BA 723 - Business Analytics Capstone

Toronto Traffic Collision Prediction Model

Rakeen Ahmed

301307050

Analytics Startup Plan

Synopsis

Project	Toronto Traffic Collision Prediction Model
Requestor	Centennial College
Date of Request	July 11, 2023
Target Quarter for Delivery	August 16, 2023
Epic Link(s)	N/A
Business Impact	The business impact of this project is to help build safer streets for the residents and road users of Toronto by minimizing traffic collisions in the city. The project will support the Toronto Traffic Services in identifying hot spots on Toronto roads where there is high risk of traffic collisions occurring, as well as provide a deeper understanding of the key factors that lead to an increased risk of collisions. The project will also allow traffic services to take proactive actions based on the identified risky road segments/intersections to mitigate collision risk, inform road design and public policy choices, as well as assist in the optimization of police resource deployment.

1.0 Business Opportunity Brief

In the last 10 years, a total number of 1,581 deaths have resulted from road accidents in Toronto,
Canada between 2011 and 2020 with an average yearly road accident-related death of 158.1. Road accidents are the 3rd leading cause of death in Toronto.

The project arose from the need to provide data-driven insights and recommendations in delivering safety measures to reduce road accidents for the Vision Zero 2.0 project which aims to eliminate collisions in several world cities, including Toronto. An important

part of the Vision Zero 2.0 project is to minimize collisions through policy and road design changes, and strategic deployment of police resources to high-risk roads and intersections.

This project will assist in the identification of roads and intersections with high collision risk by developing a traffic collision prediction model for the City of Toronto based on various data points – past collisions, weather data, average traffic speeds and volumes, etc, to predict the likelihood of a collision occurring on a particular road segment on a given day. Using these likelihood estimates, roads and intersections can be ranked or mapped according to collision risk, which will allow traffic services to determine where best to dedicate scarce police resources, as well as identify roads where design or policy changes are required to decrease the risk of collisions before they occur.

1.1 Supporting Insights

Similar collision prediction models developed in the past:

1. City of Montreal: 1905.08770.pdf (arxiv.org)
2. State of Utah: Using Machine Learning to Predict Car Accident Risk | by Daniel Wilson | GeoAI | Medium
3. New York City: Predictive Analytics on NYC Collision Data | by Anushka Sandesara | Medium

1.2 Project Gains

The successful completion will allow traffic services to determine where best to dedicate scarce police resources, as well as identify roads where design or policy changes are required to ensure safety for users.

If sufficient precautions are taken by the city to mitigate collision risks identified by the model, it should lead to a decrease in the number of collisions on Toronto roads leading

to reduced loss of life and property, as well as increased operational efficiency for the Toronto police services.

The implications of not carrying out this project would lead to preserving the status quo of high number of traffic collisions in the City of Toronto, leading to loss of life and property, as well as failure to progress on the objectives of the city's Vision Zero plan of reducing traffic collisions and fatalities.

2.0 Analytics Objective

The primary objective of this project is to develop a collision prediction model to estimate the probability of a traffic collision occurring on a given road segment or intersection within the City of Toronto in a given day.

2.1 Other Related Questions and Assumptions

We will also look at these specific to gain a better understanding of traffic collisions in Toronto:

1. Which intersections and roadways in Toronto have the highest collision rates?
2. When and where to most crashes occur?
3. How does the spatial pattern of fatal collisions differ from the spatial pattern of collisions overall?
4. Where are collisions likely to disproportionately affect vulnerably road users?

The major assumption made by this project is that the likelihood of traffic collisions occurring can be modeled and predicted by analyzing the features of a road segment or intersection such as traffic and pedestrian volume, traffic controls, and previous collision record.

2.2 Success Measures and Metrics

The success metrics used to evaluate the performance of the predictive models will be accuracy for the classification models and mean absolute percentage error (MAPE) and normalized mean absolute error.

For the project to be considered successful, it must be able to identify road segments/intersections within the City of Toronto that have high risk of traffic collisions. These road segments and intersections must be rank-able by the predicted collision probability, or the predicted number of collisions, so that the Toronto traffic services can identify which intersections to dedicate resources and design changes towards.

2.3 Methodology and Approach

Background

Traffic Collisions are one of the major public health risks in the world today. The World Health Organization describes traffic systems as one of the most dangerous systems with which people interact every day (World Health Organization, 2022). In a recent study, the WHO reported that traffic collisions are the number one cause of mortality for people between the ages of 15 to 29, claiming more than 1.3 million lives every year (World Health Organization, 2022). Toronto is no exception - Traffic Collisions are the third leading cause of mortality in Toronto after heart disease and Alzheimers, with a total of 1,581 deaths resulting from collisions in the last 10 years (Statistics Canada, 2019).

To combat this issue, many major cities around the world, including Toronto, have launched their *Vision Zero* program. The core objective of the program is to eliminate deaths and serious injuries resulting from traffic collisions through the deriving

data-driven insights based on past traffic collision data, and to inform the implementation of road infrastructure and design improvements in high-risk road segments to proactively reduce the risk of traffic collisions on Toronto roads (City of Toronto, 2020).

Problem Statement

The project arose from the need to provide data-driven insights based on a combination of road design and infrastructure, traffic volumes, and previous traffic collisions datasets to inform road safety and design improvements for the Vision Zero program, as well as to identify the key drivers of collision risk. This will be achieved through the development of a Collision Prediction Model for the City of Toronto that uses information about the road infrastructure and design, as well as past collision data to predict the collision risk of a given road segment in the city. The model can be used to identify road segments and intersections with high risk of collisions, which will help identify roads where design of policy changes are required, or dedicate scarce police resources. The Collision Prediction model can also be used to identify the key drivers of collision risk, which can inform the implementation of design changes that have a positive impact on reducing collisions.

Project Significance

The successful completion of this project will allow Toronto Police Services where best to deploy scarce police resources, as well as identify roads where design or policy improvements are required to ensure safety for users. It will also allow infrastructure planners to make decisions based on the key drivers of collisions identified by the model.

If sufficient precautions are taken by the City to mitigate collision risks on road segments identified by the model, it should lead to the decrease in the number of

collisions in Toronto - leading to reduced loss of life and property, as well as increased operational efficiency for the Toronto Police Services.

Objective

As such, the primary objective of our project is to develop a collision prediction model for Toronto using machine learning models to estimate the risk of traffic collisions occurring at a given road segment (expressed as the Collision Rate of that road segment), and to derive the key features that drive collision risk on Toronto roads.

Additionally, we will also look to use the collision prediction model to develop a risk map of each road segment in Toronto to visualize the priority roads and intersections to consider for improvements.

To do this, we will be training different types of both regression and classification models including Decision Trees, Random Forests, Neural Networks, and Boosting Algorithms to understand the key drivers of each, and select the best model based on several error metrics to use for risk mapping.

Assumptions

For this project, we make a number of key assumptions based on available data, as well as modeling traffic collisions. Some of the assumptions are listed below:

The major assumption made by this project is that collision risk can be modeled and predicted by analyzing the features of a road segment - such as traffic lights, pedestrian volumes, and number of lanes etc.

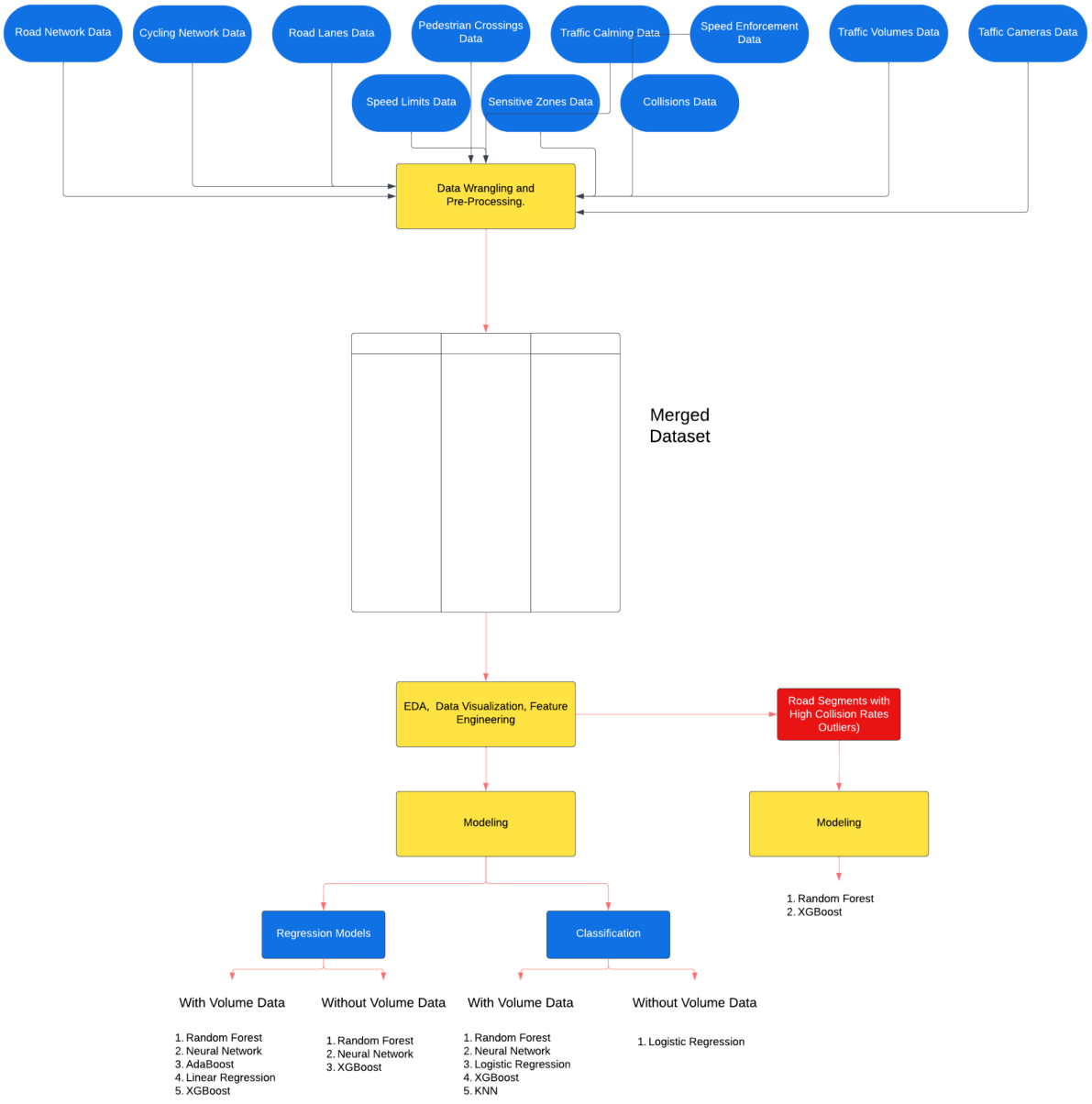
We are also assuming that the data, derived and merged from various sources, is accurate and up-to-date, and that data for the road segment features have stayed stationary throughout the observation window of this analysis - from 2016 to 2023. Additionally, we are assuming that all collisions are reported and recorded by the

Toronto Police, especially minor collisions that may not necessarily trigger a police response.

Finally, as traffic volume data was not available for all road segments, we merged traffic volumes for all road segments using a radius search of 100 meters. The assumption made was that traffic volumes did not differ significantly across a distance of 100 meters for any road segment.

Project Process Flow

The process flow diagram below outlines the project process to be followed in the implementation of this project.



First, data from various sources was cleaned, pre-processed and merged into one dataset based on their spatial characteristics. Then, we performed extensive EDA and pre-processing on the merged dataset, as well as feature engineering to create new features for modeling.

We then developed multiple regression and classification models based on the merged dataset, building separate models including and without traffic volumes, to see the effect of traffic volume on collision rates. Since collision rates included a lot of outliers, we also separated an additional dataset that included road segments with very high collision rates to model separately.

Data Summary

Data Set Introduction

The dataset used to build the collision prediction model was derived from merging multiple datasets from various sources, most of which are from open data portals from the City of Toronto, the Government of Ontario, and the Toronto Police Services. Open Data Portals are data repositories operated by public agencies that make the data they collect for their operations available to the public. Open Data is defined as “structured data that is machine-readable, freely shared, used and built on without restrictions” (K.Kabasakalis et al., 2019).

The datasets used in our analysis include:

1. **Toronto Centreline Road Network**: This dataset provided by the City of Toronto, contains the geometry of all road segments in Toronto defined by the City, as well as the centreline id, road name, addresses, road classification along with other additional information. We will be using this dataset as our base dataset for merging to understand the design features for each road segment and the associated number of collisions. For this, we filtered the dataset to keep only the necessary columns - centreline id, name, street type, feature code, and feature code description. Additionally, we also filtered the dataset for specific road classes, which will be described later.

2. **All Traffic Collisions**: This dataset is provided by the Toronto Police Services and includes all motor traffic-related collisions in the City of Toronto from 2014 - 2022 with 553,780 rows. The dataset contains the date, time, and location of all traffic collisions, and whether the collision resulted in any injuries, property damage, or fatalities. For our analysis, we will take collision information for the past 7 years - from 2016 onwards.

3. **Automated Speed Enforcement**: Includes the location and spatial geometry of all automated speed enforcement systems in Toronto.

4. **Cycling Network**: Contains information about the cycling network in Toronto, with spatial geometry column for mapping. Also contains the street name and when the bike lane was installed.

5. **Centreline Lanes**: This dataset contains road width, number of lanes, and road classification for every road segment in Toronto.

6. **Pedestrian Crossing**: Contains spatial geometry and information about all pedestrian crossings in Toronto.

7. **Traffic Calming:** Contains street names, spatial geometry and count of all traffic calming measures in Toronto - such as speed humps, traffic islands, and speed cushions.

8. **Traffic Volumes**: This dataset is provided by Toronto Transportation Services and contains the daily traffic volumes collected at specific intersections through automatic traffic recorders in 15 minute intervals. The traffic volumes are divided into various modes of transport - pedestrians, vehicles, bicycles, trucks. The data was aggregated into daily traffic volume for use in our analysis.

9. **Traffic Cameras**: Contains spatial geometry and information about all traffic cameras in the City of Toronto.

10. **Posted Speed Limits**: This file contains the posted speed limits for all road segments in Toronto, including their spatial geometry and centreline id.

11. **Sensitive Zones**: This dataset was derived from multiple datasets that contained all educational, healthcare, child care, community and seniors centers with their name and locations. A location was identified as a “sensitive zone” if any of these institutions were present.

12. **Killed or Seriously Injured (KSI)**: This dataset is also provided by the Toronto Police Services and includes a subset of all traffic collisions that resulted in major injuries or fatalities. The dataset provides some additional information about the collision such as whether a truck or bus was involved, the type of impact, age of the victim etc.

Exclusions

For our project, we will only be considering collision data for the past seven years: 2016 - 2021. Although the traffic collisions dataset contains data from 2014 onwards, we will exclude data for 2014 and 2015.

Initial Data Preparation

The initial datasets downloaded from the open data portals had to be cleaned, only keeping the necessary columns, checking for and replacing missing values, and preparing for merging. We also had to ensure that the different datasets were in the correct geometry format, as we would be merging them based on spatial information using the Python GeoPandas library. For the datasets of boolean values, such as Traffic Cameras, we added a column that would act as a flag if Traffic Camera was present in

that road segment. Finally, the datasets were exported to a GeoJSON format that could be read and merged using GeoPandas.

The data cleaning and preprocessing pipeline for a few of the datasets are provided below:

Centreline Lanes

The original shapefile downloaded from the Toronto Open Data portal did not have the geometry column in the proper format. The shapefile had to be uploaded to ArcGIS Online for processing and converting the geometry column to a LINESTRING format that we could use for analysis. The file was then extracted from ArcGIS as a GeoJSON file that could be read as a GeoPandas GeoDataFrame.

The exported dataset contained a lot of unnecessary columns that we would not need for our analysis. Only necessary columns were kept and renamed for the analysis.

```
#drop unneccesary columns
lanes = lanes[['lf_name','mar13rc2_3','numberofla','geometry']]
```

Python

```
#renaming columns
lanes = lanes.rename(columns={'lf_name':'road_name','mar13rc2_3':'road_class',
'numberofla':'number_of_lanes','width':'width'})
```

Python

```
lanes.head()
```

Python

	road_name	road_class	number_of_lanes	geometry
0	F G GARDINER XY W	City Expressway	3	LINESTRING (-79.54521 43.61462, -79.54518 43.6...
1	CENTENNIAL PARK BLVD	Collector	4	LINESTRING (-79.59005 43.64550, -79.59019 43.6...
2	BELVEDERE BLVD	Local	2	LINESTRING (-79.51511 43.65010, -79.51669 43.6...
3	LAKE SHORE BLVD W	Major Arterial	4	LINESTRING (-79.53400 43.59435, -79.53560 43.5...
4	F G GARDINER XY E	City Expressway	3	LINESTRING (-79.54509 43.61445, -79.54501 43.6...

The dataset did not contain any null values, as shown in the second figure.

```
lanes.isnull().sum()
```

0.0s Python

```
road_name          0
road_class         0
number_of_lanes    0
geometry           0
dtype: int64
```

```
#check to see if any rows are missing values for the geometry column
#no geometry column values are missing
lanes.loc[lanes.geometry.isnull(), 'geometry'].count()
```

0.0s Python

```
0
```

Next, we wanted to see the distribution of the number of lanes. From the distribution, we noticed that there were some lanes with number of lanes = 0.

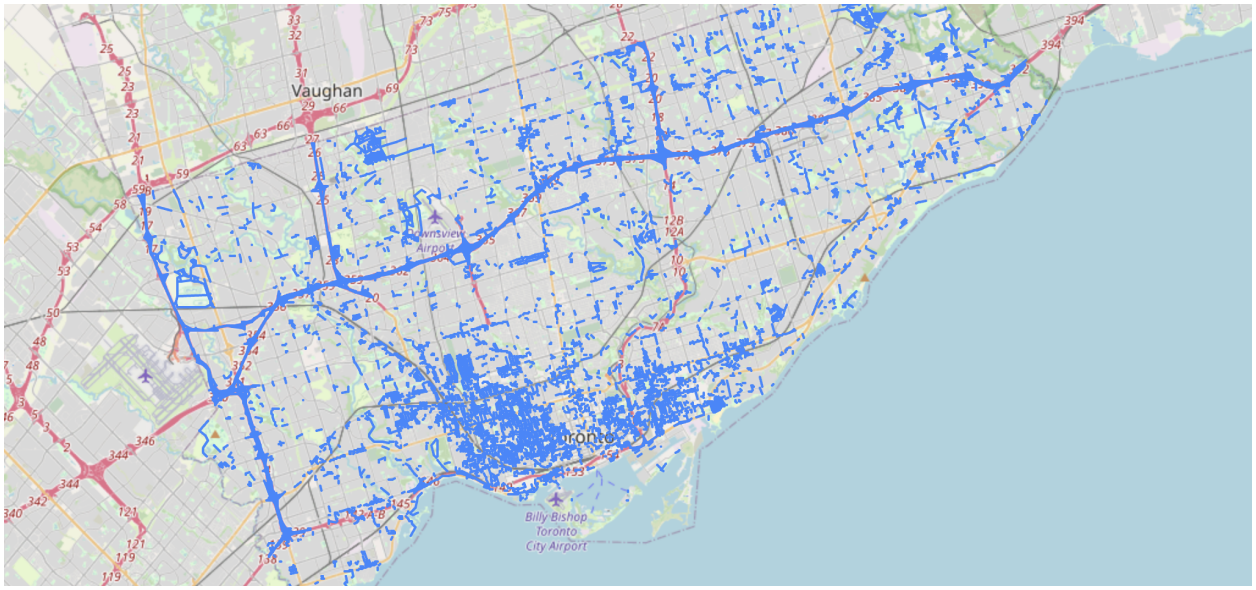
```
lanes.describe()
```

0.0s Python

	number_of_lanes
count	49451.000000
mean	1.910477
std	1.331600
min	0.000000
25%	2.000000
50%	2.000000
75%	2.000000
max	6.000000

There are some values where number of lanes 0. These will have to be treated.

Therefore, we wanted to check what type of roads had the number of lanes = 0. Upon further investigation, it was clear that road segments with 0 lanes assigned were mostly laneways or local residential roads where there are no lane markings, or provincial expressways such as the 401 or expressway ramps.



For the laneways and residential roads, as well as expressway ramps, we replaced the number of lanes with 1.

```
#if road_class is on the list and number of lanes = 0, replace number of lanes with 1
road_class_list = ['Laneway','Other','Local','Provincial Expressway Ramp','City Expressway Ramp','Unassumed Laneway','Private
Laneway','Park Road','Pending','Provincial Expressway']

lanes.loc[(lanes['road_class'].isin(road_class_list)) & (lanes['number_of_lanes']==0),'number_of_lanes'] = 1
```

Python

For the expressways, we replaced the number of lanes with 4, as most 400 series expressways in Toronto are 4 lanes.

```
road_class_list_2 = ['City Expressway','Major Arterial Ramp','Minor Arterial Ramp','Major Arterial']

lanes.loc[(lanes['road_class'].isin(road_class_list_2)) & (lanes['number_of_lanes']==0),'number_of_lanes'] = 4
```

36]

Python

Posted Speed Limits

The Posted Speed Limits dataset also had many columns that we did not need for analysis. Therefore, we only kept the necessary columns and discarded the rest.

```python
#dropping unneccessary columns

#specifying columns to keep
cols_to_keep = ['lf_name','mar13rc_11','posted_spe','geometry']

#dropping columns
speed_limits = speed_limits[cols_to_keep]
```

Python

Additionally, the dataset had 1 null value for the geometry column. As we will be using the geometry column for merging datasets, we cannot have null values for the geometry column. It is also impossible to impute or replace the null geometry value, as it would introduce false information into our dataset. Therefore, we dropped the row with the null value.

[6]
```python
speed_limits.isnull().sum()
```

```
lf_name       0
mar13rc_11    0
posted_spe    0
geometry      1
dtype: int64
```

[ ]
```python
#dropping speed limits that do not have a value for the geometry column
speed_limits = speed_limits.dropna()
```

Next, we checked the distribution of the speed limits in the dataset, and noticed that there were unnatural values for the posted speed limits in the dataset, such as a minimum speed limit of 0, or a maximum speed limit of 304050.

[10]
```python
speed_limits.posted_spe.describe()
```

Python

```
count     58956.000000
mean         54.034890
std        1276.070414
min           0.000000
25%           1.000000
50%          40.000000
75%          50.000000
max      304050.000000
Name: posted_spe, dtype: float64
```

Note: There are some unreasonable speed limits in the dataset that will have to be cleaned.

From mapping the posted speed limits of over 200 - these seemed to be inaccuracies in the dataset, as most of these road segments were local roads in North York and Scarborough. Therefore, we replaced them with a speed limit of 50 - a common speed limit for most roads in Toronto's outer boroughs.

```
#looking at the roads that have posted speed limits of over 200
#looks like mostly local roads so we can replace with median speed limit
speed_limits.loc[(speed_limits.posted_spe > 200)].explore()
```

✓ 0.0s


```
#replacing speed limits > 200 with speed limit = 50
speed_limits.loc[(speed_limits.posted_spe > 200), 'posted_spe'] = 50
```

As for the speed limits of under 10, most of them were in Toronto proper. We replaced them with the median speed limit in our dataset - 40.

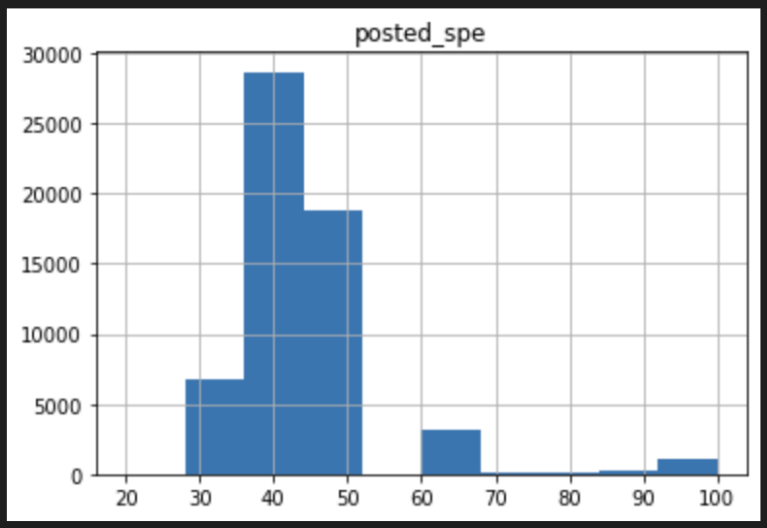
More...

```
#different types of roads but mostly local roads
#replace with median speed limit
speed_limits.loc[(speed_limits.posted_spe < 10), 'posted_spe'] = 40
```

After the data cleaning process, the final histogram of speed limits looked like the figure below, with most road segments having speed limits of between 30 km/h and 50 km/h, with some (expressways, major arterials) having higher. The cleaned dataset was then exported as a GeoJSON file for merging.

```
speed_limits.hist()
plt.show()
```

✓ 0.0s



Merging Datasets

After all the datasets were cleaned and processed, the datasets were merged with GeoPandas spatial joins using the geometry column present in each dataset. The geometry column represents the spatial information and shape of each feature within the dataset, and allows us to perform spatial operations and analysis, such as calculating distances, spatial joins, or plotting the data on a map. The process for the merge is outlined below.

First, we read in the processed datasets for merging. We printed the CRS (Coordinate Reference System) of a few of the datasets to ensure that they all had the same CRS. A Coordinate Reference System is used to define and interpret coordinates in spatial analysis, and establishes a standard way of representing geographic data to ensure consistent analysis across different datasets. If we want to merge two datasets based on spatial geometry, we need to ensure that they are using the same CRS to avoid errors in our merged dataset.

```python
#reading in all datasets
road_network = gpd.read_file('road_network_processed.geojson')
#road_network = gpd.read_file('ontario_road_network_processed.geojson')
bikelanes = gpd.read_file('cycling_network.geojson')
collisions = gpd.read_file('collisions_processed.geojson')
intersections = gpd.read_file('centreline_intersections_processed.geojson')
ksi = pd.read_csv('ksi_processed.csv')
lanes = gpd.read_file('lanes_processed.geojson')
pedestrian_crossings = gpd.read_file('pedestrian_crossover_processed.geojson')
sensitive_zones = gpd.read_file('sensitive_areas_processed_final.geojson')
speed_enforcement = gpd.read_file('speed_enforcement_processed.geojson')
speed_limits = gpd.read_file('speed_limits_processed.geojson')
traffic_calming = gpd.read_file('traffic_calming_processed.geojson')
traffic_cameras = gpd.read_file('traffic_cameras_processed.geojson')
traffic_volumes = pd.read_csv('traffic_volumes_processed_2.csv')
```

Python

```python
print(road_network.crs)
print(bikelanes.crs)
print(lanes.crs)
print(sensitive_zones.crs)
print(collisions.crs)
```

Python

```
EPSG:4326
EPSG:4326
EPSG:4326
EPSG:4326
EPSG:4326
```

We used the Toronto Centreline Road Network dataset as our base dataset for merging. We will then merge each additional feature of the road segment onto the Toronto Centreline dataset. This is because our collision prediction analysis will be based on each road segment's features and its associated collision rate. To identify each unique road segment, we set the centreline_id as the index.

Merging Cycling Network

```
#setting centreline_id id as index
#road_network = road_network.set_index('centreline_id')
road_network = road_network.set_index('centreline_id')
```

Python

```
road_network.head(2)
```

Python

centreline_id	linear_name_full_legal	linear_name_type	feature_code	feature_code_desc	geometry
914600	Morrison Street	St	201500	Local	MULTILINESTRING ((-79.50875 43.59744, -79.5098...
914601	Twelfth Street	St	201500	Local	MULTILINESTRING ((-79.50987 43.59720, -79.5103...

More...

The first dataset we merged to the Road Network data was the Cycling Network dataset. We dropped a few columns that were not needed from the cycling network dataset and added a flag for a bike lane being present.

Joining Bikelanes

```
#bikelanes data processing
bikelanes.columns = [s.strip().lower() for s in bikelanes.columns]
#drop unneeded columns
bikelanes = bikelanes.drop(columns=['objectid','segment_id','upgraded','pre_amalgamation','roadclass','cnpclass','surface','owner',
'dir_loworder','sepa_loworder','sepb_loworder','orig_loworder_infra','dir_highorder','sepa_highorder','sepb_highorder',
'orig_highorder','bylawed','editor','last_edit_date','upgrade_description','converted'])
#rename id column
bikelanes = bikelanes.rename(columns={'_id':'id'})
#set id column as index
bikelanes = bikelanes.set_index('id')
#adding a bikelane_present column
bikelanes['bikelane_present'] = "Yes"
```

[9]

Python

Then we joined the Cycling Network dataset using a geopandas sjoin_nearest function, which merges two GeoDataFrames based on a specified distance, even if an exact spatial match is not found. For this join, we used a max_distance parameter of 0.0003, which translates to a distance of 30 meters. Therefore, if a specific road segment had a bike lane within a 30 meter radius of that segment, the cycling network information would be merged to that segment. For our purposes, we would flag that road segment as having a bike lane present.

```python
#joining road network and bikelanes using a radius search of 0.0003 degrees = 30 metres.
#essentially road segments that have a bikelane present within 30 metres
join_1 = road_network.sjoin_nearest(bikelanes,how='left',max_distance=0.0003,lsuffix='road_network',
rsuffix='bikelanes')
```

Python

As we used a Left Join to merge the two datasets, all road segments with no bike lanes within 30 meters would have N/A values for the bike lane columns. To fix this, we imputed the missing values for the bikelane_present column as “NO”, indicating a bike lane was not present. We dropped the rest of the columns from the bike lane dataset as we did not require them for the analysis.

```python
join_1.bikelane_present.fillna("No",inplace=True)
```

Python

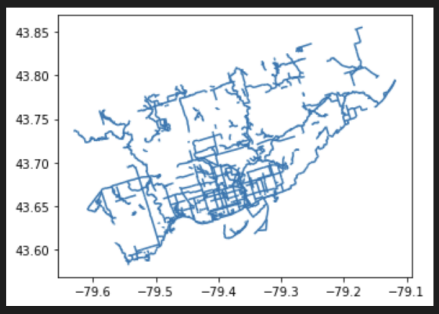
```python
join_1 = join_1.drop(columns=['index_bikelanes','installed','street_name','from_street','to_street',
'infra_loworder','infra_highorder'])
```

Python

We then checked the plot for the cycling network dataset and the merged dataset where bikelane_present = “YES” to check if the merge was successful.

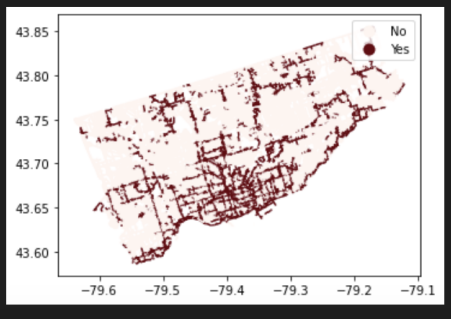
```
bikelanes.plot()
```

```
<AxesSubplot:>
```




```
#color plot by bikelane_present: red if bikelane is present, blue if bikelane is not present
join_1.plot(column='bikelane_present',legend=True,cmap='Reds')
```

```
<AxesSubplot:>
```



Merging Traffic Cameras

The Traffic Cameras dataset needed more processing, and had to be converted into a GeoDataFrame before it could be merged.

```python
#processing traffic cameras dataframe

#setting camera number as index
traffic_cameras.set_index('camera_number',inplace=True)

#dropping latitude and longitude columns
#traffic_cameras = traffic_cameras.drop(columns=['traffic_cameras_latitude','traffic_cameras_longitude'])

#renaming columns
traffic_cameras.rename(columns={'traffic_cameras_traffic_cameras':'traffic_cameras_present'},
inplace=True)

#setting CRS to EPSG:4326
traffic_cameras = traffic_cameras.to_crs('EPSG:4326')
```

Python

```python
#processing geometry column

#replacing the geometry column
traffic_cameras = traffic_cameras.drop(columns=['geometry'])

traffic_cameras['geometry'] = gpd.points_from_xy(traffic_cameras['traffic_cameras_longitude'],
traffic_cameras['traffic_cameras_latitude'],crs='EPSG:4326')

#converting to geodataframe
traffic_cameras = gpd.GeoDataFrame(traffic_cameras,geometry=traffic_cameras['geometry'])
```

Python

We set the camera number as the index, as each camera number is a unique identifier and renamed some columns. We also had to create a new geometry column from the Latitude and Longitude coordinates, as the existing geometry column was in a different CRS format. To do this, we used the GeoPandas gpd.points_from_xy function and set the CRS to EPSG:4326 and stored the values returned by the function as the new geometry column. Finally, we converted the DataFrame to a GeoDataFrame that could be merged.

```python
#spatial join on traffic cameras
join_7 = join_6.sjoin_nearest(traffic_cameras,how='left',lsuffix='road_network',
rsuffix='traffic_cameras',max_distance=0.0003)
```

```
#dropping unneccesary columns
join_7 = join_7.drop(columns=['index_traffic_cameras','traffic_cameras_latitude',
'traffic_cameras_longitude','traffic_cameras_main_road'])


#filling N/A values where traffic camera is not present
join_7.traffic_cameras_present.fillna('No',inplace=True)
```

As with the cycling network dataset, we joined the traffic cameras dataset to the base road network data using a distance of 30 metres, dropped inessential columns, and filled missing values with “NO”, indicating a traffic camera was not present in that road segment.

The remaining datasets of road infrastructure and features were merged to the base road network dataset using a similar process as the cycling network and traffic cameras datasets.

Merging Traffic Collisions

The processed traffic collisions dataset contains information about all collisions from 2016 - 2023 in Toronto, including the road segment the collision occurred in. However, each collision is recorded as a separate incident, and we required a count of total collisions for each road segment.

To derive this information from the traffic collisions dataset, we created a pivot table of the count of collisions by geometry. The pivot table provides a count of collisions at each geometry, which we can merge to the road segment dataset to get the count of collisions for each road segment. We also converted the pivot table to a GeoDataFrame for merging.

```
collisions_pivot = pd.pivot_table(collisions,values='eventuniqueid',index=['geometry'],aggfunc='count')
```

```
collisions_pivot = gpd.GeoDataFrame(collisions_pivot,geometry=collisions_pivot.index.values,crs='EPSG:4326')
```

The resulting GeoDataFrame was merged to the road network dataset with a max_distance of 30 metres. We imputed the road segments that had no collisions with a collisions_count of 0 and dropped columns that were not needed.

```
#filling in NA values
join_10.eventuniqueid.fillna(0,inplace=True)
```

```
#dropping unneccesary columns
join_10 = join_10.drop(columns=['index_collisions'])
#renaming columns
join_10.rename(columns={'eventuniqueid':'collisions_count'},inplace=True)
```

The final dataset resulting from the merge was named final_df and exported as a GeoJSON file for further EDA and analysis.

Data Dictionary

The final dataset derived from merging the different datasets contained 13,979 rows and 26 columns. There are additional three columns included in the data dictionary below that were derived from feature engineering - Collision Rate Traffic, Collision Class, and Speed Hump Present. The columns, along with their data type and a brief description, are provided in the table below:

Variable	Data Type	Description
Centreline_Id	ID	The unique id of the road segment designated by the City of Toronto.
Road Name	String	The name of the road segment.
Road Hierarchy	Category	Road Hierarchy values such as St,

		Boulevard, Avenue.
Road Classification	Category	Road Classification according to the City of Toronto Road Classification system.
Bike Lane Present	Bool	Boolean value indicating the presence of a bike lane in the specific road segment.
Number of Lanes	Integer	Number of lanes in the road segment.
Pedestrian Crossover Present	Bool	Boolean value indicating whether a pedestrian crossover is present.
Speed Hump Count	Integer	Count of speed humps in the road segment.
Traffic Island Count	Integer	Count of traffic islands in the road segment.
Speed Cushion Count	Integer	Count of speed cushions in the road segment.
Speed Enforcement Present	Bool	Boolean value indicating whether speed enforcement measures are present.
Cyclists Count	Integer	Count of cyclists measured during the Traffic Volumes count window. Expressed as daily count.
Pedestrians Count	Integer	Count of pedestrians measured during the Traffic Volumes count window. Expressed as daily count.
Trucks Count	Integer	Count of trucks measured during the Traffic Volumes count window. Expressed as daily count.
Vehicle Count	Integer	Count of motor vehicles measured

		during the Traffic Volumes count window. Expressed as daily count.
Total Count	Integer	Aggregated value of all road users including cyclists, pedestrians, trucks, and vehicles during the count window. Expressed as daily traffic volume.
Pct_Vehicles	Float	Percentage of total count that was motor vehicle traffic.
Pct_Pedestrians	Float	Percentage of total count that was pedestrian foot traffic.
Pct_Cyclists	Float	Percentage of total count that was cyclists.
Pct_Trucks	Float	Percentage of total count that was trucks.
Traffic Camera Present	Bool	Boolean value indicating whether a traffic camera was present.
Speed Limit	Integer	The posted speed limit of the road segment.
Sensitive Zone	Bool	Whether the road segment is assigned as a Sensitive Zone.
Collisions Count	Integer	Count of total collisions in the road segment from 2016-2023.
KSI Count	Integer	Count of KSI collisions in the road segment from 2016-2023.
Collision Rate Traffic	Float	The rate of collisions of the road segment by 1000 road users. Derived by

		dividing collisions count by the total count of traffic.
Collision Class	Category	Bins of collisions counts, divided into 5 classes.
Speed Hump Present	Bool	Boolean value indicating whether a speed hump was present.
Geometry	Spatial/MultiLineString	Spatial geometry column indicating the location of the road segment.

Exploratory Data Analysis (EDA)

Initial Data Exploration

Once the merged dataset is prepared and exported, we conduct EDA on the dataset to prepare for modeling. We read the merged dataframe as a GeoDataFrame called final_df.

Missing Values

We checked the merged dataset for missing values. If found, missing values can be imputed. However, the dataset does not contain any missing values.

```
#checking for null values
final_df.isnull().sum()
```
✓ 0.0s

```
centreline_id                  0
road_name                      0
road_hierarchy                 0
road_classification            0
bikelane_present               0
number_of_lanes                0
pedestrian_crossover_present   0
speed_hump_count               0
traffic_island_count           0
speed_cushion_count            0
speed_enforcement_present      0
cyclists_count                 0
pedestrians_count              0
total_count                    0
trucks_count                   0
vehicle_count                  0
pct_vehicles                   0
pct_pedestrians                0
pct_cyclists                   0
pct_trucks                     0
traffic_cameras_present        0
speed_limit                    0
sensitive_zone                 0
collisions_count               0
ksi_count                      0
geometry                       0
dtype: int64
```

Duplicate Rows

We also check the dataset for duplicate rows based on the centreline_id that may have been created during the merging process. We find that the data contains 5,993 duplicate rows. The duplicate rows are dropped, and the resulting data frame has 7,986 rows.

```
final_df.centreline_id.duplicated().sum()
```
✓ 0.0s

```
5993
```

```
#drop duplicate centreline id
final_df = final_df.drop_duplicates(subset='centreline_id', keep='first')
```

```
#checking if all dulicates were dropped
final_df.centreline_id.duplicated().sum()
```

```
0
```

```
final_df.shape
```

```
(7986, 26)
```

Feature Engineering

Feature Engineering is the process of creating new and meaningful features from existing data to improve the performance of a machine learning model. For our analysis, we engineer three new features: Collision Rate, Collision Class, and Speed Hump Present.

Collision Rate

The Collision Rate feature was derived due to the fact that road segments that have higher traffic volumes are likely to have higher collision numbers, even if the design of the road segment is not particularly dangerous. As we want to understand how the design and infrastructure of a road segment can affect the collision risk, and the differences between road segments that have high versus low collision risk, we will be using the *collision rate* of that road segment for our model.

The collision rate is derived by dividing the collisions_count of a road segment by the total daily volume of traffic. The collision rate is then multiplied by 1000 to get the collision rate per 1000 road users (including vehicles, trucks, pedestrians, and cyclists).

```
# feature engineering

#creating column for collision rate by traffic volume * 1000 (1000 road users)
final_df['collision_rate_traffic'] = (final_df['collisions_count'] / (final_df['total_count'])) * 1000
```

Collision Class

The collision class feature was derived from collision counts to bin the total number of collisions for use in classification models. The total collisions count was divided into 5 bins: less than 5 collisions, 5 - 20 collisions, 20 - 50 collisions, 50 - 80 collisions, and over 80 total collisions count.

```
#creating columns for classification - collisions_count [classes = <5,5-20,20-50,50-80,80-100,>100]
def col Loading... ass(x):
    if x < 5:
        return '<5'
    elif x >= 5 and x < 20:
        return '5-20'
    elif x >= 20 and x < 50:
        return '20-50'
    elif x >= 50 and x < 80:
        return '50-80'
    else:
        return '>80'

final_df['collision_class'] = final_df['collisions_count'].apply(collision_class)
```

Speed Hump Present

This feature was derived to use the speed hump count as a boolean flag indicating whether a speed hump was present or not in a given road segment. Speed humps were chosen as they are by far the most common traffic calming measure found in our dataset.

```python
#converting Loading... mp count to binary
def speed_hump_present(x):
    if x > 0:
        return "YES"
    else:
        return "NO"

final_df['speed_hump_present'] = final_df['speed_hump_count'].apply(speed_hump_present)
```

Correlations

After the feature engineering step, we wanted to look at the correlation heatmap of all the existing and engineered features in the dataset. As shown in the heatmap, outside of the traffic volumes and percentage columns, no features in the dataset have significant correlation with each other (refer to Appendix 1).

Data Processing

Dropping Columns

The traffic volumes and percentage columns in the dataset were correlated with each other. Therefore, we drop the percentage columns, keeping only the traffic volumes columns. We also dropped the total count column, as the information in the column was captured in the count column for other transportation modes. Additionally, we drop some other columns that are not required for the analysis or highly imbalanced- such as road name, speed cushion count, road hierarchy etc.

```python
#set centreline id as index
final_df = final_df.set_index('centreline_id')

#drop unneccesary columns
final_df = final_df.drop(columns=['road_name','geometry','pct_pedestrians','pct_cyclists','pct_trucks',
'pct_vehicles','ksi_count','speed_cushion_count','traffic_island_count','speed_hump_count','total_count',
'road_hierarchy'])
```

Python

Setting Data Types

We set the data types for the variables to the proper format for modeling, including setting the data type of categorical variables to “category” and collisions count and number of lanes from “float” to “int64”.

```python
#setting data types for variables

#setting data types to category
final_df[['road_classification','collision_class','bikelane_present','pedestrian_crossover_present',
'speed_enforcement_present','traffic_cameras_present','sensitive_zone','speed_hump_present']] = final_df
[['road_classification','collision_class','bikelane_present','pedestrian_crossover_present',
'speed_enforcement_present','traffic_cameras_present','sensitive_zone','speed_hump_present']].astype
('category')

#setting data types to integer
final_df[['collisions_count','number_of_lanes']] = final_df[['collisions_count','number_of_lanes']].
astype('int64')
```

Python

Skews and Outliers

The numerical variables in our dataset were highly skewed and contained significant outliers, as shown in the figure below. Skewed data distributions can negatively impact the performance of machine learning models and hinder their ability to learn from the data. They can also bias the models understanding of how features impact the target variable, with models assigning higher importance to features with a larger range of values. Thus, we needed to treat the skewed distributions with cap & floor using a range of 3IQR, as well as log transformation techniques.

```
final_df.skew()
```

```
C:\Users\arara\AppData\Local\Temp\ipyke

Dropping of nuisance columns in DataFra
```

```
number_of_lanes            0.849748
cyclists_count             7.360736
pedestrians_count          9.977515
trucks_count               4.593409
vehicle_count              4.349662
speed_limit                1.217062
collisions_count           4.322991
collision_rate_traffic    25.347762
dtype: float64
```

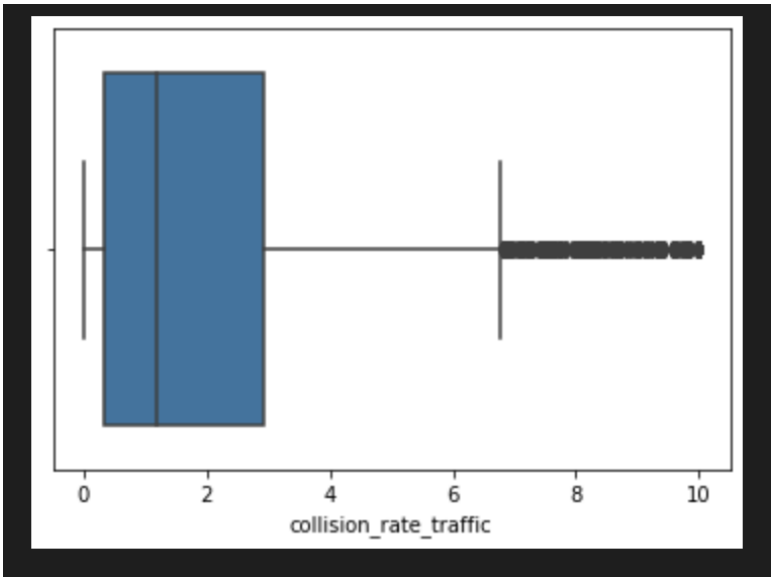
Collision Rate

Collision Rate was the variable with the highest skew. We capped the collision rate at 10, as the data distribution showed that although the maximum collision rate was 383, 75% of the road segments have collision rates of under 3. Even with capping the collision rate using a 3 IQR at 10, the data still contained some outliers, as shown in the box plot. We decided to leave those outliers to preserve the information about road segments with high collision rates. Later, we create a separate data frame of the road segments with these outlier collision rates to model them separately.

```
final_df.collision_rate_traffic.describe()
```

```
count    7986.000000
mean        2.839036
std         8.647943
min         0.000000
25%         0.344003
50%         1.199082
75%         2.919239
max       383.529412
Name: collision_rate_traffic, dtype: float64
```

```
#capping collision rate to 50, which only affects 376/7986 rows
final_df['collision_rate_traffic'] = final_df['collision_rate_traffic'].clip(0,10)
```



Collisions Count

Collisions Count was another target variable that we wanted to use that had a relatively high skew of 4.322 and thus had to be treated.

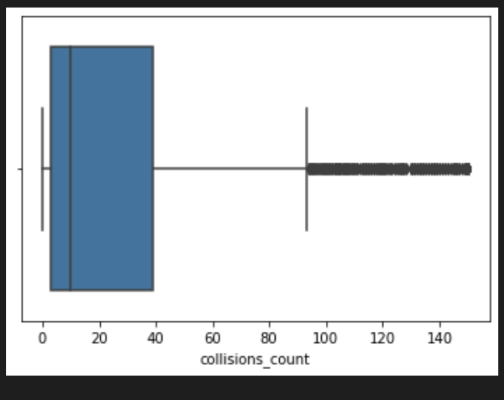
```
count    7986.000000
mean       43.746932
std        92.159074
min         0.000000
25%         3.000000
50%        10.000000
75%        39.000000
max      1064.000000
Name: collisions_count, dtype: float64
```

```
[250]
#capping collisions count 150, which only aff                7986 rows
final_df['collisions_count'] = final_df['coll                unt'].clip(0,150)
```

Loading...

```
[251]
#still quite a few outliers
sns.boxplot(x='collisions_count',data=final_df)
```

```
<AxesSubplot:xlabel='collisions_count'>
```



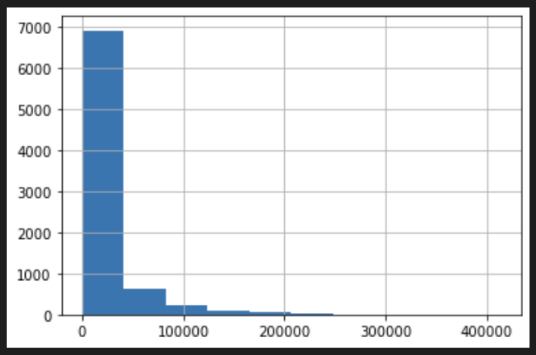
The descriptive statistics of the variable shows that the variable had significant outliers, with a minimum collision count of 0 and 75th percent of the data being under 40. However, the maximum count was 1064. Using the pandas .clip() function, the collision count was capped using a range slightly higher than 3 IQR range at 150 collisions. In our modeling step, we will be using a collisions count of over 80 as a very high collisions count.

The resulting box plot shows that the data still has quite a few outliers, but we decided to keep them so as not to lose valuable information in our dataset.

Vehicle Count

```
print(final_df.vehicle_count.describe())
final_df.vehicle_count.hist()
```

```
count      7986.000000
mean      22166.630353
std       35783.419902
min          95.000000
25%        4603.000000
50%       11004.000000
75%       22712.500000
max      412989.000000
Name: vehicle_count, dtype: float64

<AxesSubplot:>
```




```
#clipping vehicle count to 75000 using 3IQR range, which only affects 519/7986 rows
final_df['vehicle_count'] = final_df['vehicle_count'].clip(0,75000)
```

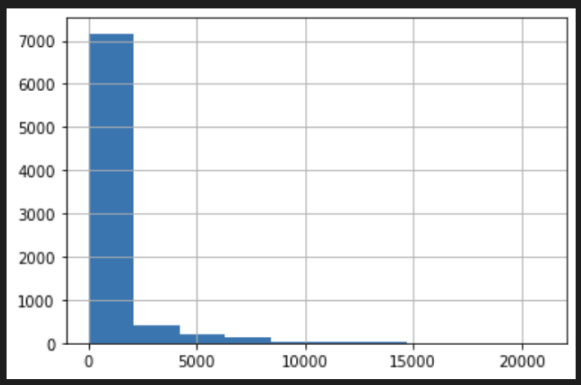
As shown by the descriptive statistics and histogram, vehicle count was also relatively skewed, with a long tail on the right. Again, we capped this variable using a range of 3 IQR, making sure not to affect too many rows.

Trucks Count

```
print(final_df.trucks_count.describe())
final_df.trucks_count.hist()
```

```
count     7986.000000
mean       879.546081
std       1781.897633
min          0.000000
25%         78.000000
50%        292.000000
75%        801.000000
max      21004.000000
Name: trucks_count, dtype: float64
```

```
<AxesSubplot:>
```




```
#clipping trucks count to 3000 using 3IQR range, which only affects 550/7986 rows
final_df['trucks_count'] = final_df['trucks_count'].clip(0,3000)
```

Trucks Count was capped at 3000 using a 3 IQR range.

Pedestrian Count

Pedestrian Count was capped at 10,000 using a 3 IQR range.

```
#clipping pedestrian count to 10000 using 3IQR range, which only affects 730/7986 rows
final_df['pedestrians_count'] = final_df['pedestrians_count'].clip(0,10000)
```

Cyclists Count

Cyclists Count was capped at 2000

```
#clipping cyclists count to 2000, which only affects 631/7986 rows
final_df['cyclists_count'] = final_df['cyclists_count'].clip(0,2000)
```

Log Transformation

After conducting Cap & Floor on our numerical variables to treat the outliers, we performed Log Transformation on our dataset to fix the skewed distributions, as well as bring the dataset values to a similar range, using a log base of 2. Log Transformation is used to reduce skewness in the dataset by compressing the range of values, making the distributions more symmetric and bringing extreme values closer to the mean. Log Transformation was chosen instead of other methods as our data is mostly positively skewed and contains outliers.

As some of our numerical variables contained values that were 0, we could not apply Log Transformation directly. Therefore, we added 1 to all numerical values in the dataset before applying the log transformation.

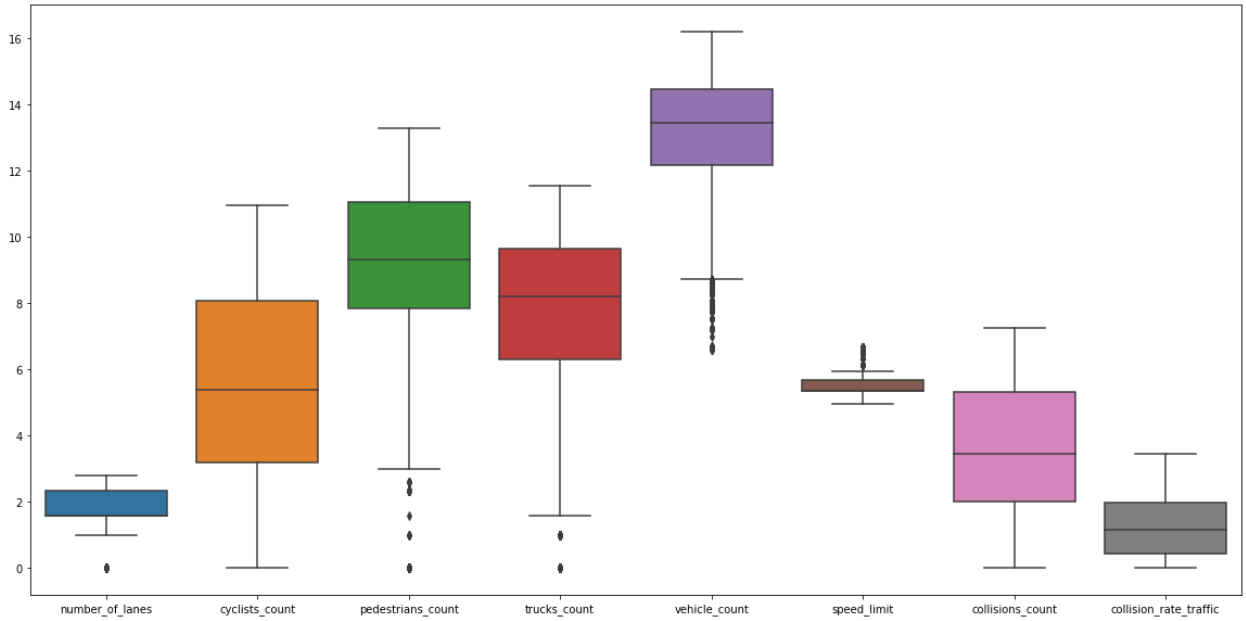
```python
#applying log transformation on the numerical variables. variables that have a value of 0 are replaced
with 1 before applying log transformation.
def log_transform(x):
    return np.log2(x+1)

final_df[['cyclists_count','pedestrians_count','trucks_count','vehicle_count','speed_limit',
'collisions_count','collision_rate_traffic','number_of_lanes']] = final_df[['cyclists_count',
'pedestrians_count','trucks_count','vehicle_count','speed_limit','collisions_count',
'collision_rate_traffic','number_of_lanes']].apply(log_transform)
```

✓ 0.0s Python

Upon re-evaluating the skews in the dataset, we found that the log transformation had successfully fixed the skewed distributions.

```
number_of_lanes          -0.233285
cyclists_count            0.167607
pedestrians_count        -0.320422
trucks_count             -0.573500
vehicle_count            -0.556127
speed_limit               0.155743
collisions_count          0.126758
collision_rate_traffic    0.569175
dtype: float64
```

The box plots of the numerical variables also show that the data is symmetrical, and all variables are in a similar range, with only vehicle count having noticeable outliers.



Summary of EDA

In summary, we started our EDA on the dataset by checking for missing values and duplicate rows, which were dropped. After that, we conducted feature engineering, creating three new features: collision rate, collision class, and speed hump present. The collision rate and collision class features will be used as targets for our regression and classification models respectively. We then checked for correlations between variables to decide which variables to include in our modeling. We discovered our dataset did not contain many correlated variables, except for the traffic counts and percentages.

Therefore in the data processing step conducted next, we dropped the traffic percentage columns, along with other columns that were not necessary to our analysis, and also set the data types of the remaining variables to the correct format.

Finally, we treated the skewed variables and outliers in our dataset by applying cap & floor, as well as log transformation to the skewed variables, and bringing them to a similar range. The final skews and box plot showed that the treated variables all had skews close to zero and were on a similar range. Therefore, we decided not to perform additional treatment on the data set such as standardization or normalization.

Modeling

To develop the final collision prediction model, we will be training different types of models with both regression and classification targets to pick the best one. We will also be further subdividing the models to take into account or leave out traffic volumes, as our initial exploratory modeling showed that traffic volumes were a significant predictor of overall collisions count, as well as collision rates. Furthermore, we will be developing models on road networks with extremely high collision rates from the outliers identified in our EDA, to assess the key drivers of collisions in these road segments.

To start the modeling process, we import the necessary libraries. Other libraries will be imported as needed later on.

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.ensemble import AdaBoostClassifier, AdaBoostRegressor
from sklearn.neighbors import KNeighborsClassifier
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from dmba import regressionSummary, classificationSummary
!pip install xgboost --quiet
import xgboost as xgb
```

Metrics

For the regression models, we will be evaluating the models on various metrics, such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and normalized Mean Absolute Percentage Error. A brief description of each of these error metrics are given below:

1. **Mean Squared Error (MSE)**: Measures the average squared distance between the predicted values and the actual values. MSE gives more weight to larger errors by penalizing larger errors heavily because of the squaring mechanism.

2. **Mean Absolute Error (MAE)**: Calculates the average absolute difference between the predicted and actual values. MAE treats all errors the same regardless of direction (positive or negative), and gives equal weight to both large and small errors.

3. **Mean Absolute Percentage Error (MAPE)**: Calculates the average percentage difference between the predicted and actual values, and is useful when evaluating the performance of a model in terms of relative errors rather than actual error values. It is more suitable for comparing error statistics of models with different target variables which may be in different scales.

4. **Normalized Mean Absolute Error**: This error statistic was derived from normalizing the Mean Absolute Error value by dividing it by the range of the target variable. This provides a way to compare error statistics between target variables with different ranges and compare the accuracy of models on a more consistent scale. The formula is shown below.

   Normalized MAE = MAE / (Valid y max - Valid y min)

To compare the performance of our regression models, we will use a combination of Mean Absolute Percentage Error and Normalized Mean Absolute Error. This is because our regression model contains two different targets - Collisions Count and Collision Rate - which are on different scales, which introduces problems with comparing them using MSE or MAE. The Normalized Mean Absolute Error statistic was selected because MAPE, although appropriate for comparing models with different targets, is sensitive to division-by-zero if the true values are close to zero, resulting in extremely large MAPE values. Thus, Normalized MAE provides a more robust metric to compare the performance of models with different targets.

To get all the regression model error statistics quickly, we defined a function called regression_summary() to print all the metrics.

```python
#creating a function to return regression summary
def regression_summary(model):
    print('Regression Statistics:\n')
    print(f'Mean Squared Error (MSE): {mean_squared_error(valid_y,model.predict(valid_X))}')
    print(f'Mean Absolute Error (MAE): {mean_absolute_error(valid_y,model.predict(valid_X))}')
    print(f'Mean Absolute Percentage Error (MAPE): {mean_absolute_percentage_error(valid_y,model.predict
    (valid_X))}')
    print(f'Normalized MAE:{mean_absolute_error(valid_y,model.predict(valid_X))/(valid_y.max()-valid_y.
    min())}')
```

Python

For the classification models, we will simply be using the Accuracy metric derived from the confusion matrix of the classification summary. The Accuracy of a classification model is calculated by summing up the number of correct predictions (true positives and true negatives) and dividing it by the total number of predictions. The formula is shown below:

Accuracy = (True Positive + True Negative) / (True Positive + True Negative + False Positive + False Negative).

Targets

As mentioned earlier, we will be using two targets for the regression models - Collisions Count and Collision Rate.

For the classification models, we will be using Collision Class as the target, which is a derived feature from binning the Collisions Count into 5 bins.

Model Descriptions

The models used throughout the modeling process include Decision Trees, Random Forests, Neural Networks, AdaBoost, Linear Regression, Logistic Regression, K-Nearest Neighbors, and XGBoost. A brief description of each of these models is given below:

1. Decision Trees: A decision tree is a flowchart-like model that makes decisions by recursively splitting the input data based on certain features. Each internal node

represents a test on a feature, each branch represents an outcome of that test, and each leaf node represents a class label or a value.

2. Random Forests: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It works by creating a multitude of decision trees and then averaging their predictions to provide a final result. This approach improves the accuracy and reduces overfitting compared to a single decision tree model.

3. Neural Networks: Neural networks are a set of algorithms inspired by the human brain's structure and functioning. They consist of interconnected artificial neurons or nodes organized in layers. Each node receives input, performs a mathematical operation, and produces an output.

4. AdaBoost: AdaBoost (Adaptive Boosting) is a boosting algorithm that combines multiple weak classifiers to create a strong classifier. Each weak classifier is trained on a subset of the data and assigned a weight. The subsequent classifiers focus more on the misclassified samples from previous classifiers, adjusting their weights accordingly. The final prediction is a weighted sum of the weak classifiers outputs.

5. Linear Regression: Linear regression is a supervised learning algorithm used for predicting continuous numerical values. It assumes a linear relationship between the input features and the target variable. The algorithm finds the best-fit line that minimizes the sum of squared differences between the predicted and actual values.

6. Logistic Regression: Logistic regression is a binary classification algorithm that predicts the probability of an instance belonging to a particular class. It uses a logistic function to map the input features to the probability output. Logistic regression is commonly used for problems with two classes or as a component in multiclass classification.

7. K-Nearest Neighbors (KNN): KNN is a non-parametric classification algorithm that predicts the class of an instance based on its nearest neighbors' class labels. It assigns the class label of the majority of the k nearest neighbors to the unseen sample. KNN is effective when the decision boundaries are non-linear.

8. XGBoost: XGBoost (Extreme Gradient Boosting) is an optimized implementation of gradient boosting, a boosting ensemble algorithm. XGBoost builds a strong predictive model by combining the predictions of multiple weak models called decision trees. It uses gradient descent optimization technique to minimize a specific loss function and improve the model's performance.

Regression Models

Collisions Count - Including Volume

For building the first set of regression models, we set the target variable as the Collisions Count, and included the Traffic Volumes features in the analysis.

We specified the predictors and outcomes, excluding the collision_class and collision_rate_traffic columns, as they were to be used as target variables for modeling.

Next, we separated the predictors and outcomes into X and y dataframes, one-hot-encoding the categorical variables in the predictors dataframe.

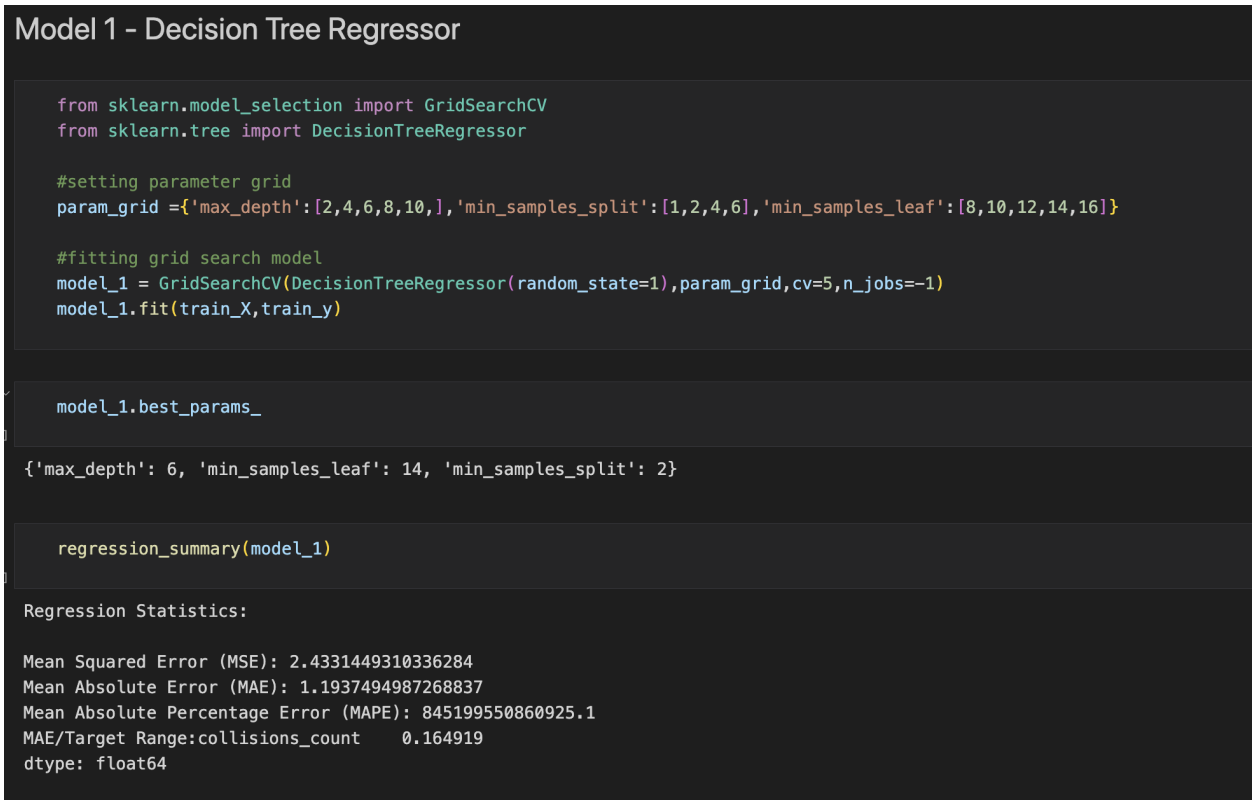
Finally, we split the X and y dataframes into training and validation sets, keeping 70% of the data for training and assigning 30% for validation.

```python
#specifying predictors and outcome
excluded_columns = ['collision_class','collision_rate_traffic']
outcome = ['collisions_count']
predictors = [s for s in final_df.columns if s not in outcome + excluded_columns]

#splitting predictors and outcome into X and y dataframes
X = pd.get_dummies(final_df[predictors],drop_first=True)
y = final_df[outcome]

#splitting data into train and test
from sklearn.model_selection import train_test_split
train_X,valid_X,train_y,valid_y = train_test_split(X,y,test_size=0.3,random_state=1)
```

Python

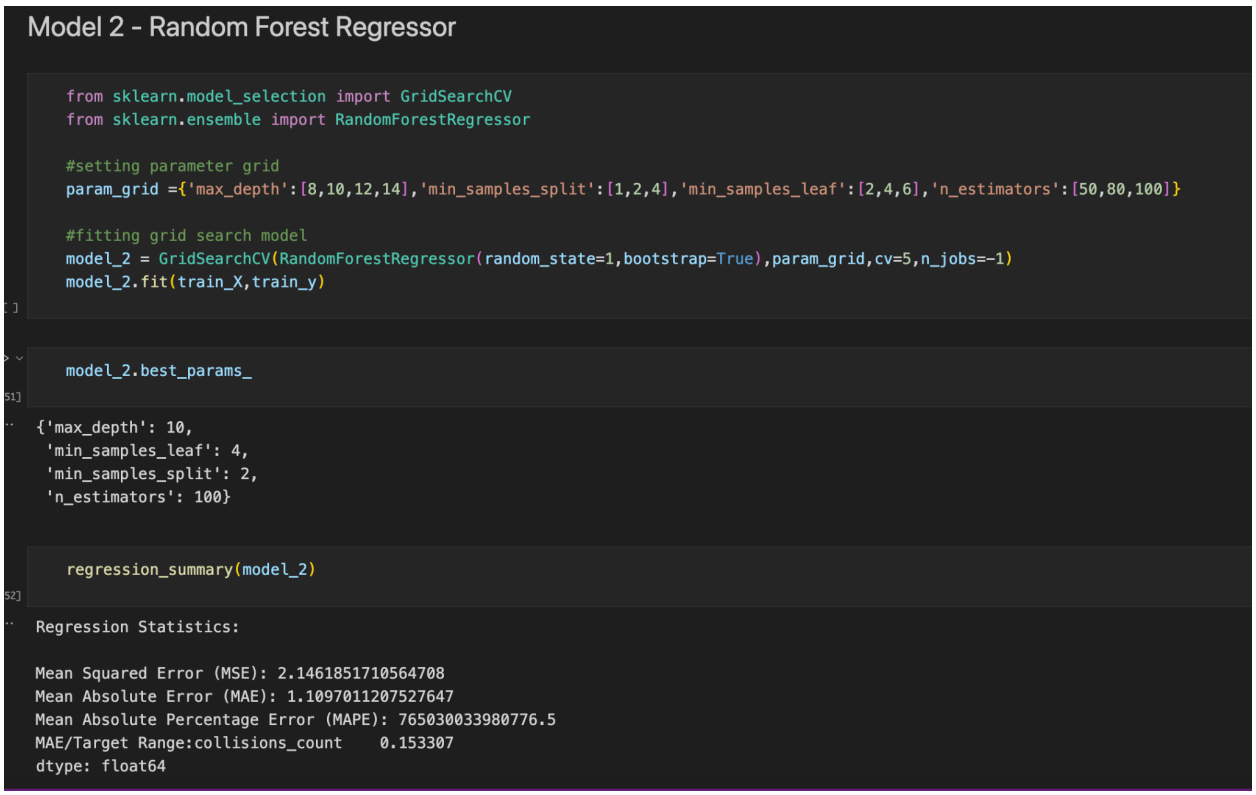
Model 1 - Decision Tree Regressor

The first model to predict Collisions Count is a Decision Tree Regressor. We used a GridSearchCV along with a Decision Tree estimator. The best parameters identified by the GridSearch were a max_depth of 6, min_samplies_leaf of 14, and min_samples_split of 2. The decision tree regressor was one of the least powerful models we used, to use as a base model for comparing the performance of models built afterwards. The performance statistics of the model are shown below along with the resulting plot of the tree. The full decision tree is shown in Appendix 2.



Model 2 - Random Forest Regressor

The second model, a Random Forest, was also fitted to predict the Collisions Count using a GridSearchCV. The code for the mode, along with the best parameters and Regression Summary, are shown below.



Model 2 - Random Forest Regressor

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

#setting parameter grid
param_grid ={'max_depth':[8,10,12,14],'min_samples_split':[1,2,4],'min_samples_leaf':[2,4,6],'n_estimators':[50,80,100]}

#fitting grid search model
model_2 = GridSearchCV(RandomForestRegressor(random_state=1,bootstrap=True),param_grid,cv=5,n_jobs=-1)
model_2.fit(train_X,train_y)
```

```
model_2.best_params_
```

```
{'max_depth': 10,
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 100}
```

```
regression_summary(model_2)
```

```
Regression Statistics:

Mean Squared Error (MSE): 2.1461851710564708
Mean Absolute Error (MAE): 1.1097011207527647
Mean Absolute Percentage Error (MAPE): 765030033980776.5
MAE/Target Range:collisions_count    0.153307
dtype: float64
```

Collisions Count - Excluding Volume

```python
#specifying predictors and outcome
excluded_columns = ['collision_class','collision_rate_traffic','total_count','trucks_count','vehicle_count','cyclists_count',
'pedestrians_count']
outcome = ['collisions_count']
predictors = [s for s in final_df.columns if s not in outcome + excluded_columns]

#splitting predictors and outcome into X and y dataframes
X = pd.get_dummies(final_df[predictors],drop_first=True)
y = final_df[outcome]

#splitting data into train and test
from sklearn.model_selection import train_test_split
train_X,valid_X,train_y,valid_y = train_test_split(X,y,test_size=0.3,random_state=1)
```

Python

Model 3 - Random Forest Regressor

To compare the performance of the models and the feature importances derived from with and without using the volume feature, we ran another Random Forest model, this time excluding the traffic volume columns for all modes. The code for the model is shown below:

Model 3 - Random Forest Regressor

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

#setting parameter grid
param_grid ={'max_depth':[4,6,8,10],'min_samples_split':[1,2,4],'min_samples_leaf':[1,2,4,6],'n_estimators':[100,200,300,400]}

#fitting grid search model
model_3 = GridSearchCV(RandomForestRegressor(random_state=1,bootstrap=True),param_grid,cv=5,n_jobs=-1)
model_3.fit(train_X,train_y)
```

```python
model_3.best_params_
```

```
{'max_depth': 8,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 400}
```

```python
regression_summary(model_3)
```

```
Regression Statistics:

Mean Squared Error (MSE): 2.3332155336522162
Mean Absolute Error (MAE): 1.1710732352940438
Mean Absolute Percentage Error (MAPE): 794572185130750.8
MAE/Target Range:collisions_count    0.161786
dtype: float64
```

However, the performance of the regression models with Collisions Count as the targets (both including and excluding volume) was not satisfactory compared to the models with Collision Rates as the target, having MAPE values much higher than the other models. Additionally, Collision Rate is a more appropriate target to use for the collision prediction model, as we are more interested in the rates of collision of a road segment due to its design features, not the volume of traffic traveling through that segment.

Therefore, we decided not to further investigate the models with Collisions Count targets for their feature importances.

Collision Rate - Including Volume

The second set of models we built included Collision Rate as the target variable. For the first subset, we included the volume metrics in the predictors, this time excluding collision_class and collisions_count columns from the analysis. The code is shown below:

```
#specifying predictors and outcome
excluded_columns = ['collision_class','collisions_count']
outcome = ['collision_rate_traffic']
predictors = [s for s in final_df.columns if s not in outcome + excluded_columns]

#splitting predictors and outcome into X and y dataframes
X = pd.get_dummies(final_df[predictors],drop_first=True)
y = final_df[outcome]

#splitting data into train and test
from sklearn.model_selection import train_test_split
train_X,valid_X,train_y,valid_y = train_test_split(X,y,test_size=0.3,random_state=1)
```

Model 4 - Random Forest Regressor

We developed a random forest regressor model using a GridSearchCV as before. The best parameters identified, as well as summary statistics of the regression are shown below:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

#setting parameter grid
param_grid ={'max_depth':[10,12,14],'min_samples_split':[1,2,4],'min_samples_leaf':[1,2,4,6],'n_estimators':[800]}

#fitting grid search model
model_4 = GridSearchCV(RandomForestRegressor(random_state=1,bootstrap=True),param_grid,cv=5,n_jobs=-1)
model_4.fit(train_X,train_y)
```

```
model_4.best_params_
```

```
{'max_depth': 12,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 800}
```

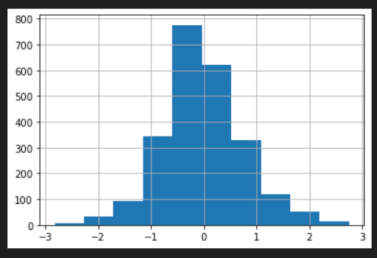
```
regression_summary(model_4)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.5911218837983653
Mean Absolute Error (MAE): 0.5873730036564754
Mean Absolute Percentage Error (MAPE): 301991510536521.4
MAE/Target Range:collision_rate_traffic    0.169789
dtype: float64
```

The model had a MAPE value of 301991510536521.4, which was among the lowest of all the models developed, and a normalized MAE of 0.169789, which was also quite low among all models.

```
#residuals normally distributed
residuals_df_4.residual.hist()
```

```
<AxesSubplot:>
```




```
feature_importance_model_4 = pd.DataFrame({'coef':model_4.best_estimator_.feature_importances_,'predictors':train_X.columns}).
sort_values(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_model_4.head()
```

	coef	predictors
4	0.313689	vehicle_count
1	0.132063	cyclists_count
2	0.113939	pedestrians_count
0	0.113149	number_of_lanes
3	0.086676	trucks_count

The residuals for this model seem almost normally distributed. However, the most important features identified are the traffic volume features.

Add Model 2 - AdaBoostRegressor

The next model was an AdaBoostRegressor, a boosting model to improve the performance of weak learning models.

```
#setting parameter grid
param_grid ={'n_estimators':[10,20,30,50,80]}

#fitting grid search model
add_model_2 = GridSearchCV(AdaBoostRegressor(random_state=1),param_grid,cv=5,n_jobs=-1)
add_model_2.fit(train_X,train_y)
```

```
add_model_2.best_params_
```

```
{'n_estimators': 10}
```

```
regression_summary(add_model_2)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.7723108602398425
Mean Absolute Error (MAE): 0.7289714093369668
Mean Absolute Percentage Error (MAPE): 404741714542612.3
MAE/Target Range:collision_rate_traffic    0.21072
dtype: float64
```

Looking at the summary statistics, this model did not perform well compared to the others, and mostly used traffic volumes counts as the most important features for prediction.

Add Model 3 - Linear Regression

The Linear Regression Model also did not perform very well according to the summary statistics. However, it was interesting to note that it did not use the traffic volume features as the most important, instead using road classifications and number of lanes for predictions.

```
add_model_3 = LinearRegression(n_jobs=-1)
add_model_3.fit(train_X,train_y)
```

```
LinearRegression(n_jobs=-1)
```

```
regression_summary(add_model_3)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.6758081231768986
Mean Absolute Error (MAE): 0.6535975259287684
Mean Absolute Percentage Error (MAPE): 294316230142221.56
MAE/Target Range:collision_rate_traffic    0.188932
dtype: float64
```

```
feature_importance_add_model_3 = pd.DataFrame({'coef':add_model_3.coef_[0],'predictors':train_X.columns}).sort_values(by='coef',
ascending=False).sort_values(by='coef',ascending=False)
feature_importance_add_model_3.head()
```

	coef	predictors
12	0.796222	road_classification_Major Arterial
14	0.613611	road_classification_Minor Arterial
9	0.491620	road_classification_Expressway
0	0.472320	number_of_lanes
20	0.423942	traffic_cameras_present_Yes

Add Model 4 - XGBoost Regressor

The XGBoost Regressor model was the best overall regression model we developed, even though it was the only model built without using GridSearch. We tested GridSearchCV on the model to determine whether it improved the performance, but there was no significant improvement.

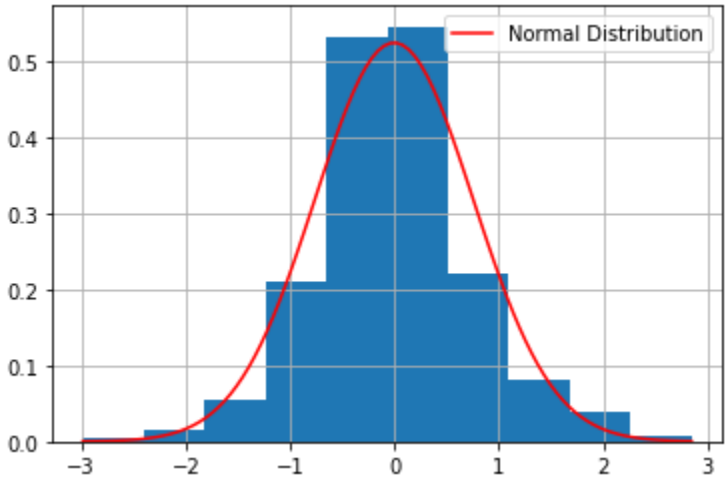
```
#setting parameter grid
param_grid ={'max_depth':[1,2,3,4,6],'learning_rate':[0.1,0.01,0.001,0.2],'n_estimators':[100,150,200,300,400]}

#fitting xgboost model
add_model_4 = xgb.XGBRegressor(random_state=1)
add_model_4.fit(train_X,train_y)
```

```
regression_summary(add_model_4)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.5766868238117391
Mean Absolute Error (MAE): 0.5669784756049971
Mean Absolute Percentage Error (MAPE): 268310550516157.44
MAE/Target Range:collision_rate_traffic    0.163894
dtype: float64
```

The XGBoost model had the lowest MAPE value of 268310550516157.44, while also having one of the lowest normalized MAE values of 0.163894. Additionally, it had the lowest MSE and MAE values of all the models at 0.57 and 0.56.

Moreover, the residuals for the model were normally distributed.



Looking at the feature importances of this model, road_classification was used as the most important features, even in the presence of traffic volume counts.

```
feature_importance_add_model_4 = pd.DataFrame({'coef':add_model_4.feature_importances_,'predictors':train_X.columns}).sort_values
(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_add_model_4.head()
```

	coef	predictors
10	0.115145	road_classification_Expressway Ramp
11	0.102791	road_classification_Local
12	0.099045	road_classification_Major Arterial
20	0.098958	traffic_cameras_present_Yes
8	0.092863	road_classification_Collector Ramp

Collision Rate - Excluding Volume

Because the traffic volume features were identified as the most important in most of our models, we wanted to see which features would impact the model predictions in the absence of traffic volume data.

Therefore, we excluded the traffic volumes columns from our analysis and set the target to collision_rate_traffic.

```
#specifying predictors and outcome
excluded_columns = ['collision_class','total_count','trucks_count','vehicle_count','cyclists_count','pedestrians_count',
'collisions_count']
outcome = ['collision_rate_traffic']
predictors = [s for s in final_df.columns if s not in outcome + excluded_columns]

#splitting predictors and outcome into X and y dataframes
X = pd.get_dummies(final_df[predictors],drop_first=True)
y = final_df[outcome]

#splitting data into train and test
from sklearn.model_selection import train_test_split
train_X,valid_X,train_y,valid_y = train_test_split(X,y,test_size=0.3,random_state=1)
```

Model 5 - Random Forest Regressor

We added a Random Forest Regressor model with a GridSearchCV to optimize hyperparameters. The code for the model, along with best parameters and regression summary are below:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

#setting parameter grid
param_grid ={'max_depth':[10,12,14],'min_samples_split':[1,2,4],'min_samples_leaf':[1,2,4,6],'n_estimators':[600,800,1000]}

#fitting grid search model
model_5 = GridSearchCV(RandomForestRegressor(random_state=1,bootstrap=True),param_grid,cv=5,n_jobs=-1)
model_5.fit(train_X,train_y)
```

```
model_5.best_params_
```

```
{'max_depth': 10,
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 600}
```

```
regression_summary(model_5)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.8870989106457856
Mean Absolute Error (MAE): 0.7734012498966811
Mean Absolute Percentage Error (MAPE): 294060853716347.0
MAE/Target Range:collision_rate_traffic    0.223563
dtype: float64
```

The best parameters found for this model were a max_depth of 10, min_samples_leaf of 4, min_saples_split of 2, and n_estimators of 600.

According to the summary statistics, this model performed well, with a low MAPE value of 294060853716347.0 although with a high normalized MAE at 0.223563.

We wanted to look at the feature importances in the absence of traffic volume counts. The diagram below shows that the model is using the number of lanes, speed limit, and road classifications as the most important features for prediction.

```
#feature importances for model 5
feature_importance_model_5 = pd.DataFrame({'coef':model_5.best_estimator_.feature_importances_,'predictors':train_X.columns}).
sort_values(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_model_5.head()
```

	coef	predictors
0	0.384530	number_of_lanes
1	0.135196	speed_limit
7	0.079134	road_classification_Local
13	0.067782	bikelane_present_Yes
6	0.061004	road_classification_Expressway Ramp

Add Model 5 - Neural Network Regressor

We also tried a Neural Network model with GridSearchCV to see if the performance would improve over the Random Forest model, but found no significant improvement in the model's error statistics.

```
add_model_5 = GridSearchCV(MLPRegressor(random_state=1,max_iter=10000),param_grid,cv=5,n_jobs=-1)
add_model_5.fit(train_X,train_y)
```

```
d:\Software\Anaconda\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:1599: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().


GridSearchCV(cv=5, estimator=MLPRegressor(max_iter=10000, random_state=1),
             n_jobs=-1,
             param_grid={'activation': ['relu', 'logistic', 'tanh'],
                         'hidden_layer_sizes': [8, 10, 12],
                         'solver': ['lbfgs', 'adam']})
```

```
add_model_5.best_params_
```

```
{'activation': 'tanh', 'hidden_layer_sizes': 8, 'solver': 'adam'}
```

```
regression_summary(add_model_5)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.8936948108376269
Mean Absolute Error (MAE): 0.7855813566906488
Mean Absolute Percentage Error (MAPE): 296164651261523.44
MAE/Target Range:collision_rate_traffic    0.227084
dtype: float64
```

Add Model 6 - XGBoost Regressor

As the XGBoost model performed well with predictions with traffic volume data included, we wanted to see how it would perform without the volume data. As shown in the summary statistics, the XGBoost model is still the best of the three models built without including the volume columns, with the lowest MAPE value but similar Normalized MAE values.

Road Classification and Number of Lanes were again identified as important features in the absence of traffic volumes data.

```
#fitting xgboost model
add_model_6 = xgb.XGBRegressor(random_state=1)
add_model_6.fit(train_X,train_y)
```

```
regression_summary(add_model_6)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.9185706874141408
Mean Absolute Error (MAE): 0.7758060573389658
Mean Absolute Percentage Error (MAPE): 285719088973938.9
MAE/Target Range:collision_rate_traffic    0.224258
dtype: float64
```

```
feature_importance_add_model_6 = pd.DataFrame({'coef':add_model_6.feature_importances_,'predictors':train_X.columns}).sort_values
(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_add_model_6.head()
```

	coef	predictors
6	0.124704	road_classification_Expressway Ramp
7	0.116194	road_classification_Local
8	0.113479	road_classification_Major Arterial
0	0.107518	number_of_lanes
9	0.050125	road_classification_Major Arterial Ramp

Classification Models

Similar to the regression models, we ran the classification models both with and without including the traffic volumes features.

Collision Class - Including Volume

We set the target variable as the collision class, excluding the collision_count and collision_rate_traffic columns from the analysis. We then one-hot-encoded the categorical variables and created the training and validation datasets with the same methodology as before.

However, one thing to note was that there was severe class imbalance in the target column - collision class, as most of the values in the collision_class columns were in the first two bins: <5 and 5-20. Machine Learning algorithms generally have difficulty learning from imbalanced datasets, which introduces bias into the models.

To rectify this, we undersampled for the majority class using the RandomUnderSampler() function from the ImbalancedLearn library. Undersampling was used instead of more sophisticated methods like SMOTE, as it usually has better performance compared to other methods. (Wallace et. al, 2011).

The initial class imbalance, as well as the class distribution after performing undersampling, is shown in the figure below.

```
train_y.collision_class.value_counts()
```

```
<5       1838
5-20     1652
20-50     915
>80       812
50-80     373
Name: collision_class, dtype: int64
```

```
train_y_resampled.value_counts()
```

```
collision_class
20-50              373
5-20               373
50-80              373
<5                 373
>80                373
dtype: int64
```

```
#specifying predictors and outcome
excluded_columns = ['collision_rate_traffic','collisions_count']
outcome = ['collision_class']
predictors = [s for s in final_df.columns if s not in outcome + excluded_columns]

#splitting predictors and outcome into X and y dataframes
X = pd.get_dummies(final_df[predictors],drop_first=True)
y = final_df[outcome]

#splitting data into train and test
from sklearn.model_selection import train_test_split
train_X,valid_X,train_y,valid_y = train_test_split(X,y,test_size=0.3,random_state=1)

#undersampling data
from imblearn.under_sampling import RandomUnderSampler
ros = RandomUnderSampler(random_state=1)
train_X_resampled, train_y_resampled = ros.fit_resample(train_X,train_y)
```

Model 6 - Random Forest Classifier

We ran a Random Forest Classifier model as the base classification model. The code for the model, as well as the classification summary statistics and feature importances, are shown below.

```
#setting parameter grid
param_grid ={'max_depth':[6,8,10],'min_samples_split':[8,10,12],'min_samples_leaf':[1,2],'n_estimators':[50,80,100],'bootstrap':
[False]}

#fitting grid search model
model_6 = GridSearchCV(RandomForestClassifier(random_state=1),param_grid,cv=5,n_jobs=-1)
model_6.fit(train_X_resampled,train_y_resampled)
```

```
model_6.best_params_
```

✓ 0.0s

```
{'bootstrap': False,
 'max_depth': 8,
 'min_samples_leaf': 1,
 'min_samples_split': 8,
 'n_estimators': 100}
```

```
classificationSummary(valid_y,model_6.predict(valid_X),class_names=['<5','5-20','20-50','50-80','>80'])
```

✓ 0.0s

```
Confusion Matrix (Accuracy 0.4812)

       Prediction
Actual    <5  5-20 20-50 50-80   >80
    <5   134    72   109    29    55
  5-20   131   231    84   215    32
 20-50    33    12    50    15    40
 50-80    65   162    34   504    47
   >80    22    13    66     7   234
```

The Random Forest Model had an accuracy of 48%, which is still a significant improvement over a random guess - which would have an accuracy of 20% with 5 classification labels.

```
#feature importances for model 6
feature_importance_model_6 = pd.DataFrame({'coef':model_6.best_estimator_.feature_importances_,'predictors':train_X.columns}).
sort_values(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_model_6.head()
```

✓ 0.0s

	coef	predictors
0	0.164776	number_of_lanes
3	0.149434	trucks_count
4	0.144378	vehicle_count
2	0.132026	pedestrians_count
11	0.109759	road_classification_Local

Add Model 7 - Neural Network Classifier

The Neural Network model performed worse than the base Random Forest Model, and is therefore not explored further.

```
#setting parameter grid
param_grid = {'hidden_layer_sizes': [8,10,12],'solver':['lbfgs','adam'],'activation':['relu','logistic','tanh']}

add_model_7 = GridSearchCV(MLPClassifier(random_state=1,max_iter=10000),param_grid,cv=5,n_jobs=-1)
add_model_7.fit(train_X_resampled,train_y_resampled)
```

```
add_model_7.best_params_
```

```
{'activation': 'logistic', 'hidden_layer_sizes': 10, 'solver': 'adam'}
```

+ Code + Markdown

```
classificationSummary(valid_y,add_model_7.predict(valid_X),class_names=['<5','5-20','20-50','50-80','>80'])
```

```
Confusion Matrix (Accuracy 0.4691)

       Prediction
Actual    <5  5-20 20-50 50-80   >80
    <5   121    66    86    50    76
  5-20   138   186    80   242    47
 20-50    33    10    46    19    42
 50-80    75   121    51   535    30
   >80    34    13    48    11   236
```

Add Model 8 - Logistic Regression

The Logistic Regression Model also performed much worse than the Random Forest model. However, the Logistic Model is easier to interpret compared to the Random Forest, and can be used to understand the direction of the impact of the feature variables.

```
classificationSummary(valid_y,add_model_8.predict(valid_X),class_names=['<5','5-20','20-50','50-80','>80'])
```

```
Confusion Matrix (Accuracy 0.4512)

       Prediction
Actual    <5  5-20 20-50 50-80   >80
    <5   102    59    97    52    89
  5-20   128   157    92   254    62
 20-50    32    15    47    15    41
 50-80    77   118    52   530    35
   >80    26    10    48    13   245
```

Add model 9 - XGBoost Classifier

Unexpectedly, the XGBoost Classifier also performed worse than the Random Forest model.

Add Model 9 - XGBoost

```
#need label encoder to transform classes to integer values
from sklearn.preprocessing import LabelEncoder

#fitting xgboost model
add_model_9 = xgb.XGBClassifier(random_state=1)
add_model_9.fit(train_X_resampled,LabelEncoder().fit_transform(train_y_resampled))
```

```
classificationSummary(LabelEncoder().fit_transform(valid_y),add_model_9.predict(valid_X),class_names=['<5','5-20','20-50','50-80',
'>80'])
```

```
Confusion Matrix (Accuracy 0.4733)

       Prediction
Actual    <5  5-20 20-50 50-80   >80
    <5   159    76    86    35    43
  5-20   134   245    75   200    39
 20-50    32     8    76    10    24
 50-80    91   209    45   435    32
   >80    34    16    61    12   219
```

```
feature_importance_add_model_9 = pd.DataFrame({'coef':add_model_9.feature_importances_,'predictors':train_X.columns}).sort_values
(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_add_model_9.head()
```

	coef	predictors
11	0.190203	road_classification_Local
0	0.101655	number_of_lanes
20	0.062931	traffic_cameras_present_Yes
4	0.051138	vehicle_count
7	0.050739	road_classification_Collector

Add Model 10 - KNN Classifier

We also tried a K-Nearest Neighbors classifier to test its performance in classification, but the performance of the model was much lower compared to the others.

```
# Run GridSearchCV to find the best hyperparameters
add_model_10 = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, n_jobs=-1)
add_model_10.fit(train_X_resampled,train_y_resampled)
```

```
add_model_10.best_params_
```

```
{'n_neighbors': 25, 'weights': 'distance'}
```

```
classificationSummary(valid_y,add_model_10.predict(valid_X),class_names=['<5','5-20','20-50','50-80','>80'])
```

```
Confusion Matrix (Accuracy 0.4345)

       Prediction
Actual    <5  5-20 20-50 50-80   >80
    <5   115    96    97    35    56
  5-20   116   251    80   196    50
 20-50    28    11    63     9    39
 50-80    95   237    45   390    45
   >80    27    18    68     7   222
```

Collision Class - Excluding Volume

Similar to the regression models, we also built classification models to see which features impact the predictions in the absence of traffic volume data.

Model 7 - Random Forest Classifier

The Random Forest Classifier was trained using GridSearchCV. It is important to note that this model had almost the same accuracy score as the Model 6, which uses traffic volume data. The accuracy for this model was 0.4775, while model 6 had an accuracy score of 0.4812.

The important features identified by this model were the number of lanes, road classification, speed limit, and whether a traffic camera was present.

```
#fitting grid search model
model_7 = GridSearchCV(RandomForestClassifier(random_state=1),param_grid,cv=5,n_jobs=-1)
model_7.fit(train_X_resampled,train_y_resampled)
```

```
model_7.best_params_
```

```
{'bootstrap': False,
 'max_depth': 6,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 50}
```

```
classificationSummary(valid_y,model_7.predict(valid_X),class_names=['<5','5-20','20-50','50-80','>80'])
```

```
Confusion Matrix (Accuracy 0.4775)

       Prediction
Actual    <5  5-20 20-50 50-80   >80
    <5   121    68    90    60    60
  5-20   140   159    73   286    35
 20-50    37     9    44    23    37
 50-80    46    91    38   583    54
   >80    31    12    54     8   237
```

```
#feature importances for model 7
feature_importance_model_7 = pd.DataFrame({'coef':model_7.best_estimator_.feature_importances_,'predictors':train_X.columns}).
sort_values(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_model_7.head()
```

	coef	predictors
0	0.287386	number_of_lanes
7	0.246173	road_classification_Local
8	0.118162	road_classification_Major Arterial
1	0.112619	speed_limit
16	0.074291	traffic_cameras_present_Yes

Add Model 11 - XGBoost Classifier

The XGBoost Model performed worse than the Random Forest model. Additionally, it used road classifications as the most important features.

Add Model 12 - Logistic Regression

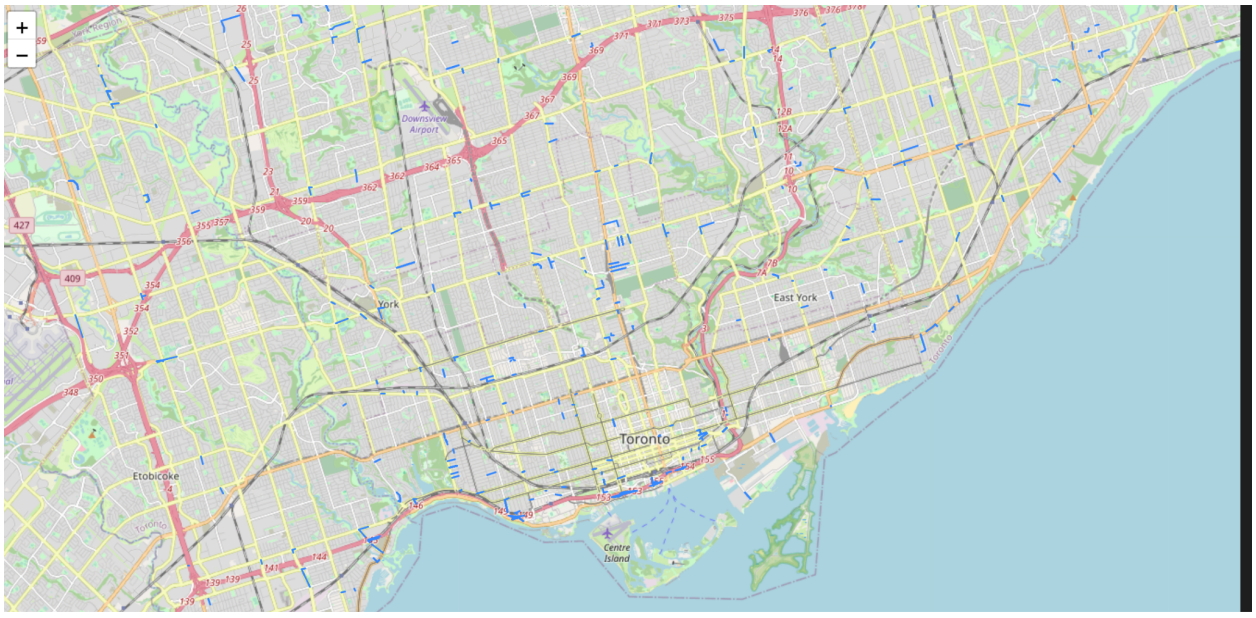
The final classification model, Logistic Regression, also performed much worse than the Random Forest model. However, it can be used to explain the direction of the feature importance of the variables.

Additional Models: Modeling Road Segments with High Collision Rates

As seen during our data processing steps, our target variable - Collision Rates was significantly skewed, which prompted us to cap Collision Rates to a maximum value of 10. However, we wanted to investigate whether there was a difference between the key drivers of collision rates for all data points versus the road segments with extremely high collision rates.

For this, we created a new dataframe of road segments with collision rates above 10, and capped rates at 200 this time, which left us with 376 rows in our dataset. We then log transformed the numerical variables in the new dataset again, and set up the training and validation sets as before.

Looking at the map of road segments with high collision rates, it is a very small subset of the original dataframe, mostly including major arterial and downtown expressways.



```
final_df_high_collision_rates = final_df[final_df['collision_rate_traffic']>10]
```

```
final_df_high_collision_rates['collision_rate_traffic'] = final_df_high_collision_rates.collision_rate_traffic.clip(0,200)
```

```
final_df_high_collision_rates.shape
```

```
(376, 16)
```

```
#applying log transformation on the numerical variables. variables that have a value of 0 are replaced with 1 before applying log
transformation.
def log_transform(x):
    return np.log2(x+1)

final_df_high_collision_rates[['cyclists_count','pedestrians_count','trucks_count','vehicle_count','speed_limit','collisions_count',
'collision_rate_traffic','number_of_lanes']] = final_df_high_collision_rates[['cyclists_count','pedestrians_count','trucks_count',
'vehicle_count','speed_limit','collisions_count','collision_rate_traffic','number_of_lanes']].apply(log_transform)
```

```
#specifying predictors and outcome
excluded_columns = ['collision_class','collisions_count']
outcome = ['collision_rate_traffic']
predictors = [s for s in final_df_high_collision_rates.columns if s not in outcome + excluded_columns]

#splitting predictors and outcome into X and y dataframes
X = pd.get_dummies(final_df_high_collision_rates[predictors],drop_first=True)
y = final_df_high_collision_rates[outcome]

#splitting data into train and test
from sklearn.model_selection import train_test_split
train_X,valid_X,train_y,valid_y = train_test_split(X,y,test_size=0.3,random_state=1)
```

Python

Add Model 13 - Random Forest Regressor

We first trained a Random Forest model with GridSearchCV on the new data frame of high collision rate road segments.

As shown in the summary statistics below, the error statistics of the new model was significantly better than any of the previous models, with much lower MAPE and Normalized MAE values of 0.10141059652484613 and 0.131632 respectively. This indicates that these models were much more accurate compared to the models built earlier.

```
#setting parameter grid
param_grid ={'max_depth':[12,14,16],'min_samples_split':[1,2,4],'min_samples_leaf':[1,2,4],'n_estimators':[100,200,400]}

#fitting grid search model
add_model_13 = GridSearchCV(RandomForestRegressor(random_state=1,bootstrap=True),param_grid,cv=5,n_jobs=-1,verbose=0)
add_model_13.fit(train_X,train_y)
```

```
add_model_13.best_params_
```

```
{'max_depth': 14,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

```
regression_summary(add_model_13)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.3819490295556462
Mean Absolute Error (MAE): 0.4432832332775152
Mean Absolute Percentage Error (MAPE): 0.10141059652484613
MAE/Target Range:collision_rate_traffic    0.131632
dtype: float64
```

```
feature_importance_add_model_13 = pd.DataFrame({'coef':add_model_13.best_estimator_.feature_importances_,'predictors':train_X.
columns}).sort_values(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_add_model_13.head()
```

	coef	predictors
2	0.240490	pedestrians_count
4	0.200806	vehicle_count
3	0.131471	trucks_count
5	0.097364	speed_limit
1	0.092375	cyclists_count

Add Model 14 - XGBoost Regressor

The second model was an XGBoost Regressor, which had even better performance compared to the Random Forest model. This model had MAPE values of 0.09966774169841133 and a Normalized MAE value of 0.131269. Additionally, the XGBoost Regressor had different feature importances compared to the Random Forest model.

```
regression_summary(add_model_14)
```

```
Regression Statistics:

Mean Squared Error (MSE): 0.4559546801807087
Mean Absolute Error (MAE): 0.4420606758602134
Mean Absolute Percentage Error (MAPE): 0.09966774169841133
MAE/Target Range:collision_rate_traffic    0.131269
dtype: float64
```

```
feature_importance_add_model_14 = pd.DataFrame({'coef':add_model_14.feature_importances_,'predictors':train_X.columns}).sort_values
(by='coef',ascending=False).sort_values(by='coef',ascending=False)
feature_importance_add_model_14.head()
```

	coef	predictors
7	0.127857	road_classification_Collector
13	0.108899	road_classification_Major Arterial Ramp
9	0.091324	road_classification_Expressway
2	0.078018	pedestrians_count
4	0.064473	vehicle_count

Model Comparison

In order to build our final collision prediction model based on the best model, we created 20+ different models throughout this project. To determine which of our models has the best performance and should be used to build the collision risk map, we are going to compare the performance of these models based on Mean Absolute Percentage Error (MAPE) and Normalized Mean Absolute Error for the regression models, and accuracy for the classification models. We will be selecting a subset of the models we developed to include in the comparison from each category (different targets, including/excluding volume etc.), as comparing 20+ models in a single table would be challenging and potentially overwhelming for the readers.

Regression Models Table

Model Name	Mean Absolute Percentage Error (MAPE)	Normalized Mean Absolute Error
Model 2- Random Forest	765030033980776.5	0.153307
Model 4 - Random Forest	301991510536521.4	0.169789
Add Model 1 - Neural Network	287371321505858.2	0.174619
Add Model 4 - XGBoost	268310550516157.44	0.163894
Model 5 - Random forest	294060853716347.0	0.223563
Add Model 6 - XGBoost	285719088973938.9	0.224258
Add Model 14 - XGBoost	0.09966774169841133	0.131269

Among the regression models, the Add Model 4 - XGBoost had the lowest combination of Mean Absolute Percentage Error and Normalized Mean Absolute Error values at 268310550516157.44 and 0.163894 respectively.

Two competing models that could also be selected were the Model 2 - Random Forest, which had the lowest overall Normalized Mean Absolute Error, and the Add Model 6 - XGBoost, which had the lowest overall MAPE. However, since the Add Model 4 - XGBoost model had both of these values as very low, it was selected as the best regression model.

Finally, even though the Add Model 14 - XGBoost had significantly lower MAPE and Normalized MAE values compared to the other models, it cannot be selected as the best model as it is built on only a small subset of the data.

Classification Models Table

Model Name	Validation Accuracy
Model 6 - Random Forest	0.4812
Add Model 7 - Neural Network	0.4691
Add Model 9 - XGBoost	0.4733
Model 7 - Random Forest	0.4775

From the classification models above, the Model 6- Random Forest model was selected as the best mode, as it had the highest classification accuracy of 0.4812 based on validation data. Additionally, this model was preferred to the next best model Model 7 - Random Forest, as it contained traffic volume information.

Model Selection

Upon evaluating the performance of all the models and selecting the best regression and classification models, we must select a final model for creating the collision risk map. The competing models for this are the Add Model 4 - XGBoost and Model 6 - Random Forest.

From the competing models, we selected Add Model 4 - XGBoost as the final model to predict traffic collision risk due to few key reasons:

1. A regression model would be more preferable to a classification model to map collision risk, as it would provide more granular information and variance about the risk involved in each road segment, compared to a classification model which can categorize road segments by estimated risk.

2. Secondly, the classification models were trained using a target of bins of collision counts, and thus does not account for the volume of daily traffic that flows through each road segment.
3. XGBoost models perform extremely well even without using GridSearch for hyperparameter optimization. This leads to XGBoost models being much faster to train compared to other models.

Thus, the Add Model 4 XGBoost was selected as the best model to use for estimating collision risk of road segments.

Recommendations and Key Findings

Feature Interpretation

After selecting the Add Model 4- XGBoost as our best model, we wanted to understand which features were identified to be important by the model in predicting collision rates.

Predictors	Coef	Odds Ratio
road_classification_Expressway Ramp	0.115145	1.122036121
road_classification_Local	0.102791	1.108259759
road_classification_Major Arterial	0.099045	1.104115984
traffic_cameras_present_Yes	0.098958	1.10401993
road_classification_Collector Ramp	0.092863	1.097311393
vehicle_count	0.081711	1.085142158
number_of_lanes	0.051327	1.052667059
road_classification_Minor Arterial	0.045857	1.04692469
sensitive_zone_Yes	0.035323	1.035954268
pedestrians_count	0.032345	1.032873785

Looking at the feature importances for the XGBoost model, it seems that four of the top five features were road classifications. The top 5 features identified as particularly important in predicting collision rates were:

- road_classification_Expressway Ramp
- road_classification_Local
- road_classification_Major Arterial
- traffic_cameras_present_Yes
- road_classification_Collector Ramp

The remaining important features that were identified were vehicle_count, number_of_lanes, road_classification_Minor Arterial, sensitive_zone_Yes, and pedestrians_count. However, as the XGBoost model is an ensemble model, it does not tell us anything about the direction of these features, and how they impact the collision rate of a road.

To understand the direction of the features identified by our best model, we looked at the coefficients of our linear regression model - Add Model 3 Linear Regression. The table below outlines selected features identified by our best model, their odds ratio, and their direction of impact on the predicted collision rate - positive or negative.

Predictor	Odds Ratio	Direction
road_classification_Major Arterial	1.10	Positive
road_classification_Minor Arterial	1.04	Positive
vehicle_count	1.08	Positive
number_of_lanes	1.05	Positive
traffic_cameras_present_Yes	1.10	Positive
road_classification_Local	1.11	Negative
road_classification_Expressway Ramp	1.12	Negative

Interpreting the Odds Ratios of the identified features along with their direction of impact:

- Collision Rates are 10% higher on roads classified as Major Arterial, and 4% higher on roads classified as Minor Arterial compared to all other road classes.
- For each additional 1000 vehicles on a road segment daily, the collision risk increases 8%.
- For each additional lane on a road, the collision risk increases 5%.
- Roads which have a traffic camera present have 10% higher risk of collisions compared to road segments without traffic cameras.
- Roads classified as Local and Expressway Ramp have 11% and 12% lower rates of collision respectively.

Recommendations

Based on our analysis and the interpretation of the features of prediction models, we recommend that the Toronto Traffic Services focus on road segments with the following features to make design and safety improvements:

- Roads that are classified as Major or Minor Arterials, such as Lawrence Avenue or St. Clair Avenue.
- Roads that have a high daily volume of vehicles flowing through every day.
- Road segments that are wide and have a high number of lanes.

Road segments with these features are recommended for design improvements as the presence of these features on road segments lead to an increased risk of collisions, even accounting for the volume of traffic. As such, safety improvements to such road segments will lead to significant reductions in the number of collisions in Toronto.

Additionally, it is recommended that the Toronto Traffic Services use the collision prediction model to identify road segments that have high predicted collision rates to focus on for design and safety improvements.

Future Work

In order to enhance the accuracy and usefulness of the traffic collision prediction model, there are several areas of future work that could be explored:

1. **Including additional road features**: In addition to the features already considered in the model, such as road classification and daily traffic volume, it would be beneficial to incorporate additional road features. Average traffic speeds data could provide valuable insights into the relationship between speed and collision risk. Similarly, considering the presence and timing of traffic signals could help identify intersections or segments with higher collision probabilities. Moreover, incorporating data from 311 requests, such as road maintenance or repair requests, could highlight areas where road conditions may contribute to collisions.
2. **Developing separate collision models**: While the current model provides a holistic view of collision risk across all road segments, future work could involve developing separate prediction models for specific scenarios. For example, creating separate models for roads with high collision counts or high pedestrian counts could provide targeted insights for implementing safety measures in those specific areas. Additionally, creating separate models for downtown areas versus suburbs could account for the differences in traffic patterns and collision risks between these regions.
3. **Mapping collision risk for all road segments**: Expanding the scope of the analysis to include all road segments in Toronto would be a valuable future endeavor. By mapping the collision risk for each road segment, traffic authorities could have a comprehensive overview of areas with higher probabilities of collisions and prioritize resources accordingly. This would enable a proactive approach to safety improvements and targeted interventions on a city-wide scale.

By continuing to address these areas in future work, the traffic collision prediction model can be further improved to identify more features that can lead to an increased risk of traffic collisions, as well as more accurately identify road segments with high collision risk.

Conclusion

In conclusion, the development of a traffic collision prediction machine learning model for the Toronto Traffic Services provides valuable insights into road segment characteristics that contribute to an increased risk of collisions. By analyzing important features such as road classification, daily traffic volume, and lane count, the model identifies road segments that are in need of design and safety improvements.

The recommendations derived from the analysis suggest that the focus should be on road segments classified as Major or Minor Arterials, those with a high daily volume of vehicles, and wide segments with a high number of lanes. Implementing safety measures on these road segments would lead to significant reductions in collisions throughout Toronto. Additionally, utilizing the collision prediction model to identify road segments with high predicted collision rates can further prioritize improvements.

For future work, it is recommended to include additional road features such as average traffic speeds, traffic signals, and data from 311 requests in the model. This will enhance the accuracy of the predictions and offer a more comprehensive understanding of collision risks. Developing separate collision models for specific scenarios, such as roads with high collision counts or high pedestrian counts, as well as distinguishing between downtown and suburban areas, would provide targeted insights for focused interventions. Moreover, mapping collision risk for all road segments in Toronto would allow for a city-wide perspective and enable proactive safety improvements.

By leveraging these recommendations and exploring future work, the Toronto Traffic Services can implement evidence-based design and safety improvements, ultimately leading to a substantial reduction in traffic collisions and ensuring safer road conditions for all residents of Toronto.

References

1. World Health Organization. "Road traffic injuries." Retrieved from: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries#:~:text=Road%20traffic%20injuries%20are%20the,result%20of%20road%20traffic%20crashes.

2. K. Kabasakalis, A. Hadjipanayis, and A. Constantinou. "Predicting Traffic Accident Hotspots with Spatial Data Science." Retrieved from: https://arxiv.org/pdf/1905.08770.pdf

3. Statistics Canada. "Road Safety and Motor Vehicle Traffic Collision Statistics." Retrieved from: https://www150.statcan.gc.ca/n1/daily-quotidien/201126/t001b-eng.htm

4. City of Toronto. "Vision Zero: Vision Zero Plan Overview." Retrieved from: https://www.toronto.ca/services-payments/streets-parking-transportation/road-safety/vision-zero/vision-zero-plan-overview/

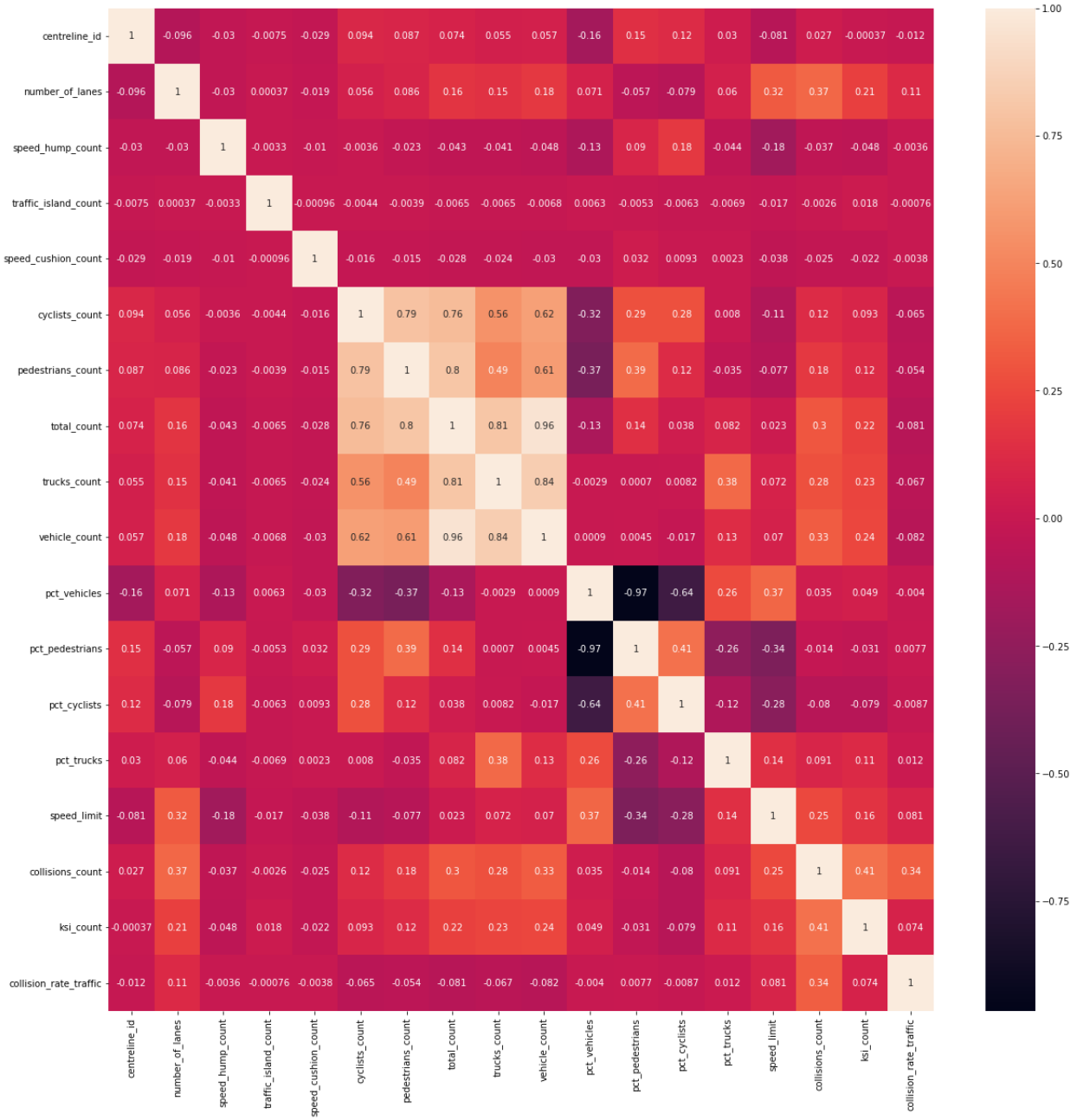
5. N. Darban. "Using Machine Learning to Predict Car Accident Risk." Retrieved from: https://towardsdatascience.com/predicting-traffic-accident-hotspots-with-spatial-data-science-cfe5956b2fd6

6. GeoAI. "Using Machine Learning to Predict Car Accident Risk." Retrieved from: https://medium.com/geoai/using-machine-learning-to-predict-car-accident-risk-4d92c91a7d57

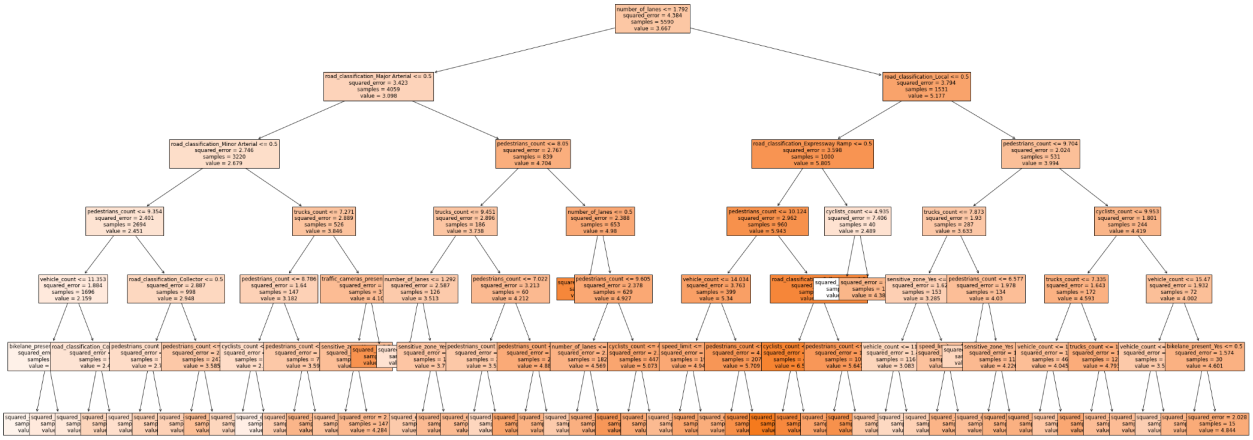
7. IEEE. "Using Machine Learning to Predict Car Accident Risk." In 2011 IEEE 14th International Conference on Intelligent Transportation Systems, 16-19 October 2011, pp. 1932-1937. doi: 10.1109/ITSC.2011.6083029. Available at: https://ieeexplore.ieee.org/document/6137280

Appendix

Appendix 1: Feature Correlation Matrix



Appendix 2: Model 1 - Decision Tree Regressor



3.0 Population, Variable Selection, Considerations

Audience/population selection:

Observation window: Although the data set contains data since 2008, we will only be taking observations from the past 7 years, so 2016 onwards.

Inclusions: All variables in the dataset will be included.

Exclusions: N/A

Data Sources: Toronto Police All Collisions, Toronto Police KSI, Toronto weather data, Traffic and Pedestrian Volumes at intersections, Average speed of roads

Audience Level: Toronto Police Traffic Services

Variable Selection:

Derived Variables:

Assumptions and data limitations:

5.0 Deliverables Timeline

Item	Major Events / Milestones	Description	Scope	Days	Date
1.	Kick-off / Formal Request				July 11, 2023
2.	Analysis Plan				July 24, 2023
3.	Gather and Merge Datasets	Merge multiple datasets into one.			July 25, 2023
4.	Data Exploration & Analysis · Issues with duplicates · Issues with Spend data	Data Exploration + Descriptive Analysis/Data Visualization to understand data structure.			July 28, 2023
5.	Data Pre-Processing and Feature Engineering	Pre-processing data for modeling + Feature engineering new model features.			July 31, 2023
6.	Modeling	Modeling and Model Performance Review + Hyperparameter Tuning.			August 7, 2023
7.	Model Deployment	Deployment of model predictions as an interactive map.			August 10, 2023
8.	Governance and Documentation	Document model governance and steps.			August 14, 2023
9.	Stakeholder Presentation	Develop stakeholder presentation about model findings.			August 16, 2023
10.	Delivery & sign-off	Final delivery of project.			August 25, 2023