc['Start_Time'].dt.strftime('%a') td =
"Time_Duration(min)" acc[td] = round((acc["End_Time"]-
acc["Start_Time"])/np.timedelta64(1,"m"))
```

## *Mapping county-year wise population*
```python
#Making County names uniform  acc =
acc.replace(to_replace='Dekalb',
value='DeKalb').replace(to_replace='Dupage',
value='DuPage').replace(to_replace='La Salle',
value='LaSalle').replace(to_replace='Dewitt',
value='De Witt').replace(to_replace='Mclean',
value='McLean').replace(to_replace = 'Saint Clair',
value = 'St.Clair').replace(to_replace = 'Mchenry',
value = 'McHenry').replace(to_replace = 'Henry',
value = 'McHenry').replace(to_replace = 'St. Clair',
value = 'St.Clair').replace(to_replace = 'Rock Island',
value = 'RockIsland')
```
```python df1 =
pd.Series(acc['County']).value_counts().reset_index().sort_values('index').reset_index(drop=Tr
u e) df1.columns = ['County', 'Frequency'] print (df1)
```
```python
#Import population data pop =
pd.read_csv("County_year_pop.csv")
pop.head()
```
```python acc_1 = pd.merge(acc,pop, how='left',
left_on=['County','Year'], right_on=['County','Year']) acc =
acc_1 acc.head()
```
```python acc[acc['Population'].isnull()][['County','Year','Population']]
# No missing population
```
### *Clubbing Weather Condition into intuitive categories*
```python acc_1 = acc.replace(to_replace=['Blowing Snow','Heavy
Snow','Light Snow',                    'Light Snow / Windy','Snow','Snow /
Windy','Snow and Sleet'],
value='Snow').replace(to_replace=['Drizzle','Heavy Drizzle',
                    'Heavy Rain','Heavy Rain / Windy','Light Drizzle',
                    'Light Drizzle / Windy','Light Freezing Drizzle',
                    'Light Freezing Rain','Light Ice Pellets','Light Rain'
                    'Light Rain / Windy','Light Rain Showers','Small Hail',
                    'N/A Precipitation','Rain','Rain / Windy',
                    'Wintry Mix','Wintry Mix / Windy','Light Rain',
```

                    'Light Rain / Windy','Rain'],
value='Rain').replace(to_replace=['Cloudy','Cloudy / Windy',                    'Mostly
Cloudy','Mostly Cloudy / Windy','Overcast',                    'Partly Cloudy','Partly Cloudy /
Windy','Scattered Clouds'],
value='Cloudy').replace(to_replace=['Clear','Fair','Fair / Windy'],
value='Clear').replace(to_replace=['Light Rain with Thunder','Heavy T-Storm',
'Heavy T-Storm / Windy','Heavy Thunderstorms and Rain','Light Thunderstorms and Rain',
                    'Light Thunderstorms and Snow','Squalls',
                    'Squalls / Windy','T-Storm','T-Storm / Windy'
                    'Thunder','Thunder / Windy','Thunder in the Vicinity',
                    'Thunderstorm','Thunderstorms and Rain','T-Storm / Windy','Thunder'],
value='Thunderstorm').replace(to_replace = ['Fog','Haze','Haze / Windy',
                    'Light Freezing Fog','Mist','Patches of Fog','Fog / Windy',
                    'Shallow Fog','Smoke'], value = 'Fog')
~~~
```python acc = acc_1 df1 =
pd.Series(acc_1['Weather_Condition']).value_counts().reset_index().sort_values('index').reset_i
ndex(drop=True) df1.columns = ['Weather_Condition', 'Frequency'] print (df1)
~~~

### *Noted some data discrepancy where some zipcodes were beginning with 4 or 5 (which do not correspond to Illinois) - excluded them*

```python
wrong_zip = ['4','5'] acc =
acc[~acc['Zipcode'].astype(str).str[0].isin(wrong_zip)]
```

## *Categorical Variables*
*Comparing Sunrise_Sunset, Civil_Twilight, Nautical_Twilight and Astronomical_Twilight to choose one*

~~~python
## Sunrise_Sunset print(acc['Sunrise_Sunset'].unique()) df1 =
pd.Series(acc['Sunrise_Sunset']).value_counts().reset_index().sort_values('index').reset_index(
d rop=True) df1.columns = ['Sunrise_Sunset', 'Frequency'] print (df1) df2 =
pd.crosstab(acc['Sunrise_Sunset'],acc['Severity']) print(df2) ```
```python ## Civil_Twilight print(acc['Civil_Twilight'].unique()) df1 =
pd.Series(acc['Civil_Twilight']).value_counts().reset_index().sort_values('index').reset_index(dr
o p=True) df1.columns = ['Civil_Twilight', 'Frequency'] print (df1) df2 =
pd.crosstab(acc['Civil_Twilight'],acc['Severity']) print(df2) ```
```python
## Nautical_Twilight print(acc['Nautical_Twilight'].unique()) df1 =
pd.Series(acc['Nautical_Twilight']).value_counts().reset_index().sort_values('index').reset_index
( drop=True) df1.columns = ['Nautical_Twilight', 'Frequency'] print (df1) df2 =
pd.crosstab(acc['Nautical_Twilight'],acc['Severity']) print(df2) ```
```python
~~~

## Astronomical_Twilight print(acc['Astronomical_Twilight'].unique())
df1 =
pd.Series(acc['Astronomical_Twilight']).value_counts().reset_index()
.sort_values('index').reset_i ndex(drop=True) df1.columns =
['Astronomical_Twilight', 'Frequency'] print (df1) df2 =
pd.crosstab(acc['Astronomical_Twilight'],acc['Severity']) print(df2)

Sunrise_Sunset, Civil_Twilight, Nautical_Twilight, Astronomical_Twilight all have very similar values and distributions. Decided to retain only 'Sunrise_Sunset' as that is more balanced. Sunrise_Sunset has 0.001% missing (1 obs) - imputing with more frequent value i.e., Night
```python acc = acc.drop(['Civil_Twilight', 'Nautical_Twilight',
'Astronomical_Twilight'], axis=1) acc['Sunrise_Sunset'].fillna('Night',inplace=True)
```

*## Binarizing target*
```python acc.loc[acc['Severity'] == 1, 'Severity'] = 0 acc.loc[acc['Severity'] == 2, 'Severity'] = 0
acc.loc[acc['Severity'] == 3, 'Severity'] = 1 acc.loc[acc['Severity'] == 4, 'Severity'] = 1 df1 =
pd.Series(acc['Severity']).value_counts().reset_index().sort_values('index').reset_index(drop=Tr
ue) df1.columns = ['Severity', 'Frequency'] print(df1)
```

## Section 2: Codes for Exploratory Data Analysis

```python acc.describe().transpose()
```

*## Dropping variables*

1. *Redundant variables - state, country, latitude, longitude etc.*
2. *Variables with >50% missing observations* ```python acc.columns

```python
## Dropping redundant variables acc = acc.drop(['index','ID', 'Source',
'TMC', 'Start_Time', 'End_Time',                'End_Lat', 'End_Lng',
'Description', 'Number', 'State', 'Street',                'Country',
'Timezone', 'Airport_Code', 'Weather_Timestamp', 'Year', 'Day',
            'Bump','Give_Way','Roundabout','Traffic_Calming','Turning_Loop',
            'Wind_Direction'], axis=1)
```

```python acc.isnull().sum()/acc.shape[0]*100
```

```python
# Dropping fields with high missing values (>50%) acc =
acc.drop(['Wind_Chill(F)','Precipitation(in)'], axis=1)
```

## Finding ouliers     outlier >
mean + 3*std_dev

```python
# Finding ouliers stats =
acc.describe().transpose() stats['outlier_max'] =
stats['mean'] + 3*stats['std'] stats['outlier_min'] =
stats['mean'] - 3*stats['std'] stats.loc[stats['min'] <
stats['outlier_min'], 'outlier'] = 1 stats.loc[stats['max'] >
stats['outlier_max'], 'outlier'] = 1 stats
```

```python
# Capping duration to mean + 3*sd
# Not capping other weather related vars as these might be meaningful extreme weather days
mean = np.mean(acc[td]) std = np.std(acc[td]) n = 3 outliers = (acc[td]-mean).abs() > std * n
acc[td][outliers] = np.nan acc[td].fillna(mean+n*std,inplace=True)
```

```python
# Capping Distance to mean + 3*sd mean =
np.mean(acc['Distance(mi)']) std =
np.std(acc['Distance(mi)']) n = 3 outliers =
(acc['Distance(mi)']-mean).abs() > std * n
acc['Distance(mi)'][outliers] = np.nan
acc['Distance(mi)'].fillna(mean+n*std,inplace=True)
```

## Missing Imputation

*Only weather related variables have missing observations.*
*We intended to use MICE to impute them.*
*Since we will be doing cross validation, so missing imputation should ideally be done separately for each fold.*
*In the interest of time, doing a course imputation with median value according to the city*

```python
acc.isnull().sum()
```

```python
city_weather = acc.groupby('City').agg({'Temperature(F)':
'median',
                                        'Humidity(%)':'median',
'Pressure(in)':'median',
                                        'Visibility(mi)':'median',
                                        'Wind_Speed(mph)':'median'},                    dropna=True)
city_weather.reset_index().head()
```

```python
acc = acc.join(city_weather, on='City', how='left', lsuffix='', rsuffix='_city')
acc['Temperature(F)'] = acc['Temperature(F)'].fillna(acc['Temperature(F)_city'])
acc['Humidity(%)'] = acc['Humidity(%)'].fillna(acc['Humidity(%)_city']) acc['Pressure(in)'] =
```

acc['Pressure(in)'].fillna(acc['Pressure(in)_city']) acc['Visibility(mi)'] =
acc['Visibility(mi)'].fillna(acc['Visibility(mi)_city']) acc['Wind_Speed(mph)'] =
acc['Wind_Speed(mph)'].fillna(acc['Wind_Speed(mph)_city']) acc =
acc.drop(['Temperature(F)_city','Humidity(%)_city','Pressure(in)_city','Visibility(mi)_city','Wind_
S peed(mph)_city'], axis=1) acc.isnull().sum()
~~~
### Imputing categorical variables with mode as per city
```python def foo(x): m = pd.Series.mode(x); return m.values[0] if not m.empty
else np.nan  check = acc[['City','Weather_Condition']] city_cat =
check.groupby('City').agg(foo) city_cat.head()
~~~
```python acc = acc.join(city_cat, on='City', how='left', lsuffix='', rsuffix='_city')
acc['Weather_Condition'] =
acc['Weather_Condition'].fillna(acc['Weather_Condition_city']) acc =
acc.drop(['Weather_Condition_city'], axis=1) acc.isnull().sum()
```
```python
## Imputing the remaining few missing observations with overall mean and mode
acc = acc.fillna(acc.mean()) acc['Weather_Condition'] =
acc['Weather_Condition'].fillna(acc['Weather_Condition'].mode()[0]) acc.isnull().sum()
```
## *Splitting data into training and test (80:20 ratio)*
```python # set X
and y y =
acc['Severity']
X = acc.drop('Severity', axis=1)
# Split the data set into training and testing data sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21, stratify=y)
```
Verifying equal distribution of target levels in training and test data
```python df1 =
pd.Series(acc['Severity']).value_counts().reset_index().sort_values('index').reset_index(drop=Tr
ue) df1.columns = ['Severity', 'Frequency'] df1['Proportion'] =
df1['Frequency']/sum(df1['Frequency'])*100 df1
```
```python df1 =
y_train.value_counts().reset_index().sort_values('index').reset_index(drop=True)
df1.columns = ['train', 'Frequency'] df1['Proportion'] =
df1['Frequency']/sum(df1['Frequency'])*100 df1
```
```python df1 =
y_test.value_counts().reset_index().sort_values('index').reset_index(drop=True)
```

~~~
df1.columns = ['test', 'Frequency'] df1['Proportion'] =
df1['Frequency']/sum(df1['Frequency'])*100 df1
```
Appending X's and Y to create 'train' and 'test' data
```python train = X_train train['Severity'] =
y_train train.reset_index(drop=True,
inplace=True) train.head()
```
```python test
= X_test
test['Severity'
] = y_test
test.reset_in
dex(drop=Tr
ue,
inplace=True
) test.head()
```
```python train.to_csv("final_train.csv")
```
```python test.to_csv("final_test.csv")
```
~~~

**Section 3: Data Transformation**

```
library(ggplot2)
library(tidyr)
library(dplyr)

#Create distributions of all variables and log transform Population
num_data <-train[, sapply(train, is.numeric)]
Pop <- data.frame(train[,c("Population")])
ggplot(Pop) +
geom_histogram(bins = 10) +
facet_wrap(~key, scales = 'free_x')

ggplot((train["Population"]))
data = data.frame(train$Population)



ggplot(data, aes(x=log(train.Population))) +
  geom_histogram()
```

```
# Binarize categorical predictors
train = train %>% mutate(Side = ifelse(Side == "R" ,1,0)) %>%
  mutate(Amenity = ifelse(Amenity == "True", 1,0), Crossing = ifelse(Crossing == "True", 1,0),
Junction = ifelse(Junction == "True", 1,0), No_Exit = ifelse(No_Exit == "True", 1,0), Railway =
ifelse(Railway == "True", 1,0), Station = ifelse(Station == "True", 1,0), Stop = ifelse(Stop ==
"True", 1,0), mutate(time_period = ifelse(Hour < 12 & Hour > 5, "morning", ifelse(Hour < 18 &
Hour >= 12, "afternoon", ifelse(Hour < 21 & Hour >= 18,"evening","night"))))

#standardize predictors
train[,c(1,4:8,22)]<-sapply(train[,c(1,4:8,22)], function(x) (x-mean(x))/sd(x))

#standardize target for NN
train$sev_std <- (train[,23]-min(train[,23]))/(max(train[,23])-min(train[,23]))

#Converting target to factor for tree based methods
train$Severity = as.factor(train$Severity)
```

## Section 4: Codes for Candidate Models and Cross Validation

```
library( nnet)
library( gbm)
library( dplyr)
library( ggplot2)
library( randomForest)
#install.packages("pROC")
library( ROCR)
library( pROC)

CVInd <- function(n,K) {
  # n is sample size; K is number of parts;
  # returns K-length list of indices for each part
  m<-floor(n/K) #approximate size of each part
  r<-n-m*K
  I<-sample(n,n) #random reordering of the indices
  Ind<-list() #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k- 1) +1): ((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1) :((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart] #indices for kth part of data
  }
  Ind
}

metric =   function(y,yhat){
          x=table(y,yhat)
          if(ncol(x)==1)
          {
            x <- cbind(x,c(0,0))
            colnames(x)<-c(0,1)
          }
       misclass = sum(diag(table(y,yhat)))/sum(table(y,yhat))
        TP = x[2,2]
        TN = x[1,1]
        FP = x[1,2]
        FN = x[2,1]
        precision =TP/(TP + FP)
        recall = TP/(TP + FN)
        f1 = 2*precision * recall/(precision + recall)
        FPR = FP/(FP + TN)
        obj = (5*FN + FP)/length(y)
        return(c(misclass,precision,recall,f1,FPR,obj))
}
```

```
# Reading in Training data
train = read.csv("FinalDatasets/train_transform.csv")
head( train)
colnames( train)
summary(train)
```

## Reading in Training Data

## Cross Validation - Logistic Regression

```
n = nrow(train_transform)
yhat = matrix(0, n,3)
set.seed(123)
for (j in 1:3){
  Ind<-CVInd(n,10)
  for (i in 1:10) {
    train = train_transform[-Ind[[i]],]
    test = train_transform[Ind[[i]],1:22]
    model = glm(Severity~., train, family = "binomial")
    yhat[Ind[[i]],j] = ifelse(predict(model, test, type = "response") >= 0.5, 1,0)
    }
}

apply(apply(yhat, 2,function(x)metric(train_transform[1:n,23],x)), 1,mean)
model1 = glm(Severity~., train_transform, family = binomial)
summary(model1)
```

```r
#Inbuilt CV on whole train to tune Shrinkage and Number of Trees
set.seed(123)
# Shrinkage Values for GBM
#shrink <- c(0.1, 0.3, 0.5)
#shrink <- c(0.01, 0.05)
shrink <- c(0.1)
# n.tree for GBM
#ntree <- c(10000, 9000, 8000, 7000)
#ntree <- c(50000, 20000)
ntree <- c(20000)
n.models <- 1
n=nrow(train)
best.iter_GBM <- matrix(0,n.models,1)
CV.deviance_GBM <- matrix(0 ,n.models,1)
counter <- 0
for(shrink_val in shrink)
{
  counter = counter+1
  out<-gbm(Severity~., data=train[,c(1:23)],
           distribution="bernoulli",
           n.trees=ntree[counter], #smaller ntree for larger shrinkage for efficiency
```

## Cross Validation - Gradient Boosting Machine

```r
           shrinkage=shrink_val,
           interaction.depth=3,
           bag.fraction = .5,
           train.fraction = 1,
           n.minobsinnode = 10,
           cv.folds = 10,
           keep.data=TRUE,
           verbose=FALSE)
  best.iter <- gbm.perf(out,method="cv")
  best.iter_GBM[counter,1] <- best.iter
  CV.deviance_GBM[counter,1] <- out$ cv.error[best.iter]
}

CV_deviance_GBM <- cbind(shrink,best.iter_GBM[,1],CV.deviance_GBM[,1])
print(CV_deviance_GBM) #Determine best shrinkage-ntree combination from here

best_ntree <- 18235
best_shrinkage <- 0.1

#Obtaining accuracy metrics of the best model as obtained above on our CV folds
set.seed(123)
n.models <- 1
Nrep<-3 #number of replicates of CV
K<-10 #K-fold CV on each replicate
n=nrow(train)
y<-train[[23]]
yhat=matrix(0,n,n.models)

CV.rate_GBM <-matrix(0,Nrep,n.models)
Prec_GBM <-matrix(0,Nrep,n.models)
Recall_GBM <-matrix(0,Nrep,n.models)
f1_GBM <-matrix(0,Nrep,n.models)
fpr_GBM <-matrix(0,Nrep,n.models)
obj_GBM <-matrix(0,Nrep,n.models)

for (j in 1:Nrep) {
 Ind<-CVInd(n,K)
 for (k in 1:K) {
       out<-gbm(Severity~., data=train[-Ind[[k]],c(1:23)],
               distribution="bernoulli",
               n.trees= best_ntree,#best ntree as per CV deviance
               shrinkage= best_shrinkage,
               interaction.depth=3,
               bag.fraction = .5,
               train.fraction = 1,
               n.minobsinnode = 10,
               keep.data=TRUE,
               verbose=FALSE)

       phat<-predict.gbm(out,train[Ind[[k]],c(1:23)],n.trees= best_ntree,type='response');
       yhat[Ind[[k]],1]<-as.numeric(phat >= 0.5)
  } #end of k loop

   CV.rate_GBM[j,]= apply(yhat,2, function(x) metric(y,x)[1])
   Prec_GBM[j,]= apply(yhat,2, function(x) metric(y,x)[2])
   Recall_GBM[j,]= apply(yhat,2, function(x) metric(y,x)[3])
   f1_GBM[j,]= apply(yhat,2,function(x) metric(y,x)[4])
   fpr_GBM[j,]= apply(yhat,2,function(x) metric(y,x)[5])
   obj_GBM[j,]= apply(yhat,2,function(x) metric(y,x)[6])

} #end of j loop
CV.rateAve_GBM<- apply(CV.rate_GBM,2,mean);  #averaged CV misclass rate
CV.rateSD_GBM <- apply(CV.rate_GBM,2,sd);  #SD of CV misclass rate
PrecAve_GBM <- apply(Prec_GBM,2,mean);  #averaged CV Precision
RecallAve_GBM <- apply(Recall_GBM,2,mean);  #averaged CV Recall
f1Ave_GBM <- apply(f1_GBM,2,mean);  #averaged CV f1 score
fprAve_GBM <- apply(fpr_GBM,2,mean);  #averaged CV FPR
objAve_GBM <- apply(obj_GBM,2,mean);  #averaged CV obj score
CV_stats_GBM <- cbind(best_shrinkage,best_ntree,CV.rateAve_GBM,PrecAve_GBM,
                    RecallAve_GBM,f1Ave_GBM,fprAve_GBM,objAve_GBM)
print(CV_stats_GBM)
```

### Cross Validation - Neural Network

```
# Grid search for neural network
size_vals <- c(5,10,20)
decay_vals <- c(0.01, 0.05, 0.1, 0.5, 1)
n.models = 15 #number of different models to fit

set.seed(123)

Nrep<-3 #number of replicates of CV
K<-10 #K-fold CV on each replicate
```

```
n=nrow(train)
y<-train[[24]]
yhat=matrix(0,n,n.models)

CV.rate_NN <-matrix(0,Nrep,n.models)
Prec_NN <-matrix(0,Nrep,n.models)
Recall_NN <-matrix(0,Nrep,n.models)
f1_NN <-matrix(0,Nrep,n.models)
fpr_NN <-matrix(0,Nrep,n.models)
obj_NN <-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
 Ind<-CVInd(n,K)
 for (k in 1:K) {
    counter <- 0
     for(size_val in size_vals)
     {
       for(decay_val in decay_vals)
       {
        counter = counter+1
        out<-nnet(sev_std~.,train[-Ind[[k]],c(1:22,24)],linout=F,skip=F,size=size_val,
                  decay=decay_val, maxit=1000,trace=F,MaxNWts = 84581)
        phat<-as.numeric(predict(out,train[Ind[[k]],c(1:22,24)]));
        yhat[Ind[[k]],counter]<-as.numeric(phat >= 0.5)
       }
     }
 } #end of k loop
 CV.rate_NN[j,]= apply(yhat,2,function(x) metric(y,x)[1])
 Prec_NN[j,]= apply(yhat,2,function(x) metric(y,x)[2])
 Recall_NN[j,]= apply(yhat,2,function(x) metric(y,x)[3])
 f1_NN[j,]= apply(yhat,2,function(x) metric(y,x)[4])
 fpr_NN[j,]= apply(yhat,2,function(x) metric(y,x)[5])
 obj_NN[j,]= apply(yhat,2,function(x) metric(y,x)[6])
} #end of j loop

CV.rateAve_NN<- apply(CV.rate_NN,2,mean);  #averaged CV misclass rate
CV.rateSD_NN <- apply(CV.rate_NN,2,sd);  #SD of CV misclass rate

PrecAve_NN <- apply(Prec_NN,2,mean);  #averaged CV f1 score
RecallAve_NN <- apply(Recall_NN,2,mean);  #averaged CV f1 score
f1Ave_NN<- apply(f1_NN,2,mean);  #averaged CV f1 score
fprAve_NN<- apply(fpr_NN,2,mean);  #averaged CV f1 score
```

```
objAve_NN<- apply(obj_NN,2,mean);  #averaged CV f1 score

CV_stats_NN <-
cbind(size_vals,decay_vals,CV.rateAve_NN,PrecAve_NN,RecallAve_NN,f1Ave_NN,fprAve_NN
,objAve_NN)
CV_sd_NN <- cbind(size_vals,decay_vals,CV.rateSD_NN)
```
```

## Cross Validation - Classification Tree

```
# Step 1: Build a huge (for low complexity value) decision tree on the entire training dataset.
This will include an in-built 10 fold cross validation.
# Step 2: For the optimal cp value, test the accuracy on our 10 folds for consistency with the
other models
# Step 3: Report cv error from step 2

setwd('C:/Users/aakan/Desktop/Work/7. Q2 Courses/1. Pred Analytics 2/5. Project')
train <- read.csv('train_transform.csv')
library(rpart)
```

```
# Step 1: Build a huge (for low complexity value) decision tree on the entire training dataset.
This will include an in-built 10 fold cross validation.
control <- rpart.control(minbucket = 5, cp = 0.0001, maxsurrogate = 0, usesurrogate = 0, xval =
10)
CRT.tr <- rpart(Severity ~ .,train[,c(1:23)], method = "anova", control = control)
printcp(CRT.tr)
plotcp(CRT.tr)
write.csv(cv_tree,'cv_tree.csv')

# Step 2: After studying the cross validation errors for each cp value, it appears that for cp =
0.000242, the CV error is the lowest
# Step 3: So we will use this for cross validation on our folds and compute performance metrics

Nrep<-3 #number of replicates of CV
K<-10 #K-fold CV on each replicate
n=nrow(train)
y<-train[[23]]
n.models = 1
set.seed(123)
yhat=matrix(0,n,n.models)
CV.rate_DT <-matrix(0,Nrep,n.models)
Prec_DT <-matrix(0,Nrep,n.models)
```
```

```
Recall_DT <-matrix(0,Nrep,n.models)
f1_DT <-matrix(0,Nrep,n.models)
fpr_DT <-matrix(0,Nrep,n.models)
obj_DT <-matrix(0,Nrep,n.models)
for (j in 1:Nrep)
  {
  Ind<-CVInd(n,K)
  for (k in 1:K)
    {

        control <- rpart.control(minbucket = 5, cp = 0.0001)
        CRT.tr <- rpart(Severity ~ .,train[-Ind[[k]],c(1:23)], method = "anova", control = control)
        CRT.tr1 <- prune(CRT.tr, cp=0.000243)
        phat<-predict(CRT.tr1,train[Ind[[k]],c(1:22)])
        yhat[Ind[[k]],1]<-as.numeric(phat >= 0.5)

  } #end of k loop
  CV.rate_DT[j,]= apply(yhat,2,function(x) metric(y,x)[1])
  Prec_DT[j,]= apply(yhat,2,function(x) metric(y,x)[2])
  Recall_DT[j,]= apply(yhat,2,function(x) metric(y,x)[3])
  f1_DT[j,]= apply(yhat,2,function(x) metric(y,x)[4])
  fpr_DT[j,]= apply(yhat,2,function(x) metric(y,x)[5])
  obj_DT[j,]= apply(yhat,2,function(x) metric(y,x)[6])
} #end of j loop

CV.rateAve_DT <- apply(CV.rate_DT,2,mean);  #averaged CV misclass rate
CV.rateSD_DT <- apply(CV.rate_DT,2,sd);  #SD of CV misclass rate

PrecAve_DT <- apply(Prec_DT,2,mean);  #averaged CV f1 score
RecallAve_DT <- apply(Recall_DT,2,mean);  #averaged CV f1 score
f1Ave_DT <- apply(f1_DT,2,mean);  #averaged CV f1 score
fprAve_DT <- apply(fpr_DT,2,mean);  #averaged CV f1 score
objAve_DT <- apply(obj_DT,2,mean);  #averaged CV f1 score

CV_stats_DT <-
cbind(CV.rateAve_DT,PrecAve_DT,RecallAve_DT,f1Ave_DT,fprAve_DT,objAve_DT)
CV_sd_DT <- cbind(CV.rateSD_DT)
```

## Cross Validation - Random Forest

```
#Tuning RF using out-of-bag error rate
set.seed(123)
train$Severity <- as.factor(train$ Severity)
# nodesize for RF
node <- c(3, 5, 10, 20, 50, 100, 150, 200, 250, 300)
y<-train[[23]]
yhat=matrix(0,n,n.models)
CV.rate_RF <-matrix(0,Nrep,n.models)
Prec_RF <-matrix(0,Nrep,n.models)
Recall_RF <-matrix(0,Nrep,n.models)
f1_RF <-matrix(0,Nrep,n.models)
fpr_RF <-matrix(0,Nrep,n.models)
obj_RF <-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    out <- randomForest(Severity~.,
                     data=train[-Ind[[k]],c(1: 23)],
                     mtry=10,
                     ntree = 1000,
                     nodesize = best_node,
                     importance = TRUE)
    yhat[Ind[[k]],1]<-predict(out,train[Ind[[k]],c(1:23)])
  } #end of k loop
  CV.rate_RF[j,]= apply(yhat,2, function(x) metric(y,x)[1])
  Prec_RF[j,]= apply(yhat,2, function(x) metric(y,x)[2])
  Recall_RF[j,]= apply(yhat,2, function(x) metric(y,x)[3])
  f1_RF[j,]= apply(yhat,2,function(x) metric(y,x)[4])
  fpr_RF[j,]= apply(yhat,2,function(x) metric(y,x)[5])
  obj_RF[j,]= apply(yhat,2,function(x) metric( y,x)[6])
} #end of j loop
CV.rateAve_RF<- apply(CV.rate_RF,2,mean);  #averaged CV misclass rate
CV.rateSD_RF <- apply(CV.rate_RF,2,sd);  #SD of CV misclass rate
PrecAve_RF <- apply(Prec_RF,2,mean);  #averaged CV Precision
RecallAve_RF <- apply(Recall_RF,2,mean);  #averaged CV Recall
f1Ave_RF <- apply(f1_RF,2,mean);  #averaged CV f1 score
fprAve_RF <- apply(fpr_RF,2,mean);  #averaged CV FPR
objAve_RF <- apply(obj_RF,2,mean);  #averaged CV obj score
CV_stats_RF <- cbind(best_node,CV.rateAve_RF,PrecAve_RF,
                RecallAve_RF,f1Ave_RF,fprAve_RF,objAve_RF)
print(CV_stats_RF)
```

### Cross Validation - Projection Pursuit Regression

```
# Evaluation metrics
compute = function(y, yhat){
  cm = as.matrix(table(y, yhat))
  mis_rate = 1-sum(diag(cm))/sum(cm)
  p = cm[2,2]/(cm[2,2]+cm[1,2])
  r = cm[2,2]/(cm[2,2]+cm[2,1])
  f1 = 2*p*r/(p+r)
  fpr = cm[2,1]/(cm[2,1]+cm[1,1])
  cost = (cm[2,1] + 5*cm[1,2])/sum(cm)
  return(c(mis_rate, p, r, f1, fpr, cost))
}

# 3 replicates of 10-fold CV
Nrep<-3 #number of replicates of CV
K<-10 #K-fold CV on each replicate
n.models = 6 #number of different models to fit
n=nrow(train)
y<-train$Severity
phat=matrix(0,n,n.models)
yhat=matrix(0,n,n.models)
metrics=array(0, dim = c(6,n.models,3))

set.seed(123)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    out <- ppr(Severity~., data=train[-Ind[[k]],c(1:8,10:18,21,23)], nterms=1)
    phat[Ind[[k]],1]<-as.numeric(predict(out,train[Ind[[k]],]))
    out <- ppr(Severity~., data=train[-Ind[[k]],c(1:8,10:18,21,23)], nterms=2)
    phat[Ind[[k]],2]<-as.numeric(predict(out,train[Ind[[k]],]))
    out <- ppr(Severity~., data=train[-Ind[[k]],c(1:8,10:18,21,23)], nterms=5)
    phat[Ind[[k]],3]<-as.numeric(predict(out,train[Ind[[k]],]))
    out <- ppr(Severity~., data=train[-Ind[[k]],c(1:8,10:18,21,23)], nterms=8)
    phat[Ind[[k]],4]<-as.numeric(predict(out,train[Ind[[k]],]))
    out <- ppr(Severity~., data=train[-Ind[[k]],c(1:8,10:18,21,23)], nterms=10)
    phat[Ind[[k]],5]<-as.numeric(predict(out,train[Ind[[k]],]))
    out <- ppr(Severity~., data=train[-Ind[[k]],c(1:8,10:18,21,23)], nterms=15)
    phat[Ind[[k]],6]<-as.numeric(predict(out,train[Ind[[k]],]))
  } #end of k loop
  yhat[,1]=as.numeric(phat[,1]>=0.5)
  yhat[,2]=as.numeric(phat[,2]>=0.5)
  yhat[,3]=as.numeric(phat[,3]>=0.5)
  yhat[,4]=as.numeric(phat[,4]>=0.5)
  yhat[,5]=as.numeric(phat[,5]>=0.5)
  yhat[,6]=as.numeric(phat[,6]>=0.5)
  metrics[,,j] = apply(yhat, 2, function(x) compute(y, x))
} #end of j loop
apply(metrics,1:2,mean)
```

## Cross Validation - Support Vector Machine

```
library(e1071)
set.seed(123)
for (i in 1:3) {
   Ind<-CVInd(n,10)
   for (j in 1:10) {
      train = train_transform[-Ind[[j]],]
      test = train_transform[Ind[[j]],1:22]
      model = svm(factor(Severity)~., train, kernel = "radial")
      predicted1[Ind[[j]], i] = ifelse(predict(model, test) == 2, 1,0)
    }
}

set.seed(123)
for (i in 1:3) {
   Ind<-CVInd(n,10)
   for (j in 1:10) {
      train = train_transform[-Ind[[j]],]
      test = train_transform[Ind[[j]],1:22]
      model = svm(factor(Severity)~., train, kernel = "sigmoid")
      predicted2[Ind[[j]], i] = ifelse(predict(model, test) == 2, 1,0)
    }
}

metric(train_transform$Severity, apply(predicted1, 1, mean))
metric(train_transform$Severity, apply(predicted2, 1, mean))
```

## Cross-validation - Naive Bayes

```
library(e1071)
set.seed(123)
Nrep<-1 #number of replicates of CV
K<-10 #K-fold CV on each replicate
library(caret)
n.models = 1
```

```
n=nrow(train)
y = train$Severity

yhat=matrix(0,n,n.models*Nrep)

nb.results<-data.frame(misclass=double(),precision=double(),recall = double(),f1 = double(),fpr
= double(),obj = double())

for (j in 1:Nrep) {
 Ind<-CVInd(n,K)
     for (k in 1:K) {
       out<-naiveBayes(factor(Severity)~.,train[-Ind[[k]],])
       yhat[Ind[[k]],1]<-predict(out, train[Ind[[k]],], type="class")
     }
     result = metric(train$Severity,yhat[,1])
     nb.results[nrow(nn.results)+1,]<-c(result[1],result[2],result[3],result[4],result[5],result[6])
   }
```

## Section 4: Codes for Final Model Results

## Fitting Final Model on Training Data

```r
train$Severity = as.factor(train$Severity)
out <- randomForest(Severity~.,
                    data=train[,1:23],
                    mtry=10,
                    ntree = 1000,
                    nodesize = 5,
                    importance = TRUE)
plot( out)
print( out)

importance( out)
varImpPlot( out)

#Better plot of variable importance
imp <- varImpPlot(out)
imp <- as.data.frame(imp)
imp$varnames <- rownames(imp) # row names to column
rownames(imp) <- NULL
imp$var_categ <-
c("location","location","traffic/road","weather","weather","weather","weather","weather",
"weather"
,

"traffic/road","traffic/road","traffic/road","traffic/road","traffic/road","traffic/road","traffic/road",
                  "traffic/road","time","time","time","location","time") #var category

ggplot(imp, aes(x=reorder(varnames, MeanDecreaseAccuracy),
               y=MeanDecreaseAccuracy, color=as.factor(var_categ))) +
  geom_point() +
  geom_segment(aes(x=varnames,xend=varnames,y=0,yend=MeanDecreaseAccuracy)) +
  scale_color_discrete(name="Variable Group") +
  ylab("Mean Decrease in Accuracy") +
  xlab("Variable Name") +
  coord_flip()

print(imp)
```

```
#Transforming test data
test = read.csv("FinalDatasets/final_test.csv",stringsAsFactors = FALSE)
head( test)
colnames( test)
summary( test)
#Take log transform of predictors that are right skewed
test$Population <- log(test$Population+1)
```

## Model performance on test data

```
test$Time_Duration <- log(test$Time_Duration+1)

# Binarize categorical predictors
test = test %>% mutate(Side = ifelse(Side == "R" ,1,0)) %>%
  mutate(Amenity = ifelse(Amenity == TRUE, 1,0), Crossing = ifelse(Crossing == TRUE, 1,0),
Junction = ifelse(Junction == TRUE,  1,0), No_Exit = ifelse(No_Exit == TRUE, 1, 0), Railway =
ifelse(Railway == TRUE, 1,0), Station = ifelse(Station == TRUE, 1,0), Stop = ifelse(Stop ==
TRUE, 1,0), Traffic_Signal = ifelse(Traffic_Signal == TRUE, 1,0), Sunrise_Sunset =
ifelse(Sunrise_Sunset == "Day", 1,0))

# Converting to factor
cols <- c( "Weather_Condition",
          "Month","time_period","Weekday")
test[cols] <- lapply(test[cols], factor)

#standardize predictors including the binary variables created above
test[,c(1:4,6:10,12:20,23:24)]<-sapply(test[,c(1:4,6:10,12:20,23:24)], function(x)
(x-mean(x))/sd(x))

write.csv(test,'test_transform.csv')
predict.randomForest
yhat<-predict(out,test)
metric(test[,26],yhat)

#ROC Curve
yhat <- predict(out, test, type='prob')
pROC_obj <- roc(test$Severity,yhat[,2],
            smoothed = TRUE,
            # arguments for ci
            ci=TRUE, ci.alpha=0.95,  stratified=FALSE,
            # arguments for plot
            plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
            print.auc=TRUE, show.thres=TRUE)

sens.ci <- ci.se(pROC_obj)
plot(sens.ci, type="shape", col="lightblue")
## Warning in plot.ci.se(sens.ci, type = "shape", col = "lightblue"): Low
## definition shape.
plot(sens.ci, type="bars")
```

```
names( train)
#Partial Dependence Plots
par(mfrow=c(2,3))
for (i in c(22,17,1,2,20,19)) partialPlot(out,
                                           pred.data=train,
                                           x.var = names(train)[i],
```

## Model Interpretation & Insights

```
                                           xlab = names(train)[i],
                                           main=NULL)
```