KSI Project: Phase 2 Report

ICET, Centennial College

Supervised Learning

Group 1:

Sachin Kaushik - 301379467

Antony Tibursias Brightes Machado- 301380198

Namneet - 301279374

Seyed Nariman Hosseini- 301321754

Muhammed Ikbal Ekinci - 301309283

Aug 13, 2024

Table of content

Executive summary

The project aimed to develop a predictive service that estimates the likelihood of fatal collisions in the city of Toronto based on historical data collected by the Toronto Police Department. The service is intended for use by both the police department and the general public, providing insights that can enhance safety measures and inform personal precautions. Key findings include the identification of critical features influencing collision outcomes and the effectiveness of various machine learning algorithms in predicting fatalities. Given this focus, selecting a predictive model that aligns with the specific goals of minimizing false negatives is essential, thereby maximizing the recall of the "Killed" class. Additionally, the Area Under the Curve (AUC) score is considered to assess the model's overall ability to distinguish between the "Killed" and "Survived" classes.

Overview of Solution

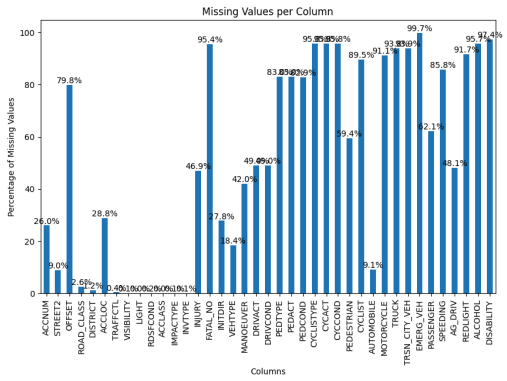
Tools and Libraries used

In our project, we utilized several tools and libraries to build and deploy our machine-learning model:

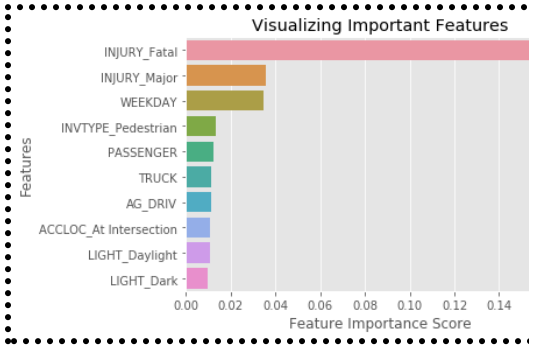
1. **Python**: The primary programming language used for the entire project.
2. **scikit-learn**:
    - **Pipeline and ColumnTransformer**: Used for creating a robust data preprocessing pipeline.
    - **SimpleImputer**: For handling missing data.
    - **OneHotEncoder**: For encoding categorical features.
    - **StandardScaler**: For scaling numeric features.
    - **Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, SVM**: Used for modelling and evaluation.
    - **SMOTE (Synthetic Minority Over-sampling Technique)**: For handling class imbalance.
3. **pandas**: For data manipulation and preparation, including loading data into DataFrames and handling data preprocessing steps.
4. **NumPy**: Used for numerical operations and handling arrays.
5. **pickle**: For saving the trained model and preprocessing pipeline for later use.
6. **Flask**: A lightweight web framework used to create the API for deploying the model, allowing real-time predictions.
7. **matplotlib and seaborn**: Used for data visualization, including plotting performance metrics and visualizing data distributions.
8. **Cross-Validation and Evaluation Metrics**: Built-in scikit-learn tools for model validation and evaluation, such as cross_val_score and metrics like accuracy_score, precision_score, recall_score, f1_score, and roc_auc_score.

A bit about Data

- Data Imbalanced
- High missing values column count



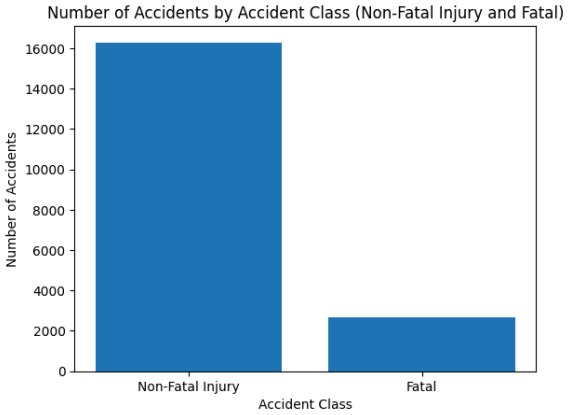
- The injury column has a high correlation to the target column leading to overfitting



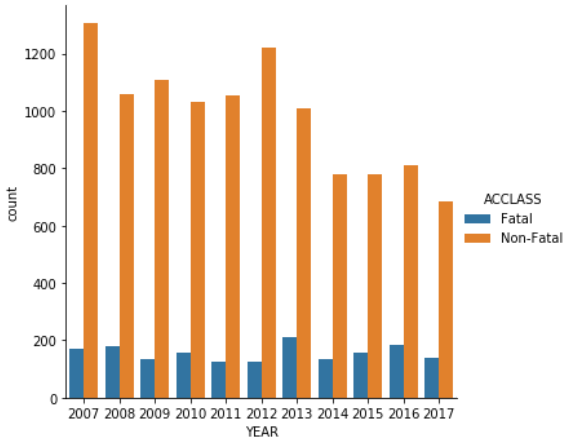
- **Columns with Binary Values (Yes/No)**: ['PEDESTRIAN', 'CYCLIST', 'AUTOMOBILE', 'MOTORCYCLE', 'TRUCK', 'TRSN_CITY_VEH', 'EMERG_VEH', 'PASSENGER', 'SPEEDING', 'AG_DRIV', 'REDLIGHT', 'ALCOHOL', 'DISABILITY']

Graphical Representation of Data

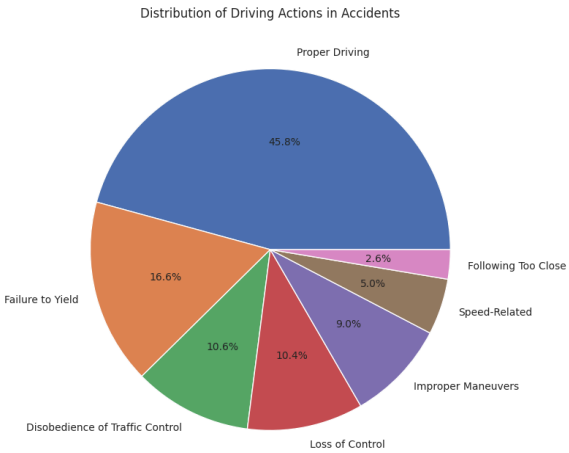
Target Class (ACCLASS: Fatal/NonFatal) Distribution



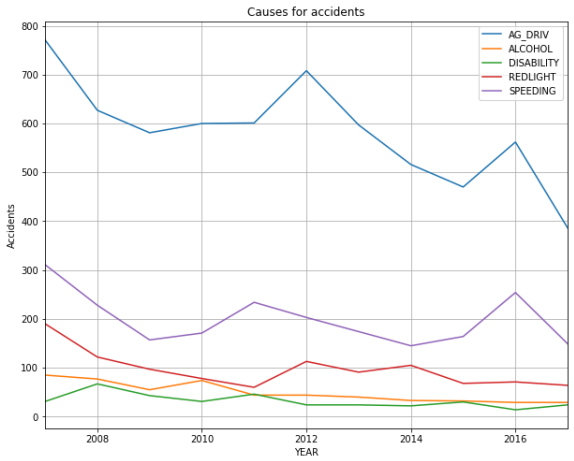
Fatal/ Non-Fatal over years



Condition of Driver:

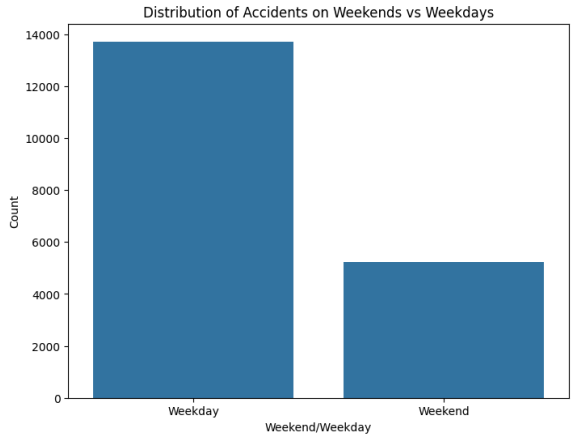


Cause of Accidents over the years:

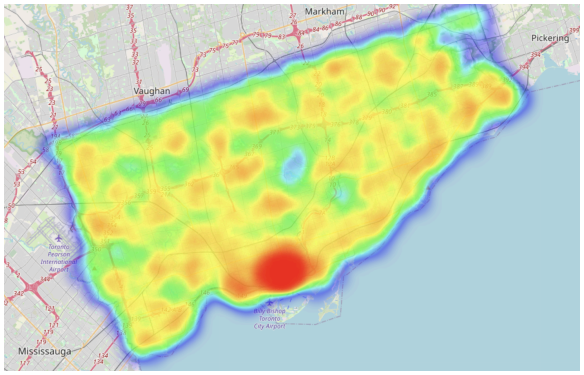


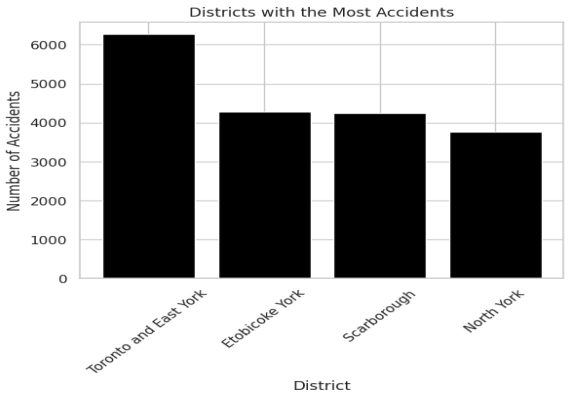
Accidents comparison with the type of the Day:

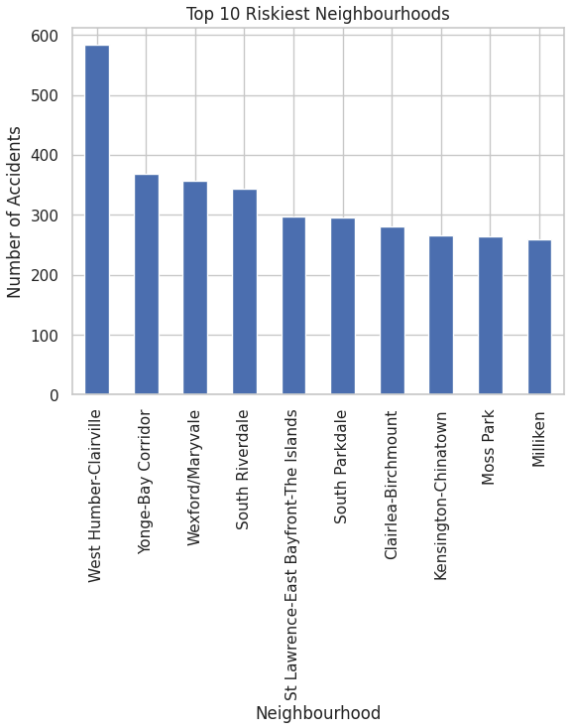
Observation: Almost 14000 in 5 days and about 5000 (1/3rd) in two days alone. Can be interpreted as weekends are more prone to having accidents.



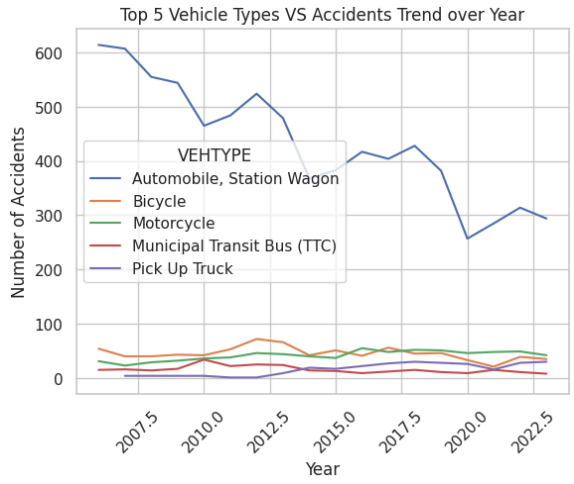
Accident Density visualised on map: Overall similar distribution but as aligned with population ratio, Downtown had the most accidents.

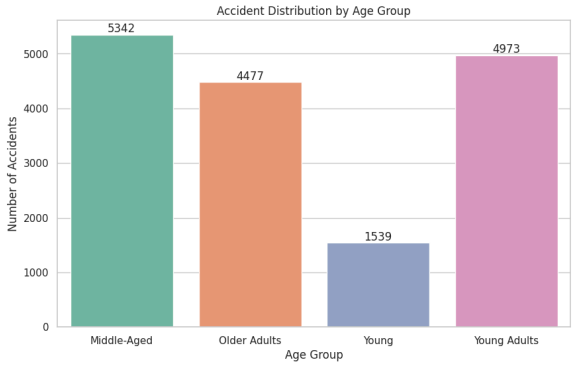






Accident Trend over time: when explored with Vehicle types.
Observation: The only significant change in vehicular accident type is Automobile/ Station Wagon (Cars). Others had no major change.





Feature selection

Data Cleaning

- **Binary Columns Handling:** Converted binary columns from Yes/No values to 1/0 for the following columns.
- **Missing Values Handling**:
    - **Dropped Columns: Removed columns with more than 60% missing data, particularly those with categorical, unique, or non-pattern values.**
    - **INVAGE Missing Values**: Filled missing or 'unknown' values in the INVAGE column with the mode value.
    - **ROAD_CLASS Missing Values**: Dropped rows with missing values in the ROAD_CLASS column.
    - **TRAFFCTL Missing Values**: Filled missing values in the TRAFFCTL column with 'Unknown'.
- **Categorical Values Handling**:
    - **One-Hot Encoding**: Applied one-hot encoding to categorical columns such as INVAGE_GROUPED, DISTRICT, DRIVCOND_GROUPED, DRIVACT_GROUPED, VEHTYPE_GROUPED, and TRAFFCTL_GROUPED.
    - **Boolean Encoding**: Converted 'Yes'/'No' values to 1/0 for columns like AG_DRIV, ALCOHOL, DISABILITY, REDLIGHT, SPEEDING, PEDESTRIAN, AUTOMOBILE, and PASSENGER.

- Feature Engineering:

Additional features were created to capture important aspects such as whether the accident occurred on a weekend, at an intersection, or involved specific vehicle types.

- **Season Extraction**: Extracted the season from the date column and created a new column Season.
- **Day of the Week**: Created a Weekday/Weekend column to

represent the day of the week from the date.

- **Grouping Values**:
  - **INVAGE Grouping**: Grouped age categories into broader groups (e.g., 'Young', 'Young Adults', 'Middle-Aged', 'Older Adults').
  - **DRIVCOND Grouping**: Mapped detailed driving conditions to broader categories like 'Normal', 'Distracted/Inattentive', 'Impaired', 'Medical/Physical Issues', and 'Unknown/Other'.
  - **VEHTYPE Grouping**: Mapped specific vehicle types to broader categories such as 'Automobiles and Light Vehicles', 'Bicycles and Motorcycles', 'Public Transit and Buses', 'Trucks and Heavy Vehicles', 'Emergency Vehicles', and 'Others'.
  - **TRAFFCTL Grouping**: Mapped specific traffic control types to broader categories like 'No Control', 'Traffic Signals and Signs', and 'Controlled Intersections'.

Pipeline Preparation

We used scikit-learn's Pipeline and ColumnTransformer classes in the get_pipeline_x_y function to handle the following tasks:

- **Feature Identification**: The pipeline automatically separates numeric and categorical features in the dataset.
- **Handling Missing Data**: For categorical features, we fill missing values with 'missing' using SimpleImputer. For numeric features, we use the median value.
- **Feature Encoding**: Categorical features are one-hot encoded with OneHotEncoder, which manages unknown categories well.
- **Feature Scaling**: Numeric features are standardized with StandardScaler to ensure they're on a similar scale.
- **Data Splitting**: The data is split into training and testing sets, with 20% reserved for testing

These pipelines are combined using ColumnTransformer to create a full preprocessing pipeline.

The function also splits the data into training and testing sets.

Data modeling

Feature Importance

Extra Trees Classifier: Used to determine the most important features influencing the target variable (Fatal). Techniques like Chi-Square were used to measure the degree or correlation with the target feature.

To reduce model complexity, Features having unique identifiers, duplicate data, random values or features with no relation to the occurrence of

an accident were removed like Accident number, Hood_140, etc.

Features selected for training

Numeric features: VISIBILITY, LIGHT, RDSFCOND, Driver, PEDESTRIAN, CYCLIST, AUTOMOBILE, MOTORCYCLE, TRUCK, TRSN_CITY_VEH, EMERG_VEH, PASSENGER, SPEEDING, AG_DRIV, REDLIGHT, ALCOHOL, DISABILITY, Primary_Road, Weekend, At Intersection, Fatal_Injury

Categorical features: DISTRICT, DRIVACT_GROUPED, INVAGE_GROUPED, DRIVCOND_GROUPED, VEHTYPE_GROUPED, TRAFFCTL_GROUPED

Model Building

Models Evaluated

We tested five different classification models, each selected for their specific strengths:

- Logistic Regression: A simple model that's easy to interpret and works well with linear data. We used the 'saga' solver and balanced class weights to handle class imbalance.
- Decision Tree: A non-linear model that captures complex feature interactions. We also used balanced class weights here.
- Random Forest: An ensemble of decision trees, which is generally accurate and less prone to overfitting. We used 100 trees and balanced class weights.
- Gradient Boosting: Another ensemble method that builds models sequentially to improve accuracy. It's effective for many types of problems.
- Support Vector Machine (SVM): A versatile algorithm good for both linear and non-linear classification. We enabled probability estimates and used balanced class weights.

Model Training and Evaluation Process

For each model, we followed these steps:

- SMOTE Integration: We used SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance in our dataset.
- Model Training: Each model was trained on the preprocessed training data.
- Prediction: The trained models were then used to make predictions on the test set.
- Performance Metrics: We assessed each model using:
    - Accuracy: How often the model's predictions were correct.
    - Precision: The percentage of correct positive predictions.
    - Recall: The percentage of actual positives correctly identified.
    - F1 Score: A balance between precision and recall.
    - AUC (Area Under the ROC Curve): The model's ability to distinguish between classes.
- Cross-Validation: To ensure the results were reliable, we performed 3-fold cross-validation, focusing on the AUC score.

Model Comparison and Selection

To identify as many "Killed" cases as possible, the model selection process prioritizes **recall**. Recall measures the proportion of actual positive cases (in this instance, individuals who were killed) that the model correctly identifies. A higher recall is crucial for ensuring that the model correctly captures the majority of the "Killed" cases, even at the potential expense of increasing false positives. The **AUC score** is also considered to evaluate the model's overall performance in distinguishing between the two classes.

After evaluating all models, we compared their performance across all metrics, especially the AUC score, which is crucial for our imbalanced dataset. The model with the highest AUC score was selected as the best model:

```
best_model = max(results, key=lambda x: results[x]['AUC'])
print(f"\nBest model based on AUC: {best_model}")
```

This method ensured we chose a model that not only performed well overall but also effectively identified both fatal and non-fatal accidents.

Model Persistence

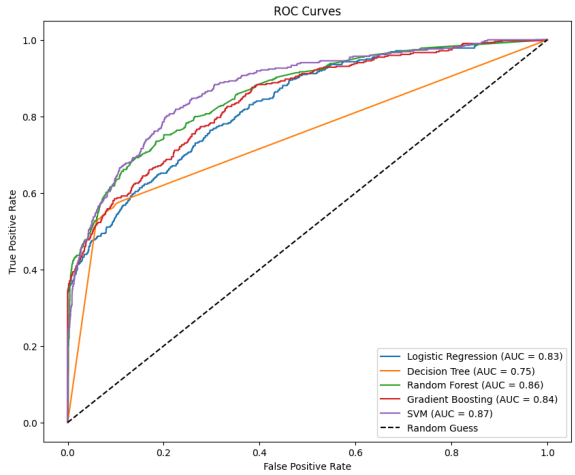
Once we identified the best model, we saved it along with the full preprocessing pipeline for future use:

```
with open('best_model.pkl', 'wb') as file:
  pickle.dump(best_model, file)

with open('full_pipeline_transformer.pkl', 'wb') as file:
  pickle.dump(full_pipeline_transformer, file)
```

This step is vital for deploying the model in our Flask API, as it ensures that the same preprocessing and model are used in both training and prediction environments.

Model Performance Results



Detailed Analysis

1. **Support Vector Machine (SVM)**:
   - **Accuracy**: 86.12%
   - **Precision**: 48.77%
   - **Recall**: 66.35%
   - **F1 Score**: 56.22%
   - **AUC**: 0.8741
   - **CV AUC**: 0.8722
   - **Classification Summary**:
     - Survived (0): 2,549
     - Killed (1): 570
2. **Logistic Regression**:
   - **Accuracy**: 78.33%
   - **Precision**: 34.00%
   - **Recall**: 65.16%
   - **F1 Score**: 44.68%
   - **AUC**: 0.8287
   - **CV AUC**: 0.8364
   - **Classification Summary**:

        - Survived (0): 2,316
        - Killed (1): 803
3. **Decision Tree**:
    - **Accuracy**: 87.98%
    - **Precision**: 55.47%
    - **Recall**: 53.22%
    - **F1 Score**: 54.32%
    - **AUC**: 0.7456
    - **CV AUC**: 0.7544
    - **Classification Summary**:
        - Survived (0): 2,717
        - Killed (1): 402
4. **Random Forest**:
    - **Accuracy**: 90.29%
    - **Precision**: 72.66%
    - **Recall**: 44.39%
    - **F1 Score**: 55.11%
    - **AUC**: 0.8588
    - **CV AUC**: 0.8541
    - **Classification Summary**:
        - Survived (0): 2,863
        - Killed (1): 256
5. **Gradient Boosting**:
    - **Accuracy**: 90.86%
    - **Precision**: 84.90%
    - **Recall**: 38.90%
    - **F1 Score**: 53.36%
    - **AUC**: 0.8413
    - **CV AUC**: 0.8483
    - **Classification Summary**:
        - Survived (0): 2,927
        - Killed (1): 192

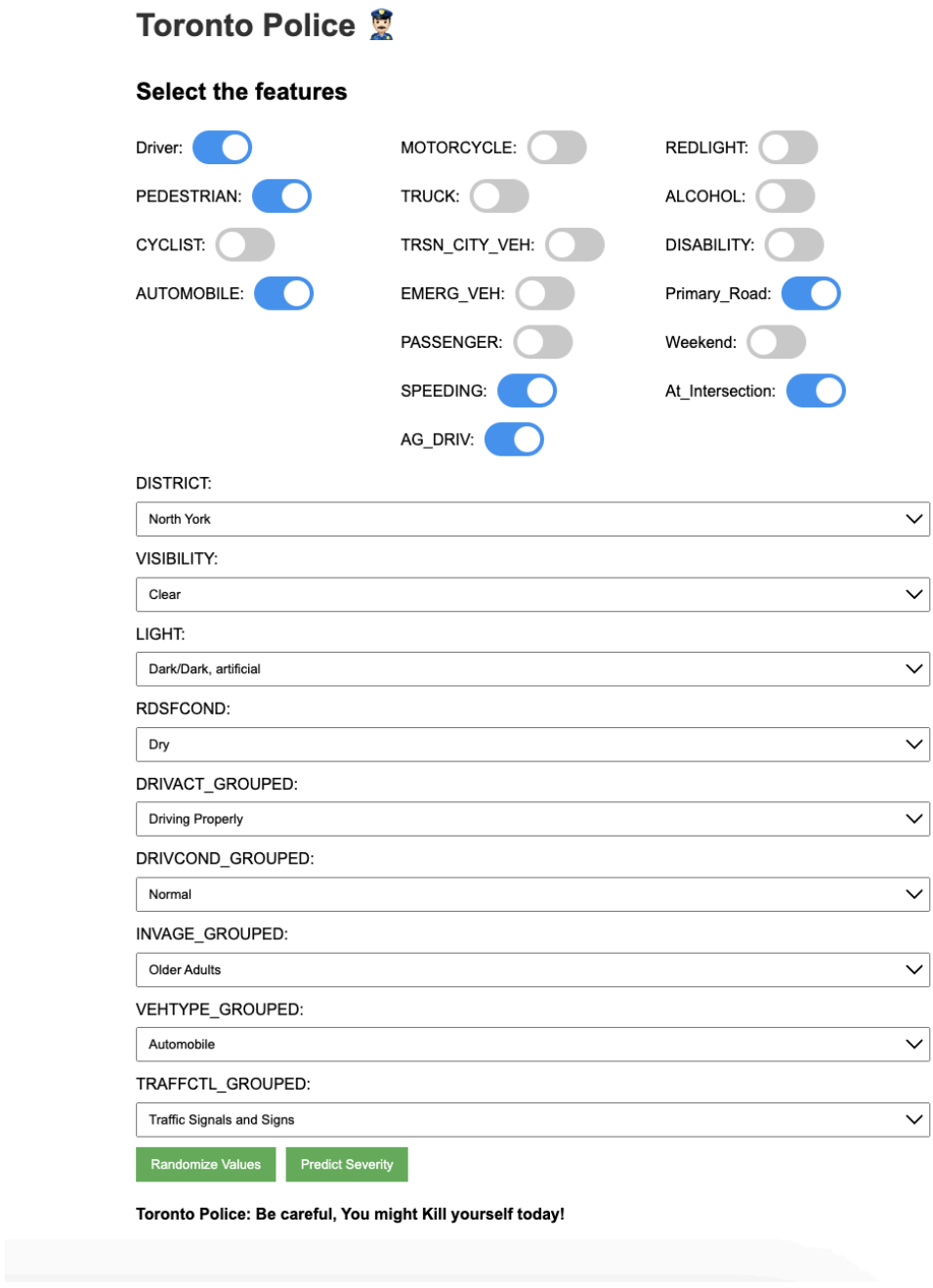
Analysis and Conclusion

- **Support Vector Machine (SVM)** is the most suitable model for this case study if the priority is to maximize the number of "Killed" cases identified. SVM has the highest recall rate of **66.35%** and a strong **AUC score of 0.8741**, indicating a robust ability to distinguish between "Killed" and "Survived" individuals. This model is the preferred choice for this analysis when focusing on recall.
- **Logistic Regression** also performs well, with a recall of **65.16%** and an AUC score of **0.8287**, making it a viable alternative.
- Other models like **Decision Tree**, **Random Forest**, and **Gradient Boosting** have lower recall rates, making them less suitable for the specific objective of identifying as many "Killed" cases as possible, despite their higher precision and accuracy in some cases.

Flask Integration for API Development

To deploy our model, we created a Flask-based API for real-time predictions:

- **API Structure**: The API has two main routes:
    - The home route (/) provides a web interface for user input.
    - The prediction route (/predict) handles POST requests with input data.
- **Data Processing**: Incoming data is processed using the saved preprocessing pipeline to ensure consistency with the training data.
- **Prediction Generation**: The processed data is fed into our best model to generate predictions.
- **Response Formatting**: The API returns predictions in a JSON format, showing predicted severity (fatal or non-fatal) classification.
- **Error Handling**: The API includes basic error handling to manage any issues with input data or processing.


Toronto Police
Select the features
Driver:
PEDESTRIAN:
CYCLIST:
AUTOMOBILE:
MOTORCYCLE:
TRUCK:
TRSN_CITY_VEH:
EMERG_VEH:
PASSENGER:
SPEEDING:
AG_DRIV:
REDLIGHT:
ALCOHOL:
DISABILITY:
Primary_Road:
Weekend:
At_Intersection:
DISTRICT:
North York
VISIBILITY:
Clear
LIGHT:
Dark/Dark, artificial
RDSFCOND:
Dry
DRIVACT_GROUPED:
Driving Properly
DRIVCOND_GROUPED:
Normal
INVAGE_GROUPED:
Older Adults
VEHTYPE_GROUPED:
Automobile
TRAFFCTL_GROUPED:
Traffic Signals and Signs
Randomize Values
Predict Severity
Toronto Police: Be careful, You might Kill yourself today!