AI Final Project Report

Alejandro Akifarry, Jungyu Lee, Minyoung Seol

COMP247: Supervised Learning

Professor Blessing Ajiboye

August 15, 2023

TABLE OF CONTENTS

Executive Summary

Problem

Nowadays, road safety is increasingly more important due to the increase in population and the need to travel in order to survive. And in order for the police department to provide better security measures and the public to be more safe when traveling, it can be helpful to know the likelihood of a fatal accident when some factors are present. These factors include the age of the people involved, the condition of the road, the visibility of the road, etc.

Solution

In order to increase the general road safety, it can be helpful to train a model that can take the factors mentioned above and predict whether it will result in a fatal accident or not. Using this model, both the police department as well as the public can benefit from being safe in general. And in order to make it easy to access and use, the predictive model will be able to be used on a website.

Key Findings

After doing some data exploration and visualization on the dataset, we first discovered that aggressive/distracted driving conditions are the most common among all accidents while alcohol is surprisingly the least common. We also discovered that the most common type of vehicle involved in these accidents over the decade is automobiles. Lastly, we discovered that interestingly, the Summer of 2013 had a relatively higher number of fatal accidents.

Overview of Solution

In order to implement the solution mentioned in the Executive Summary part of this report, we first had to understand the Killed or Seriously Injured (KSI) dataset from the Toronto Police website. After understanding what each field in the dataset means, we then explored the data as well as performing some data visualizations that led us to the discovery of some interesting key findings. After that, we transformed some columns by filling in missing values and even changing the values themselves as well as dropping some columns to make it better for training the model later. We then split the data into train and test before putting them into pipelines which we created to streamline the whole pre-processing process.

Later on, we used the prepared train data to train several models before making them predict the test data. After we have the models predict the test data, we were able to decide on the best model for our features by examining metrics like the accuracy score. Finally, we pickled the model itself and turned it into an API using the Flask framework, which can then be used in a website to predict whether an accident is going to be fatal or non-fatal.

Data Exploration and Findings

The first thing we did before we start the actual exploration of the dataset is checking out the fields within the dataset as well as the description of each, which can be obtained from the documentation on the official website:

Field	Field Name	Description
1	X	Geographic Coordinate
2	Y	Geographic Coordinate
3	INDEX_	Unique Identifier
4	ACCNUM	Accident Number
5	YEAR	Year Collision Occurred
6	DATE	Date Collision Occurred (time is displayed in UTC downloaded as a CSV)
7	TIME	Time Collision Occurred
8	STREET1	Street Collision Occurred
9	STREET2	Street Collision Occurred
10	OFFSET	Distance and direction of the Collision
11	ROAD_CLASS	Road Classification
12	DISTRICT	City District
13	WARDNUM	City of Toronto Ward collision occurred
14	LATITUDE	Latitude
15	LONGITUDE	Longitude
16	LOCCOORD	Location Coordinate
17	ACCLOC	Collision Location
18	TRAFFCTL	Traffic Control Type
19	VISIBILITY	Environment Condition

Field	Field Name	Description
20	LIGHT	Light Condition
21	RDSFCOND	Road Surface Condition
22	ACCLASS	Classification of Accident
23	IMPACTYPE	Initial Impact Type
24	INVTYPE	Involvement Type
25	INVAGE	Age of Involved Party
26	INJURY	Severity of Injury
27	FATAL_NO	Sequential Number
28	INITDIR	Initial Direction of Travel
29	VEHTYPE	Type of Vehicle
30	MANOEUVER	Vehicle Manoeuver
31	DRIVACT	Apparent Driver Action
32	DRIVCOND	Driver Condition
33	PEDTYPE	Pedestrian Crash Type - detail
34	PEDACT	Pedestrian Action
35	PEDCOND	Condition of Pedestrian
36	CYCLISTYPE	Cyclist Crash Type - detail
37	CYCACT	Cyclist Action
38	CYCCOND	Cyclist Condition
39	PEDESTRIAN	Pedestrian Involved In Collision
40	CYCLIST	Cyclists Involved in Collision
41	AUTOMOBILE	Driver Involved in Collision
42	MOTORCYCLE	Motorcyclist Involved in Collision
43	TRUCK	Truck Driver Involved in Collision
44	TRSN_CITY_VEH	Transit or City Vehicle Involved in Collision

Field	Field Name	Description
45	EMERG_VEH	Emergency Vehicle Involved in Collision
46	PASSENGER	Passenger Involved in Collision
47	SPEEDING	Speeding Related Collision
48	AG_DRIV	Aggressive and Distracted Driving Collision
49	REDLIGHT	Red Light Related Collision
50	ALCOHOL	Alcohol Related Collision
51	DISABILITY	Medical or Physical Disability Related Collision
52	HOOD_158	Unique ID for City of Toronto Neighbourhood (new)
53	NEIGHBOURHOOD_158	City of Toronto Neighbourhood name (new)
54	HOOD_140	Unique ID for City of Toronto Neighbourhood (old)
55	NEIGHBOURHOOD_140	City of Toronto Neighbourhood name (old)
56	DIVISION	Toronto Police Service Division
57	ObjectId	Unique Identifier (auto generated)

Reference:

https://torontops.maps.arcgis.com/sharing/rest/content/items/c0b17f1888544078bf650f3b8b04d35d/data

We explored brief information of each incident by printing the first 5 rows using df_ksi.head(). Columns X and Y represent the geographic coordinates of the incident's location, and the remaining columns represent the index that identifies each incident(row).

```
In [4]: df_ksi.head()
Out[4]:
              X             Y  ...  DIVISION  ObjectId
0 -8.829728e+06  5.419071e+06  ...       D55         1
1 -8.829728e+06  5.419071e+06  ...       D55         2
2 -8.854874e+06  5.414091e+06  ...       D22         3
3 -8.829728e+06  5.419071e+06  ...       D55         4
4 -8.829728e+06  5.419071e+06  ...       D55         5

[5 rows x 57 columns]
```

After that, we also briefly explored each column's overall information such as the number of missing values or the type.

```
In [6]: df_ksi.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17488 entries, 0 to 17487
Data columns (total 57 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   X                 17488 non-null  float64
 1   Y                 17488 non-null  float64
 2   INDEX_            17488 non-null  int64
 3   ACCNUM            17488 non-null  int64
 4   YEAR              17488 non-null  int64
 5   DATE              17488 non-null  object
 6   TIME              17488 non-null  int64
 7   STREET1           17488 non-null  object
 8   STREET2           15896 non-null  object
 9   OFFSET            3028 non-null   object
 10  ROAD_CLASS        17112 non-null  object
 11  DISTRICT          17453 non-null  object
 12  WARDNUM           16963 non-null  float64
 13  LATITUDE          17488 non-null  float64
 14  LONGITUDE         17488 non-null  float64
 15  LOCCOORD          17393 non-null  object
 16  ACCLOC            12038 non-null  object
 17  TRAFFCTL          17454 non-null  object
 18  VISIBILITY        17470 non-null  object
 19  LIGHT             17488 non-null  object
 20  RDSFCOND          17465 non-null  object
```

Statistical assessments including means, averages, correlations. The correlation between 'YEAR' and 'INDEX_' columns is approximately 0.874. The mean of the 'X' column is approximately -8,838,301, and the standard deviation of the 'Y' column is approximately 8,655,506. It can be seen in the screenshots below.

```
In [8]: df_ksi.corr()
Out[8]:
                X         Y    INDEX_  ...  LONGITUDE  FATAL_NO  ObjectId
X        1.000000  0.418305  0.010089  ...   1.000000  0.074813  0.020431
Y        0.418305  1.000000  0.022885  ...   0.418305  0.019322  0.006643
INDEX_   0.010089  0.022885  1.000000  ...   0.010089  0.204953  0.862236
ACCNUM   0.012913  0.040005  0.828143  ...   0.012913  0.237229  0.687832
YEAR     0.017399  0.004153  0.874233  ...   0.017399  0.136423  0.991528
TIME     0.015348  0.032838  0.044155  ...   0.015348  0.059189  0.045215
WARDNUM  0.009450  0.010731  0.013679  ...   0.009450 -0.034983  0.008572
LATITUDE 0.418251  1.000000  0.022883  ...   0.418251  0.019338  0.006637
LONGITUDE 1.000000 0.418305  0.010089  ...   1.000000  0.074813  0.020431
FATAL_NO 0.074813  0.019322  0.204953  ...   0.074813  1.000000  0.207997
ObjectId 0.020431  0.006643  0.862236  ...   0.020431  0.207997  1.000000

[11 rows x 11 columns]
```

```
In [7]: df_ksi.describe()
Out[7]:
                X             Y  ...    FATAL_NO      ObjectId
count  1.748800e+04  1.748800e+04  ...  773.000000  17488.000000
mean  -8.838301e+06  5.420763e+06  ...   29.399741   8744.500000
std    1.160273e+04  8.655506e+03  ...   17.974056   5048.495089
min   -8.865305e+06  5.402162e+06  ...    1.000000      1.000000
25%   -8.846498e+06  5.413284e+06  ...   14.000000   4372.750000
50%   -8.838366e+06  5.419556e+06  ...   28.000000   8744.500000
75%   -8.829649e+06  5.427830e+06  ...   43.000000  13116.250000
max   -8.807929e+06  5.443099e+06  ...   78.000000  17488.000000

[8 rows x 11 columns]
```

We then tried to understand the percentage of missing values for each column. Due to the presence of numerous null values, it is hard to produce meaningful graphs and visualizations. For this reason, we will present them in the Feature Selection and Data Modeling part, after handling some of the missing data. Below is the percentage of missing values in each column.

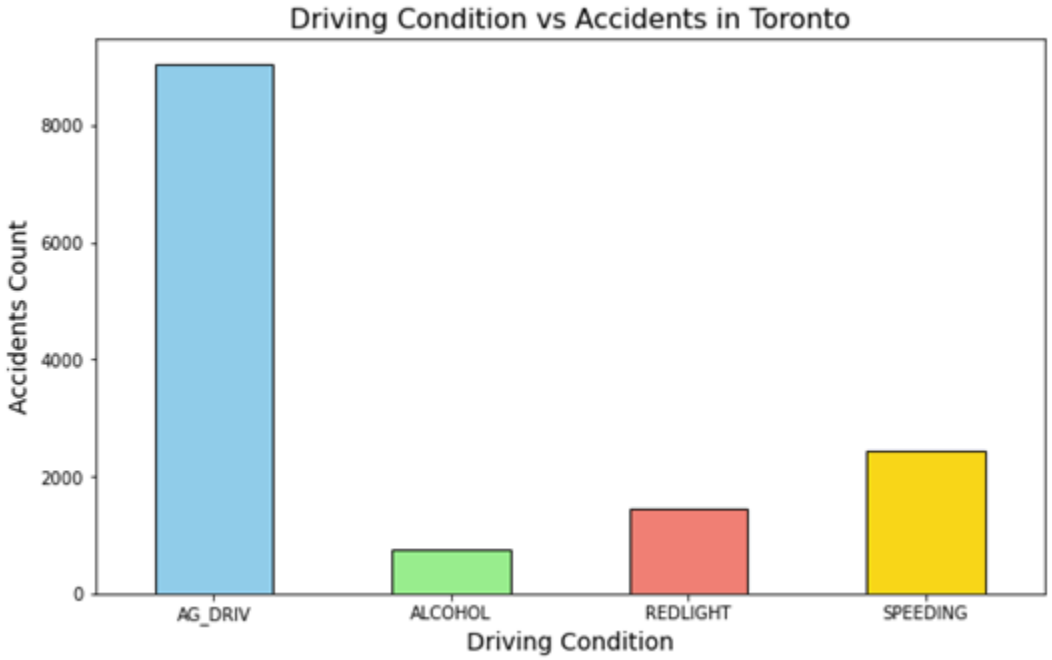
```
In [10]: print(df_ksi.isna().sum() / len(df_ksi) * 100)
X                 0.000000
Y                 0.000000
INDEX_            0.000000
ACCNUM            0.000000
YEAR              0.000000
DATE              0.000000
TIME              0.000000
STREET1           0.000000
STREET2           9.103385
OFFSET           82.685270
ROAD_CLASS        2.150046
DISTRICT          0.200137
WARDNUM           3.002059
LATITUDE          0.000000
LONGITUDE         0.000000
LOCCOORD          0.543230
ACCLOC           31.164227
TRAFFCTL          0.194419
VISIBILITY        0.102928
LIGHT             0.000000
RDSECOND          0.131519
```

Before we start with the graphs, here are the tools and libraries that were used to produce our data visualizations:

- Matplotlib
- Seaborn
- Folium

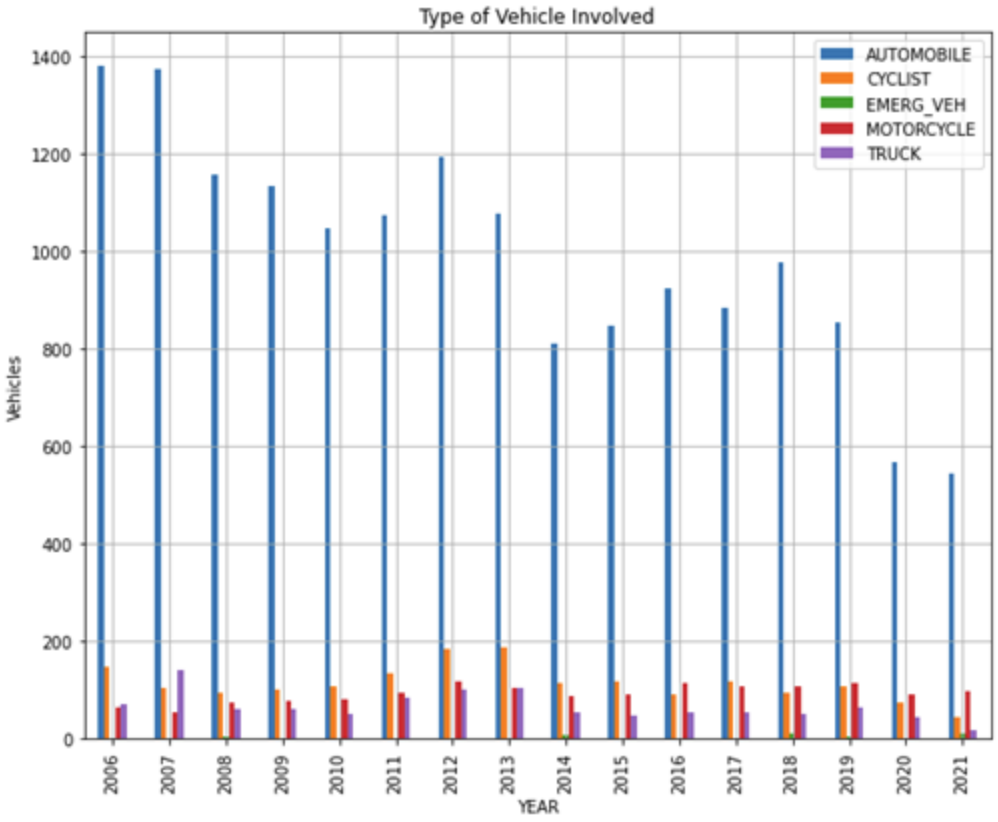
Only Matplotlib is used in this part of the report, and the other 2 libraries will be used in the next parts of the report.

Firstly, we used the Matplotlib library to plot the graph that compares the occurrence of accidents in each driving condition which provides a visual analysis of which conditions cause accidents most frequently. This also helps us determine the percentage of accidents in each condition relative to the total number of accidents.

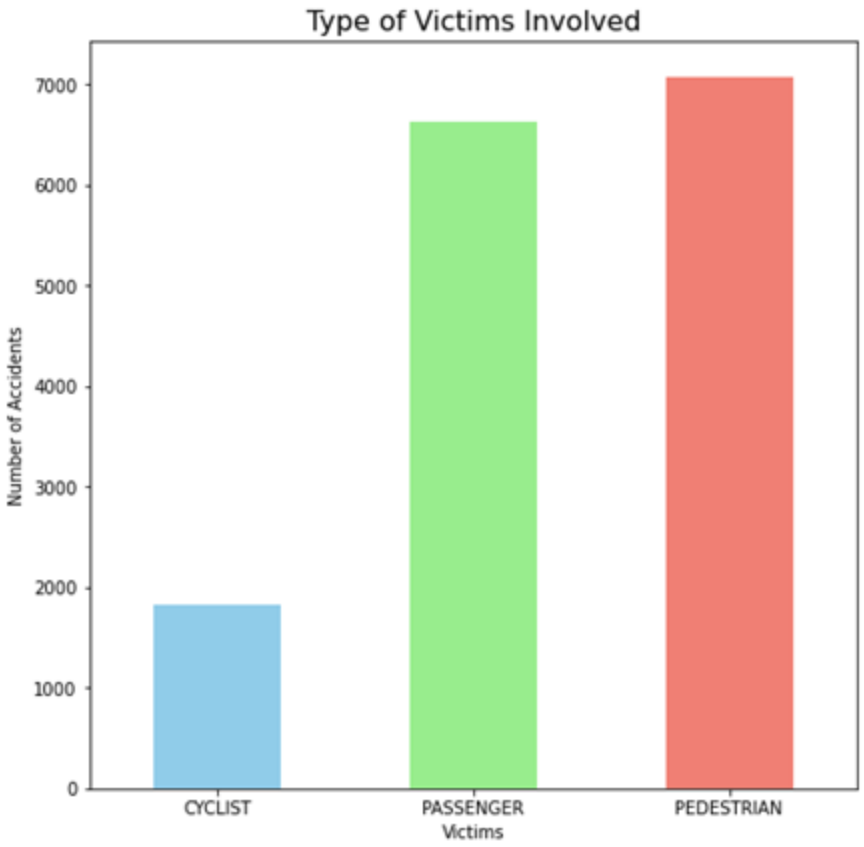


It can be seen from the graph that aggressive driving condition (AG_DRIV) had the biggest percentage among all driving conditions.

After that, we used Matplotlib again to visualize the types of vehicles involved, and we found that over decades, the majority of accidents were caused by automobiles.



Finally, we visualized the type of victims involved using a bar graph from Matplotlib.



From the graph above, it appears that most of the victims are pedestrians and passengers.

Feature Selection

For feature selection, we first dropped the columns for unique identifiers. The columns that we think fit this description are: 'INDEX_', 'ObjectId', 'ACCNUM', 'FATAL_NO'.

We then noticed that for these columns: 'PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY', there are only “Yes” and “nan” values, so we assumed that “nan” is “No”, and we filled the missing values with “No”s.

After that, there are some location columns that are too specific, and we think that they are not worth keeping for the predictions. And those are: ‘X’, ‘Y’, 'ACCLOC', 'WARDNUM',

'HOOD_158', 'NEIGHBOURHOOD_158', 'HOOD_140', 'NEIGHBOURHOOD_140', 'STREET1', 'STREET2', 'ROAD_CLASS', 'LOCCOORD'. Subsequently, we drop these columns due to the amount of missing values as well as too many specific values, which we deemed not worth keeping for predictions: 'INJURY', 'INITDIR', 'VEHTYPE', 'MANOEUVER', 'DRIVACT', 'DRIVCOND', 'DIVISION', 'TRAFFCTL', 'IMPACTYPE', 'INVTYPE'.

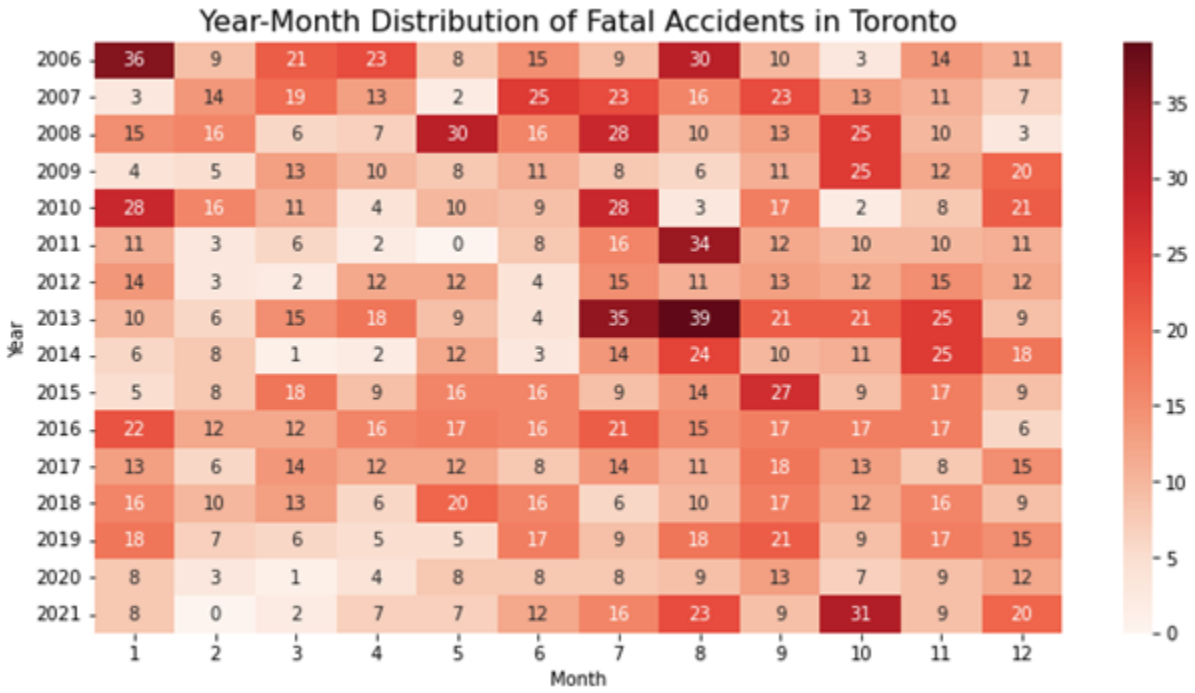
Afterwards, we dropped these columns (after doing some data visualization) because we think that there are better columns for prediction like LIGHT, VISIBILITY, and RDSFCOND: 'YEAR', 'MONTH', 'DAY', 'HOUR', 'MINUTES', 'WEEKDAY', 'LATITUDE', 'LONGITUDE'. Now that we dropped all those columns, we're going to do some data modeling in the next part, which also includes addition and removal of some columns. Therefore, here are the final set of columns that will be used: 'DISTRICT', 'VISIBILITY', 'LIGHT', 'RDSFCOND', 'INVAGE', 'PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY', 'FATAL'. Note that the FATAL column is the target column and the other ones are feature columns.

Data Modeling

For this part, we first transformed the columns DATE and TIME into MONTH, DAY, HOUR, MINUTES, WEEKDAY columns for data visualization purposes as well as data modeling. We then dropped the columns DATE and TIME as they were no longer needed. However, after further discussion, we decided to drop those new columns also since we thought there are better alternatives. This is why in the previous part we mentioned that we dropped those new columns that are technically not in the original dataset.

Afterwards, for these columns: 'PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY', like mentioned previously, we noticed that they only have “Yes” and “nan” values, so, we assumed that “nan” is “No”, and we transformed the “Yes” values to 1’s and the “nan” values to 0’s to prepare them for training and testing.

Consequently, using the ACCLASS column, we built our target column named FATAL. About how we did it, we noticed that the ACCLASS column has values “Non-Fatal Injury”, “Property Damage Only”, and “Fatal”. So, for each row in ACCLASS, if the values are either “Non-Fatal Injury” or “Property Damage Only”, we put a 0 in the FATAL column, and only when it is “Fatal” that we put 1 in the FATAL column. In addition to that, if there are any missing values, we put a 0. After creating this FATAL column, we also plotted these 2 graphs using Seaborn and Folium respectively before removing the time related columns:



The above graph is a plot that shows the Year-Month Distribution of Fatal Accidents. And it appears that the Summer of 2013 had a relatively high number of fatal accidents.

The above graph is a heatmap that shows the distribution of fatal accidents around GTA.

Later on, in the DISTRICT column, we notice that among the values, there are 2 very similar values, which are “Toronto East York” and “Toronto and East York”. Because of the similarity, we decided to just replace all “Toronto East York” with “Toronto and East York”. After this, we dropped the columns that has more than 80% missing values, which are: 'OFFSET', 'PEDTYPE', 'PEDACT', 'PEDCOND', 'CYCLISTYPE', 'CYCACT', 'CYCCOND'.

Next thing we did is for the INVAGE column. We noticed that the age ranges are too small and narrow like “0 to 4”, “5 to 9”, and so on until “90 to 94” and finally “over 95”. So we decided to broaden them up a bit by making the ranges into these 4:

- 0 to 29
- 30 to 59

- 60 and over
- Unknown (for missing values) 18194

Then, we notice that in the FATAL column that we created, the dataset is unbalanced. The 0 values have an occurrence of 15621 while the 1 values have 2573 occurrences, which are around 86% and 14% of the total rows respectively. And to balance them, we did oversampling, which results in both values having 15621 occurrences after the procedure. Note that as mentioned before, here are the final set of columns: 'DISTRICT', 'VISIBILITY', 'LIGHT', 'RDSFCOND', 'INVAGE', 'PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY', 'FATAL'. Again, note that the FATAL column is the target column and the other ones are feature columns.

After all those transformations, we defined and utilized 2 pipelines, 1 for numeric features and 1 for categorical features. The pipeline for numeric features consists of a SimpleImputer that fills in missing values with the median of the column. On the other hand, the pipeline for categorical features consists of a SimpleImputer that fills in missing values with the most frequent value of the column as well as a OneHotEncoder that ignores unknown values. Then, we split the train and test data, we split 80% train data and 20% train data, and to do that, we used StratifiedShuffleSplit that does 10 re-shuffles. And as we've mentioned before, we did oversampling for the target column FATAL as it's originally unbalanced (around 86%-14%).

Model Building

As for the models, we experimented with these classifiers: Logistic Regression Classifier, Decision Tree, Support Vector Machine, Random Forest, Multi-Layer Perceptron Neural Network. Then, we used GridSearchCV and RandomizedSearchCV with 5-fold cross-validation

to find the best parameters for each one. Here are the screenshots of hyperparameters and the best parameters for each model:

- Logistic Regression

```
# Setting the hyperparameter grid for tuning
param_grid_lrclf = {
    'lrclf__solver': ['lbfgs', 'saga'],
    'lrclf__max_iter': [400, 500, 600, 700],
    'lrclf__tol': [0.001, 0.01, 0.1]
}

# Grid search for hyperparameter tuning
grid_search_lrclf = GridSearchCV(
    estimator=pipeline_lrclf,
    param_grid=param_grid_lrclf,
    cv=cv,
    error_score='raise',
    scoring='accuracy')
```

```
Best parameters of logistic regression
classifier:
{'lrclf__max_iter': 400,
'lrclf__solver': 'saga', 'lrclf__tol':
0.1}
```

- Decision Trees

```
# Setting the parameter distributions for random search
param_dist_dtclf = {
    'dtclf__criterion': ['gini', 'entropy'],
    'dtclf__max_depth': [None, 10, 20, 30, 40, 50],
    'dtclf__min_samples_split': [2, 5, 10],
    'dtclf__min_samples_leaf': [1, 2, 4]
}

# Random search for hyperparameter tuning
random_search_dtclf = RandomizedSearchCV(estimator=pipeline_dtclf,
                                         param_distributions=param_dist_dtclf,
                                         n_iter=50,
                                         cv=cv,
                                         error_score='raise',
                                         scoring='accuracy',
                                         random_state=42)
```

```
Best parameters of Decision Tree:
{'dtclf__min_samples_split': 2,
'dtclf__min_samples_leaf': 1,
'dtclf__max_depth': 50,
'dtclf__criterion': 'gini'}
```

- SVM

```
param_grid_svc = {
    'svc__kernel': ['rbf'],
    'svc__C': [1, 10],
    'svc__gamma': ['scale']
}

# Grid search for hyperparameter tuning
grid_search_svc = GridSearchCV(estimator=pipeline_svc,
                               param_grid=param_grid_svc,
                               cv=cv,
                               error_score='raise',
                               scoring='accuracy')
```

```
Best parameters of SVM:
{'svc__C': 10, 'svc__gamma': 'scale',
'svc__kernel': 'rbf'}
```

- Random Forests

```
# Parameter grid for Random Forest
param_dist_rf = {
    'rf__n_estimators': [10, 50, 100, 150, 200],
    'rf__max_depth': [None, 10, 20, 30, 40, 50],
    'rf__min_samples_split': [2, 5, 10],
    'rf__min_samples_leaf': [1, 2, 4],
    'rf__bootstrap': [True, False]
}

# Random search for hyperparameter tuning
random_search_rf = RandomizedSearchCV(estimator=pipeline_rf,
                                      param_distributions=param_dist_rf,
                                      cv=cv,
                                      error_score='raise',
                                      scoring='accuracy',
                                      random_state=42)
```

```
Best parameters of Random Forest:
{'rf__n_estimators': 150,
'rf__min_samples_split': 10,
'rf__min_samples_leaf': 1,
'rf__max_depth': 30, 'rf__bootstrap':
False}
```

- Neural Networks (Multi-Layer Perceptron)

```
# Simplified hyperparameters to be tuned
param_dist_mlp = {
    'mlp__hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'mlp__activation': ['relu', 'tanh'],
    'mlp__solver': ['adam'],
    'mlp__learning_rate_init': [0.001, 0.01],
    'mlp__early_stopping': [True],
    'mlp__validation_fraction': [0.1],
    'mlp__n_iter_no_change': [10]
}

# Random search for hyperparameter tuning
random_search_mlp = RandomizedSearchCV(estimator=pipeline_mlp,
                                       param_distributions=param_dist_mlp,
                                       n_iter=10,  # Number of parameter s
                                       cv=cv,
                                       error_score='raise',
                                       scoring='accuracy',
                                       random_state=42)
```

```
Best parameters of MLP:
{'mlp__validation_fraction': 0.1,
'mlp__solver': 'adam',
'mlp__n_iter_no_change': 10,
'mlp__learning_rate_init': 0.01,
'mlp__hidden_layer_sizes': (50, 50),
'mlp__early_stopping': True,
'mlp__activation': 'relu'}
```

And next, we used the models with those best parameters to predict the test data. The results are the following:

```
Results for Logistic Regression:
Accuracy: 0.6371
Precision: 0.6341
Recall: 0.6479
F1 Score: 0.6409
Confusion Matrix:
[[1957 1168]
 [1100 2024]]
```

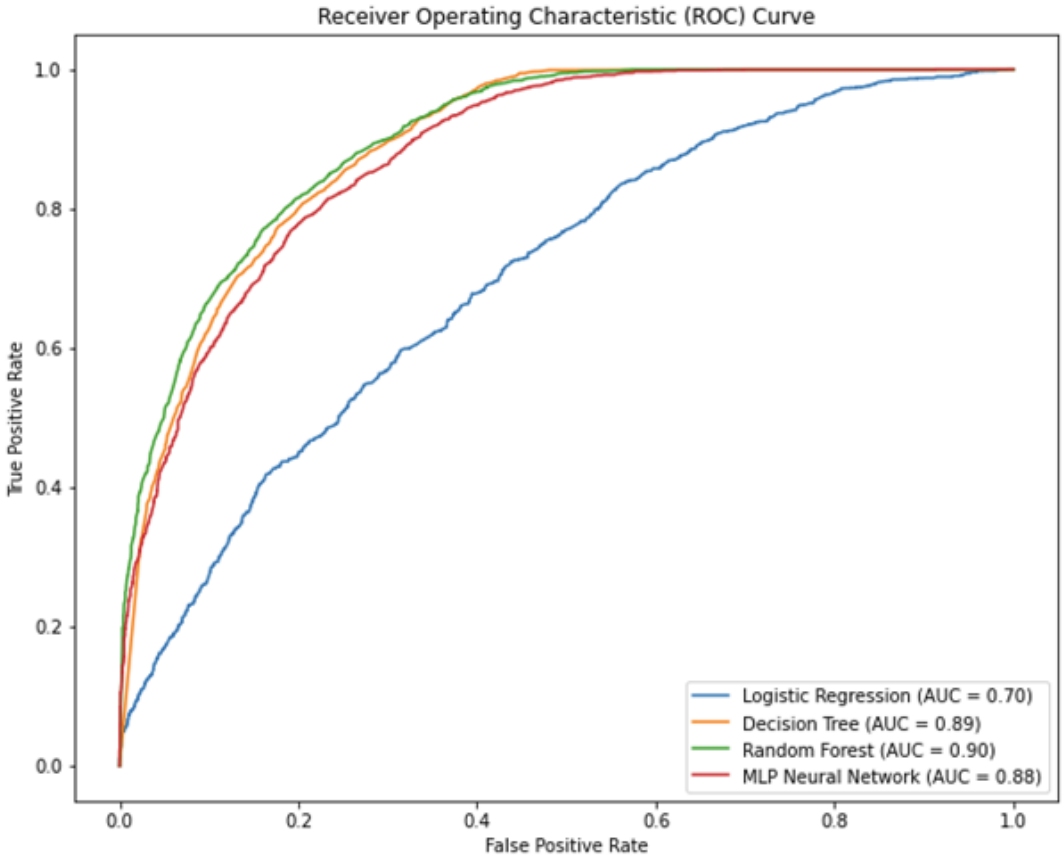
```
Results for Decision Tree:
Accuracy: 0.7993
Precision: 0.7763
Recall: 0.8409
F1 Score: 0.8073
Confusion Matrix:
[[2368  757]
 [ 497 2627]]
```

```
Results for SVM:
Accuracy: 0.7812
Precision: 0.7648
Recall: 0.8121
F1 Score: 0.7878
Confusion Matrix:
[[2345  780]
 [ 587 2537]]
```

```
Results for Random Forest:
Accuracy: 0.8057
Precision: 0.7866
Recall: 0.8390
F1 Score: 0.8120
Confusion Matrix:
[[2414  711]
 [ 503 2621]]
```

```
Results for MLP Neural Network:
Accuracy: 0.7876
Precision: 0.7686
Recall: 0.8230
F1 Score: 0.7949
Confusion Matrix:
[[2351  774]
 [ 553 2571]]
```

From these results, it seems like Random Forest is the best due to the fact that it has the highest accuracy. We then double checked by plotting the Receiver Operating Characteristic (ROC) curve and calculating the Area Under Curve (AUC):



Due to the computational cost, the ROC curve for SVC was not drawn as the 'probability' parameter is set to False by default.

```
The best-performing model is Random
Forest with an AUC of 0.90
```

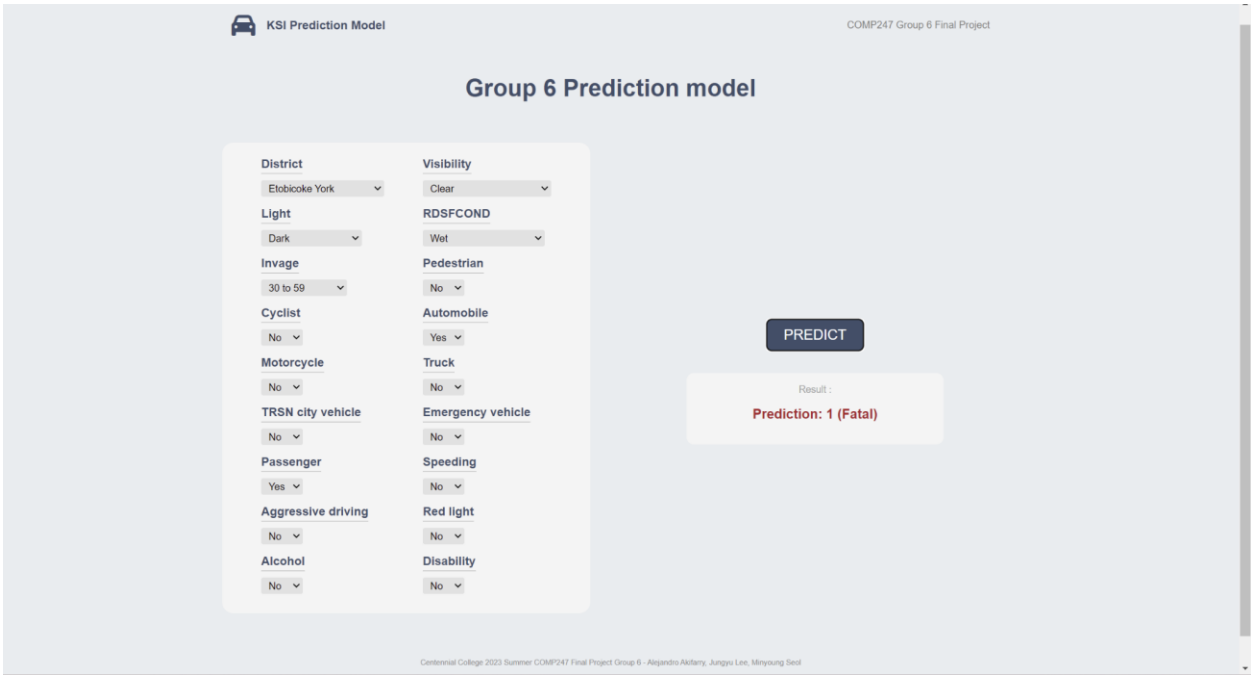
Therefore, based on our feature selection, data modeling, and model building of those 5 models, we concluded that the best performing model to be deployed is the Random Forest model with these parameters:

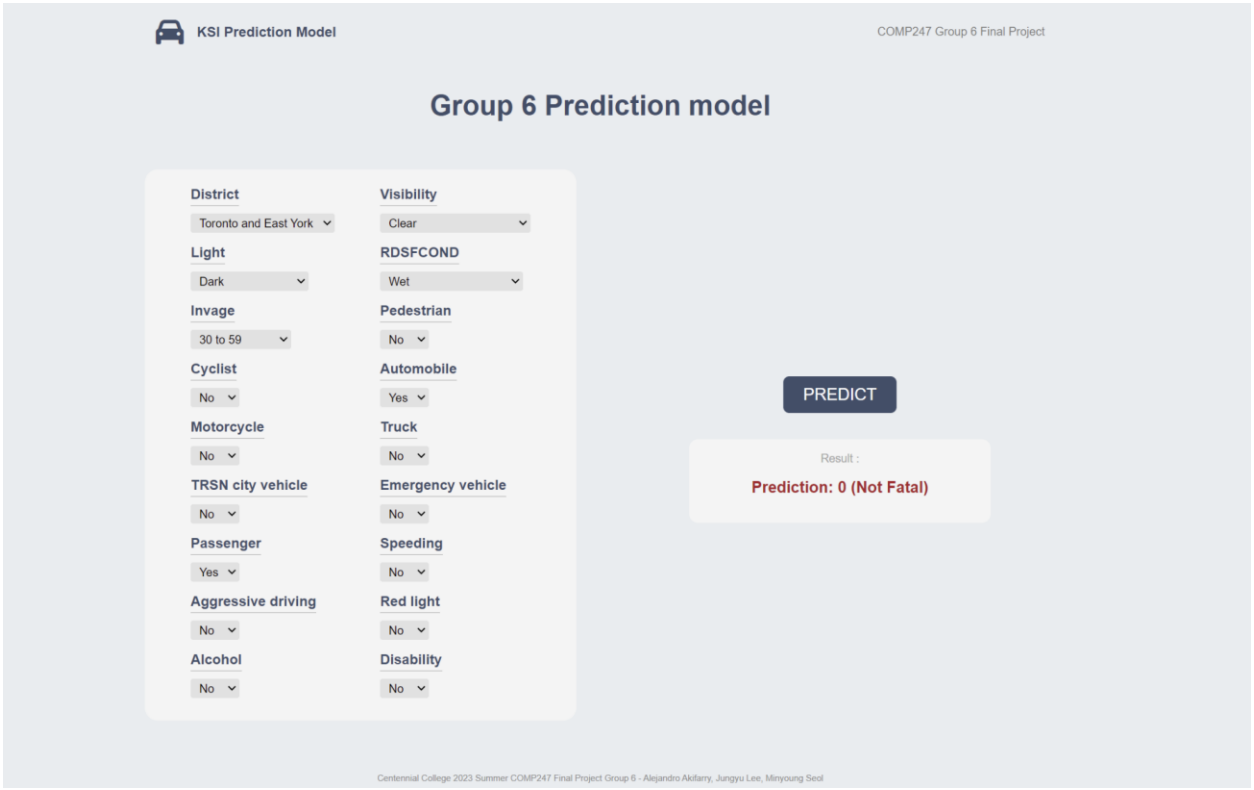
- rf__n_estimators: 150
- rf__min_samples_split: 10
- rf__min_samples_leaf: 1
- rf__max_depth: 30

- rf__bootstrap: False

Conclusion

To bring it all together, we first started some data exploration for the KSI dataset, in which we found some interesting key findings like the fact that aggressive/distracted driving conditions causes more accidents than alcohols in the dataset as well as that the type of vehicle involved the most in those accidents are automobiles. We then did some feature selection where we removed the features that we deemed not worth using and data transformation to allow it to be used to train predictive models. Consequently, we trained 5 models, Logistic Regression Classifier, Decision Tree, Support Vector Machine, Random Forest, Multi-Layer Perceptron Neural Network, and after using them to predict the test data, we deduced that the Random Forest model is the best one for our case since it has the highest accuracy score as well as AUC among the other models. Finally, we pickled that model to be made into an API using the Flask framework and made a front-end to be used by the public. And here are 2 screenshots of the final product (in the next page of this report):


KSI Prediction Model
COMP247 Group 6 Final Project
Group 6 Prediction model
District
Etobicoke York
Visibility
Clear
Light
Dark
RDSFCOND
Wet
Invage
30 to 59
Pedestrian
No
Cyclist
No
Automobile
Yes
Motorcycle
No
Truck
No
TRSN city vehicle
No
Emergency vehicle
No
Passenger
Yes
Speeding
No
Aggressive driving
No
Red light
No
Alcohol
No
Disability
No
PREDICT
Result :
Prediction: 1 (Fatal)
Centennial College 2023 Summer COMP247 Final Project Group 6 - Alejandro Akifarry, Jungyu Lee, Minyoung Seol


KSI Prediction Model
COMP247 Group 6 Final Project
Group 6 Prediction model
District
Toronto and East York
Visibility
Clear
Light
Dark
RDSFCOND
Wet
Invage
30 to 59
Pedestrian
No
Cyclist
No
Automobile
Yes
Motorcycle
No
Truck
No
TRSN city vehicle
No
Emergency vehicle
No
Passenger
Yes
Speeding
No
Aggressive driving
No
Red light
No
Alcohol
No
Disability
No
PREDICT
Result :
Prediction: 0 (Not Fatal)
Centennial College 2023 Summer COMP247 Final Project Group 6 - Alejandro Akifarry, Jungyu Lee, Minyoung Seol