Analyzing collision events in Seattle and predicting severe occurrences

Sayan Das

1. Introduction and background

The safety of drivers, passengers, pedestrians and property is an ever-growing concern when it comes to the operation of vehicles on the road. With **6,452,000 motor vehicle crashes** and **37,133 crash-related deaths** in 2017 alone, significant costs in terms of life, money and property are incurred as a result of collisions each year.

To make things more complicated, the number of vehicles in the US keeps growing every year, creating a need to adopt greater safety measures on the road. Insurance companies play a significant role in minimizing and handling automobile collisions, and they do so through services provided by charging insurance premiums. These insurance rates could be more accurately predicted based on factors such as location, weather, road dimensions, speed of traffic, date/time, whether the driver was DUI etc., most of which are dependent on the location.

In this report, we use collision data from the city of Seattle to implement a machine learning model to predict the severity of collisions using the above variables and others. Furthermore, we explore the variables in detail to find patterns and insights related to the geography of the area, trends across time and traffic variables that affect collisions.

Some of the biggest insurers in the US such as *Statefarm*, *Esurance* and *Allstate* have stated location as one of their top criteria for determining rates. These and other insurance companies could use this model to determine more accurate rates for their customers. Moreover, they could also provide useful information to their clients such as looking out for red flags when buying expensive cars in more accident prone locations which would likely increase their insurance premiums. Another audience for this model could be rideshare companies as well as companies such as *Google* and *Apple* who provide mapping apps. Furthermore, traffic control departments could collect this data firsthand and make it available to the aforementioned companies.

2. Data description

The data was acquired from the City of Seattle Open Data Portal and consists of **212,760 instances** of vehicle collisions in Seattle while the timeframe of the data ranges from **2004-2018**. Spanning mostly the Seattle area, the data for each variable was extracted using the *ArcGIS REST API* in *.csv* format and made available on the Seattle GeoData page.

A number of collision event features such as date of collision, location of collision, type of collision (sideswipe, rear-end, parked car etc.), number of people involved, number of fatalities, junction type (intersection, driveway junction, mid-block etc.), weather conditions, etc. are described which are analyzed in this report.

In total, **40 variables** are included in the dataset but this number is reduced to include only the ones crucial for building the model. As location played an important role in the analysis, addresses for each instance were extracted using Python's *reverse_geocoder* library and added to the dataset. This library was used with latitudes and longitudes as inputs to retrieve the neighborhood for each instance, stored in a variable called *Neighborhood* in the dataframe.

The *Tomtom API* was used to retrieve the free-flow traffic speed at the closest road to each coordinate. This data was retrieved as a *JSON* file which was then parsed using the *requests* and *json* libraries in Python and finally stored as *Speed* in the dataframe. Additionally, the *HERE API* was used to extract the road length and road congestion data. Both these sets of information were estimated using the closest road to the provided coordinate and stored as *Road Length* and *Road Congestion* respectively in the main dataset.

3. Data wrangling and cleaning

The original dataset called *Collisions.csv* was loaded as a dataframe into a Jupyter Notebook, hosted by the *SageMaker* machine learning (ML) platform on *Amazon Web Services (AWS).* The shape of the dataset was confirmed to be 212,760 rows by 40 columns.

a. Selecting initial variables

The following variables were removed from the dataset as they would provide little information during further analysis or during the modeling process.

Redundant variables

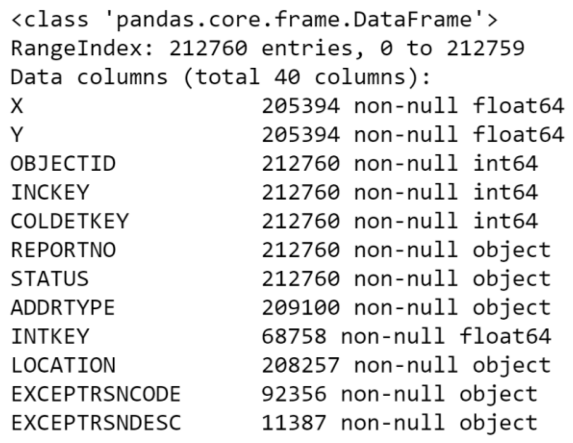
The *LOCATION* variable provides specific and descriptive addresses which are difficult to interpret. Instead, the *Latitude* and *Longitude* variables are used for more accurate location data. *INCDATE* only contains date information while the variable *INCDTTM* contains both date and time data making it more useful for analysis. *SDOT_COLCODE* contains the same information as *SDOT_COLDESC* which contains more detailed information. Similarly, *ST_COLCODE* contains the same information as *ST_COLDESC*. *SEVERITYCODE* was removed as it contains the same information as *SEVERITYDESC* except it contains character codes instead of text.

Variables with many unique categories

OBJECTID contains the same values as the default dataframe indexes with as many unique values as the number of rows in the dataset which would not provide much information for our model. The case is the same for *INCKEY*, *COLDETKEY* and *REPORTNO*. Both *SEGLANEKEY* and *CROSSWALK* contains more than 2000 unique categories making it difficult to analyze

Variables with many missing values

The number of missing values for each feature was determined using the *df.info()* method in Python which provided a list of all the variables along with the number of non-null values in each as well as the data type of each variable providing a good overview of the structure of the dataset. *Fig 1* shows a snippet of the output.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 212760 entries, 0 to 212759
Data columns (total 40 columns):
X                   205394 non-null float64
Y                   205394 non-null float64
OBJECTID            212760 non-null int64
INCKEY              212760 non-null int64
COLDETKEY           212760 non-null int64
REPORTNO            212760 non-null object
STATUS              212760 non-null object
ADDRTYPE            209100 non-null object
INTKEY              68758 non-null float64
LOCATION            208257 non-null object
EXCEPTRSNCODE       92356 non-null object
EXCEPTRSNDESC       11387 non-null object
```

Fig 1. Snippet showing the number of missing values and data type for each variable

Using the table shown in *Fig 1*, variables with many missing values were excluded such as *INTKEY* (68,578 entries), *EXCEPTRSNCODE* (92,356 entries), *EXCEPTRSNDESC* (11,387 entries), *INATTENTIONIND* (29,116 entries), *PEDROWNOTGRNT* (4983 entries), *SDOTCOLNUM* (127205 entries) and *SPEEDING* (9492 entries).

b. Renaming columns

The columns for the remaining variables were renamed to make them more human-readable and less cryptic. For example, the name *INCDTTM* was converted to *Incident Date and Time* whereas *SDOT_COLDESC* was changed to *SDOT Collision Description*.

c. Handling missing values and formatting

From the remaining 22 variables, 11 contained missing values which were visualized using the *missingno* library in Python. This provides a clearer understanding of the missing values relative to each variable as shown in *Fig 2*.

Several patterns were noticeable from this matrix. Variables such as *DUI, Weather, Road Condition* and *Light Condition* have missing values for essentially the same rows. *Longitude* and *Latitude* share the same missing values while *Address Type* shares missing values with these variables for certain rows but not all. *Junction Type* consists of a unique set of missing values while other variables such as *Status*, *Severity Description* and *Number of Injuries* contain almost no missing values. Variables containing very few missing values cannot be distinguished well using this matrix but this information can be confirmed from the table formed using the *.info()* function.

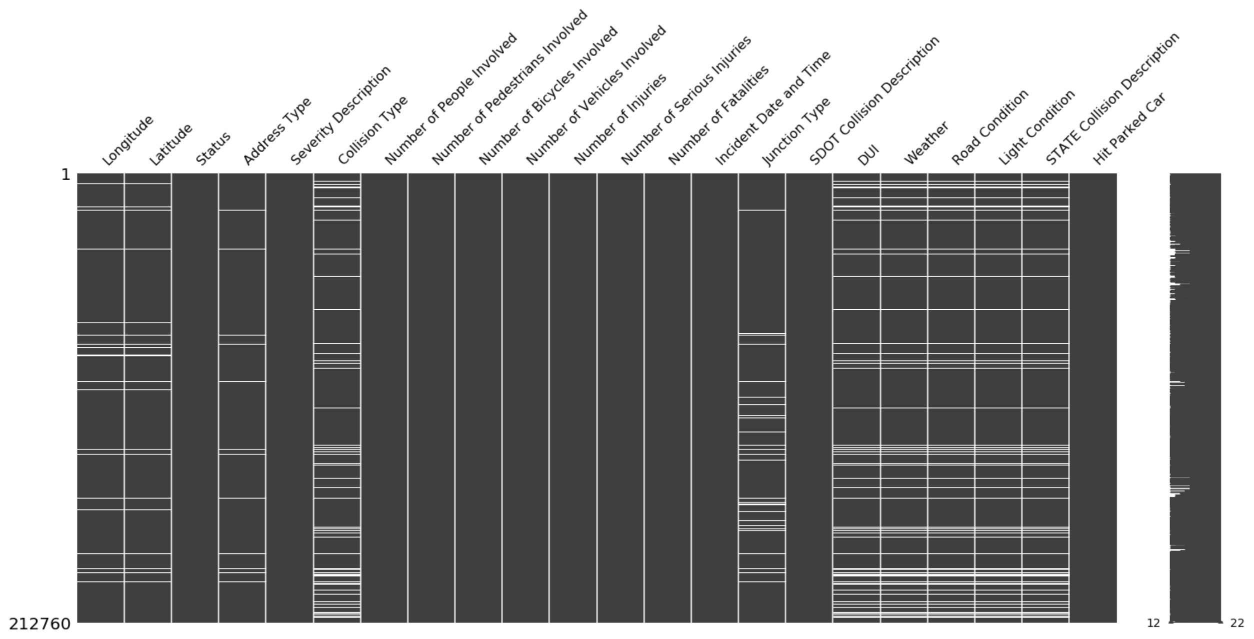


Fig 2. Visualization of missing values across all variables in the dataset.

The following variables were cleaned and formatted further.

Latitude and Longitude

Since only 7,366 instances were missing for this variable, these observations were simply dropped as the total number of instances would still be 205,394 which is significant. Attempting to fill in these values would be a tedious task provided they are specific location data.

Address Type

From *Fig 2*, it was observed that the missing values for *Address Type* are contained within the missing values for the coordinates. Hence simply dropping the null values for the coordinates also resulted in the null values for *Address Type* being dropped.

Weather

The weather-related variables provided the greatest challenge when it came to dealing with missing values. Not only did they contain a decent amount of null values but also contained categories such as ‘Unknown’ and ‘Other’ which essentially represented missing values. *Fig 3* shows the distribution of categories for *Weather* while *Fig 4* provides the actual frequencies for each category in a table.

Fig 3 was generated using the *countplot* method of Python’s *seaborn* library while *Fig 4* was created using the *value_counts()* pandas dataframe method.

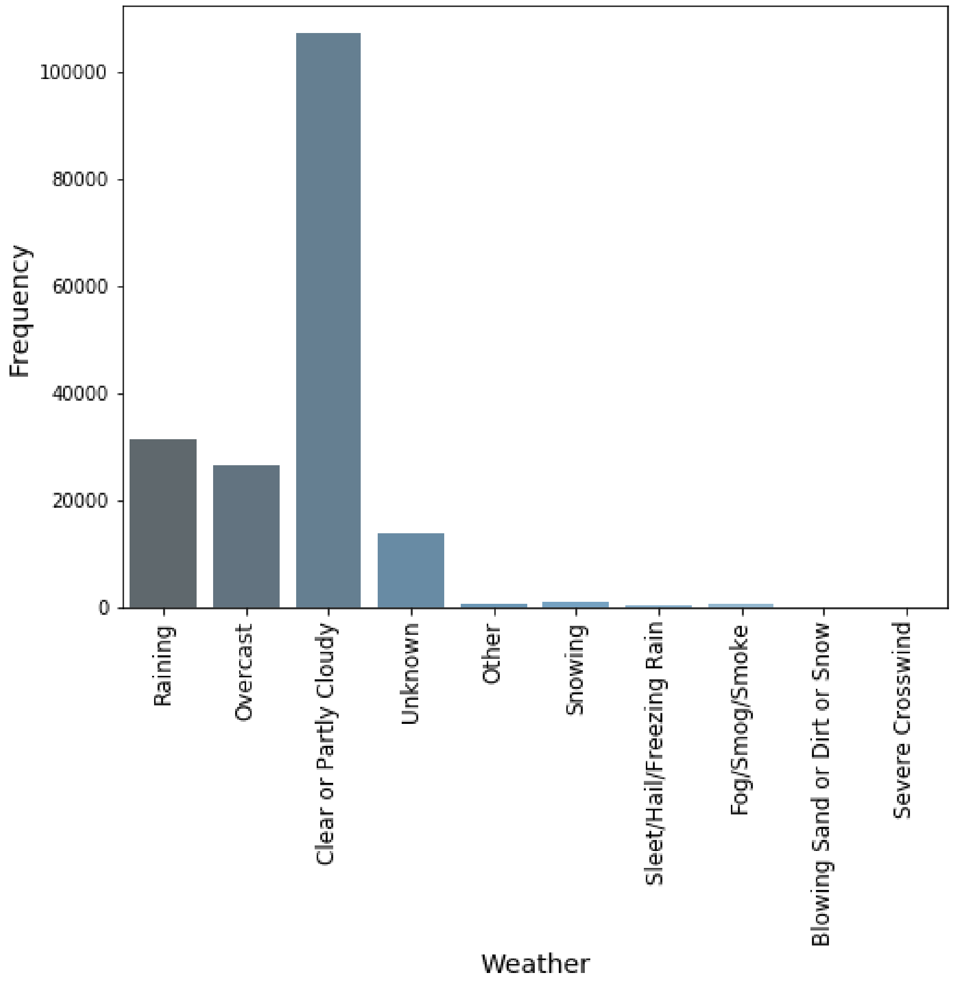


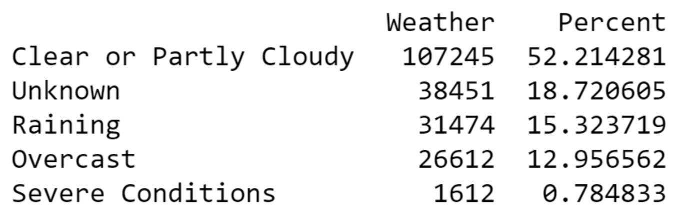
Fig 3. Visualization of weather categories

```
Clear or Partly Cloudy         107245
Raining                         31474
Overcast                        26612
Unknown                         13824
Snowing                           882
Other                             745
Fog/Smog/Smoke                    548
Sleet/Hail/Freezing Rain          109
Blowing Sand or Dirt or Snow       48
Severe Crosswind                   25
```

Fig 4. Table showing frequencies for each weather category

Several cleaning techniques were adopted for this variable. All missing values (23,882 instances), the ‘other’ category (745 instances) and the ‘unknown’ category (13,824 instances) were grouped into a single category called *Unknown* consisting of 38,451 observations.

Since categories such as ‘Snowing’, ‘Fog/Smog/Smoke’, ‘Severe Crosswind’, ‘Sleet/Hail/Freezing Rain’ *and* ‘Blowing Sand or Dirt or Snow’ had distinctly lower frequencies as compared to other categories they were grouped into a single category called ‘Severe Conditions’, making the categories more interpretable. *Fig 5* shows the frequencies of the newly formed weather categories.



```
                        Weather    Percent
Clear or Partly Cloudy   107245  52.214281
Unknown                   38451  18.720605
Raining                   31474  15.323719
Overcast                  26612  12.956562
Severe Conditions          1612   0.784833
```

Fig 5. Frequencies of new weather categories

Despite combining all the severe categories into one, it’s proportion is only 0.785%. Another noticeable category is ‘Unknown’ which has now risen to a sizeable 18.72% and is the second largest category. Removing this category would eliminate a sizeable chunk of instances. Instead we decided to keep this category.

Road Condition

Road Condition also contained a significant amount of missing values as well as some categories which could be combined for better interpretation and handling. Once again, the ‘unknown’ and ‘other’ categories were combined with the null values to form a ‘Unknown’ category. ‘Snow/Slush’ and ‘Ice’ were combined into a new category called ‘Snow/Ice’ whereas ‘Standing Water’ and ‘Oil’ were combined with the existing ‘Wet’ category. Finally, ‘Sand/Mud/Dirt’ went into the existing ‘Dry’ category.

Light Condition

Categories ‘Dark - No Street Lights’ and ‘Dark - Street Lights Off’ were combined into a single all-encompassing category representing dark conditions whereas ‘Dusk’ and ‘Dawn’ were combined into a single category as well. The biggest difference between the two are the actual times of the day when they occur but our *Time* variable should account for that variability.

DUI

This variable had 2 different types of binary distinctions when it came to whether the driver was under the influence or not. The first binary distinction contained the classes ‘Y’ and ‘N’, while the second one contained ‘1’ and ‘0’. Since It was self-explanatory that ‘1’ *corresponds to* ‘Y’ and ‘0’ corresponds to ‘N’, we simply replaced the letters to the numbers which will not only come in handy during the modeling process but is also easier to understand. It also contained 23,705 missing values which were converted to the ‘Unknown’ category.

Junction Type, Collision Type and SDOT Collision Description

For these variables, the only cleaning involved adding the missing values to an ‘Unknown’ category.

Severity Description

Since this was the target variable, it was crucial to handle it appropriately. Fortunately, it contained no missing values but the categories had to be re-aligned. An initial look at the frequency for each class revealed that the more severe categories such as ‘Serious Injury Collision’ and ‘Fatality Collision’ had much fewer instances than the non-severe cases – 1.42% and 0.15% respectively. This caused a **class imbalance problem** which would make predicting the severity class challenging as shown later on.

Therefore, to increase the severe class representation, the ‘Fatality Collision’ and ‘Serious Injury Collision’ categories were combined to form the ‘Severe’ category whereas ‘Property Damage Only Collision’ and ‘Injury Collision’ were converted to a ‘Non-severe’ class. Another benefit of having binary classes is that it simplifies the problem and makes the variable easier to work with. It would also make the results more actionable by making the interpretation of the results easier.

Finally, we also had to deal with the ‘Unknown’ category for this variable. This category was combined with the ‘Not Severe’ category for the following reasons:

- For crucial variables such as *Speed*, the distribution for the unknown category was similar to the non-severe one, both of which have a median of roughly 40mph. The severe median was distinctly higher, see *Fig 6*.

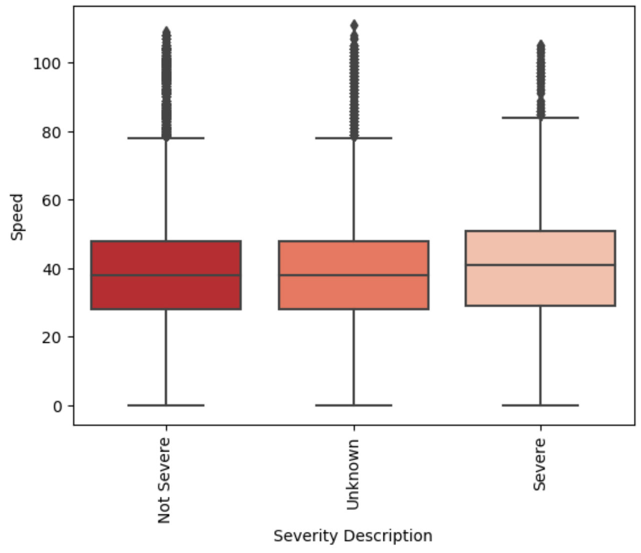
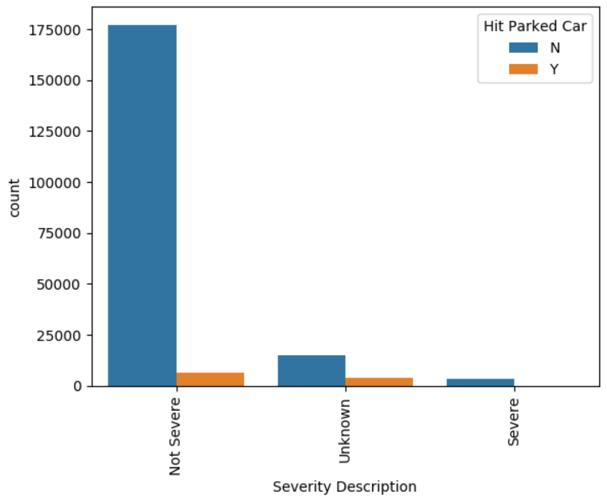


Fig 6. Distribution of severe categories for the speed variable

- The distributions for the ‘Unknown’ category, as seen from some bivariate analysis, are more consistent with the non-severe distributions than the severe ones. For example, if we look at variables such as ‘Address Type’ and ‘Junction Type’, the distribution of the ‘Unknown’ category matches more with the non-severe class, see *Fig 7*.



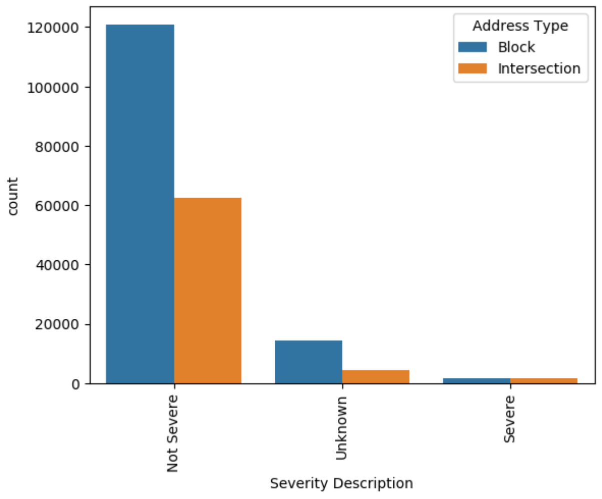


Fig 7. Visualization of unknown categories for the *Hit Parked Car* and *Neighborhood* variables.

- Also, there are more chances of mis-classification if the unknown category is combined with severe as that would push that category from ~1.5% to ~10% which is a big jump and would potentially provide false results. On the other hand, combining with the non-severe category would increase the size from ~89% to ~98% which would not have such a major impact on the results as the non-severe category is already much larger.

d. Transforming date variables

The *Incident Date and Time* variable was split into *Year*, *Month*, *Date* (day of the month), *Day of the Week* and *Hour* variables. This was done by first converting the *Incident Date and Time* variable from 'string' to 'datetime' format using the *.to_datetime()* function in pandas. Then the new date and time variables were extracted from *Incident Date and Time* using attributes of the datetime format. For example, the year for a datetime value can be extracted by using the *.year* attribute. The other variables were extracted in similar fashion and had integer values. The *Day of the Week* variable was ordered from 0-6 with Monday being the first day.

e. Dealing with outliers

The only variables where we had to deal with outliers were *Road Length* and *Road Congestion* as they contained a few '999' values which were purposely added in order to catch error returns from the *HERE API* during data retrieval.

4. Exploratory data analysis

a. Geographical analysis

In this section, the coordinates are used to visualize the crash locations across Seattle in order to get an idea of the areas of high collision density as well as areas of greater severity.

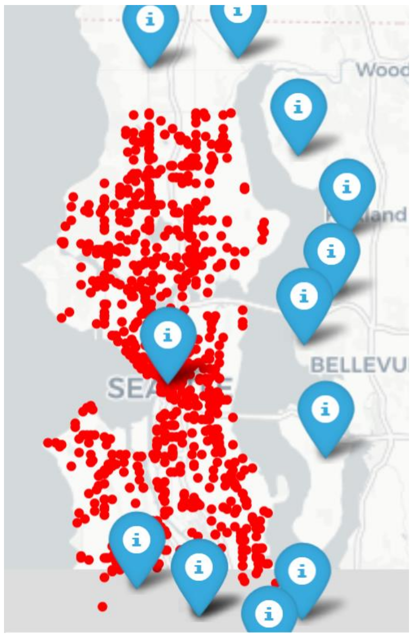


Fig 8. Folium map of Seattle showing the spread of neighborhoods across the city

Fig 8 shows all neighborhoods in Seattle as blue markers, created using Python's *Folium* library. It is instantly noticeable that the neighborhood right at the center of the map (Seattle) contains majority of the collision instances with the other neighborhoods lying at the fringes of the city.

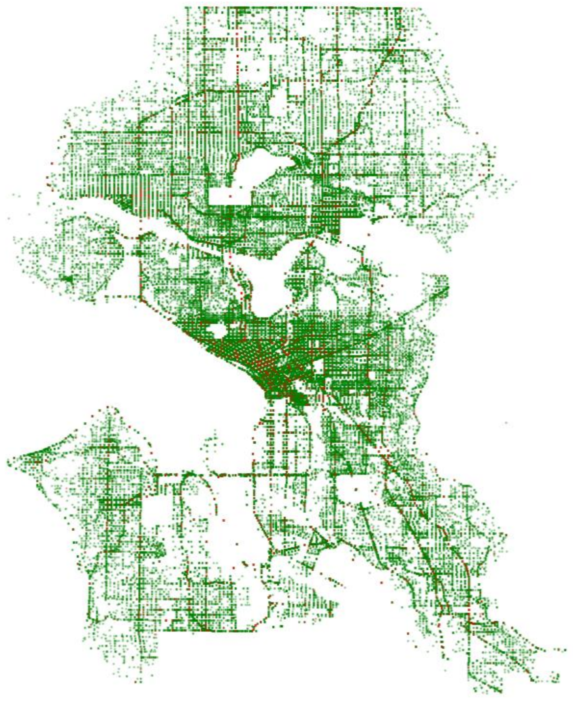


Fig 9. Scatterplot representation of the collision incidents across Seattle using coordinates

Fig 9 provides a clearer visualization of all the collisions as a scatter plot of the coordinates. Due to the fairly large number of instances, the plot essentially maps out the entire Seattle area. On the map, the green points denote non-severe collisions while the red points denote severe cases. As expected, the green instances are much more prevalent. Despite the sparse nature of the severe points, they can still be seen across the map.

The severe and non-severe incident locations are plotted separately in *Fig 10* to get a better understanding of the geographical trends. The first plot, shown in green, displays the non-severe cases. The second plot shows the severe cases but they appear sparse due to the low frequency of the severe class. The third plot is the same as the second one but with the transparency increased so that the points can be visualized better.

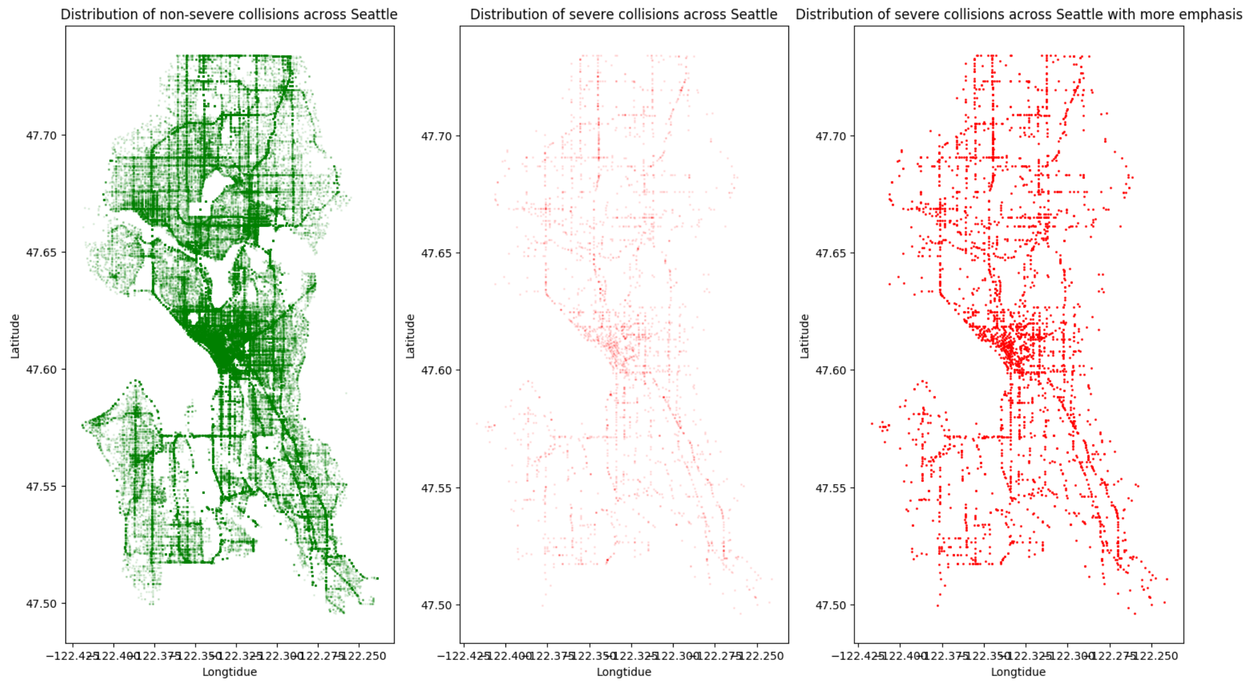

Distribution of non-severe collisions across Seattle
Distribution of severe collisions across Seattle
Distribution of severe collisions across Seattle with more emphasis
Latitude
Longtidue
47.70
47.65
47.60
47.55
47.50
−122.425
−122.400
−122.375
−122.350
−122.325
−122.300
−122.275
−122.250

Fig 10. Separate scatterplots of only severe and non-severe incident locations

In the first plot, the center of Seattle sees the highest density of collisions with several streets towards the north also depicting dense regions. The density is relatively lower in the south. Here are some key areas where the density of severe collisions are relatively higher:

- *Center of the city*: highest density of crashes
- *Aurora Ave North*: road going north
- *Rainier Ave South*: road going south-east
- *15th Ave Northwest*: road going north-west
- *Lake City Way Northeast*: road going north-east
- *24th Ave East*: towards the east

Most of these areas are also dense for the non-severe cases but the non-severe plot also has other regions of higher density as opposed to the severe plot which mostly emphasizes just a few areas.

Fig 11 shows the distribution of severe and non-severe cases across all the neighborhoods. A glaring issue with the first plot in the figure is that the distribution of the severe category can barely be noticed. We therefore visualize this category separately in the second plot. For both severity cases, **Seattle** is by far the most prevalent neighborhood. This makes sense as most of the other neighborhoods are at the edges of the city as we observed from the folium map earlier.

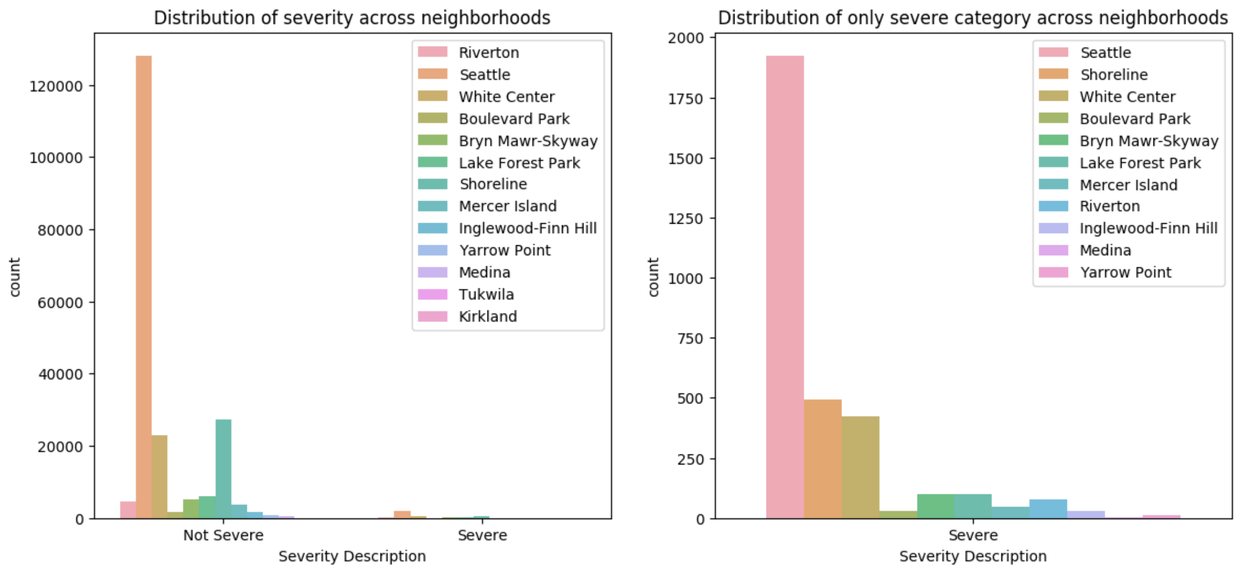


Fig 11. Distribution of non-severe and severe collisions across neighborhoods

The next two most frequent neighborhoods are **Shoreline** and **White Center** for both severity categories. After that, neighborhoods such as **Lake Forest Park**, **Bryn Mawr-Skyway** and **Riverton** follow. An important point to note here is that the distribution of neighborhoods for both severity cases are almost identical. Hence, there is not much information to differentiate non-severe and severe instances.

b. Analyzing weather related variables

Once again we look at the weather variables (*Fig 12*), this time with clearer categories and binary severity classes (*Fig 13*). For both cases, 'Clear or Partly Cloudy' conditions are the most prevalent possibly because there is more traffic on the roads under those conditions which in turn causes more collisions.

In order to normalize this issue, we would need to determine the rate of traffic flow under each condition and then divide the amount for each category by the corresponding traffic rate. Unfortunately, we don't have access to such data at the moment. When it comes to the other weather categories, the distribution is once again similar for both severity cases.

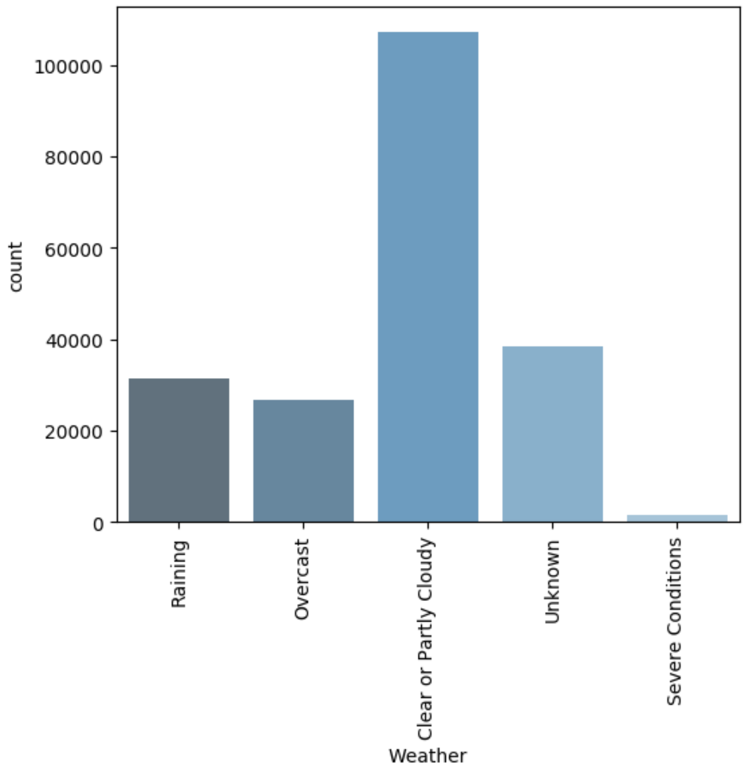


Fig 12. Distribution of weather categories.

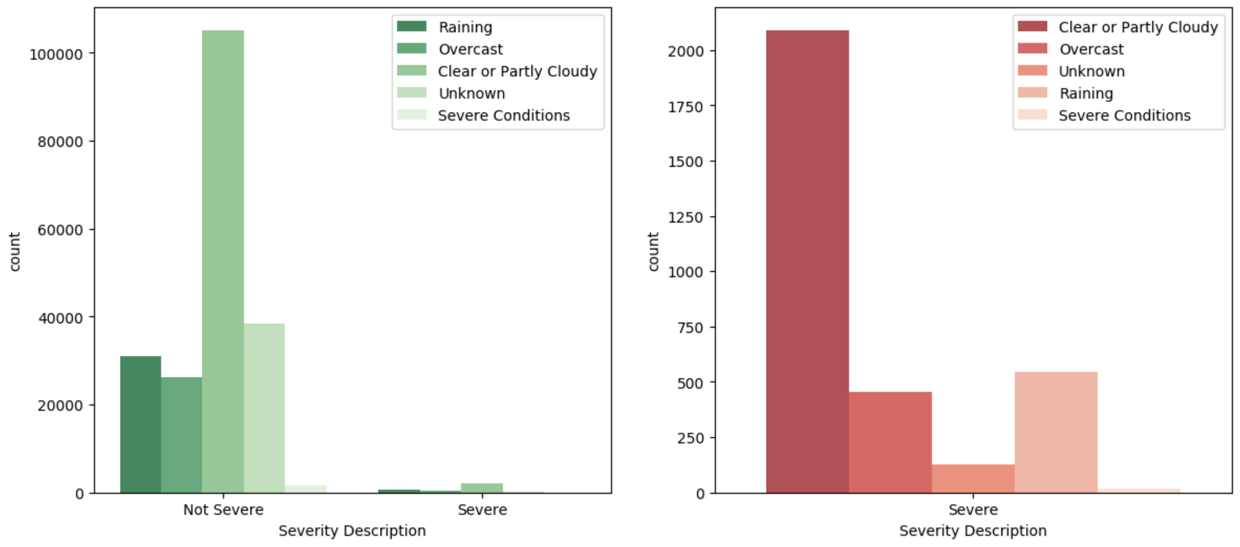


Fig 13. Distribution of non-severe and severe collisions across weather.

The case is the same for *Light Condition* as most of the collisions are during the daytime (*Fig 14*) and there is nothing to indicate an obvious difference between severe and non-severe cases. The distribution of light conditions across severity categories are similar, as shown in *Fig 15*.

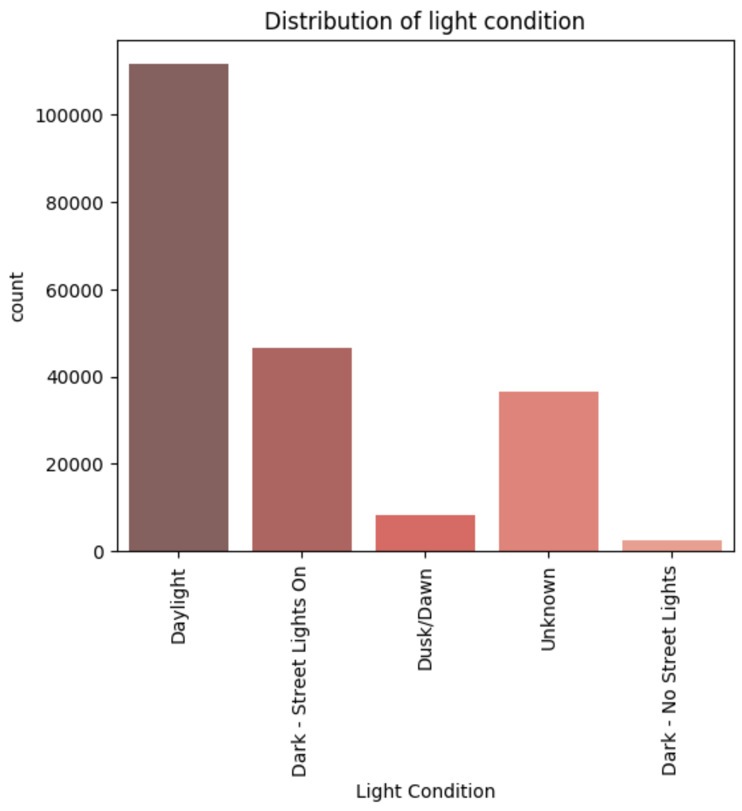


Fig 14. Distribution of light condition categories.

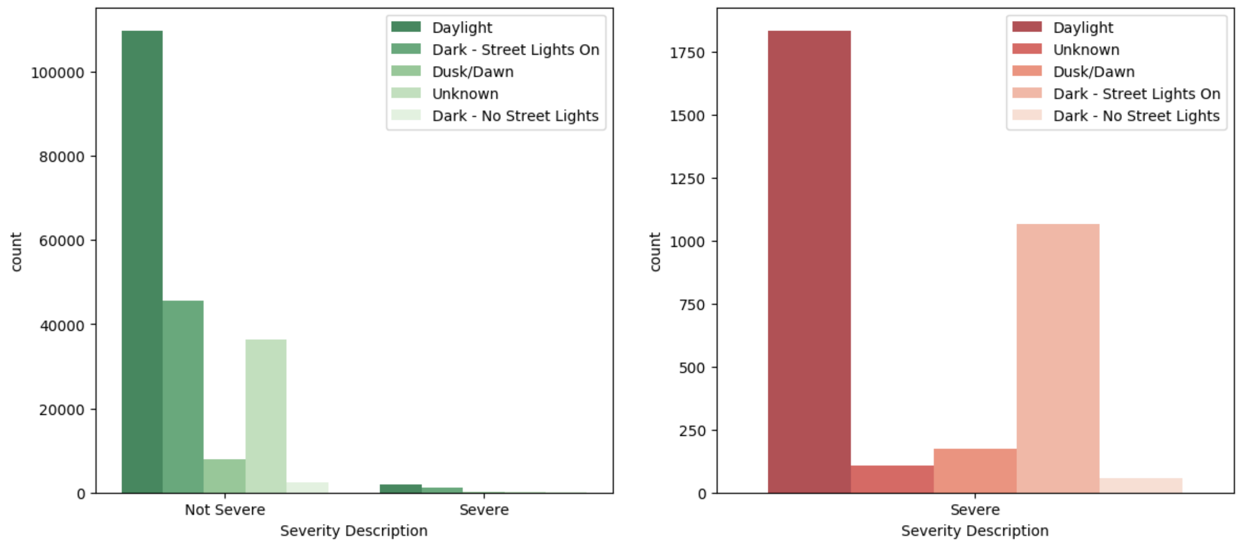


Fig 15. Distribution of non-severe and severe collisions across light condition.

Road conditions follow the same trend as weather and light conditions as seen in *Fig 16* and *Fig 17*. The 'Dry' category is the most predominant which could again be because most cars operate during these conditions whereby causing more accidents. For both severity classes, wet conditions are less frequent while icy conditions are rare.

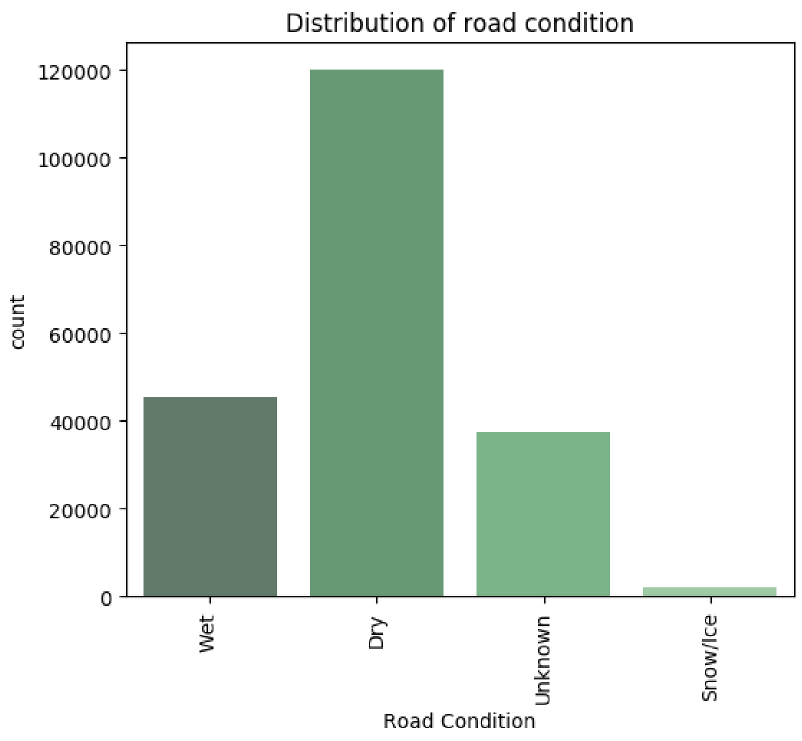


Fig 16. Distribution of road condition categories.

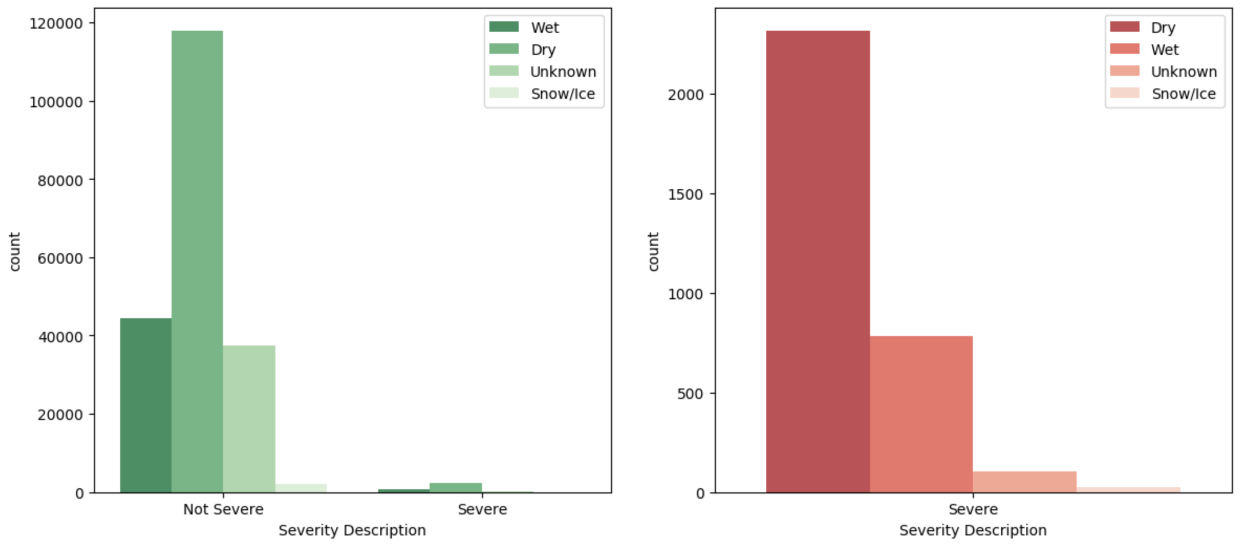


Fig 17. Distribution of non-severe and severe collisions across road condition

c. Analyzing address related variables

The analysis of *Address Type* shows that the 'Intersection' category contains half the number of instances as that for the 'Block' category, shown in *Fig 18*.

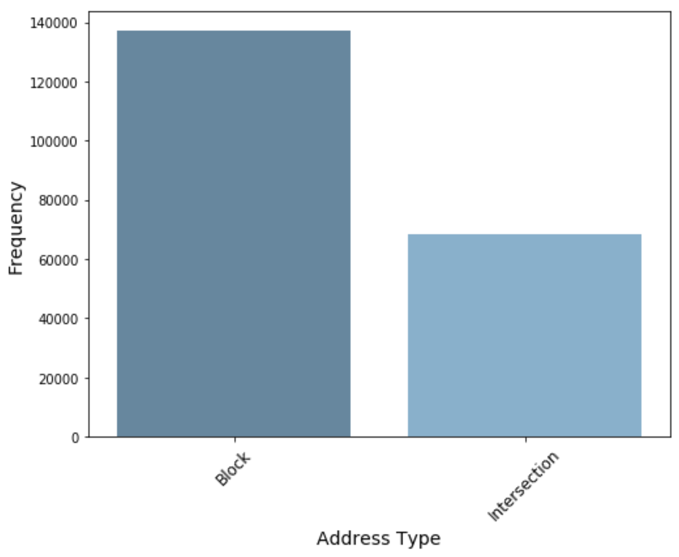


Fig 18. Distribution of address type classes.

The second plot in *Fig 19* shows that 'Intersection' has a slightly higher frequency than 'Block' for the severe class whereas the non-severe class shows double the number of instances for 'Block' as that for 'Intersection' which also matches the overall trend for *Address Type*. Hence, this could be a telling distinction for our model.

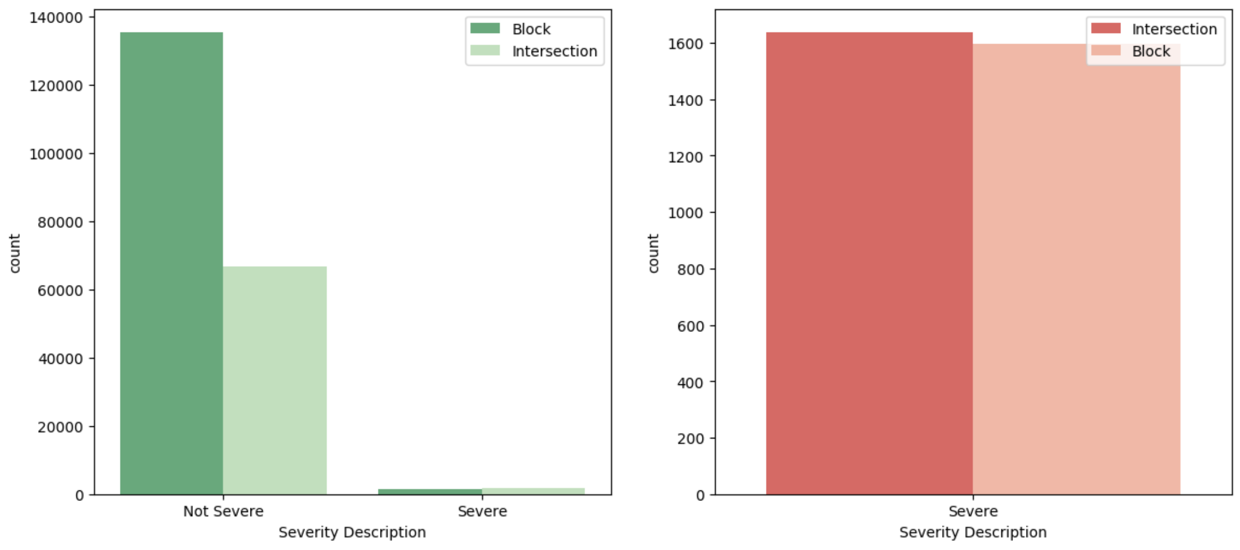


Fig 19. Distribution of severity across address type.

For *Junction Type*, once again 'Intersection' collisions are more frequent for the severe class whereas 'Mid-Block' is more prevalent for non-severe cases, as shown in *Fig 20*. We are mainly considering the two most frequent classes for each severity as they cover most of the distribution.

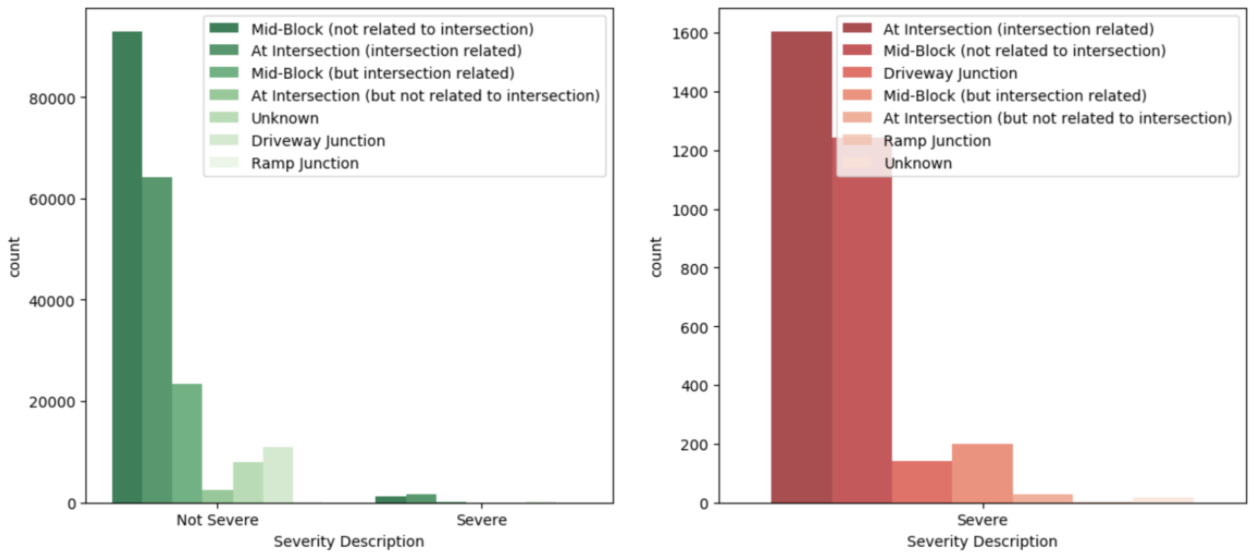


Fig 20. Distribution of severity across junction type.

d. Analyzing variables related to the road

For the speed variable, the histograms for the two severity cases as well as the overall speed distribution are almost identical, as shown in *Fig 21*. Hence, not much information can be gathered from this data. However, the boxplot in *Fig 22* reveals that the severe cases have a slightly higher median. The severe class has a median of ~41mph while non-severe has a median of ~38mph.

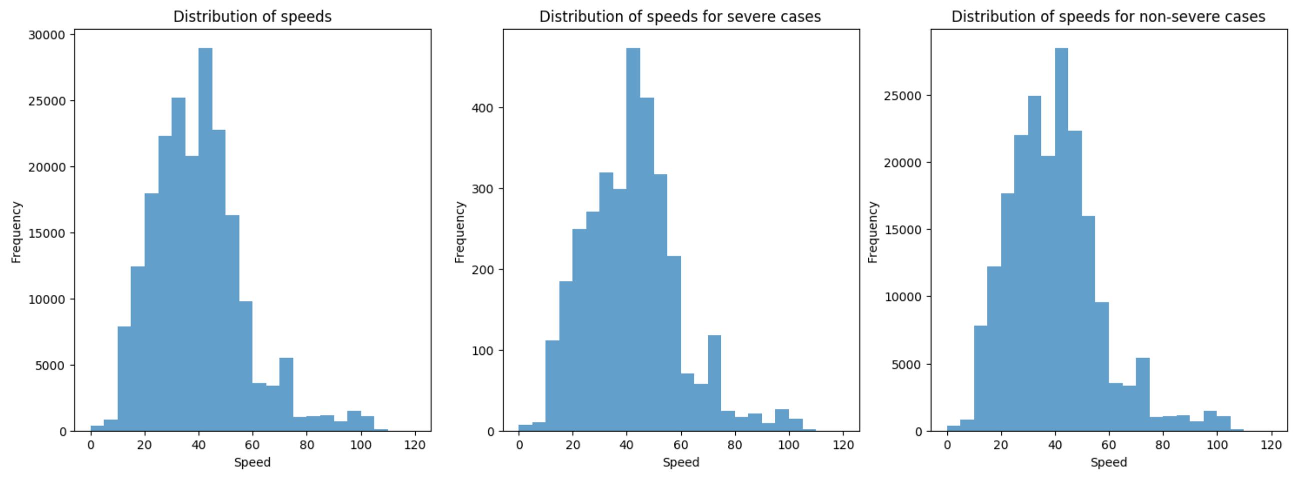


Fig 21. Distributions speed for different severity classes.

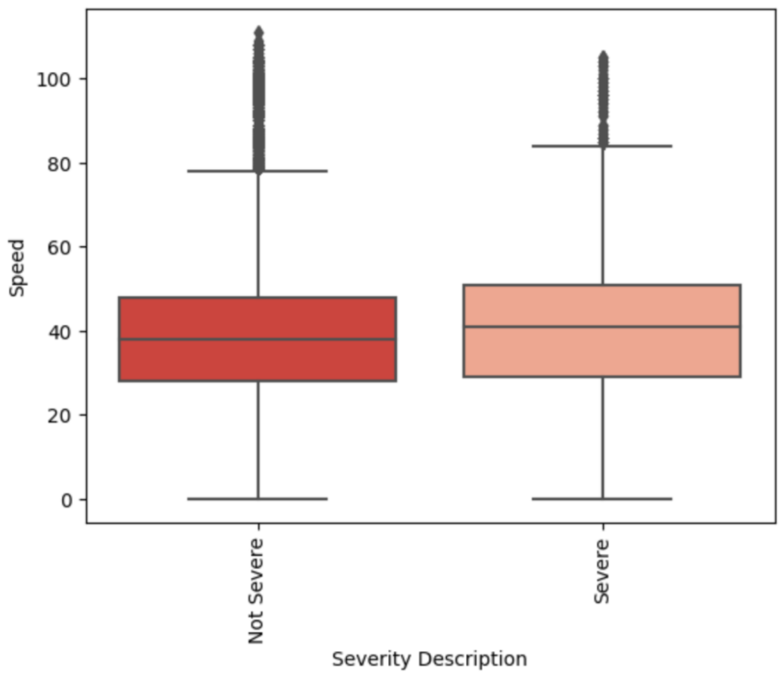


Fig 22. Boxplot of the speed variable across severity.

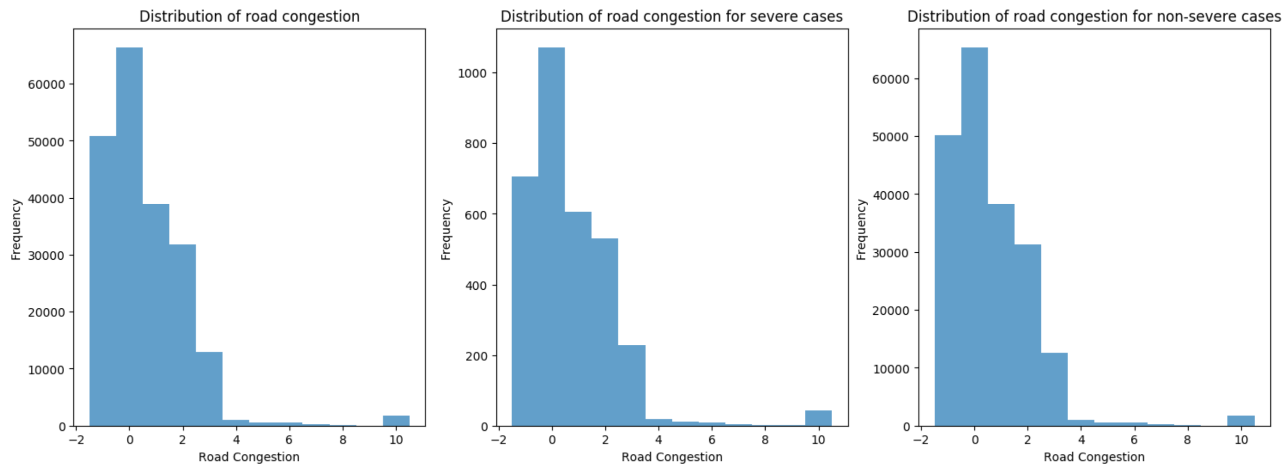


Fig 23. Distributions of road congestion for different severity classes.

The distributions of road congestion across severity are almost identical (*Fig 23*). The median for non-severe instances is slightly lower (0.50823) than the severe (0.63665), shown in *Fig 24*. Note that for this metric, values **closer to 1 have higher congestion** rates while those **closer to 0 have lesser congestion**.

Similar to road congestion, the distributions for road length is also tight across the severity classes (*Fig 26*) where the median only differs by ~0.04 (*Fig 27*). Overall, despite the fact that speed, road congestion and road length don't provide a clear distinction between the severity levels, they could still prove to be useful for our model.

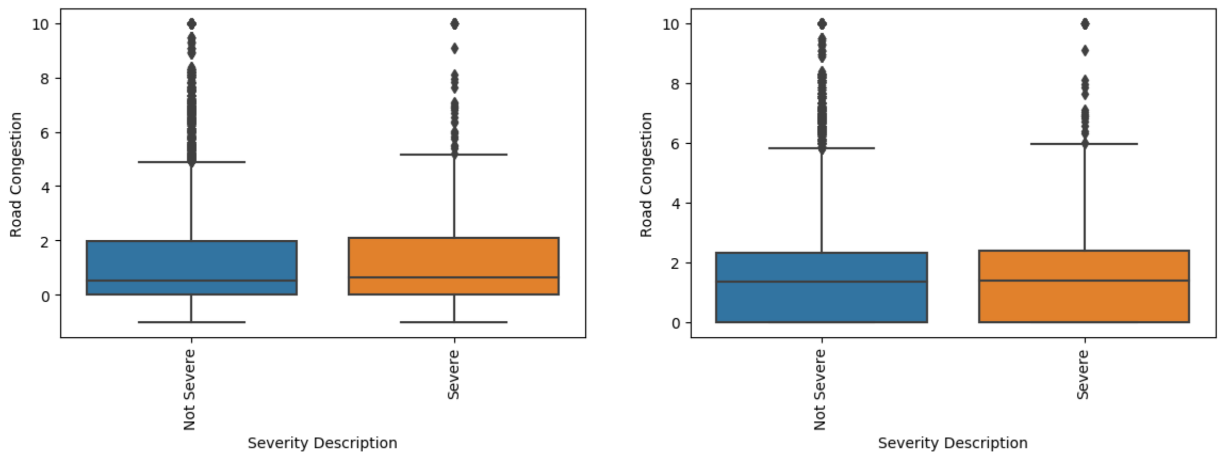


Fig 24. Boxplots of road congestion with the '-1' value included (left) and excluded (right)

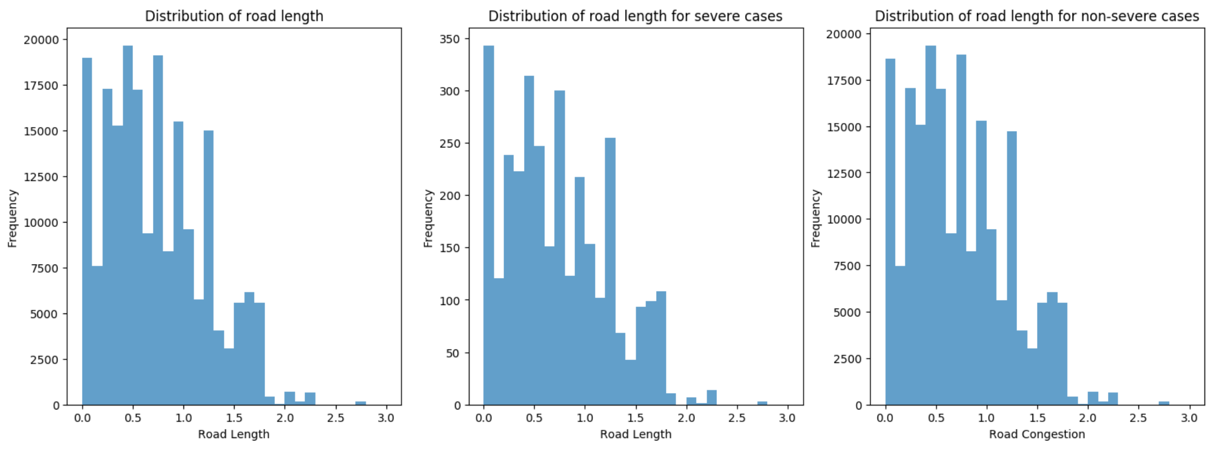


Fig 25. Distribution of road length across severity classes

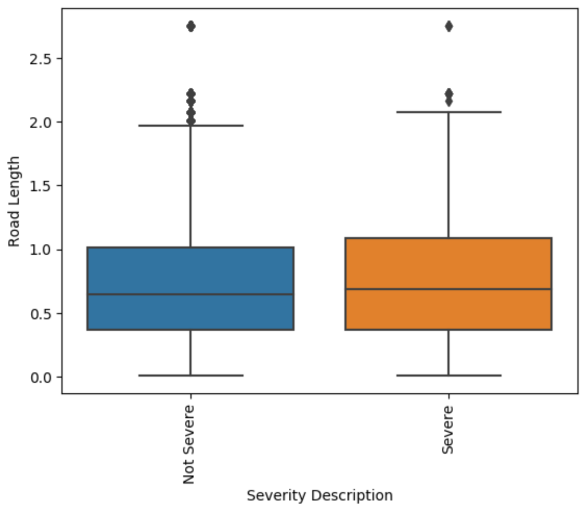


Fig 26. Boxplot of road length across severity.

e. Timeseries analysis

The plot in *Fig 27* displays the frequency of collisions across years showing an overall downward pattern. A closer look reveals a downward trend from 2006 to 2011 followed by an upward pattern till 2016. From 2017 to 2019, there is again a downward movement.

Looking at the average severe collisions rate over the years in *Fig 28*, there is a sharp increase from 2005-2006 followed by a slump till 2008. After another increase till 2009, there was a downward trend till 2011. There was another one year increase till 2012 followed by yet another downward pattern till 2015 followed by a continuous increase till 2019. **Overall, there is a downward trend from 2006 to 2015 followed by an upward trend since.**

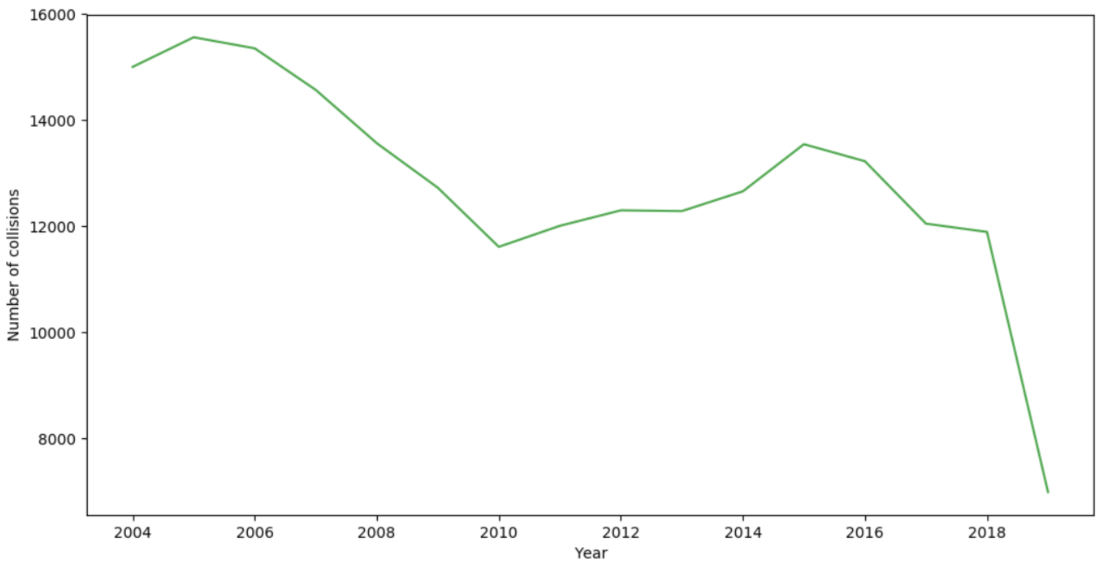


Fig 27. Yearly timeseries trend of collision frequencies.

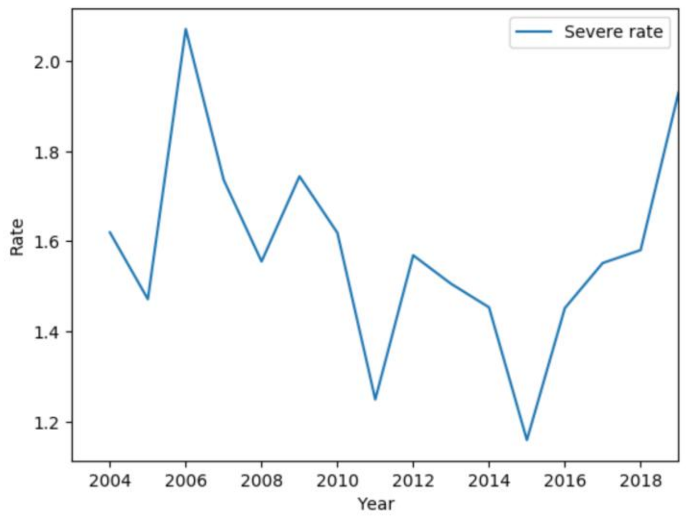


Fig 28. Average collision severity rates across years.

Fig 29 shows the average severity rate across months where July and August clearly have a higher rate than the other months. April - June as well as the winter months of Feb and Dec see the lowest rates. The monthly and weekly collision frequencies are plotted in *Fig 30* which shows the respective trends across time from 2004-2019.

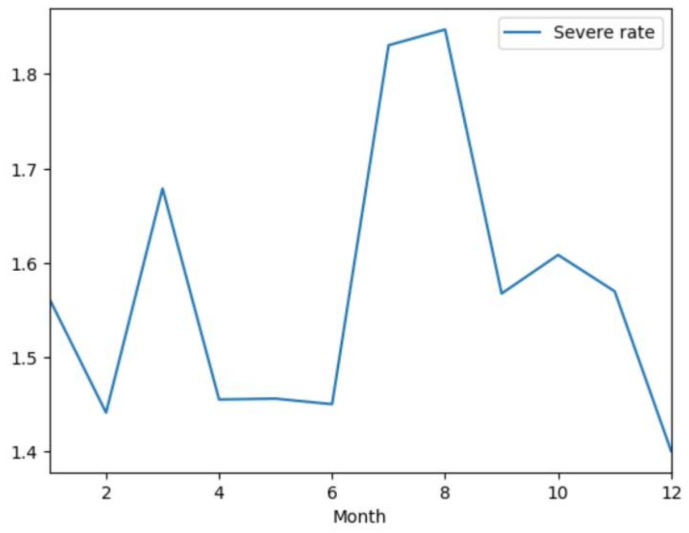


Fig 29. Average collision severity rates across months.

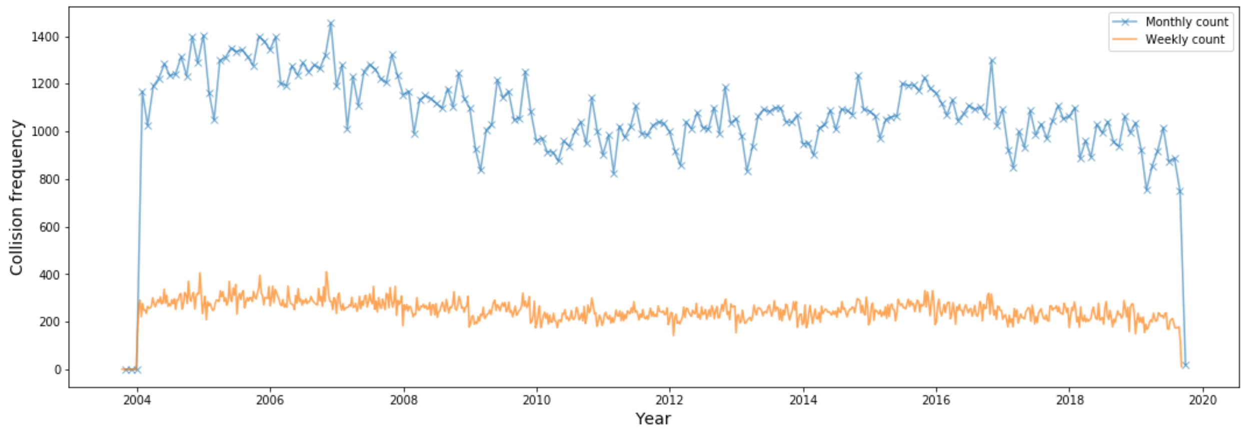


Fig 30. Monthly and weekly collision frequencies across time.

Fig 31 shows the average yearly, monthly and weekly collision rates across time. Note that the severe and non-severe class labels had to be converted to integers (1 for severe and 0 for non-severe). Hence, points that are higher in value represent more severe collision instances. All points have values much closer to 0 due to class imbalance. Due to the much larger proportion of non-severe instances in the dataset, an average of points at a certain point in time is likely to be skewed towards 0.

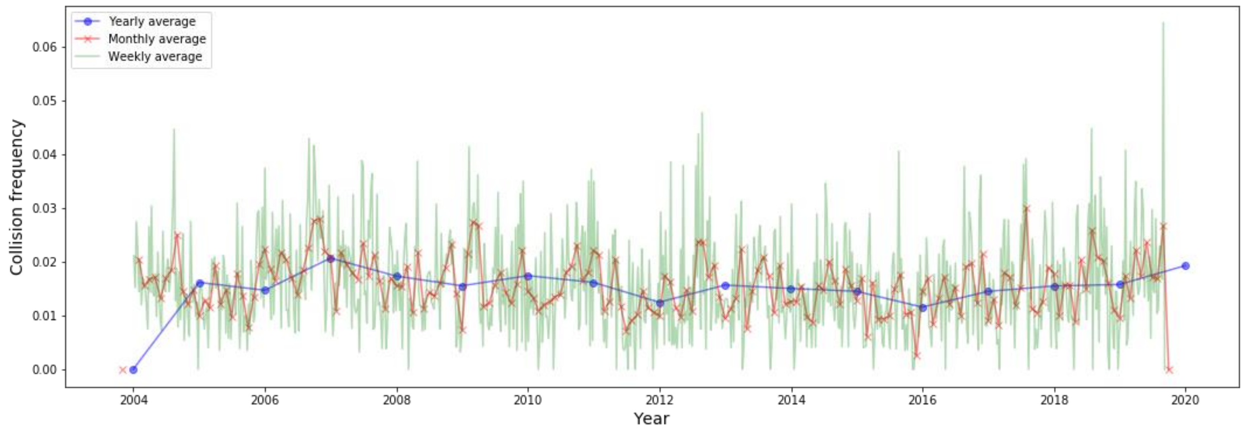


Fig 31. Average yearly, monthly and weekly collision rates across time.

The hourly average severity collision rates are plotted in *Fig 32* with **'0' indicating 12am and '23' indicating 11pm**. Some trends clearly stand out. The hours between 1am and 6am see a higher rate of severe collisions. The case is similar for 4pm-6pm, 9pm and 11pm. Hence, between 4pm and 11pm, there is certainly an increasing trend in severe collision rate. Surprisingly the rate is the lowest at 12am. The hours between 8am and 3pm see the lowest severe collision rate which is somewhat expected since the light conditions tend to be the most ideal during this period.

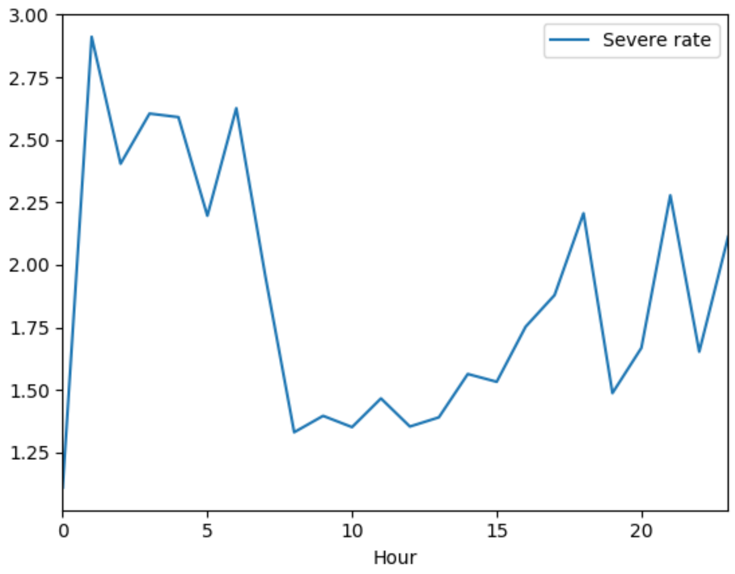


Fig 32. Average severity rate across hours of the day

From the plots in *Fig 33*, the downward trend for the non-severe case is much more pronounced with a higher moving average. The curve is smoother and has a distinct downward trend. For severe collisions, we can observe a general downward trend but a dip and then a rise between day 5 and 10 in the month. One way in which we can differentiate between the severity levels is by emphasizing the fact that the early days of the month see lesser severe collisions.

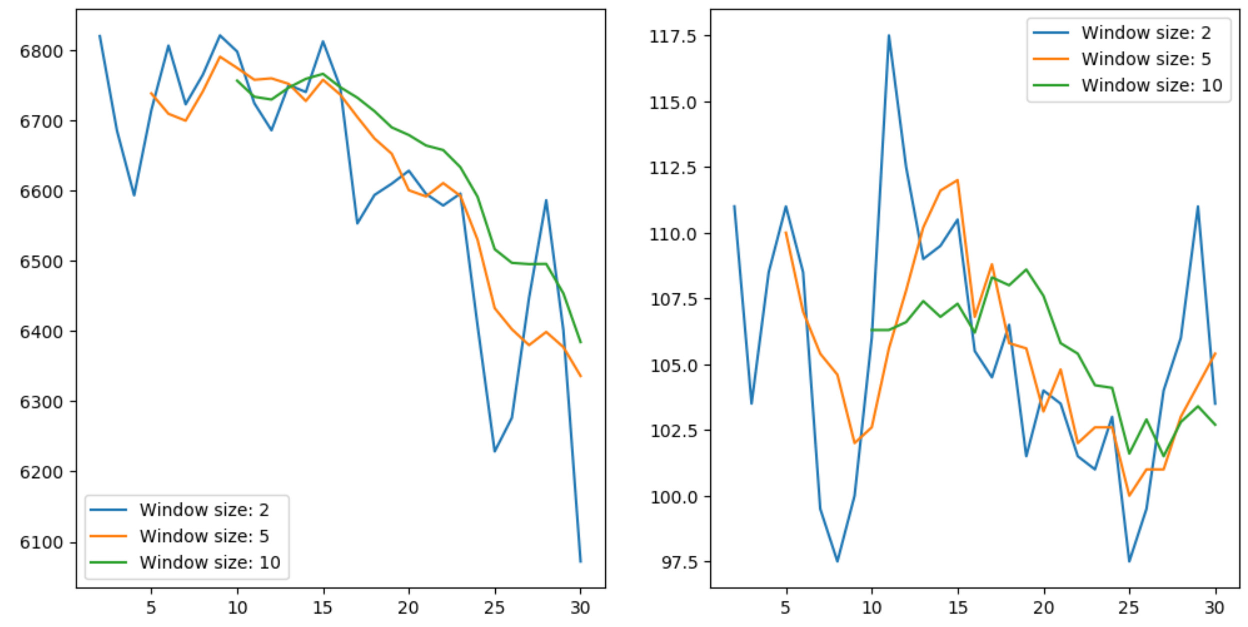


Fig 33. Average non-severe (left) and severe (right) collision rates across days of the month.

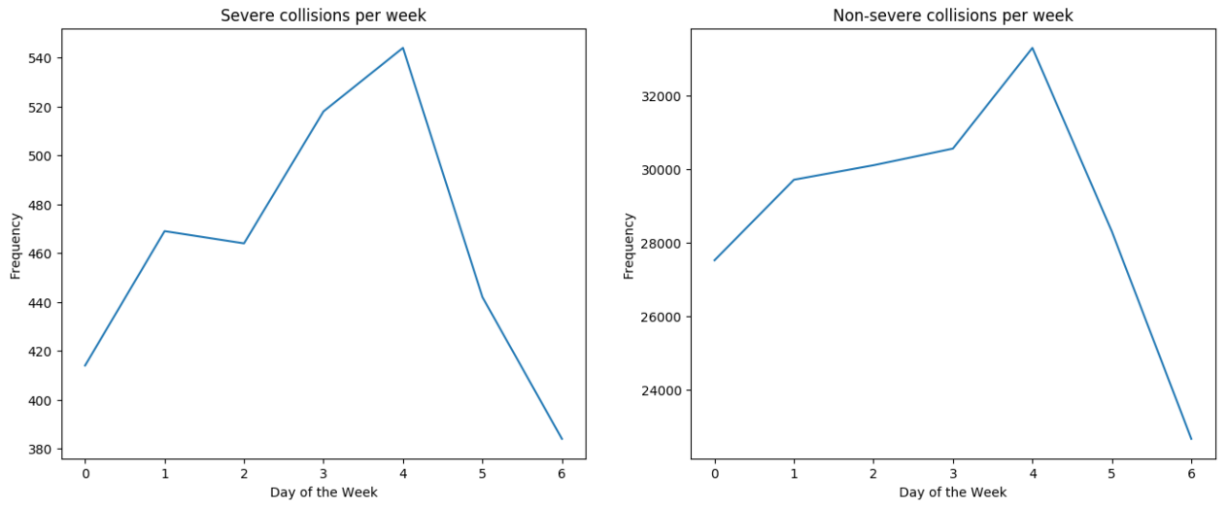


Fig 34. Average severe (left) and non-severe (right) collision rates across days of the week.

The trends for both severe and non-severe cases for day of the week are very similar as shown in *Fig 34*. In both cases, Friday (4) sees the highest amount of collisions while Sunday (6) sees the lowest. These numbers make sense since as we progress through the week, the number of collisions increase gradually hitting the peak on Friday.

5. Modeling

Feature Selection

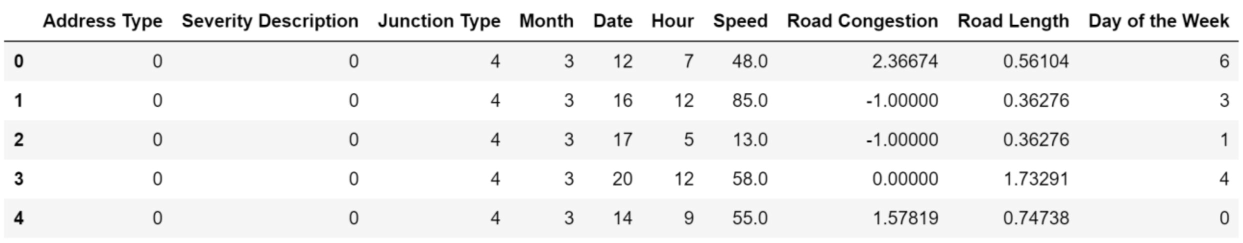
Variables that only provide information about the collision incident once it has occurred were dropped from the dataset as they could not be used for predicting the severity of future collisions. These are variables such as *Number of People Involved* and *SDOT Collision Description*.

Weather, *Light Condition*, *Road Condition* and *Neighborhood* were also dropped following initial modeling as they did not necessarily improve the model. In fact, removing the *Neighborhood* feature improved the model performance. The following features were retained:

- *Address Type*
- *Junction Type*
- *Month*
- *Hour*
- *Day of the Week*
- *Speed*
- *Road Congestion*
- *Road Length*
- *Severity Description*

Pre-processing

The categorical variables were converted to discrete integers using the *LabelEncoder* function in Python. Features that were encoded are *Address Type, Junction Type, Month, Hour, Day of the Week* and *Severity Description*. The encoded dataframe is shown in *Fig 35*.



	Address Type	Severity Description	Junction Type	Month	Date	Hour	Speed	Road Congestion	Road Length	Day of the Week
0	0	0	4	3	12	7	48.0	2.36674	0.56104	6
1	0	0	4	3	16	12	85.0	-1.00000	0.36276	3
2	0	0	4	3	17	5	13.0	-1.00000	0.36276	1
3	0	0	4	3	20	12	58.0	0.00000	1.73291	4
4	0	0	4	3	14	9	55.0	1.57819	0.74738	0

Fig 35. Snippet of dataframe with label encoding for categorical variables

Note that feature scaling was not implemented initially as our selected model was *LightGBM* which did not require the features to be scaled as opposed to an algorithm such as *Logistic Regression* which is a linear model and hence required scaling for optimum performance.

The dataset was then split into the explanatory variables (*X*) and the target variable (*Y*) for modeling and evaluation.

Train test split

The datasets are further split into train and test samples. The train sample size was chosen as 70% of the original size. During the optimization phase, the test sample will be considered a validation sample. What we mean by this is that the train sample would be used for parameter tuning and cross-validation where it will be split into several random train and test samples in order to determine the best average score.

Oversampling of minority class

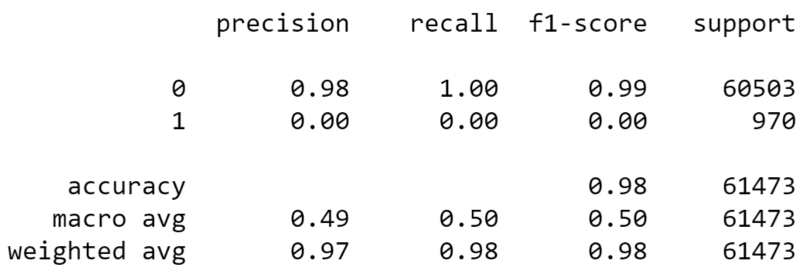
In the case where weather and neighborhood variables were included, *Synthetic Minority Oversampling Technique (SMOTE)* improved the model performance (ROC AUC score). SMOTE creates synthetic data points like the minority instances, making the representation of the two classes nearly the same whereby fixing the imbalance issue to an extent.

However, oversampling was not used for the final baseline model as it decreased the scoring metric. This is not surprising as oversampling only works well when the initial model is robust to begin with. In our case, the class imbalance was severe making oversampling ineffective.

Modeling, prediction and evaluation

We fit the training data with the *LightGBM* classifier and tested the model by making predictions on the *x_test* sample. For evaluating the model, **accuracy**, **precision**, **recall**, and **ROC AUC** were used as the main metrics, all of which were imported from the *sklearn.metrics* library. A first look at the results reveals the **accuracy score to be 0.9842**. The precision and recall scores values are same.

Another way of quickly visualizing these values is by using the *classification_report* function. Although, it does not show the accuracy and roc_auc scores. Looking at the classification report reveals the bigger picture, shown in *Fig 36*.



```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99     60503
           1       0.00      0.00      0.00       970

    accuracy                           0.98     61473
   macro avg       0.49      0.50      0.50     61473
weighted avg       0.97      0.98      0.98     61473
```

Fig 36. Classification report for baseline model

The table above shows that the model is only predicting the majority class. This can be observed from the individual precision or recall value for the '1' (severe) class which is a 0. In fact, if the predicted values are categorized into severe and non-severe values, we notice that only one case was predicted as 'severe' or '1' which is actually a wrong prediction. This can be further analyzed using a confusion matrix, shown in *Fig 37*.

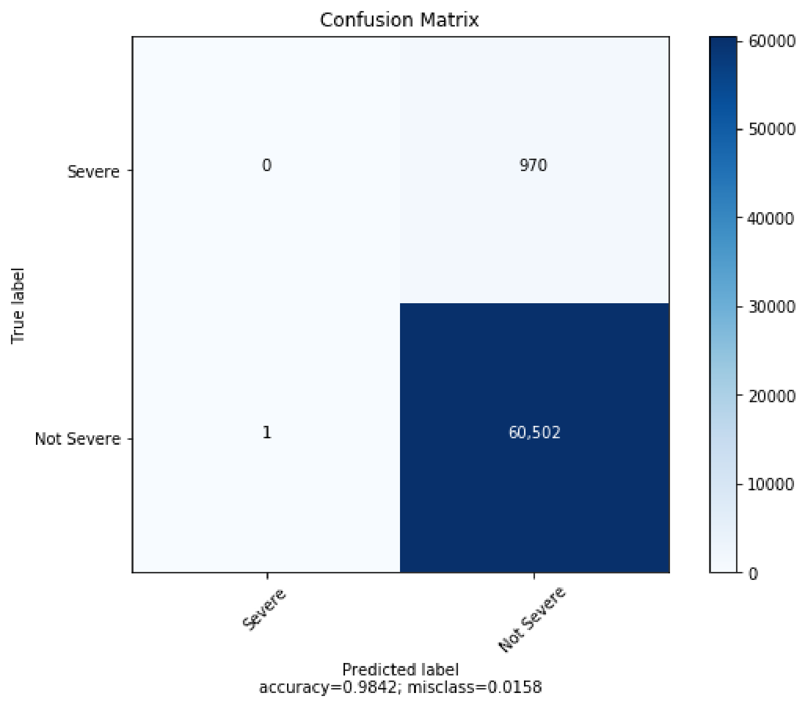


Fig 37. Confusion matrix for the baseline model.

As seen in *Fig 37*, 60,502 instances were correctly predicted as being non-severe (true negative), 970 were incorrectly predicted as non-severe (false negative), 1 was incorrectly predicted as severe (false positive) and none were predicted correctly as severe (true positive). In order to explore this behavior further, we looked at the ROC AUC curve, shown in *Fig 38*.

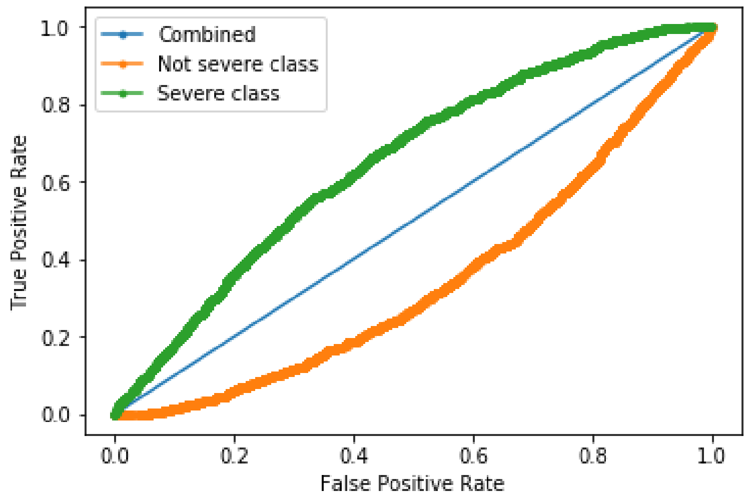


Fig 38. ROC AUC curve for combined, non-severe and severe classes.

The calculated **ROC AUC score for this model was 0.654**. This value is low mostly due to the large class imbalance in the data. The plot shown in *Fig 38* displays the ROC AUC curves for both the severe and non-severe classes as well as the combined ROC AUC. Moving along the curve corresponds to changing our threshold value for the predictions. Therefore, depending on the application, different thresholds can be set in order to achieve a certain specification.

Model selection and optimization

Grid search was performed in order to tune the hyperparameters of the model. Cross-validation was used within the grid search algorithm with 5 folds in order to determine the best average ROC AUC score. This ensured that the optimization would not cause the obtained parameters to overfit the model.

Parameters **maximum depth**, **number of leaves**, **learning rate**, and **minimum data in each leaf** were tuned to values of **5, 10, 0.1 and 20** respectively while the **best score was 0.6397**. Applying the above parameters to the *LightGBM* model improved the baseline score from 0.654 to 0.659, which was further improved to 0.66 after applying scaling for the features using the *MinMaxScaler* function from *sklearn*. The resulting ROC AUC curve is shown in *Fig 39*.

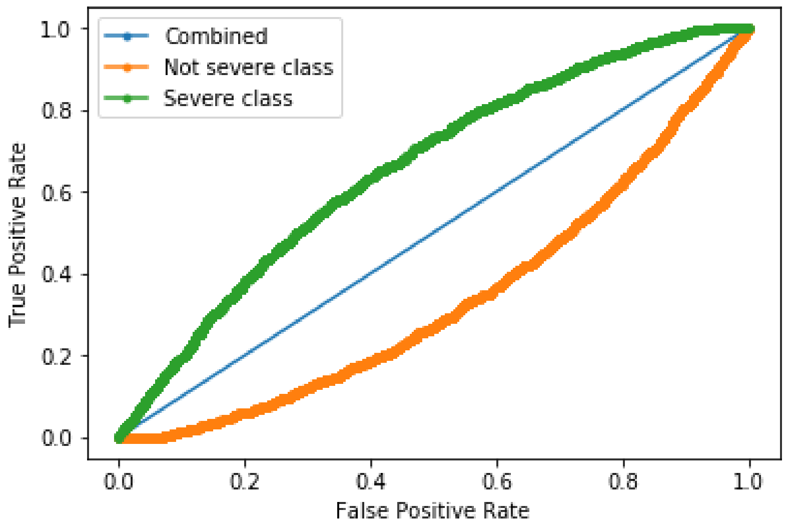


Fig 39. ROC AUC curve for combined, non-severe and severe classes after grid search and feature scaling.

The confusion matrix in *Fig 40* shows that even now, none of the severe cases are being predicted due to class imbalance. Hence, the thresholds were tuned to increase the number of TP results and decrease the number of FN results. *Fig 41* shows a sweep of threshold values from 0 to 1 with a step size of 0.01 to visualize the change in the above metrics.

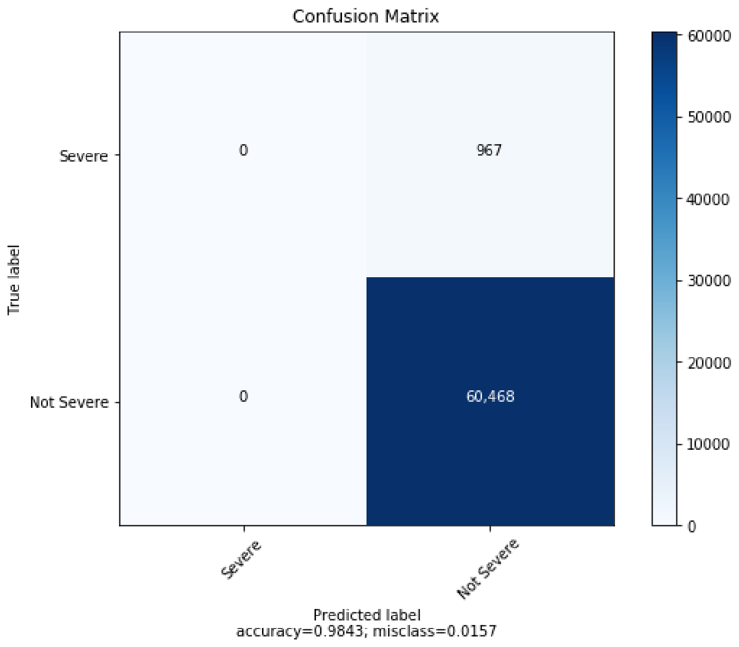


Fig 40. Confusion matrix for the optimized model.

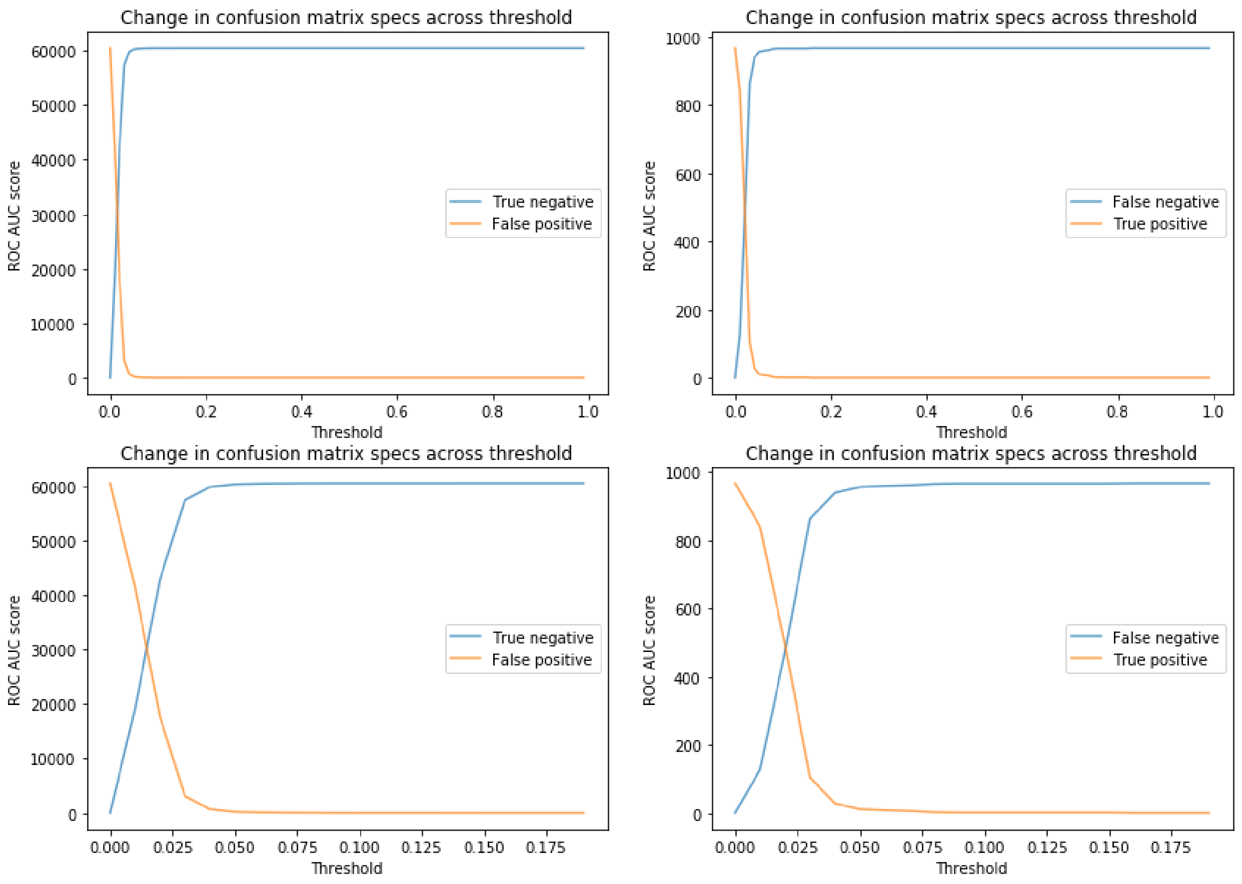


Fig 41. Confusion matrix metrics across thresholds.

Looking at the plots above, the number of TP increases and FN decreases with decreasing threshold. We start to see significant increases in TP values only at a threshold of less than 0.05. At about 0.025, the TP and FN are the same.

The trend is same for TN and FP. The interesting thing to note here is that as we decrease the threshold, our accuracy decreases while our AUC remains the same as we are just moving along the AUC curve. As threshold decreases, events go from TN to FP whereby decreasing the accuracy and increasing the misclassification error which can be seen in *Fig 42*. Similarly, values go from FN to TP for decreasing threshold.

In our application where detecting severe collisions is the most important criteria, this is a fair tradeoff to consider.

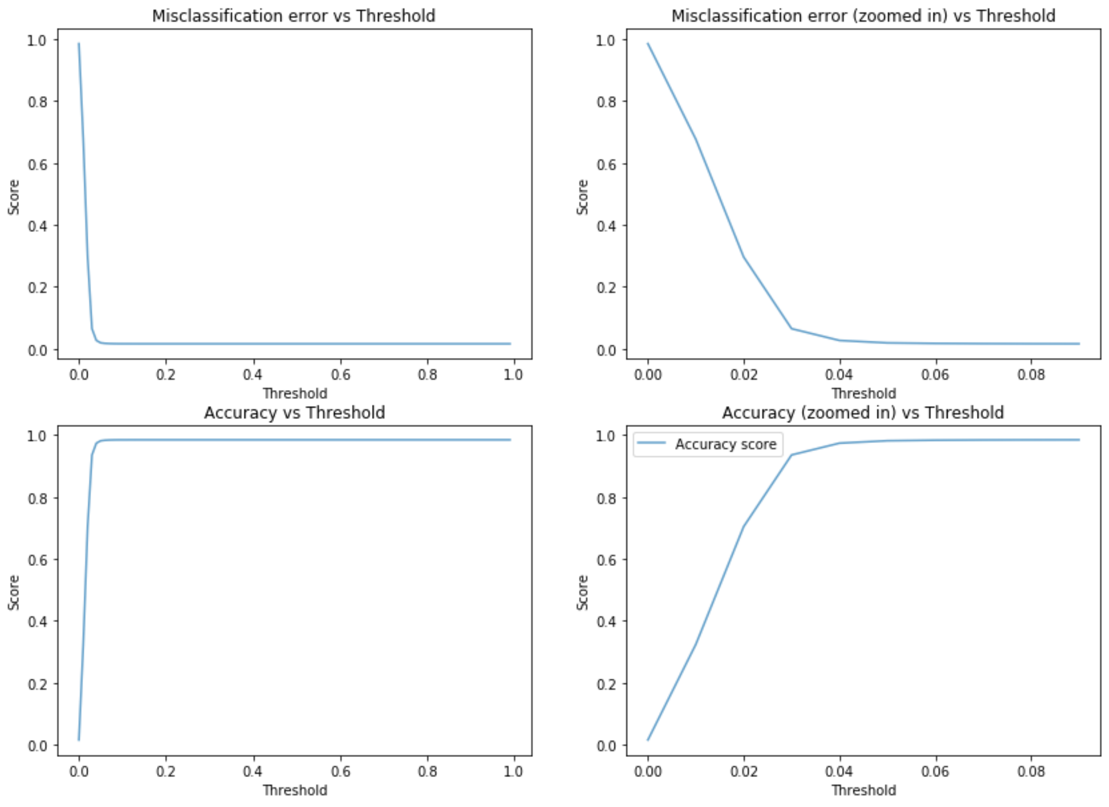


Fig 42. Accuracy and misclassification error across thresholds.

Fig 43 shows the most important features for the model based on the training data. *Road Length* and *Speed* are the best estimators followed closely by *Hour*. This makes sense as we had observed some association between these variables and severity during EDA. The least important features are *Day of the Week* and *Address Type*. For practical purposes, the importance numbers are more relative than absolute. For example, *Address Type* is not necessarily a bad feature, it just has less predictive power than a variable like *Road Length*.

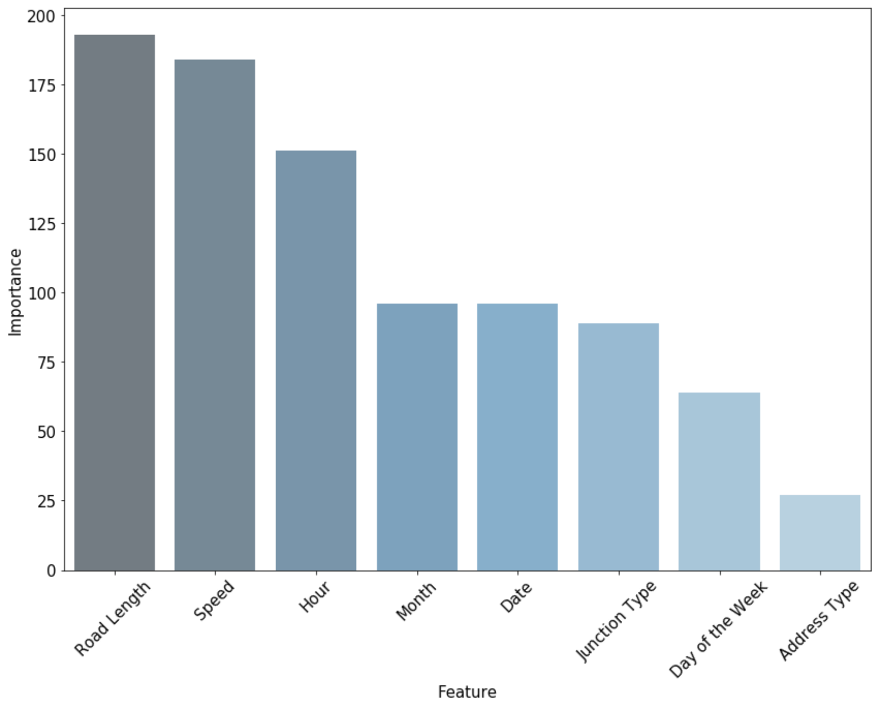


Fig 43. Important model features.

Fig 44 shows ROC AUC scores for train and test samples across the *number of leaves* and *max depth* hyperparameters. Since the number of leaves is the outer loop, the points at 0, 10, 20 and so on represent different number of leaves while the points between 0-10, 10-20 and so on represent different max depth values. The test sample performance remains fairly flat as compared to the train sample. Hence, higher train sample performance means greater overfitting. Based on this, points between 10-20 offer the best model in terms of overfitting. From point 20, each increase in test performance increases train performance by a higher rate, the ideal points being between 10-40. Anything above that would cause more and more overfitting.

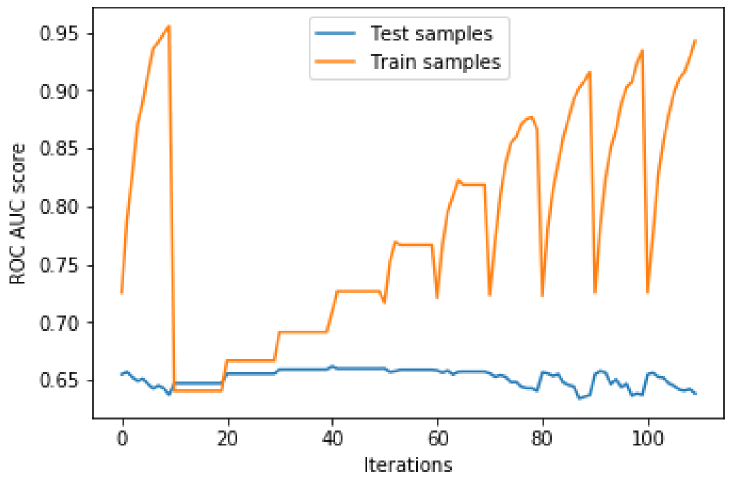


Fig 44. ROC AUC scores for train and test samples across hyperparameters.

Other models

Other baseline models that were considered were *Logistic Regression* and *Random Forest* which yielded **ROC AUC scores of 0.629 and 0.545** respectively. For the logistic regression model, the meeting point between the TP and FN values during the threshold sweep was much earlier. Therefore, it would also be a reasonable model for the application of collision severity prediction as the threshold would only need to be adjusted by a small value.

6. Conclusion

We learned that certain areas of the city are more prone to severe collisions than others with the center of the city being the most at risk. Additionally, routes along major roads going north, south and east also appeared to have a relatively higher density of severe incidents.

Class imbalance made prediction of the severe class difficult. Since not many of the features explained the variance in severity classes well, the final model was not robust enough in terms of accuracy, ROC AUC, precision and recall scores. Hence, oversampling methods such as SMOTE were not effective in dealing with the imbalance problem. Surprisingly, location-based features such as *Weather*, *Road Condition*, *Light Condition* and *Neighborhood* did not provide much useful information to differentiate between the two severity categories. The more important variables turned out to be the ones related to traffic flow, road dimensions, date and time.

An optimized LightGBM model provided a ROC AUC score of 0.66 for the severe class. Although, the default threshold of 0.5 prevented the model from predicting any severe instances. Therefore, the threshold had to be lowered to less than 0.1 to see some correctly predicted severe cases whereby decreasing false negatives and increasing false positives. However, this was achieved at the expense of greater false positives and lesser true negatives as a result.

A good threshold to use is 0.02 which yields 493 true positives, 474 false negatives, 17697 false positives and 42771 true negatives. At this threshold, the accuracy is reduced to 0.7042 with a corresponding misclassification error of 0.2957. Since we are more interested in predicting severe collisions at any cost, even a lower threshold would work which would generate more true positives. Despite a higher false positive rate, it would be better to have many false alarms if it meant that we could improve the true positive rate as well.

The optimized parameters provided the least overfitting model. This was observed when the performance using both train and test scores were plotted for different values of parameters maximum depth and number of leaves which are often used for regularizing the model.

Since the data covered collisions only in the Seattle area and majority of the features were location-based, the results of the model for predicting severe collisions cannot be generalized to the US. However, similar analytical techniques can be adopted for different cities and states across the country as well as other countries to create useful models and gain insights through exploratory data analysis (EDA). It can be assumed that random sampling was adopted during the retrieval of this data for purposes of any statistical analysis.

7. Appendix

Jupyter Notebooks

- https://github.com/saychelsea11/Springboard-Projects/blob/master/Capstone1_Project/Cleaning_and_wrangling_Seattle_collision_data.ipynb
- https://github.com/saychelsea11/Springboard-Projects/blob/master/Capstone1_Project/binary_class_conversion_handling_unknown_category.ipynb
- https://github.com/saychelsea11/Springboard-Projects/blob/master/Capstone1_Project/adding_neighborhood_speed_and_road_variables.ipynb
- https://github.com/saychelsea11/Springboard-Projects/blob/master/Capstone1_Project/exploratory_data_analysis.ipynb
- https://github.com/saychelsea11/Springboard-Projects/blob/master/Capstone1_Project/modeling_predicting_severe_collisions.ipynb

Data

- http://data-seattlecitygis.opendata.arcgis.com/datasets/5b5c745e0f1f48e7a53acec63a0022ab_0/geoservice