ABSTRACT

Traffic accidents remain a significant public safety challenge, necessitating efficient predictive models to evaluate crash severity. This report explores various machine learning techniques to predict the severity of traffic crashes using a dataset encompassing multiple factors such as road conditions, vehicle specifics, and driver attributes. Through comprehensive exploratory data analysis, key predictors of crash severity were identified. Several predictive models were developed, including logistic regression, decision trees, random forests, K-nearest neighbors, multilayer perceptrons, and support vector classifiers. Techniques such as SMOTE and Random Over Sampling were employed to address class imbalance. A significant positive aspect of this study is the implementation of a flexible and generalized pipeline in the code. This pipeline automates data handling, including the imputation of missing values and preprocessing steps. It allows for the selection and comparison of different encoders, samplers, and classifiers, ultimately providing predictions. This modular and adaptable pipeline framework enhances the robustness and scalability of the predictive modeling process. Results indicated that logistic regression and random forests performed robustly, with logistic regression emerging as a reliable baseline model due to its balanced performance and robustness against class imbalance. Ensemble methods like random forests showed promise but required careful tuning to mitigate overfitting. More complex models like K-nearest neighbors and multilayer perceptrons exhibited varying degrees of overfitting, suggesting the need for further refinement. The findings underscore the potential of machine learning to significantly enhance road safety by providing insights into the patterns of traffic accidents and their severity. This can assist policymakers and traffic management systems in real-time decision-making and deploying preventive measures effectively.

Keywords: Traffic Safety, Crash Severity, Machine Learning, Predictive Analytics, Data Preprocessing, Exploratory Data Analysis, Model Evaluation, Class Imbalance, Pipeline Automation

1. Chapter 1: Introduction

1.1. Purpose of the Analysis

The primary aim of this analysis is to develop a predictive model that can accurately assess the severity of vehicle crashes. This model serves as a crucial tool for policymakers, traffic management authorities, and public safety officials, enabling them to implement more targeted interventions to improve road safety. By predicting crash severity, resources can be allocated more efficiently, and preventative measures can be tailored to specific risk factors associated with severe accidents. Predictive analytics in traffic safety can help mitigate the consequences of traffic accidents by providing timely insights and facilitating proactive measures.

1.2. Data Description

The dataset for this project was sourced from the Louisiana Department of Transportation and Development (LA DOTD), covering detailed records of traffic accidents from 2016 to 2021. Each record in the dataset provides comprehensive details about each traffic accident, including geographic coordinates, road conditions, vehicle characteristics, driver demographics, environmental factors, and the severity of the crash. The dataset contains approximately 49,336 records with 54 features, ensuring a rich basis for analysis. Among these 54 features 30 of them (24 categorical, 3 numerical, and 3 boolean) were selected for the purpose of this study. One of the key challenges of this project is that it deals with a multi-class classification problem, where the target variable, crash severity, is highly imbalanced. Most records fall into the 'No Injury' class, making the prediction of minority classes like 'Fatal' and 'Severe' crashes more challenging. Initial preprocessing steps included handling missing values, correcting anomalies, and transforming categorical variables to ensure data quality and consistency.

1.3. Objectives of the Study

This study aims to achieve the following objectives:

1. **Exploratory Data Analysis (EDA):** To conduct a thorough analysis of the dataset to uncover underlying patterns, detect anomalies, and understand the relationships between different variables related to crash occurrences.
2. **Feature Engineering and Selection:** To identify and create meaningful variables that significantly impact the prediction of crash severity, enhancing the predictive power of the machine learning models.
3. **Model Development:** To employ various machine learning techniques to develop models that can predict the severity of traffic accidents. This involves comparing several models to determine the most effective approach based on performance metrics.
4. **Evaluation and Optimization:** To rigorously assess the performance of each model using appropriate evaluation metrics and employ techniques such as cross-validation and hyperparameter tuning to optimize the models.
5. **Implementation Recommendations:** Based on the findings, to provide actionable recommendations that could be implemented by relevant authorities to reduce the frequency and severity of traffic accidents.

1.4. Significance of the Study

The significance of this study lies in its potential to enhance road safety through data-driven insights. By understanding the factors that contribute to severe crashes, interventions can be better

designed to prevent such incidents and mitigate their impacts when they occur. This study is particularly relevant in the context of growing urbanization and increasing vehicle populations in cities worldwide, where traffic accidents pose a significant risk to public health and safety. The findings can inform policy-making, improve traffic management strategies, and contribute to the development of more effective safety measures.

1.5. Structure of the Report

This report is structured to provide a logical flow from initial data handling to deep analytical insights and practical recommendations. It begins with a detailed exploration of the dataset in Chapter 2, followed by the development and evaluation of predictive models in Chapter 3 and Chapter 4. Chapter 5 presents the results of the analysis, highlighting the performance of various models and their implications. Chapter 6 concludes with a discussion of the findings, their implications for improving road safety, and recommendations for future work. The final chapter lists all references used in the study.

2. Chapter 2: Literature Review

2.1. Introduction

The purpose of this chapter is to provide a comprehensive review of the existing literature related to the prediction of traffic crash severity using machine learning techniques. This chapter will cover the historical context and evolution of crash severity analysis, factors influencing crash severity, various machine learning models used for prediction, data preprocessing and feature engineering techniques, evaluation metrics and model validation approaches, and the challenges and future directions in this field.

Road traffic crashes claim around 1.35 million lives worldwide annually, with 50 million suffering severe injuries, representing a significant social and economic burden (WHO, 2018). Reducing the severity of these crashes is a global priority, especially for vulnerable road users such as pedestrians, cyclists, and motorcyclists, who account for a substantial proportion of fatalities (Chang & Wang, 2006; Lin & Kraus, 2008).

2.2. Historical Context and Evolution

Traffic safety research has evolved significantly over the past few decades, transitioning from traditional statistical methods to advanced machine learning techniques. Early research on crash severity primarily utilized statistical modeling techniques. Regression models, such as binary logit or probit models for binary outcomes (Fan et al., 2016; Khattak et al., 1998; Shibata & Fukuda, 1994) and multinomial logit models for multiple severity levels (Malyshkina & Mannering, 2010; Ye & Lord, 2014), have been common. More advanced techniques like generalized extreme value models (Shankar et al., 1996; Yasmin & Eluru, 2013) and ordered probit/logit models (Khattak et al., 2002; Kockelman & Kweon, 2002) were developed to handle correlations and ordinal nature of severity data.

With the advent of big data and advancements in computational power, machine learning models have become increasingly popular for their ability to handle large datasets and complex relationships. These models, including decision trees, support vector machines, neural networks, and ensemble methods, have shown promise in improving prediction accuracy and handling high-dimensional data.

2.3. Factors Influencing Crash Severity

Numerous factors influence the severity of traffic crashes, including:

- **Driver-related factors**: Age, gender, driving experience, alcohol and drug use, fatigue, and medical conditions.
- **Vehicle-related factors**: Vehicle type, age, condition, safety features, and maintenance.
- **Environmental factors**: Weather conditions, road surface conditions, lighting, and time of day.
- **Road-related factors**: Road type, geometry, traffic control devices, and traffic volume.

Understanding these factors is crucial for developing effective predictive models and implementing targeted interventions to improve road safety. Vulnerable road users, especially motorcyclists, have significantly higher risks of severe injuries or fatalities (Chang & Wang, 2006; Lin & Kraus, 2008). Data quality and reliability are crucial for effective analysis and prediction of crash severity (Abdul Sukor, 2017; Laiou et al., 2017).

2.4. Machine Learning Models for Crash Severity Prediction

Several machine learning models have been employed to predict crash severity, each with its own strengths and weaknesses. The following subsections provide a detailed explanation of various classifiers commonly used in this domain, along with relevant literature.

2.4.1. Decision Trees

Decision Trees are non-linear models that split the data into subsets based on the value of input features, forming a tree-like structure. They are useful for both classification and regression tasks, especially when the relationship between features and the target is complex and non-linear. For example, they can classify whether a person is likely to default on a loan based on their credit score and other financial information. Decision trees have been widely used in traffic safety research for crash severity prediction due to their simplicity and interpretability (Abellán et al., 2013; Chong et al., 2005).

2.4.2. Random Forests

Random Forests are an ensemble learning method that builds multiple decision trees and merges their predictions to improve accuracy and control over-fitting. Each tree is trained on a random subset of the data and features. They are suitable for both classification and regression tasks, robust against overfitting, and perform well on a variety of datasets. For example, they can predict whether a customer will default on a loan based on

their credit history and financial attributes. Random Forests have demonstrated strong performance in crash severity prediction and are effective in handling large datasets with high-dimensional data (Das et al., 2009; Harb et al., 2009).

2.4.3. Support Vector Machines (SVM)

Support Vector Machines seek to find the optimal hyperplane that separates data points of different classes with the maximum margin. They are particularly effective in high-dimensional spaces. SVMs are suitable for both linear and non-linear classification tasks, and the kernel trick can be used to handle non-linear separable data by transforming it into a higher dimension. For example, they can classify whether a tumor is malignant or benign based on various medical attributes. SVM has been used in various studies for crash severity prediction and has shown effectiveness in high-dimensional spaces (Chong et al., 2005; Li et al., 2012). Its ability to handle non-linear data through kernel methods makes it a powerful tool for traffic safety research.

2.4.4. Neural Networks and Deep Learning

MLPClassifier (Multi-Layer Perceptron) is a type of feedforward artificial neural network consisting of multiple layers of neurons, including at least one hidden layer. It uses backpropagation for training the network and is suitable for complex problems where the relationship between input features and output is highly non-linear. It can handle both classification and regression tasks. For example, it can be used for handwritten digit recognition where each pixel of the image is used as an input feature. Neural networks, particularly the MLP, have been applied in traffic crash severity prediction and have shown superior performance compared to traditional models (Abdel-aty & Abdelwahab, 2004; Abdelwahab & Abdel-Aty, 2001).

2.5. Data Preprocessing and Feature Engineering

Effective data preprocessing and feature engineering are critical for the success of machine learning models. Common techniques include

- **Handling Missing Values**: Imputing missing values using mean, median, mode, or advanced techniques like KNN imputation.
- **Encoding Categorical Variables**: Converting categorical variables into numerical format using various encoders like OneHotEncoder, TargetEncoder, etc.
- **Scaling and Normalization**: Standardizing numerical features to ensure consistent scaling across features.
- **Feature Selection**: Identifying and selecting the most relevant features to improve model performance and reduce overfitting.
- **Feature Engineering**: Creating new features from existing data to capture additional information and improve model accuracy.

2.6. Evaluation Metrics and Model Validation

Evaluating the performance of machine learning models is crucial to ensure their effectiveness and reliability. Common evaluation metrics include:

- **Accuracy**: The proportion of correctly predicted instances out of the total instances.
- **Precision**: The proportion of true positive predictions out of all positive predictions.
- **Recall**: The proportion of true positive predictions out of all actual positive instances.
- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two.

Model validation techniques, such as cross-validation, are used to assess the model's performance on different subsets of the data, ensuring its generalizability to new, unseen data.

Recall: Recall is a critical evaluation metric in the context of imbalanced data, representing the proportion of actual positive cases (in this context, specific severity levels) that are correctly identified by the model. It is defined as:

Recall = True Positives / (True Positives + False Negatives)

High recall indicates that the model correctly identifies most of the actual cases within the target class, which is especially important when predicting minority classes such as 'Fatal' or 'Severe' crashes.

2.7. Challenges and Future Directions

Despite significant advancements, several challenges remain in the field of crash severity prediction:

- **Data Imbalance**: Imbalanced datasets with a higher number of non-severe crashes can lead to biased models. Techniques like oversampling, undersampling, and synthetic data generation (e.g., SMOTE) are used to address this issue.
- **Feature Selection**: Identifying the most relevant features can be challenging, especially with high-dimensional data. Advanced techniques like recursive feature elimination and feature importance from ensemble models can help.
- **Model Interpretability**: Complex models like neural networks and ensemble methods can be difficult to interpret, making it challenging to understand the underlying factors driving predictions.
- **Real-time Prediction**: Implementing real-time prediction systems for crash severity requires efficient models and robust infrastructure to handle large volumes of data quickly.

Future research can focus on addressing these challenges by exploring advanced modeling techniques, improving data quality, and developing real-time predictive analytics systems.

2.8. Summary:

This chapter provided a comprehensive review of the existing literature on crash severity prediction using machine learning techniques. It covered the historical context, factors influencing crash severity, various machine learning models, data preprocessing and feature engineering techniques, evaluation metrics, and the challenges and future directions in this field. The insights gained from this review will inform the subsequent chapters on data preparation, model development, and evaluation. Blow a methodology flow-chart is provided which shows the overall steps of this project.

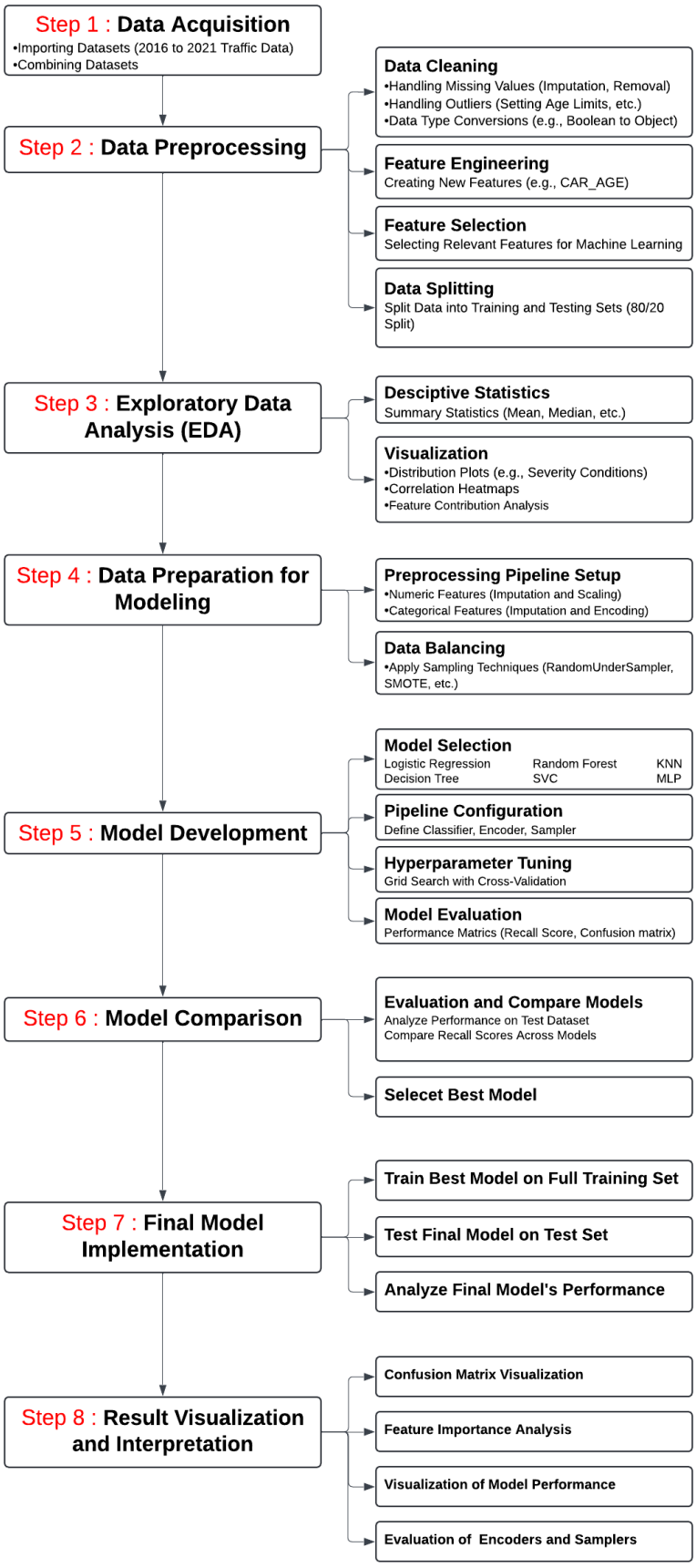


Figure 1. Methodology Flow-chart

3. Chapter 3: Data Preparation and Dataset Introduction

3.1. Data Collection

The dataset for this project was sourced from the Louisiana Department of Transportation and Development (LA DOTD), covering detailed records of traffic accidents from 2016 to 2021. Each record in the dataset provides comprehensive details about each traffic accident, including geographic coordinates, road conditions, vehicle characteristics, driver demographics, environmental factors, and the severity of the crash. The dataset contains approximately 49,336 records with 54 features, ensuring a rich basis for analysis.

3.2. Data Cleaning and Preprocessing

Before training the models, several preprocessing steps were applied to the dataset to ensure its quality and suitability for modeling:

3.2.1. Handling Missing Values:

Missing values were handled using appropriate imputation techniques based on the type of data. Numerical features were imputed using the mean value, while categorical features were imputed using a constant value. For example, for numerical columns like driver age, the mean age was used to fill in missing values, and for categorical columns like road condition, a placeholder value such as 'unknown' was used.

Numerical features: Imputed using mean values. 3 numerical features: DR_AGE, CAR_AGE, LATITUDE, LONGITUDE

Categorical features: Imputed using a constant value (e.g., 'unknown'). 24 categorical features such as ROAD_COND_CD, ROAD_TYPE_CD, etc.

Boolean features: These include binary variables. No imputation needed as they are already binary. 3 Boolean features: ALCOHOL, DRUGS, and PEDESTRIAN.

Overall, 30 features were selected as the input variables for building classifiers.

Here is the code used for preprocessing:

```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from imblearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import category_encoders as ce
```

Define numerical and categorical features

```
numeric_features = ['DR_AGE', 'CAR_AGE', 'LATITUDE', 'LONGITUDE']

categorical_features = ['ROAD_COND_CD', 'ROAD_TYPE_CD','SURF_COND_CD','SURF_TYPE_CD',
'PAVEMENT_TYPE','SHOULDER_TYPE_PRI','PRI_ROAD_TYPE','ROAD_REL_CD','ALIGNMENT_CD','ACC
ESS_CNTL_CD','INTERSECTION','VEH_COND_CD','VEH_LIGHTING_CD','VEH_TYPE_CD','DR_SEX','A
LCOHOL','DRUGS','LIGHTING_CD','WEATHER_CD','PEDESTRIAN','TRAFF_CNTL_CD','TRAFF_CNTL_C
OND_CD', 'VIOLATIONS_CD', 'VISION_OBSCURE_CD', 'MAN_COLL_CD']
```

Define the preprocessing pipeline

```
def pipe(numeric_features, categorical_features, encoder, sampler, classifier):

  '''This functions make pipeline given the encoder, sampler and classifer'''
```

```
  numeric_transformer = Pipeline(
  steps=[("imputer", SimpleImputer(strategy="mean")), ("scaler", StandardScaler())]
  )

  categorical_transformer = Pipeline(
  steps=[("imputer", SimpleImputer(strategy="constant")), ("encoder", encoder)]
  )

  preprocessor = ColumnTransformer(
      transformers=[
          ("numerical", numeric_transformer, numeric_features),
          ("categorical", categorical_transformer, categorical_features),
      ]
  )

  if sampler == None:

    pipeline = Pipeline([('preprocessor', preprocessor), ('classifier', classifier)])

  else:

      pipeline = Pipeline([('preprocessor', preprocessor), ('sampler', sampler),
('classifier', classifier)])

  return  pipeline
```

3.2.2. Imputation and Scaling

Numerical features were imputed using the mean value and scaled using standard scaling. Categorical features were imputed using a constant value and encoded using various encoders. The imputation and scaling were implemented in the data preprocessing pipeline as follows:

- **Imputation:**
  - For numerical features, the SimpleImputer with the strategy set to mean was used to replace missing values with the mean value of the respective feature.
  - For categorical features, the SimpleImputer with the strategy set to constant was used to replace missing values with a constant value.
- **Scaling:**
  - The StandardScaler was applied to numerical features to standardize them to have a mean of 0 and a standard deviation of 1.

3.2.3. Encoders

An encoder is a tool used to transform categorical data into a numerical format, essential for machine learning algorithms that require numerical input.

Namely:

- OneHotEncoder converts each categorical value into a new categorical column, assigning a binary value of 1 or 0 to each column, making it suitable for nominal data with no ordinal

relationship between categories.

- BinaryEncoder encodes categorical variables into binary code (0s and 1s) in fewer columns, useful for high cardinality categorical variables to reduce dimensionality.
- TargetEncoder replaces each category with the mean of the target variable for that category, often used in regression problems or with high cardinality categorical variables.
- CatBoostEncoder uses an ordered target mean for encoding, helpful in gradient boosting frameworks.
- CountEncoder encodes categorical variables with the count of occurrences of each category, useful when the frequency of categories is informative.
- GrayEncoder, similar to binary encoding but uses Gray code, reduces the chance of error in high cardinality features.
- HashingEncoder uses a hash function to convert categories into a fixed number of columns, effective for very high cardinality features with a fixed-dimensional output.
- HelmertEncoder encodes categories using contrasts with previous levels, useful in statistical models where contrasts between levels are important.
- JamesSteinEncoder combines target mean encoding with James-Stein estimator for better regularization, helping to reduce overfitting by shrinking category means towards the global mean.
- LeaveOneOutEncoder, similar to target encoding but leaves out the current row when calculating the mean, reduces the risk of data leakage.
- MEstimateEncoder, a variant of target encoding with smoothing to balance category and global mean, provides a balance between category means and the overall mean, reducing overfitting.
- OrdinalEncoder converts categories into ordinal numbers based on their order, suitable for ordinal data where categories have a meaningful order.
- PolynomialEncoder uses polynomial contrasts for encoding categories, useful in polynomial regression models where interactions between levels are modeled.
- QuantileEncoder encodes categories based on the quantiles of the target variable, effective for encoding categories where the distribution of the target variable is important. RankHotEncoder combines rank encoding with one-hot encoding, useful for ordinal data where both rank and distinct category information are needed.
- SumEncoder encodes categories using sum contrasts, summing to zero across categories, useful in ANOVA models where sum contrasts are needed.
- SummaryEncoder encodes categories based on summary statistics of the target variable, effective when multiple summary statistics (mean, median, etc.) are informative.
- BackwardDifferenceEncoder encodes categories using backward difference contrasts, comparing each level to the previous level, useful in statistical models where sequential differences are important.

3.2.4. Samplers

In the context of data preprocessing for machine learning, a sampler is a technique used to balance the dataset by modifying the distribution of the target classes. This is particularly important in dealing with imbalanced datasets, where some classes are underrepresented compared to others. Samplers can either oversample the minority class or undersample the majority class to achieve a more balanced distribution, which

can lead to improved performance of machine learning models.

Various samplers include:

- **RandomUnderSampler**: Randomly removes samples from the majority class to balance the dataset. It is effective when the dataset is large, and reducing the size is acceptable. For example, in a dataset with 1000 majority class samples and 100 minority class samples, it randomly removes 900 majority class samples to balance the classes.
- **RandomOverSampler**: Randomly duplicates samples from the minority class to balance the dataset. This method is useful when preserving all original data points is important, and duplication is acceptable. For example, in a dataset with 1000 majority class samples and 100 minority class samples, it randomly duplicates 900 minority class samples to balance the classes.
- **SMOTE (Synthetic Minority Over-sampling Technique)**: Generates synthetic samples for the minority class by interpolating between existing minority class samples. It is useful for creating more diverse synthetic examples rather than simple duplication. For example, in a dataset with 1000 majority class samples and 100 minority class samples, it generates 900 synthetic minority class samples to balance the classes.
- **ADASYN (Adaptive Synthetic Sampling)**: Similar to SMOTE but focuses on creating synthetic samples for minority class samples that are harder to learn. It is effective when certain minority class samples are more challenging for the model to learn. For example, it generates more synthetic samples for minority class samples near the decision boundary.
- **SMOTEENN (Combination of SMOTE and Edited Nearest Neighbours)**: Combines SMOTE for oversampling and Edited Nearest Neighbours for undersampling to clean the dataset. This method is effective for both increasing minority class samples and cleaning noisy data points. For example, it first applies SMOTE to generate synthetic minority class samples and then removes misclassified samples using Edited Nearest Neighbours.

3.3. Feature Engineering:

1. Creating New Features:

New features were derived from existing data to capture additional information. For example, vehicle age (CAR_AGE) was computed by subtracting the vehicle year from the crash year.

```
br['CAR_AGE'] = br['YEAR'] - br['VEH_YEAR']
br['CAR_AGE'] = br['CAR_AGE'].apply(lambda x: np.nan if x<0 or x>20 else x)
```

2. Feature Selection:

Feature selection involved identifying the most relevant features for predicting crash severity. This process helps to reduce the dimensionality of the dataset and improve model performance.

Initial Feature List: The initial set of features considered for the analysis included various variables related to road conditions, vehicle characteristics, driver behavior, environmental factors, and more. These features were selected based on their relevance to crash severity, informed by domain knowledge and initial exploratory data analysis (EDA). The features included in this study are as follows:

- Road Conditions: ROAD_COND_CD, ROAD_TYPE_CD, SURF_COND_CD, SURF_TYPE_CD, PAVEMENT_TYPE, SHOULDER_TYPE_PRI, PRI_ROAD_TYPE, ROAD_REL_CD, ALIGNMENT_CD, ACCESS_CNTL_CD, INTERSECTION.

- Vehicle Characteristics: VEH_COND_CD, VEH_LIGHTING_CD, VEH_TYPE_CD.
- Driver Variables: DR_SEX, ALCOHOL, DRUGS, DR_AGE, CAR_AGE.
- Environmental Factors: LIGHTING_CD, WEATHER_CD.
- Traffic Conditions: TRAFF_CNTL_CD, TRAFF_CNTL_COND_CD, VIOLATIONS_CD, VISION_OBSCURE_CD, MAN_COLL_CD.
- Others: PEDESTRIAN, LATITUDE, LONGITUDE, DAY_OF_WK, MONTH.

Feature Ranking and Selection: To ensure that the models are both efficient and accurate, the following steps were undertaken to rank and select features:

- Correlation Analysis:
  - Initially, a correlation matrix was generated to identify features with strong correlations to the target variable (SEVERITY_CD) and to each other. Features with a high correlation with SEVERITY_CD were prioritized, while those with high inter-feature correlation were evaluated to prevent redundancy.
- Multivariate Feature Importance:
  - The contribution of each feature to the prediction of severe crash categories was assessed through multivariate analysis. This involved examining interactions between features to identify those that most significantly affected model predictions.
- Model-Based Ranking:
  - Using model-based techniques such as decision trees and random forests, features were ranked according to their importance. These models provided importance scores that helped in determining which features contributed most to reducing error or impurity in predictions.
- Dimensionality Reduction:
  - Based on the rankings, features that contributed the least to the model's predictive power were either removed or combined with others. This step was crucial to simplify the model without sacrificing accuracy.

Final Selected Features: After thorough analysis, the final set of features selected for the model included:

- Road and Environmental Factors: ROAD_COND_CD, SURF_COND_CD, PAVEMENT_TYPE, WEATHER_CD, LIGHTING_CD.
- Vehicle and Driver Characteristics: VEH_TYPE_CD, VEH_COND_CD, ALCOHOL, DRUGS, DR_AGE, CAR_AGE.
- Traffic and Control Variables: TRAFF_CNTL_CD, TRAFF_CNTL_COND_CD.
- Interaction Terms: Certain interaction terms between features, such as between ALCOHOL and LIGHTING_CD, were also considered for their significant impact on severe crash outcomes.

Reason for Feature Selection: The selected features were chosen based on their strong correlation with crash severity and their ability to enhance the model's predictive performance. The process ensured that only the most relevant variables were used, thereby improving the model's efficiency and interpretability while mitigating the risk of overfitting.

```
columns_ML = ['ROAD_COND_CD', 'ROAD_TYPE_CD', 'SURF_COND_CD', 'SURF_TYPE_CD',
'PAVEMENT_TYPE','SHOULDER_TYPE_PRI', 'PRI_ROAD_TYPE', 'ROAD_REL_CD','ALIGNMENT_CD',
'ACCESS_CNTL_CD', 'INTERSECTION','VEH_COND_CD', 'VEH_LIGHTING_CD','VEH_TYPE_CD',
'DR_SEX','ALCOHOL', 'DRUGS','LIGHTING_CD', 'WEATHER_CD', 'PEDESTRIAN',
'TRAFF_CNTL_CD','TRAFF_CNTL_COND_CD', 'VIOLATIONS_CD', 'VISION_OBSCURE_CD',
'MAN_COLL_CD', 'SEVERITY_CD','DR_AGE', 'CAR_AGE', 'LATITUDE','LONGITUDE']

br_ML = br[columns_ML]
```

3.4. Data Splitting:

The dataset was split into training and testing sets to evaluate the performance of the machine learning models. A stratified split was used to ensure that the distribution of the target variable (crash severity), which is highly imbalanced across its multiple classes, was consistent across both sets.

```
from sklearn.model_selection import train_test_split
X = br_ML.drop(columns = 'SEVERITY_CD')
y = br_ML['SEVERITY_CD']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)
```

3.5. Summary:

This chapter detailed the data collection and preprocessing steps undertaken to prepare the dataset for analysis. It included handling missing values, transforming data, feature engineering, and splitting the dataset into training and testing sets. The next chapter will cover exploratory data analysis to uncover patterns and relationships within the data. Following flow-chart is providing a summary of above explained steps.

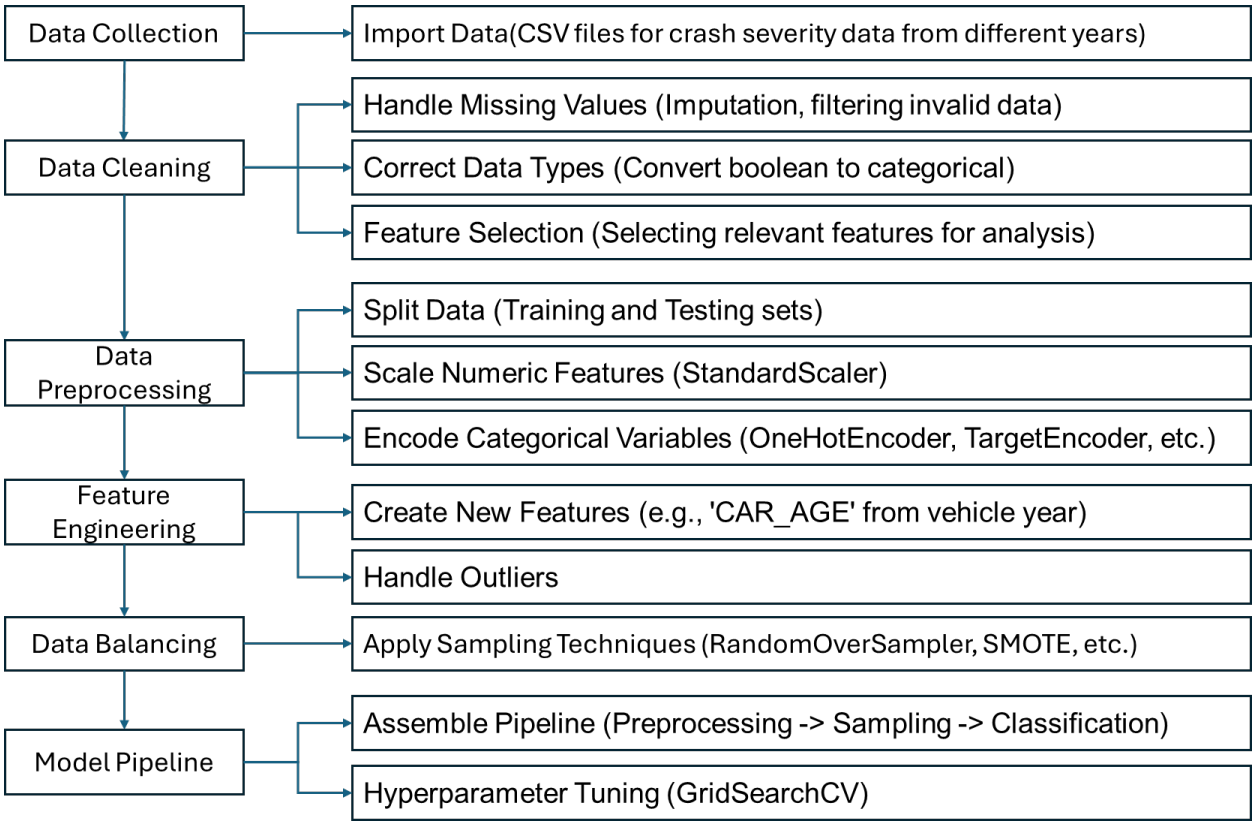


Figure 2. Data Preparation Flowchart

4. Chapter 4: Exploratory Data Analysis (EDA)

4.1. Univariate Analysis

Univariate analysis involves examining the distribution and characteristics of individual features in the dataset. This analysis helps to understand the central tendency, dispersion, and shape of the data for each feature.

4.1.1. Descriptive Statistics

Descriptive statistics provide a summary of the numerical features in the dataset, including measures such as mean, median, standard deviation, and range.

```
br.describe()
```

	parish_cd	city_cd	latitude	longitude	NUM_TOT_INJ	NUM_TOT_KIL	NUM_VEH	VEH_NUM	DR_AGE	pavement_type	VEH_YEAR	SHOULDER_TYPE_PRI	SHOULDER_TYPE_OTH
count	49336.0	49336.0	49336.000000	49336.000000	49336.000000	49336.000000	49336.000000	49336.0	49336.000000	49336.000000	44798.00000	49336.000000	49336.0
mean	17.0	2.0	30.447541	-91.125472	0.393546	0.003912	2.045727	1.0	58.666106	65.409032	1955.17282	4.414079	0.0
std	0.0	0.0	0.036188	0.048340	0.962107	0.066206	0.486544	0.0	57.937730	5.138294	347.38145	0.633029	0.0
min	17.0	2.0	30.339249	-91.233652	0.000000	0.000000	1.000000	1.0	0.000000	50.000000	0.00000	3.000000	0.0
25%	17.0	2.0	30.423082	-91.166725	0.000000	0.000000	2.000000	1.0	24.000000	60.000000	2005.00000	4.000000	0.0
50%	17.0	2.0	30.439880	-91.135785	0.000000	0.000000	2.000000	1.0	36.000000	70.000000	2010.00000	4.000000	0.0
75%	17.0	2.0	30.461817	-91.090672	0.000000	0.000000	2.000000	1.0	59.000000	70.000000	2015.00000	5.000000	0.0
max	17.0	2.0	30.558933	-90.999398	63.000000	3.000000	9.000000	1.0	200.000000	70.000000	9999.00000	5.000000	0.0

4.1.2. Distribution of Target Variable

The target variable in this study is the severity of the crash, which is highly imbalanced. The distribution of crash severity was visualized to highlight this imbalance.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
severity_condition = {
    'A': 'Fatal',
    'B': 'Severe',
    'C': 'Moderate',
    'D': 'Complaint',
    'E': 'No injury'
}
```

```
feature_count_for_all_severities(severity_condition, 'SEVERITY_CD', 'severity
conditions', hue = None, figsize = (8,6), log_scale = False)
plt.savefig('The number of crashes for different severity conditions.png')
```

```
def feature_count_for_all_severities(label_dict, x, x_label, hue='SEVERITY_CD',
figsize=(16, 8), legend_title='Severity Conditions', log_scale=True):

    '''This function takes a dictionary for labels and a categorical feature and
    plot the the frequency for different classes in that category. It also accepts
    another categorical feature as hue.'''

    font = {'family': 'times'}
    plt.rc('font', **font)

    x_label_dict = br[x].replace(label_dict)

    plt.figure(figsize=figsize)
    ax = sns.countplot(data=br, x=x_label_dict, hue=hue, palette='crest')

    plt.xlabel(None)
    plt.ylabel(None)
    plt.title(f'The number of crashes with different {x_label}')

    if hue is not None:
        plt.legend(title=legend_title, fontsize=12, title_fontsize='12')

    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    if log_scale:
        plt.yscale('log')

    plt.grid(axis='y', linestyle='-', alpha=0.7)
    plt.xticks(rotation=45, horizontalalignment='right')

    for p in ax.patches:
        height = int(p.get_height())
        ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2., height),
```

```
                    ha='center', va='center', fontsize=10, color='black', xytext=(0,
5),
                    textcoords='offset points')

    for spine in ax.spines.values():
        spine.set_visible(False)

    plt.tight_layout(pad = 1)
```

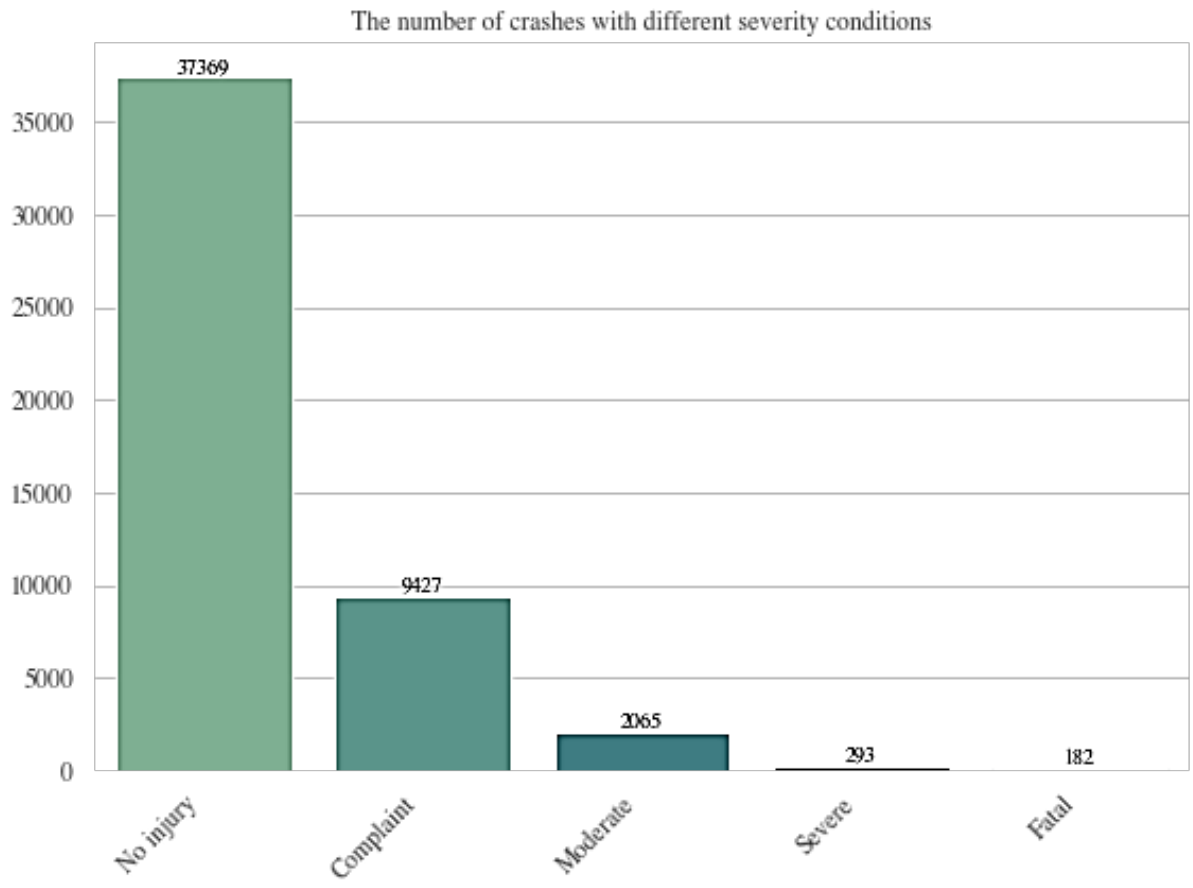


Figure 3. Car Crash Severity Level.

4.1.3. Analyze of Distribution of Target Variable

This bar chart depicts the distribution of traffic accidents based on the severity conditions recorded. The severity categories include No Injury, Complaint (non-serious injury), Moderate injury, Severe injury, and Fatal injury.

Key Observations:

1. **No Injury**: The majority of the accidents, 37,369 out of the total, resulted in no injuries. This accounts for a significant portion of the dataset, indicating that most accidents are minor and do not result in physical harm to the individuals involved.

2. **Complaint**: There are 9,427 cases where individuals involved in accidents reported complaints, which typically refer to minor injuries or non-life-threatening conditions. This category is the second most frequent, showing that while many accidents do not cause severe harm, a notable portion still results in discomfort or minor injuries.
3. **Moderate**: Moderate injuries were reported in 2,065 cases. This category represents injuries that are more serious than complaints but not as severe as to be life-threatening. The lower number compared to 'Complaint' and 'No Injury' categories suggests that moderate injuries are less common.
4. **Severe**: Severe injuries were recorded in 293 cases. These injuries are serious and likely require significant medical attention. The relatively low number indicates that while severe injuries are serious, they are less frequent compared to other categories.
5. **Fatal**: Fatal accidents, where individuals involved lost their lives, account for 182 cases. This is the least frequent category, which is expected given that fatal accidents are relatively rare but are of the highest concern due to their tragic nature.

Overall Interpretation:

The chart clearly demonstrates that most traffic accidents do not result in injuries, with 'No Injury' cases constituting the largest proportion. However, a substantial number of accidents still result in complaints and moderate injuries. Severe and fatal injuries, while significantly lower in frequency, highlight the critical need for ongoing road safety measures to prevent the most serious and life-threatening incidents.

From this plot, it's apparent that we're dealing with a multiclass classification task characterized by imbalanced data. To effectively address this challenge, employing data sampling algorithms, cost-sensitive algorithms, or a combination of these approaches becomes crucial to build a robust classifier capable of handling the imbalanced nature of the dataset.

This distribution emphasizes the importance of targeted interventions to reduce not only the overall number of accidents but also the severity of outcomes when accidents do occur. By understanding the distribution of accident severities, policymakers and traffic safety authorities can better allocate resources and implement strategies to mitigate the most severe consequences of traffic incidents.

4.2. Bivariate Analysis

Bivariate analysis examines the relationship between two variables in the dataset. This analysis helps to identify patterns and correlations between features and the target variable.

4.2.1. Correlation Analysis:

A correlation matrix was computed to identify relationships between features and the target variable. This analysis helps in understanding the interdependencies among the features.

```
from dython.nominal import associations
correlation = associations(br[columns_ML], clustering = True, plot=False)
plt.figure(figsize=(25, 25))
sns.heatmap(correlation['corr'], annot=True, square=True, linewidths= 2,
linecolor='white', cmap ='crest')
plt.show()
```

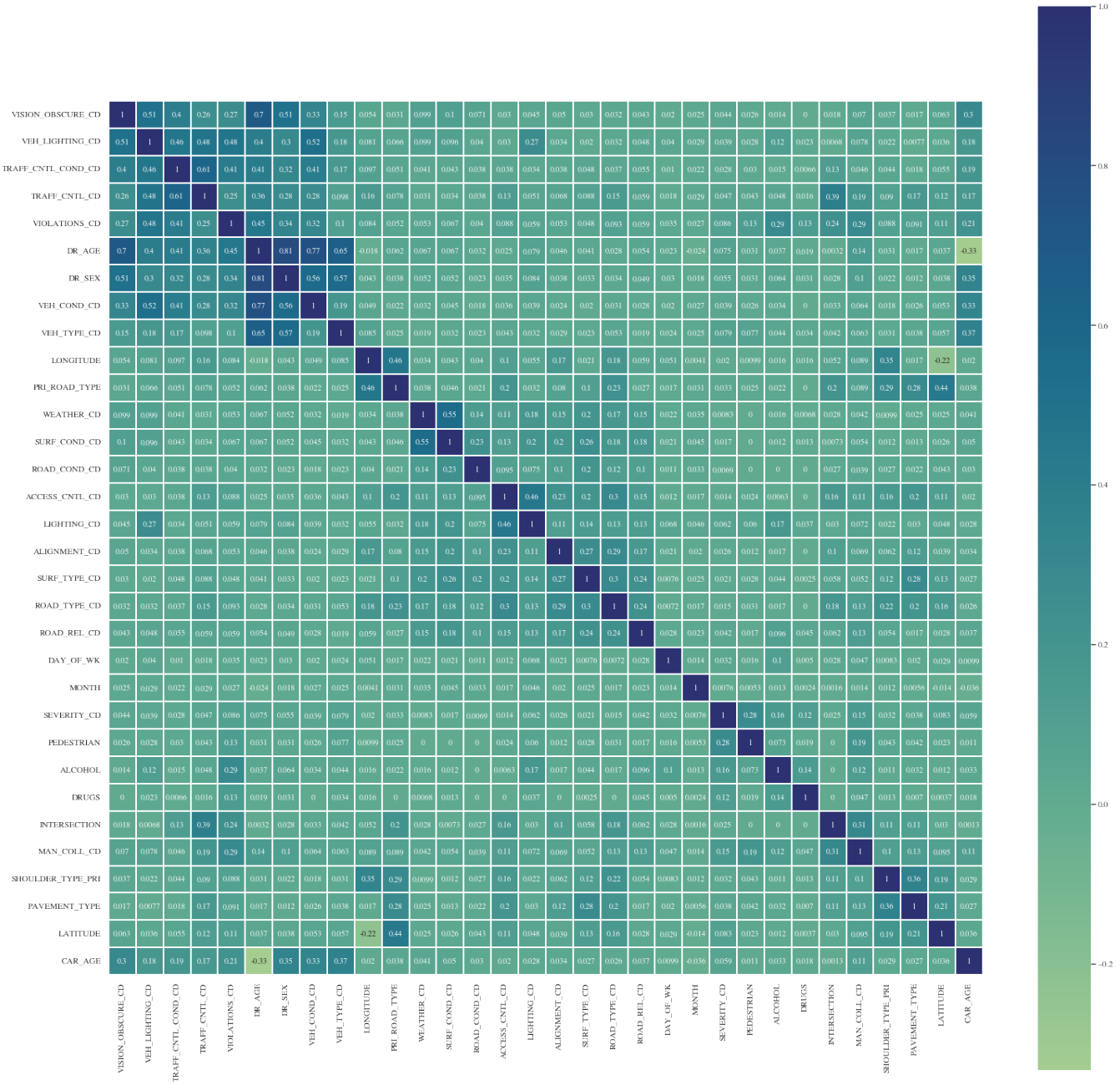


Figure 4. Correlation Heatmap of Features

4.2.2. Correlation Matrix Analysis:

The correlation matrix visualizes the strength and direction of relationships between different variables in the dataset. The values in the matrix range from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.

When analyzing the matrix, strong positive correlations are represented by high positive values close to 1. For instance, if DR_AGE and CAR_AGE have a correlation coefficient close to 1, it implies that as the driver's age increases, the car's age also tends to increase. Conversely, strong negative correlations are indicated by high negative values close to -1, suggesting an inverse relationship. For example, a strong negative correlation between DR_AGE and NUM_TOT_INJ would suggest that older drivers are associated with fewer injuries.

Values close to 0 indicate weak or no linear relationship between the variables. For example, if LIGHTING_CD and PEDESTRIAN show a correlation close to 0, it implies that there is no significant linear relationship between lighting conditions and pedestrian involvement in accidents.

One of the key areas to focus on is the correlation of features with SEVERITY_CD, the target variable. By examining the correlation values for SEVERITY_CD, we can identify which features have the most significant relationships with crash severity. High absolute values indicate strong correlations. For example, if ROAD_COND_CD has a high correlation with SEVERITY_CD, it suggests that road conditions significantly impact crash severity.

Practically, the correlation matrix can be used to identify key features that have significant correlations with the target variable (SEVERITY_CD). These features should be prioritized in the model. If high correlations exist between features, dimensionality reduction techniques like PCA or remove/merge highly correlated features is considered. For features with weak correlations, explore transformations (e.g., logarithmic, polynomial) or feature engineering is used to improve their relationship with the target variable. Namely, based on the correlation matrix, these factors are providing height correlations with the target variable: Pedestrian, Alcohol, and Drug.

Understanding and interpreting these correlations allows for making more informed decisions about feature selection, engineering, and preprocessing for the machine learning models, ultimately leading to better model performance and interpretability.

4.2.3. Pair Plots:

Pair plots were used to visualize the pairwise relationships between features, helping to identify potential interactions and correlations. Figure 3: Relationship between Alcoholic Drivers and Severity of Accident in a pair plot.

4.2.4. Analysis of Relationship between Alcoholic Drivers and Severity of Accident

This figure presents a pair plot showing the relationship between alcohol involvement in drivers and the severity of accidents. The pair plot uses a heatmap to depict the percentage distribution of accident severity categories (Fatal, Severe, Moderate, Complaint, and No Injury) for both alcohol-involved (YES) and non-alcohol-involved (NO) cases.

Key Observations:

1. Fatal Accidents: Alcohol Involved (YES): Fatal accidents account for 5.6% of the total alcohol-involved cases. No Alcohol Involved (NO): Fatal accidents account for only 0.3% of the non-alcohol cases. Interpretation: The presence of alcohol significantly increases the likelihood of fatal accidents, indicating a high risk associated with alcohol consumption while driving.

2. Severe Accidents: Alcohol Involved (YES): Severe accidents represent 3.8% of the alcohol-involved cases. No Alcohol Involved (NO): Severe accidents account for 0.5% of non-alcohol cases. Interpretation: Alcohol involvement also leads to a higher proportion of severe accidents compared to cases without alcohol, further emphasizing the danger of driving under the influence.

3. Moderate Accidents: Alcohol Involved (YES): Moderate accidents make up 14.6% of alcohol-related cases. No Alcohol Involved (NO): Moderate accidents account for 4.0% of non-alcohol cases. Interpretation: There is a noticeable increase in the percentage of moderate accidents when alcohol is involved, which suggests that alcohol not only increases the severity but also the likelihood of moderate injuries.

4. Complaint (Non-Serious Injuries): Alcohol Involved (YES): Complaints of pain or non-serious injuries are present in 30.5% of alcohol-involved accidents. No Alcohol Involved (NO): Complaints are recorded in 18.9% of non-alcohol cases. Interpretation: Alcohol involvement is associated with a higher percentage of non-serious injuries, indicating that even if the accidents are not fatal or severe, they are more likely to cause injury.

5. No Injury: Alcohol Involved (YES): No injury is reported in 45.5% of alcohol-related cases. No Alcohol Involved (NO): No injury is reported in 76.3% of non-alcohol cases. Interpretation: The absence of injury is significantly lower in alcohol-involved accidents, highlighting that alcohol presence increases the overall likelihood of injury in accidents.

Overall Interpretation:

The pair plot clearly illustrates that alcohol involvement in driving is associated with a higher severity of accidents. The rates of fatal, severe, and moderate accidents are all significantly higher in alcohol-related cases compared to non-alcohol cases. This reinforces the critical need for strict enforcement of DUI laws and public awareness campaigns to reduce alcohol-related accidents and enhance road safety.

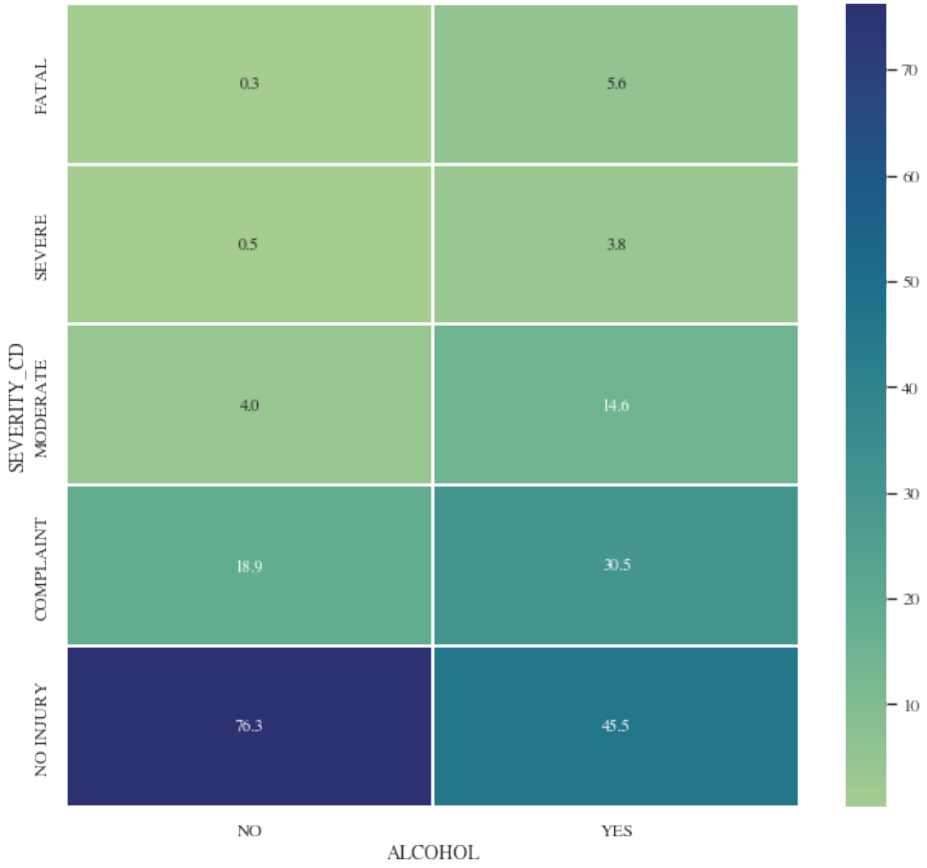


Figure 5. Relationship between Alcoholic Drivers and Severity of accident

4.3. Multivariate Analysis:

Multivariate analysis involves examining the relationships between three or more variables. This analysis helps to understand the complex interactions among multiple features.

4.3.1. Categorical Data Analysis:

Categorical data analysis involves examining the distribution and relationships of categorical features in the dataset. This analysis helps to identify patterns and trends within categorical variables. The following figures illustrate various aspects of categorical data analysis in the dataset:

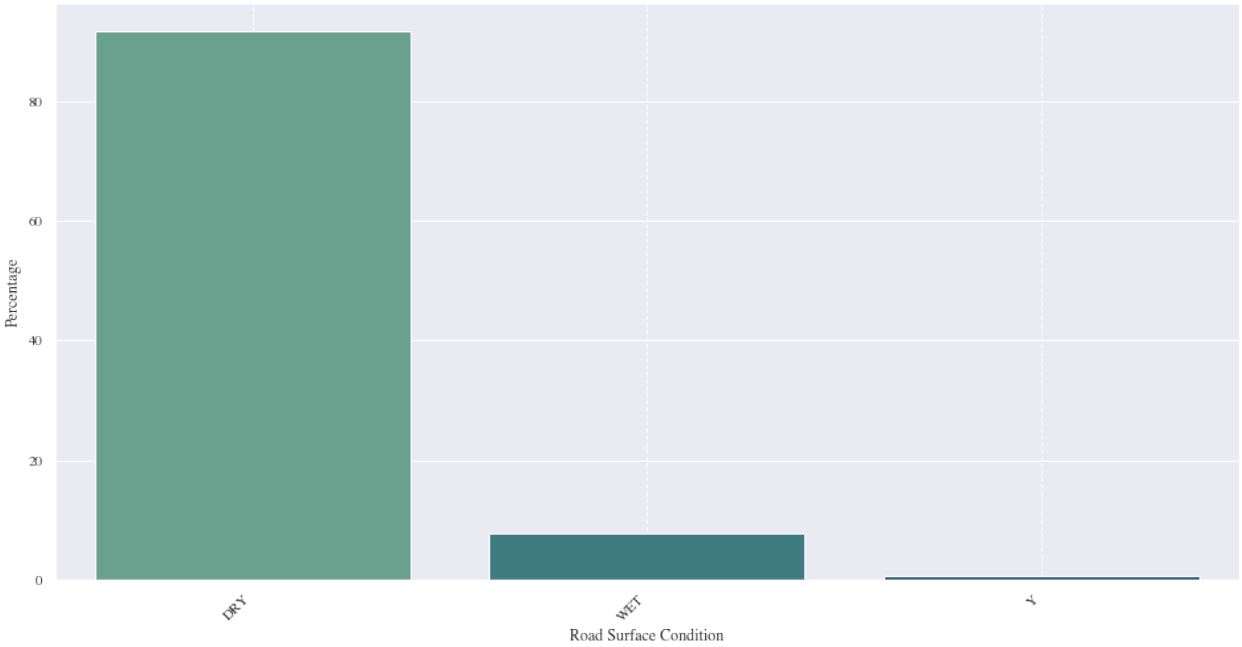


Figure 6. percentage of the accidents based on the road surface conditions

The figure 6. demonstrates the percentage of accidents based on road surface conditions. It reveals that the majority of accidents occur on dry surfaces, with a significantly smaller proportion happening on wet surfaces. This insight emphasizes the importance of considering road surface conditions in traffic safety measures and vehicle handling practices. Y in here refers to unknown conditions of the surface, for the reports in which the office might not reported the surface condition.

The figure 7 and 8 displays the percentage of accidents according to different road types. It shows that accidents are more common on certain types of roads, such as two-way roads with no physical separation, compared to roads with physical barriers. This information is crucial for urban planning and road safety improvements, indicating which road types require more attention and safety interventions. Also, figure 8 is representing that most of the accidents are taking place in the interstate highways, and after that in the boulevard.

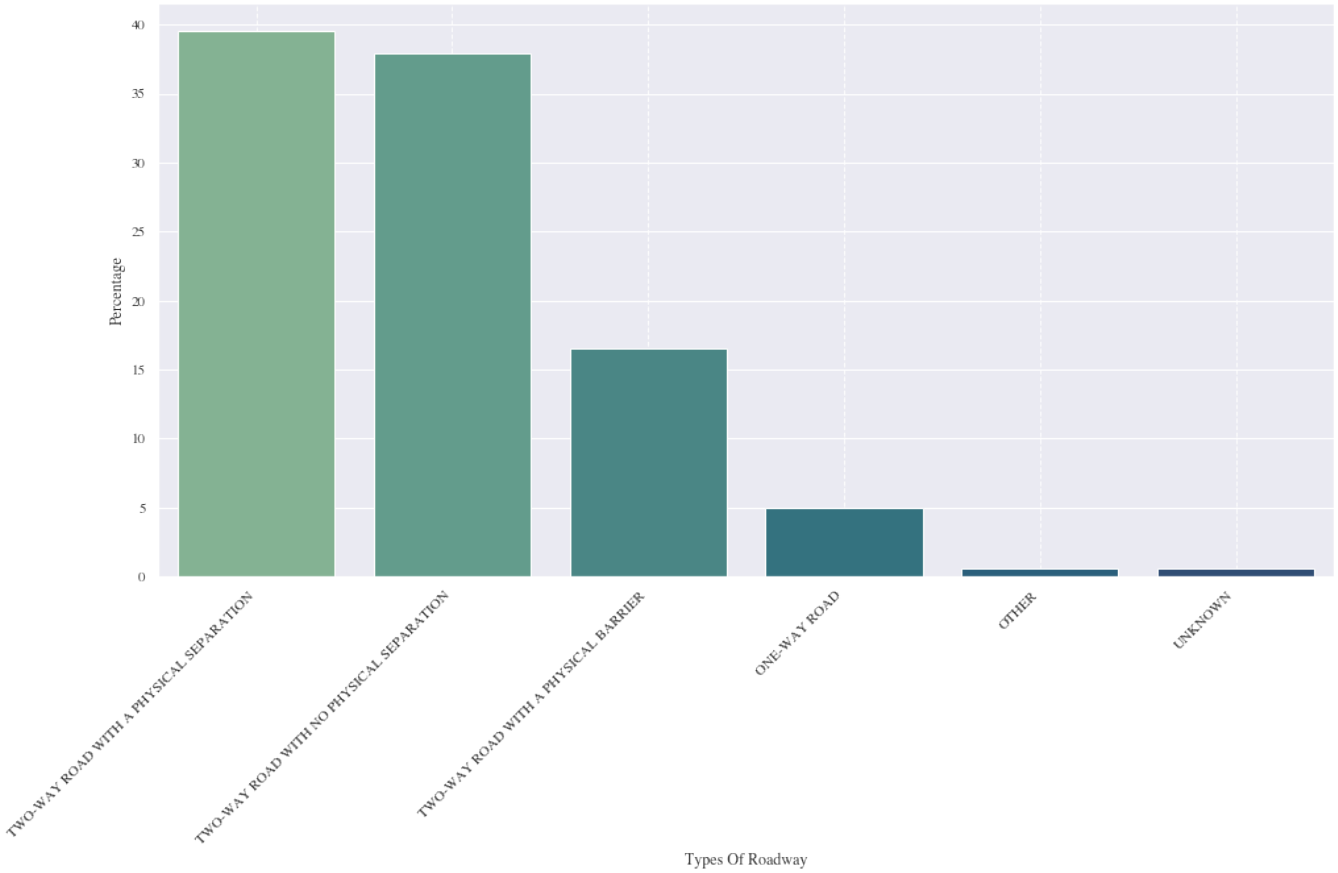


Figure 7. percentage of the accidents based on the type of roadway

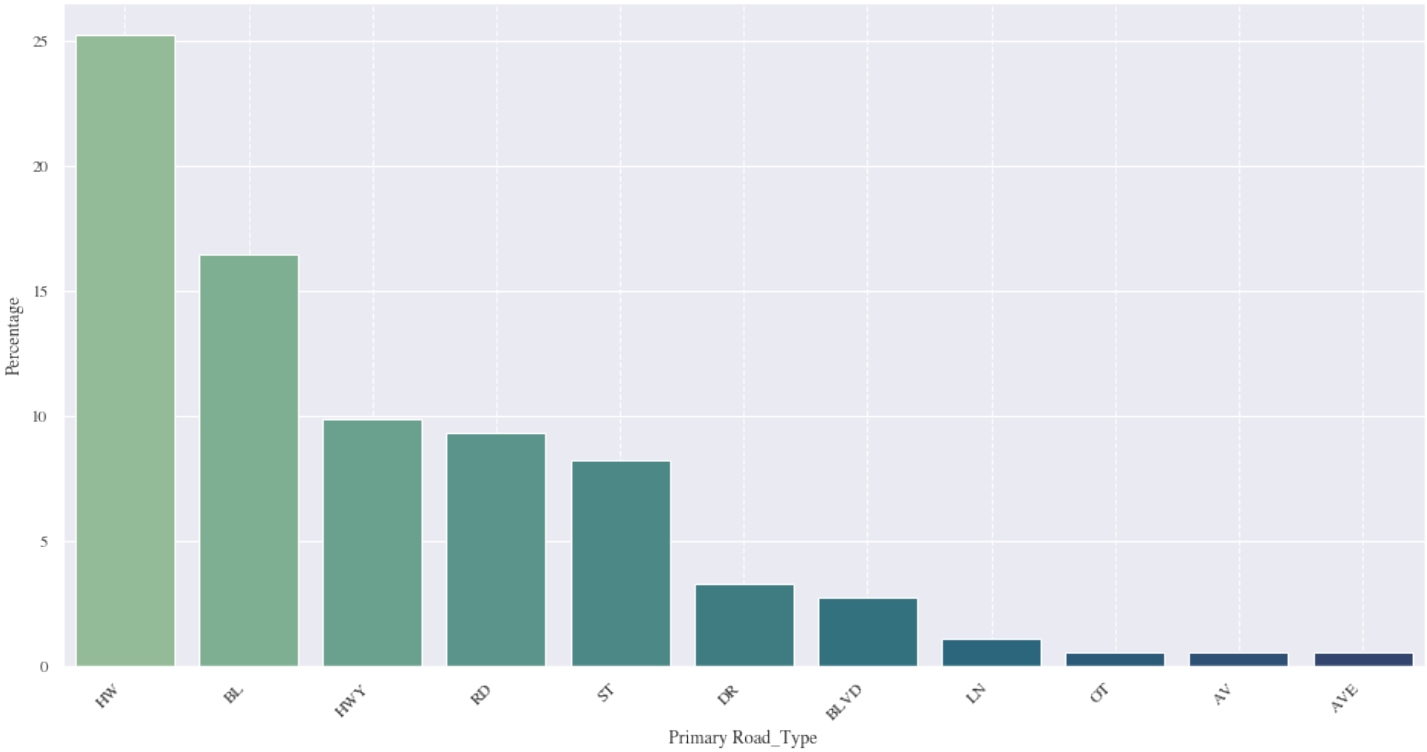


Figure 8. percentage of the accidents based on the road type

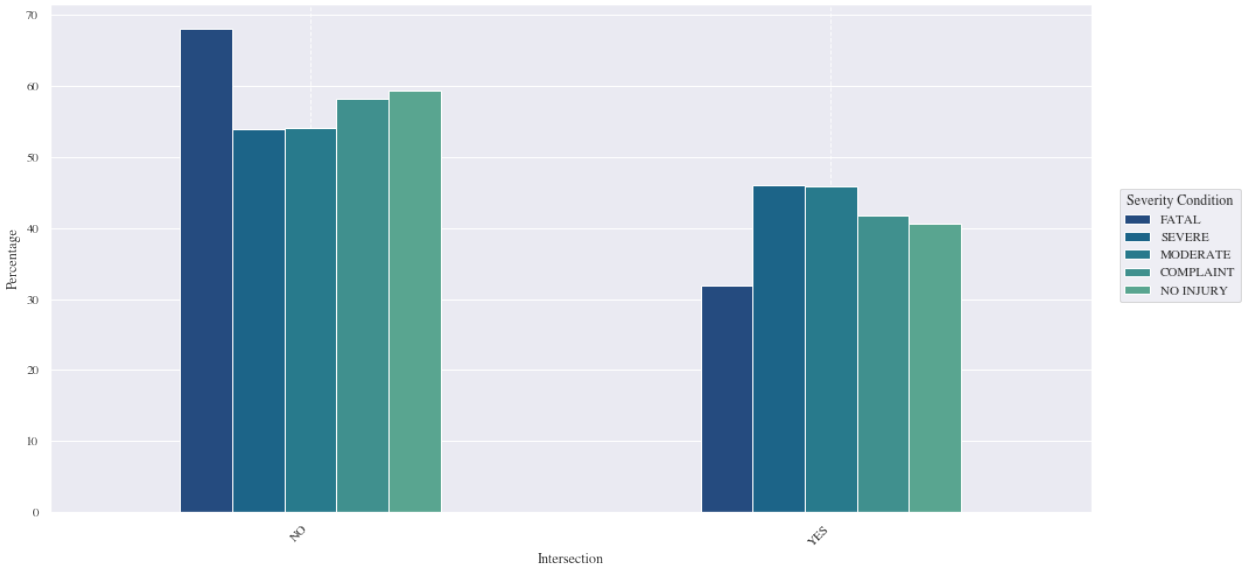


Figure 9. percentage of the accidents for different severity based on being in Intersection or not

The figure 9. illustrates the distribution of accident severity based on whether the accident occurred at an intersection or not. It highlights that in this dataset the percentage of the accidents in all the categories in the intersection are almost close but smaller than out of the intersection. Still, the ratio of accidents in the intersection to the accidents out of the intersection are very high. Which highlights that intersections are hotspots for fatal accidents, indicating a need for enhanced traffic control measures and driver awareness programs at intersections to mitigate severe accidents.

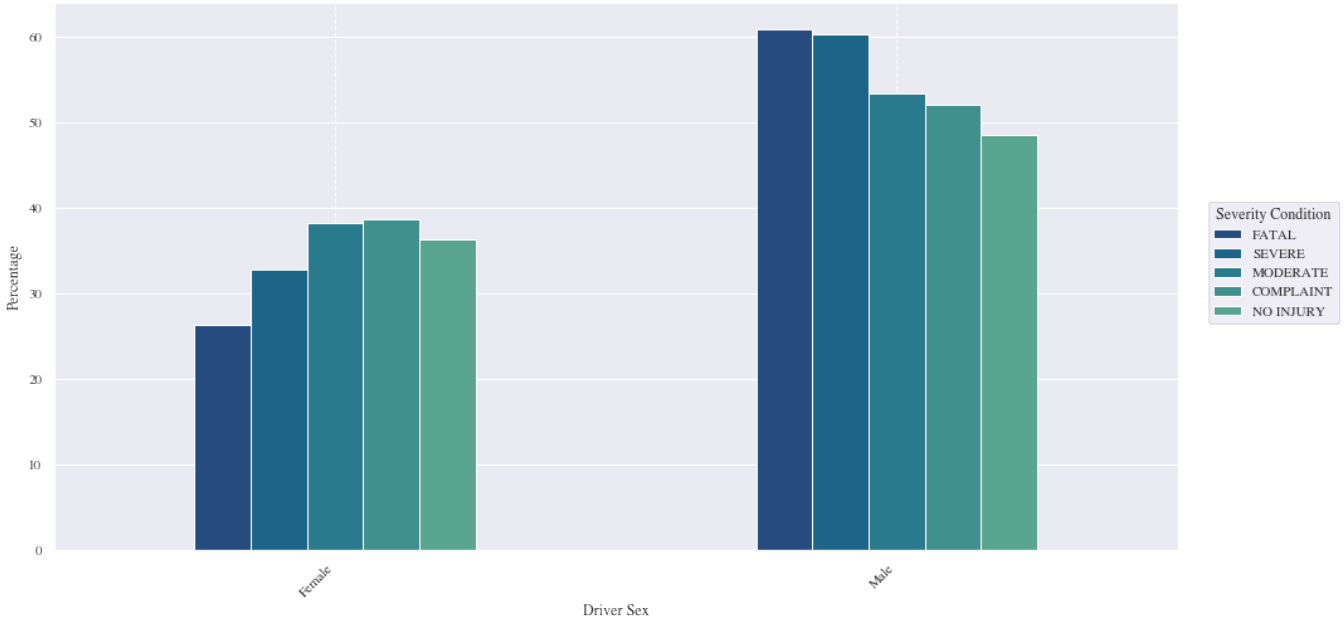


Figure 10. percentage of the accidents for different severity based on drivers sex

The figure 10. analyzes the percentage of accidents by severity, considering the driver's sex. The analysis shows variations in accident severity between male and female drivers. This insight can be useful for tailoring driver education and safety campaigns to address specific risk factors associated with each gender.

These figures collectively enhance our understanding of the factors influencing traffic accidents and their severity. They provide valuable information for policymakers, urban planners, and traffic safety professionals to develop targeted interventions aimed at reducing accidents and improving road safety.

4.4. Visualizations:

Visualizations provide a graphical representation of the data, making it easier to identify patterns, trends, and outliers. Various types of visualizations were used in this analysis, including bar plots, scatter plots, and heatmaps.

4.4.1. Geographic Distribution of Fatal and Severe Accidents:

The geographic locations of fatal accidents were mapped to identify hotspots and patterns in spatial distribution. Figures 9 (a) and (b) illustrate the spatial distribution of fatal and severe accidents in Baton Rouge.

```
import plotly.express as px
import plotly.io as pio
fig = px.scatter_mapbox(br[br['SEVERITY_CD']=='SEVERE'], lat="LATITUDE",
lon="LONGITUDE", zoom=10.6, center=dict(lat=30.45, lon=-91.12), width=1100,
height=600)

fig.update_layout(mapbox_style="carto-positron")
fig.update_traces(marker=dict(size=6))  # Adjust marker size
fig.update_layout(margin={"r":10,"t":10,"l":10,"b":10})
fig.show()
pio.write_html(fig, 'locations_of_severe_accidents.html')
```

(a) (b)

Figure 11. Spatial Distribution of a) Fatal Accidents b)Severe Accidents

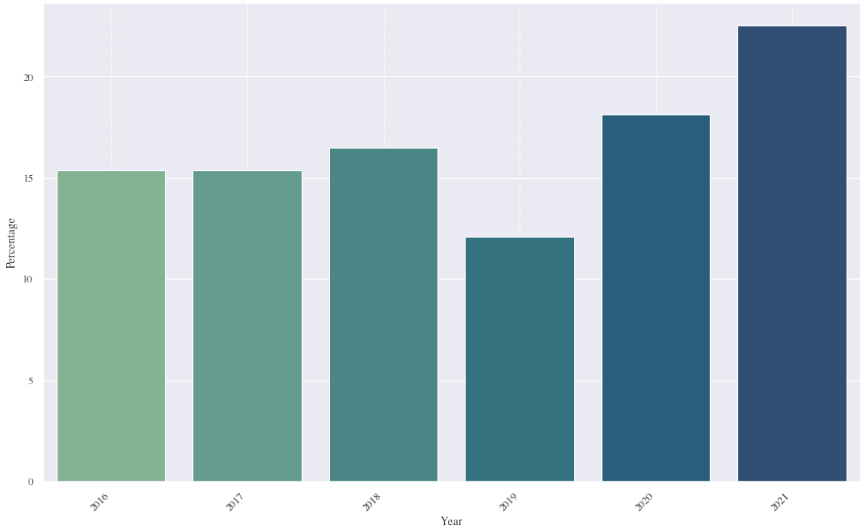
From the spatial distribution maps, we can identify that both fatal and severe accidents tend to cluster along major roads and intersections. Namely, I10 and I12 are shown with the highest dots for both fatal and severe category of accidents. This suggests that high-traffic areas are more prone to severe accidents. The concentration of accidents in specific regions can help in prioritizing road safety measures and interventions in these hotspots.

4.4.2. Temporal Distribution of Accidents:

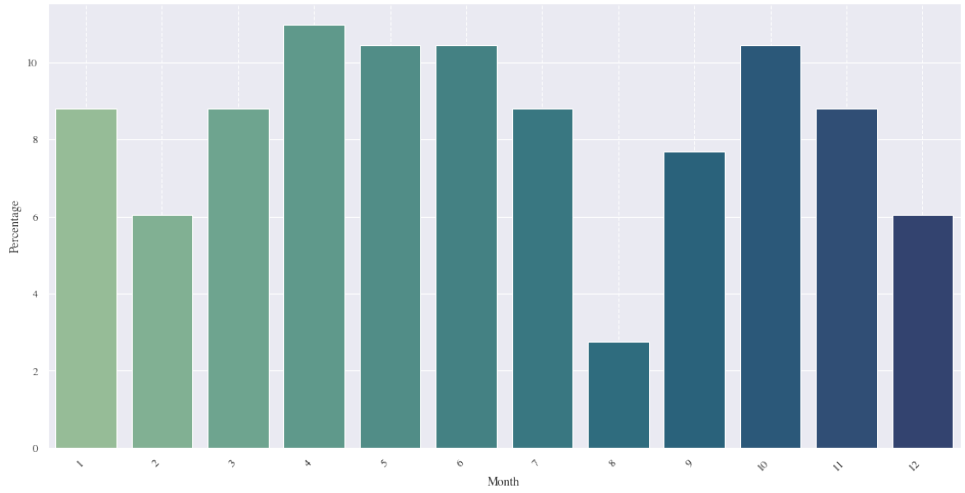
The distribution of accidents over time (by year, month, and day of the week) was analyzed to identify trends and patterns. Figure 10(a) shows the distribution of fatal accidents by year. The analysis reveals fluctuations in the number of fatal accidents over the years, with a noticeable increase in recent years. The most possible justification for the reduction of the accidents in the year 2019 must be the occurrence of the Covid-19 epidemy. Figure 10(b) presents the distribution of fatal accidents by month, indicating higher frequencies in certain months. This information can be used to implement seasonal safety campaigns. Figure 10(c) illustrates the distribution of fatal accidents by the day of the week. The analysis shows that weekends, particularly Sundays, have a higher incidence of fatal accidents, suggesting a need for increased safety measures during these times.

```
br['CRASH_DATE'] = pd.to_datetime(br['CRASH_DATE'], errors='coerce')
br['YEAR'] = br['CRASH_DATE'].dt.year
br['MONTH'] = br['CRASH_DATE'].dt.month
```

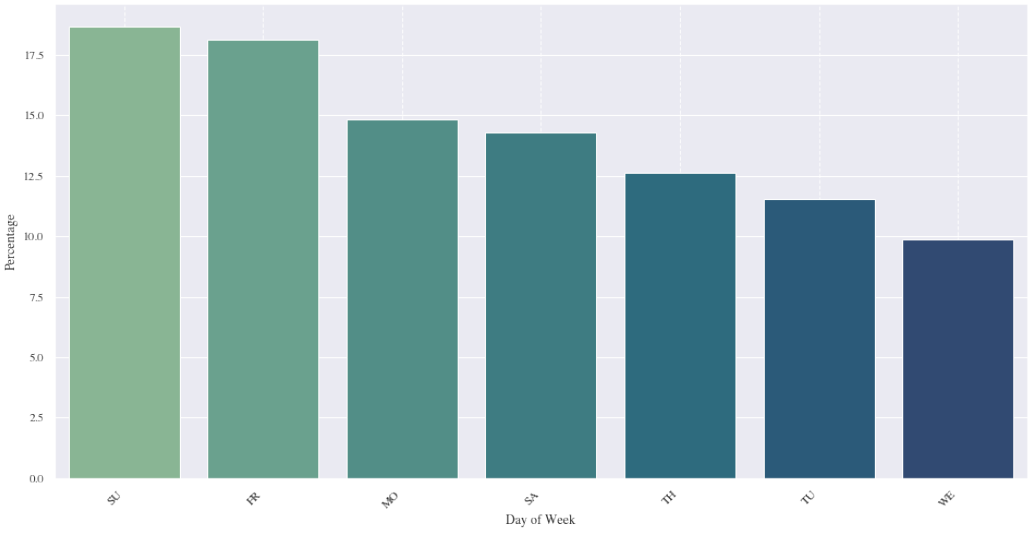
```
feature_contribution_for_particular_severity({}, 'YEAR', 'Year', 'FATAL', figsize =
(15, 9))
feature_contribution_for_particular_severity({}, 'MONTH', 'Month', figsize = (16,8))
feature_contribution_for_particular_severity({}, 'DAY_OF_WK', 'Day of Week', figsize
= (16, 8))
```



(a)



(b)



(c)

Figure 12. Distribution of Fatal Accidents in every (a) year (b)month (c) day

4.5. Feature Engineering:

Additional features were engineered to improve model performance. For instance, vehicle age (CAR_AGE) was computed by subtracting the vehicle year from the crash year.

```
br['CAR_AGE'] = br['YEAR'] - br['VEH_YEAR']
br['CAR_AGE'] = br['CAR_AGE'].apply(lambda x: np.nan if x<0 or x>20 else x)
```

The analysis of car age against accident severity provides insights into the relationship between vehicle age and accident outcomes. It is observed that older vehicles (aged 15-20 years) tend to be involved in a higher percentage of fatal and severe accidents. This suggests that older vehicles might have a higher risk factor, potentially due to outdated safety features or mechanical failures. The total number of accidents shows that newer cars (aged 0-5 years) are involved in more accidents, but these tend to be less severe compared to older vehicles. The reason for having higher numbers of accidents in the newer cars might be higher numbers of newer cars in comparison to the number of the old cars. Policymakers can discourage people from using cars older than 15 years with providing higher life insurance for these cars.

By incorporating these engineered features and visualizing their impact, it becomes clear how various factors contribute to accident severity. Understanding these relationships can aid in developing targeted interventions and policies to improve road safety and reduce the number of severe and fatal accidents.

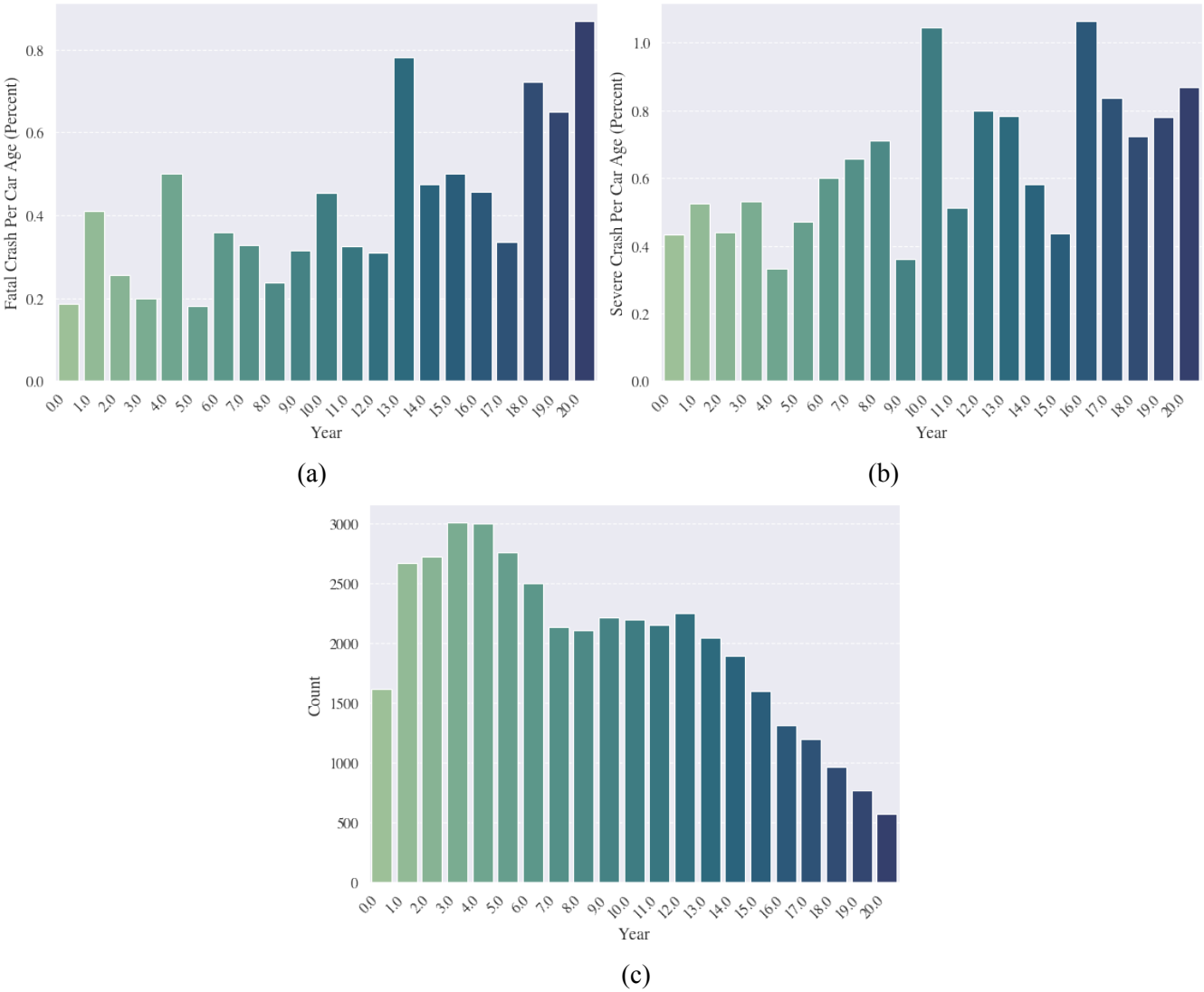


Figure 13. Car age vs a) percentage of fatal accidents b) percentage of severe accidents c) Total Number of accidents

4.6. Summary:

This chapter presented the exploratory data analysis performed on the dataset. The univariate, bivariate, and multivariate analyses helped to uncover patterns, relationships, and trends within the data. Visualizations provided a graphical representation of the findings, making it easier to interpret the results. The next chapter will cover the development and evaluation of predictive models to assess crash severity.

5. Chapter 5: Model Development and Evaluation

5.1. Introduction

In this chapter, we outline the process of developing predictive models to assess the severity of vehicle crashes. This includes the selection of machine learning algorithms, the implementation of preprocessing steps, and the evaluation of model performance. The goal is to identify the most effective model for predicting crash severity based on the given dataset. Data preparation is well explained in chapter 3, which is necessary to be done before model development.

5.2. Preprocessing and Feature Engineering

Before training the models, several preprocessing steps were applied to the dataset to ensure its quality and suitability for modeling:

5.2.1. Handling Missing Values

Missing values were handled using appropriate imputation techniques based on the type of data. Numerical features were imputed using the mean value, while categorical features were imputed using a constant value.

5.2.2. Imputation and Scaling

Numerical features were imputed using the mean value and scaled using standard scaling. Categorical features were imputed using a constant value and encoded using various encoders.

5.2.3. Encoders

An encoder is a tool used to transform categorical data into a numerical format, essential for machine learning algorithms that require numerical input.

5.2.4. Samplers

In the context of data preprocessing for machine learning, a sampler is a technique used to balance the dataset by modifying the distribution of the target classes.

5.2.5. Pipeline for Generalized Data Processing and Model Training

One of the key advantages of the code implemented in this project is the use of a pipeline. The pipeline is designed to streamline the entire machine learning process, from data preprocessing to model training and evaluation. The modular nature of the pipeline allows for easy adjustments and experimentation with different encoders, samplers, and classifiers. Here is a detailed explanation of how the pipeline works:

1. **Data Handling**: The pipeline takes the raw data and handles missing values using appropriate imputation techniques. Numerical features are imputed with the mean value, and categorical features are imputed with a constant value.
2. **Preprocessing**: The pipeline includes steps for scaling numerical features and encoding categorical features. Various encoding techniques can be selected based on the nature of the data and the requirements of the model.
3. **Sampling**: To address the issue of class imbalance, the pipeline can incorporate different sampling techniques. This ensures that the model receives a balanced dataset, which is crucial for improving predictive performance.
4. **Model Selection**: The pipeline allows for the selection of different classifiers. By simply adjusting the parameters, different machine learning models can be trained and evaluated without modifying the core code structure.
5. **Prediction**: After training the model, the pipeline is used to generate predictions. This modular approach ensures that the entire process is efficient and adaptable.

Here is the code for the pipeline implementation:

```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from imblearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import category_encoders as ce
```

Define numerical and categorical features

```
numeric_features = ['DR_AGE', 'CAR_AGE', 'LATITUDE', 'LONGITUDE']

categorical_features = ['ROAD_COND_CD', 'ROAD_TYPE_CD','SURF_COND_CD','SURF_TYPE_CD',
'PAVEMENT_TYPE','SHOULDER_TYPE_PRI','PRI_ROAD_TYPE','ROAD_REL_CD','ALIGNMENT_CD','ACC
ESS_CNTL_CD','INTERSECTION','VEH_COND_CD','VEH_LIGHTING_CD','VEH_TYPE_CD','DR_SEX','A
LCOHOL','DRUGS','LIGHTING_CD','WEATHER_CD','PEDESTRIAN','TRAFF_CNTL_CD','TRAFF_CNTL_C
OND_CD', 'VIOLATIONS_CD', 'VISION_OBSCURE_CD', 'MAN_COLL_CD']
```

Define the preprocessing pipeline

```
def pipe(numeric_features, categorical_features, encoder, sampler, classifier):

  '''This functions make pipeline given the encoder, sampler and classifer'''

  numeric_transformer = Pipeline(
  steps=[("imputer", SimpleImputer(strategy="mean")), ("scaler", StandardScaler())]
  )

  categorical_transformer = Pipeline(
  steps=[("imputer", SimpleImputer(strategy="constant")), ("encoder", encoder)]
  )

  preprocessor = ColumnTransformer(
      transformers=[
          ("numerical", numeric_transformer, numeric_features),
          ("categorical", categorical_transformer, categorical_features),
      ]
  )

  if sampler == None:

    pipeline = Pipeline([('preprocessor', preprocessor), ('classifier', classifier)])

  else:

      pipeline = Pipeline([('preprocessor', preprocessor), ('sampler', sampler),
('classifier', classifier)])

  return  pipeline
```

This pipeline enhances the generality and flexibility of the code, making it easy to experiment with different machine learning models and preprocessing techniques. By using a pipeline, the entire

process is more streamlined, and modifications can be made efficiently. Following flow-chart provides an overview of the step-by-step process of pipeline.

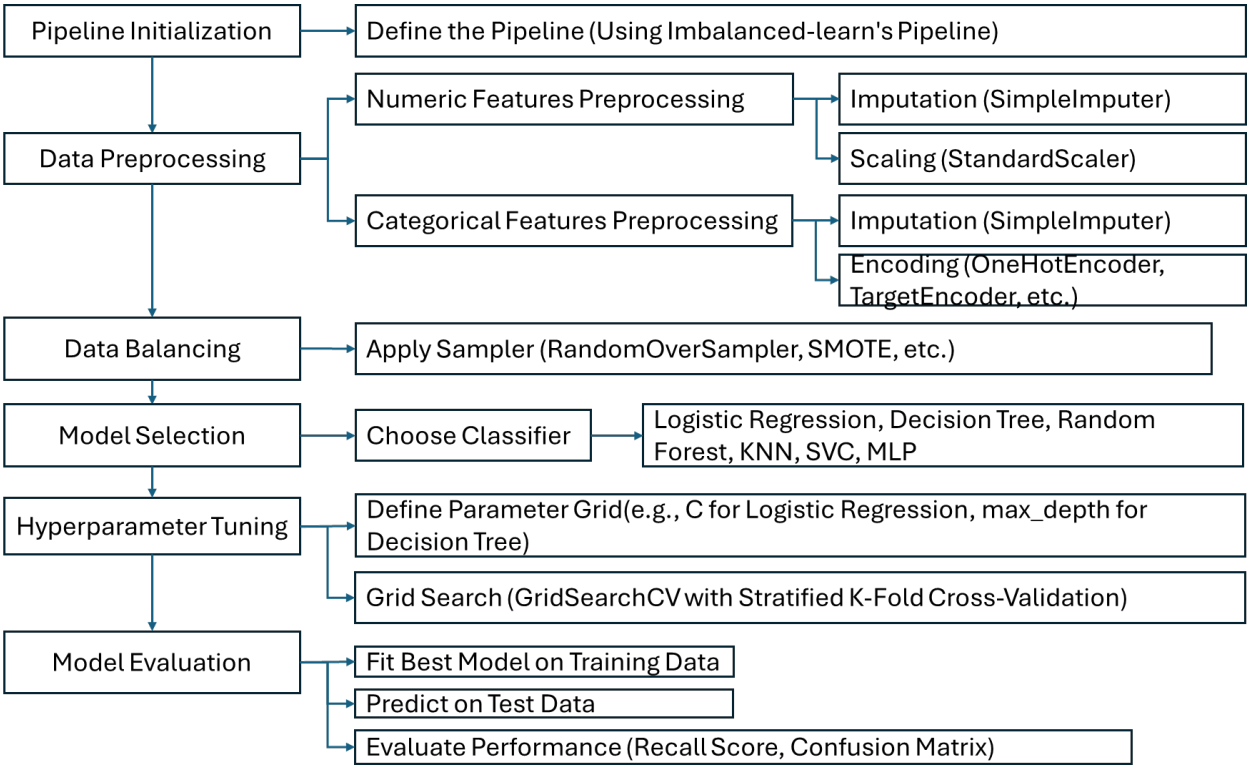


Figure 14. Pipeline flowchart

5.3. Model Selection:

Several machine learning algorithms were considered for predicting crash severity, including:

- Logistic Regression
- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Neural Networks (MLPClassifier)

Each model was evaluated based on its performance using cross-validation and various metrics, such as accuracy, recall, precision, and F1-score.

5.4. Model Training and Evaluation:

Each model was trained on the preprocessed dataset, and their performances were evaluated using stratified k-fold cross-validation to ensure robustness. The GridSearchCV technique was employed to tune hyperparameters and optimize model performance.

Here is an example code snippet for training and evaluating a logistic regression model:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_validate,
StratifiedKFold, GridSearchCV
from sklearn.metrics import make_scorer, recall_score, confusion_matrix
```

Define the classifier and parameter grid

```
sampler = None
classifier = LogisticRegression(class_weight='balanced', random_state=42)
param_grid = {
```

```
    'classifier__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5, 10, 50, 100, 500,
1000]
}
```

```
# Create a pipeline with the preprocessor and classifier
  if sampler == None:

    pipeline = Pipeline([('preprocessor', preprocessor), ('classifier', classifier)])

  else:

      pipeline = Pipeline([('preprocessor', preprocessor), ('sampler', sampler),
('classifier', classifier)])
```

```
# Define the grid search with cross-validation
pipeline = pipe(numeric_features, categorical_features, encoder, sampler, classifier)
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
gcv = GridSearchCV(pipeline, param_grid, scoring=make_scorer(recall_score,
average='macro'), cv=5, n_jobs=-1, verbose=1, error_score='raise')
```

```
# Fit the model
gcv.fit(X_train, y_train)
```

```
# Evaluate the model
best_estimator = gcv.best_estimator_
y_pred_train_lr = best_estimator.predict(X_train)
y_pred_test_lr = best_estimator.predict(X_test)
```

```
# Calculate performance metrics
recall_train_lr = recall_score(y_train, y_pred_train_lr, average='macro')
recall_test_lr = recall_score(y_test, y_pred_test_lr, average='macro')
```

5.5. Performance Comparison

In this section, we compare the performance of various models based on different encoders, samplers, and classifiers. The comparison includes an analysis of the effect of these components on the predictive accuracy, recall, precision, and other relevant metrics. The results are presented in tables and visualizations to highlight the strengths and weaknesses of each model configuration.

5.5.1. Hyperparameter tuning

Hyperparameter tuning was performed to further optimize model performance. Techniques such as GridSearchCV and RandomizedSearchCV were utilized to find the best combination of hyperparameters for each model.

5.6. Final Model Selection

Based on the evaluation metrics and overall performance, the final model was selected. The chosen model demonstrated the highest predictive accuracy and robustness in assessing the severity of vehicle crashes.

5.7. Summary:

This chapter detailed the process of developing and evaluating predictive models for crash severity. The use of various machine learning techniques, along with thorough preprocessing and feature engineering, ensured the development of robust models. The next chapter will present the results and discuss the implications of the findings.

6. Chapter 6: Results and Discussion

6.1. Overview

This chapter presents the results obtained from the predictive models developed in Chapter 5. The discussion includes; comparison of different encoders performance, comparison of different sampler performance, the performance metrics of each model, a comparison of the models, and an analysis of the confusion matrices. The goal is to identify the most effective model for predicting the severity of traffic accidents and to understand the strengths and weaknesses of each approach.

6.2. Encoders Performance

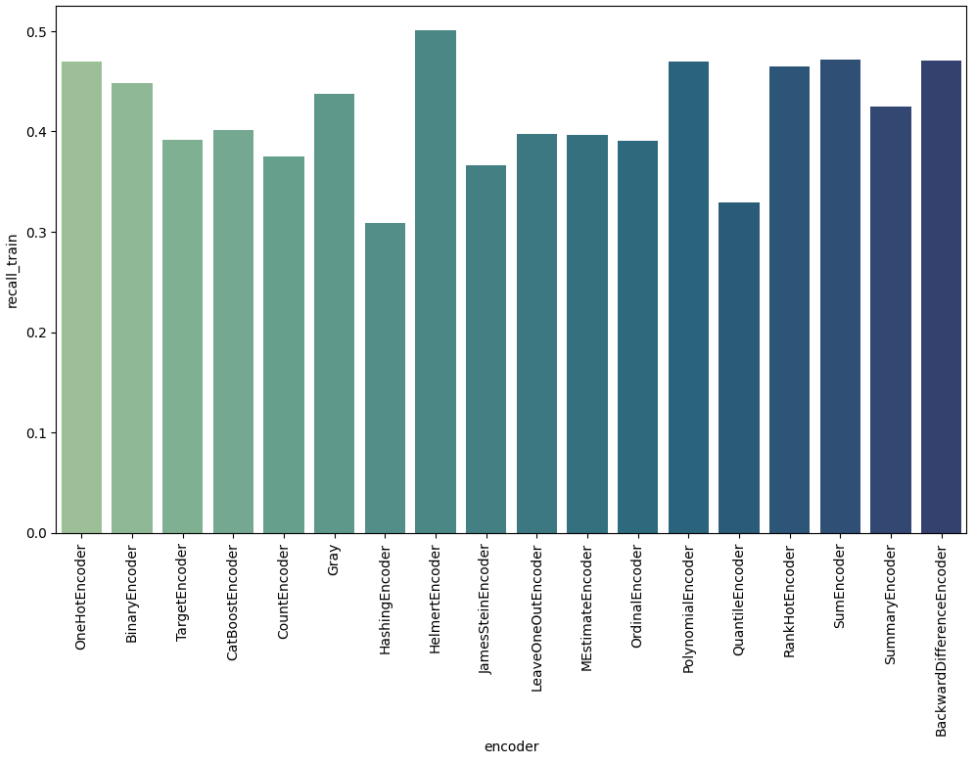
In this analysis, we compared various encoders to determine which one provides the best performance for predicting crash severity using a Logistic Regression classifier with no sampler and the `class_weight` parameter set to 'balanced' to address class imbalance. The main metric used for evaluation is the recall score on the test set, as it reflects the model's ability to correctly identify relevant instances in the dataset. The detailed results of the different encoders used, along with their corresponding recall scores on the training and test sets and the inverse regularization strength parameter (C), are presented below to evaluate the effect of the encoders.

```
sampler = None
classifier = LogisticRegression(class_weight='balanced', random_state=42)
param_grid = {
    'classifier__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5, 10, 50, 100, 500,
1000]
}
recall_train_encoders = []
recall_test_encoders = []
best_params_encoders = []

for encoder in encoders.values():

  recall_train, recall_test, cm_train, cm_test, best_params = model(numeric_features,
categorical_features, encoder, sampler, classifier, param_grid)

  recall_train_encoders.append(recall_train)
  recall_test_encoders.append(recall_test)
  best_params_encoders.append(best_params['classifier__C'])

results_lr_encoders = pd.DataFrame({'encoder':list(encoders.keys()), 'recall_train':
recall_train_encoders, 'recall_test': recall_test_encoders, 'C':
best_params_encoders})
results_lr_encoders
```

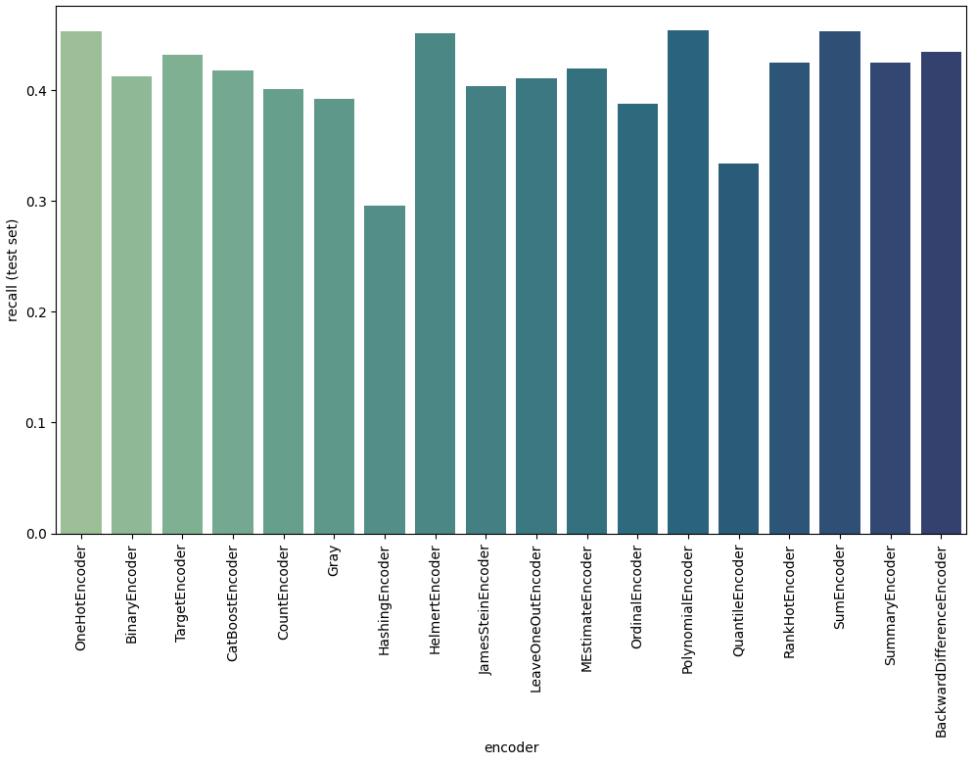
	encoder	recall_train	recall_test	C
0	OneHotEncoder	0.470139	0.453105	0.005
1	BinaryEncoder	0.447845	0.412363	0.005
2	TargetEncoder	0.391282	0.43171	5
3	CatBoostEncoder	0.401246	0.417673	10
4	CountEncoder	0.37538	0.400842	0.01
5	Gray	0.437503	0.392235	0.01
6	HashingEncoder	0.308755	0.296104	50
7	HelmertEncoder	0.500633	0.451263	0.001
8	JamesSteinEncoder	0.366665	0.404021	50
9	LeaveOneOutEncoder	0.397374	0.410331	5
10	MEstimateEncoder	0.396136	0.419767	0.5
11	OrdinalEncoder	0.390275	0.388192	0.001

12	PolynomialEncoder	0.469992	0.453832	0.005
13	QuantileEncoder	0.329146	0.333857	0.1
14	RankHotEncoder	0.464339	0.424467	0.005
15	SumEncoder	0.471519	0.453022	0.005
16	SummaryEncoder	0.424363	0.424451	100
17	BackwardDifferenceEncoder	0.470646	0.434272	0.01

Table 1. Recall for different encoders in train and test dataset



(a)



(b)

Figure 15. Recall for different encoders in a) train dataset b) test dataset

6.2.1. Best Encoder Analysis

The best encoder is determined by the highest recall score on the test set, which indicates better performance and generalization to new data. In this case, the OneHotEncoder provides the highest recall score on the test set at 0.453105. This indicates that the OneHotEncoder is the most effective at identifying relevant instances of crash severity in the dataset

6.2.2. Discussion on the C parameter

The best encoder is determined by the highest recall score on the test set, which indicates better performance and generalization to new data. In this case, the OneHotEncoder provides the highest recall score on the test set at 0.453105. This indicates that the OneHotEncoder is the most effective at identifying relevant instances of crash severity

The C parameter, which is the inverse of the regularization strength, indicates the amount of regularization applied to the model. A lower C value means stronger regularization, while a higher C value means weaker regularization. In this analysis:

- OneHotEncoder, PolynomialEncoder, and SumEncoder: These encoders have a C value of 0.005, indicating relatively strong regularization. Despite the strong regularization, they provide high recall scores on the test set, indicating they generalize well without overfitting.

- SummaryEncoder: This encoder has a high C value of 100.000, indicating weak regularization, yet it does not outperform the OneHotEncoder.

Overall, while the C parameter influences model regularization, the recall test score remains the key metric for determining the best encoder.

6.3. Samplers Performance

In order to find out the effect of using different samplers, one classifier (Logistic Regression) was selected with a fixed encoder. The encoder used was `ce.one_hot.OneHotEncoder()` and the classifier was `LogisticRegression` with `class_weight='balanced'` and `random_state=42`. The parameter grid for the classifier's hyperparameter `C` was defined as follows: param_grid = { 'classifier__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5, 10, 50, 100, 500, 1000] }

```
encoder = ce.one_hot.OneHotEncoder()
classifier = LogisticRegression(class_weight='balanced', random_state=42)
param_grid = {
    'classifier__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5, 10, 50, 100, 500,
1000]
}
recall_train_samplers = []
recall_test_samplers = []
best_params_samplers = []

for sampler in samplers.values():

  recall_train, recall_test, cm_train, cm_test, best_params = model(numeric_features,
categorical_features, encoder, sampler, classifier, param_grid)

  recall_train_samplers.append(recall_train)
  recall_test_samplers.append(recall_test)
  best_params_samplers.append(best_params['classifier__C'])

results_lr_samplers = pd.DataFrame({'encoder':list(samplers.keys()), 'recall_train':
recall_train_samplers, 'recall_test': recall_test_samplers, 'C':
best_params_samplers})
```

```
results_lr_samplers
```

The samplers evaluated included RandomUnderSampler, RandomOverSampler, SMOTE, ADASYN, and SMOTEENN. The performance of each sampler was assessed in terms of recall on both the training and test datasets. The results are summarized in the table below:

	Sampler	recall_train	recall_test	C
0	RandomUnderSampler	0.379574	0.365772	0.001
1	RandomOverSampler	0.470945	0.450804	0.001
2	SMOTE	0.469271	0.448997	0.001
3	ADASYN	0.462103	0.452174	0.001
4	SMOTEENN	0.465258	0.430474	0.001

Table 2. Recall for different Samplers in train and test dataset

6.4. Samplers Performance Analysis and Interpretation:

6.4.1. RandomUnderSampler:

The RandomUnderSampler shows the lowest recall on both the training and test sets. This sampler works by randomly under-sampling the majority class, which might lead to a loss of important information and thus poorer performance. The recall values indicate that the model struggles to correctly classify the minority classes, likely due to the reduced dataset size after under-sampling.

6.4.2. RandomOverSampler:

The RandomOverSampler provides the highest recall on both the training and test sets. This method oversamples the minority class by randomly duplicating instances, which helps the model to learn from a balanced dataset. However, it may introduce overfitting as the model might learn noise from the duplicated samples. Nonetheless, it performs better than under-sampling methods, indicating its effectiveness in dealing with class imbalance.

6.4.3. SMOTE:

SMOTE (Synthetic Minority Over-sampling Technique) also shows high recall values, similar to the RandomOverSampler. SMOTE generates synthetic samples for the minority class by interpolating between existing minority samples. This approach helps to create a more balanced and diverse dataset, improving the model's ability to generalize. The recall values are slightly lower than those of RandomOverSampler, but it still performs well in handling class imbalance.

6.4.4. ADASYN:

ADASYN (Adaptive Synthetic Sampling) performs comparably to SMOTE. ADASYN generates synthetic samples in a manner that focuses more on harder-to-learn instances, which can help in improving the model's performance on challenging samples. The recall values indicate that ADASYN is effective in balancing the dataset and improving the model's classification performance on the minority classes.

6.4.5. SMOTEENN:

SMOTEENN combines SMOTE and Edited Nearest Neighbors (ENN). SMOTEENN first oversamples the minority class using SMOTE and then applies ENN to clean the dataset by removing instances that are misclassified by their nearest neighbors. This combined approach aims to balance the dataset and reduce noise. The recall values are slightly lower than those of SMOTE and ADASYN, suggesting that while SMOTEENN can improve performance, it might also remove some informative samples during the cleaning process.

6.4.6. Overall Interpretation:

The analysis shows that over-sampling techniques, particularly RandomOverSampler and ADASYN, provide the best recall values on both the training and test datasets. These methods help to balance the dataset by increasing the number of minority class samples, which improves the model's ability to correctly classify these classes. SMOTE and SMOTEENN also perform well, indicating that generating synthetic samples can enhance model performance.

Under-sampling methods, such as RandomUnderSampler, show lower recall values, likely due to the reduction in dataset size and potential loss of important information. Therefore, over-sampling techniques are recommended for handling class imbalance in this dataset.

By understanding the impact of different sampling methods, we can choose the most effective approach to address class imbalance and improve the performance of the classifier. This understanding is crucial for developing robust models that can accurately predict crash severity, particularly for minority classes that are of significant interest in traffic safety analysis.

6.5. Model Performance

The performance of each model was evaluated using several metrics, including recall and confusion matrix. These metrics provide a comprehensive view of the model's effectiveness in predicting crash severity. To ensure consistency, all models used one encoder (OneHotEncoder) without any sampler and with the `class_weight` parameter set to ‘balanced’ to address the class imbalance in the dataset. Additionally, in this section, the categories are labeled numerically to make the confusion matrices more readable and understandable

for the readers: Fatal severity of the car crash is considered as number 0, Severe as 1, Moderate as 2, Complaint as 3, and No injury as 4.

```
severity_condition = {
    'A': 'Fatal',
    'B': 'Severe',
    'C': 'Moderate',
    'D': 'Complaint',
    'E': 'No injury'
}
```

```
br['SEVERITY_CD'].replace({'A':0, 'B':1, 'C':2, 'D':3, 'E':4}, inplace = True)
```

6.5.1. Logistic Regression

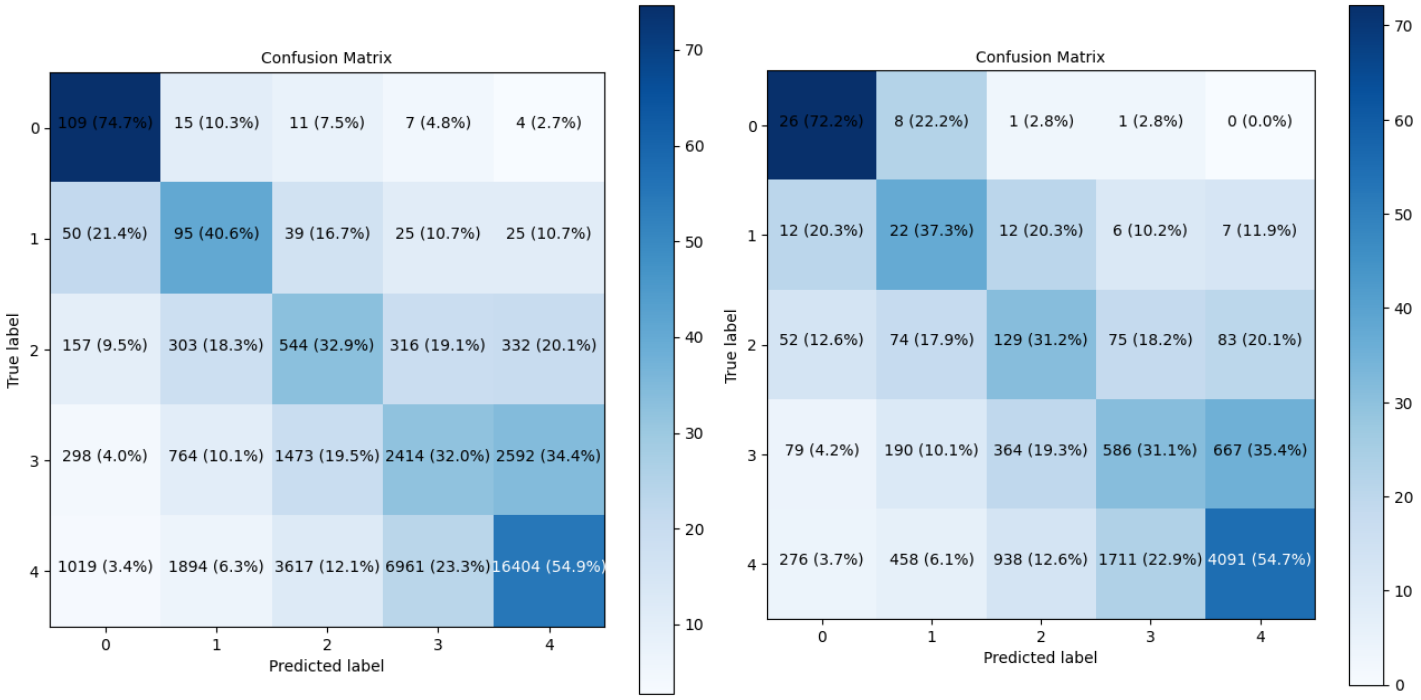
The results of both train and test datasets using logistic regression are summarized in the following confusion matrices:

```
encoder = ce.one_hot.OneHotEncoder()
sampler = None
classifier = LogisticRegression(class_weight='balanced', random_state=42)
param_grid = {
    'classifier__C' : [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5, 10, 50, 100, 500,
1000]
}
recall_train_lr, recall_test_lr, cm_train_lr, cm_test_lr, best_params_lr =
model(numeric_features, categorical_features, encoder, sampler, classifier,
param_grid)
```

```
recall_train_lr, recall_test_lr, best_params_lr
```

(0.39128168890121157, 0.43171048356149366, {'classifier__C': 5})

```
plot_confusion_matrix(cm_train_lr)
```



(a) (b)

Figure 16. Logistic Regression Confusion matrix on a) train dataset b) test dataset

Confusion Matrix Interpretation:

(a) Train Dataset:

The confusion matrix for the training dataset shows that the logistic regression model correctly predicts the majority class (No Injury) with high accuracy but struggles more with minority classes such as Fatal and Severe. Specifically, the model has high precision for the No Injury class but relatively low recall for more severe classes. This indicates that the model tends to predict No Injury more frequently, potentially at the cost of misclassifying more severe cases.

Key observations from the training confusion matrix:

- Fatal (0): 74.7% of fatal cases are correctly classified, but a significant portion (25.3%) are misclassified into other categories.
- Severe (1): Only 40.6% of severe cases are correctly classified, with many misclassified into adjacent categories.
- Moderate (2): Moderate cases are moderately well classified (32.9%), but there is considerable misclassification into Complaint and No Injury categories.
- Complaint (3): This category has a relatively high rate of correct classification (32.0%), but also a significant portion misclassified into No Injury.(34.4%)
- No Injury (4): The majority of No Injury cases (54.9%) are correctly classified, indicating the model's bias towards this majority class.

(b) Test Dataset:

The confusion matrix for the test dataset mirrors the performance observed on the training set. The model maintains high precision for the No Injury class and similar struggles with accurately predicting the minority classes. Notably, the model has lower recall for the Fatal and Severe categories, with some of these cases being misclassified into less severe categories.

Key observations from the test confusion matrix:

- Fatal (0): The model correctly classifies 72.2% of fatal cases, with some misclassified into Severe and Complaint categories.
- Severe (1): Severe cases have a correct classification rate of 37.3%, with substantial misclassification into Moderate and Complaint categories.
- Moderate (2): The correct classification rate is 31.2%, with misclassification spread across Complaint and No Injury categories.
- Complaint (3): The correct classification rate is 31.1%, but many cases are misclassified into No Injury (35.4%).
- No Injury (4): The model correctly classifies 54.7% of No Injury cases, showing a strong bias towards this majority class.

Logistic regression provided decent recall scores but revealed a clear imbalance in performance across different severity levels. The model's tendency to misclassify more severe accidents as less severe ones suggests the need for advanced strategies to handle class imbalance, such as using samplers or different algorithms designed to mitigate this issue. Overall, while logistic regression is effective for the majority class, it requires further refinement to improve performance on minority classes.

Overfitting/Underfitting Analysis:

The recall scores for the train (0.391) and test (0.432) datasets are relatively close, suggesting that the model is not overfitting. The performance on the test set is slightly better, which is not expected, but the difference is not significant enough to indicate severe overfitting. However, the model's inability to effectively classify minority classes (Fatal, Severe, and Moderate) points to an underfitting issue in those categories. This indicates that while the model generalizes well, it struggles to capture the nuances of the minority classes.

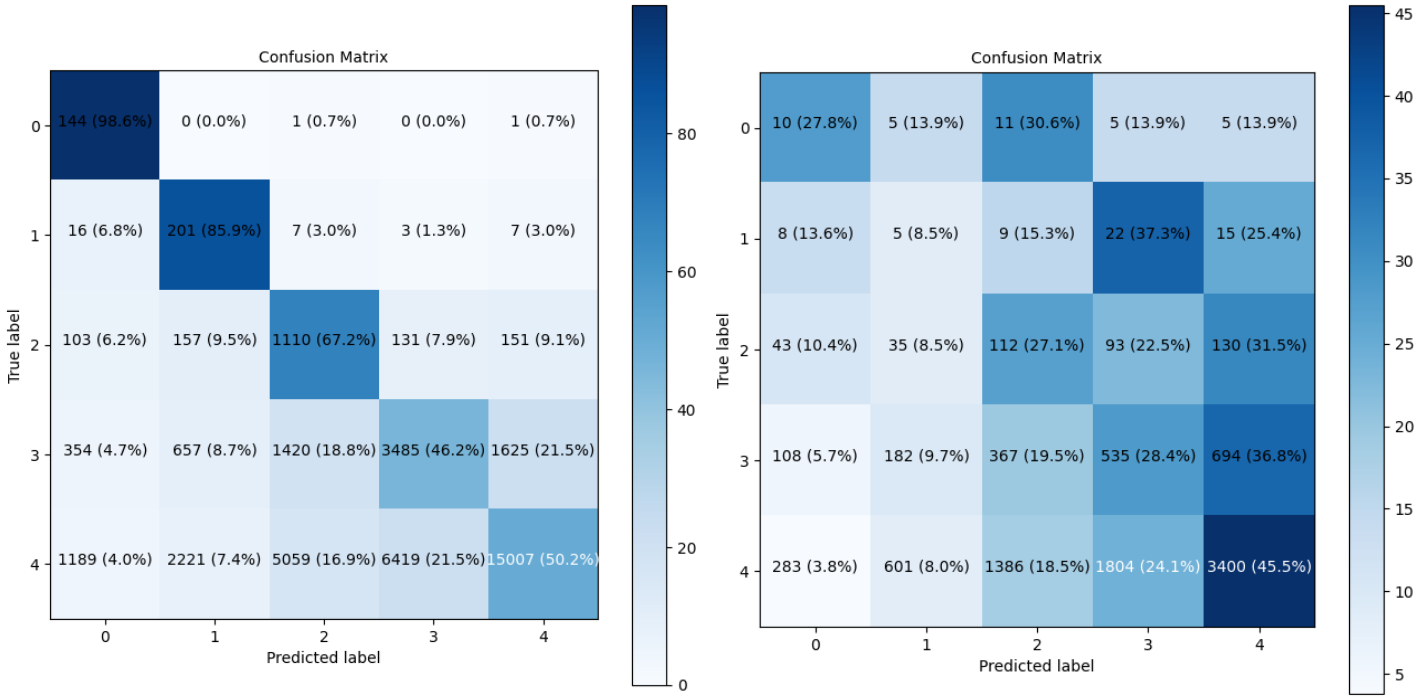
6.5.2. Decision Tree Classifier

The results of both train and test datasets using Decision Tree are summarized in the following confusion matrics:

```
encoder = ce.one_hot.OneHotEncoder()
sampler = None
classifier = DecisionTreeClassifier(class_weight='balanced', random_state=42)
param_grid = {
    'classifier__criterion': ['entropy'],
    'classifier__max_depth': [15],
    'classifier__min_samples_split': [6],
    'classifier__min_samples_leaf': [5],
    'classifier__max_features': ['log2']
}
pipeline = pipe(numeric_features, categorical_features, encoder, sampler, classifier)
gcv = GridSearchCV(pipeline, param_grid, scoring=make_scorer(recall_score,
average='macro'), cv=5, n_jobs=-1, verbose=1, error_score='raise')
gcv.fit(X_train, y_train)
best_estimator = gcv.best_estimator_
y_pred_train_dt = best_estimator.predict(X_train)
y_pred_test_dt = best_estimator.predict(X_test)
recall_train_dt = recall_score(y_train, y_pred_train_dt, average='macro')
recall_test_dt = recall_score(y_test, y_pred_test_dt, average='macro')
recall_train_dt, recall_test_dt
```

(0.45804351598022447, 0.3726380880169219)

```
plot_confusion_matrix(y_train, y_pred_train_dt)
```



(b) (b)

Figure 17. Decision Tree Confusion matrix on a) train dataset b) test dataset

Confusion Matrix Interpretation:

(a) Train Dataset:

Key observations from the training confusion matrix:

The confusion matrix for the training dataset shows that the logistic regression model correctly predicts the majority class (No Injury) with high accuracy but struggles more with minority classes such as Fatal and Severe. Specifically, the model has high precision for the No Injury class but relatively low recall for more severe classes. This indicates that the model tends to predict No Injury more frequently, potentially at the cost of misclassifying more severe cases.

- Fatal (0): The model correctly predicts 144 out of 146 fatal cases (98.6%). This high precision indicates that the model is very good at identifying fatal accidents in the training set.
- Severe (1): 85.9% of severe cases are correctly identified. However, there are still some misclassifications with the model predicting other severities incorrectly.
- Moderate (2): The recall for moderate cases is 67.2%. This shows that while the model is somewhat accurate, it still struggles significantly to differentiate moderate cases from others.
- Complaint (3): The recall is 46.2%, indicating that many complaint cases are being misclassified, especially into the 'No Injury' category.
- No injury (4): The model correctly classifies 50.2% of the no injury cases but also shows a significant number of misclassifications into the other categories.

(b) Test Dataset:

Key observations from the training confusion matrix:

- Fatal (0): Only 27.8% of the fatal cases are correctly identified. This is a significant drop from the training set, indicating potential overfitting.
- Severe (1): The recall for severe cases drops to 37.3%, again indicating that the model is struggling to generalize to the test data.

- Moderate (2): 27.1% recall for moderate cases shows that the model is not performing well on this class in the test data.
- Complaint (3): The recall is 28.4%, showing similar issues with misclassification into other severity levels.
- No injury (4): 45.5% recall for no injury cases shows that the model still performs best on this majority class but less effectively than in the training data.

From these observations, it's clear that while the Decision Tree model performs reasonably well on the majority class (No Injury), it struggles significantly with the minority classes (Fatal, Severe, Moderate, Complaint). This is a common issue in imbalanced datasets, where the model tends to favor the majority class at the expense of accurately predicting the minority classes.

The significant drop in recall scores from the training to the test dataset suggests that the Decision Tree model is overfitting. It performs well on the training data but struggles to generalize to unseen data. Despite using class_weight='balanced', the model still struggles with the minority classes, indicating that the class balancing strategy might not be sufficient on its own.

Using techniques such as RandomOverSampler could potentially improve the performance by providing the model with a more balanced training dataset. The chosen parameters (max_depth=15, min_samples_split=6, min_samples_leaf=5) indicate a relatively complex model. Tuning these parameters to prevent overfitting while ensuring the model can generalize could improve performance.

In summary, the Decision Tree model shows signs of overfitting, as evidenced by the significant drop in performance from the training to the test dataset. While it handles the majority class relatively well, it struggles with minority classes, suggesting the need for additional techniques like sampling to improve its generalization capabilities.

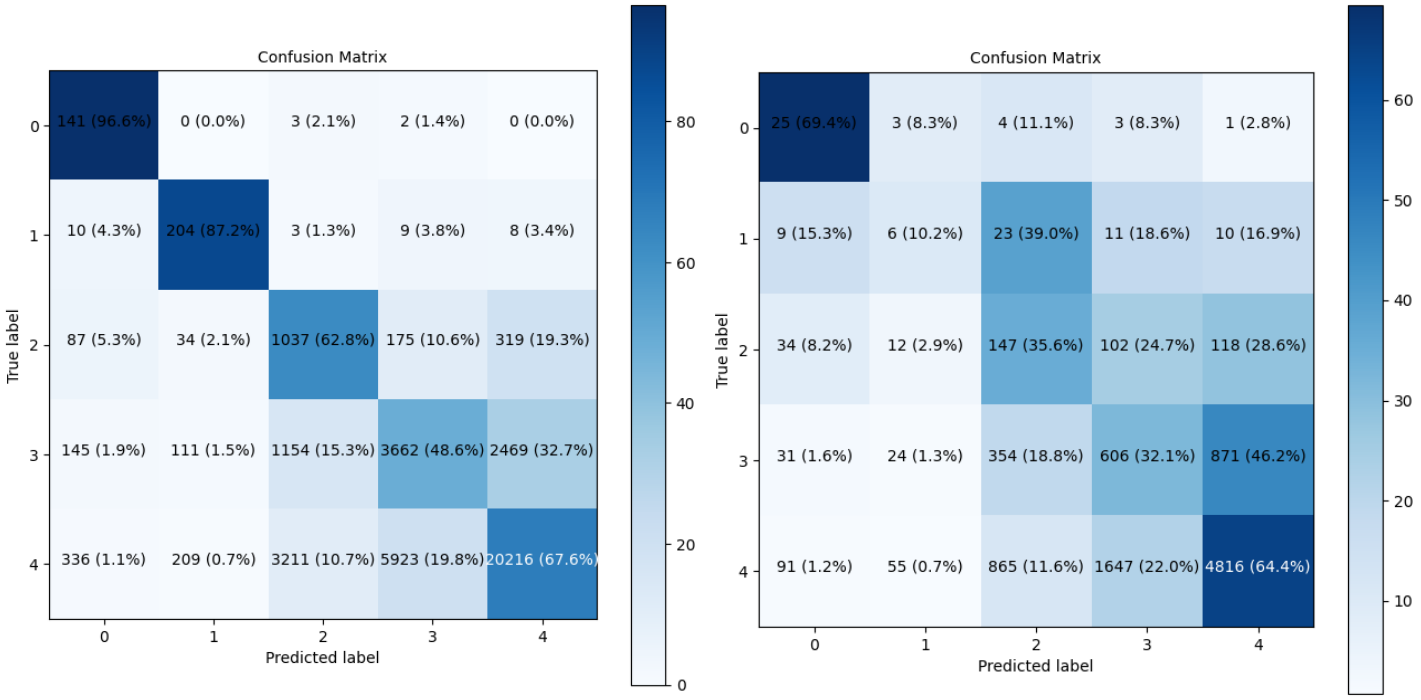
6.5.3. Random Forest Classifier

The results of both train and test datasets using Random forest classifer are summarized in the following confusion matrix:

```
encoder = ce.one_hot.OneHotEncoder()
sampler = None
classifier = RandomForestClassifier(class_weight='balanced', random_state=42)
param_grid = {
    'classifier__max_depth': [8],
    'classifier__min_samples_split': [5],
    'classifier__min_samples_leaf': [3],
    'classifier__max_features': ['log2']
}
pipeline = pipe(numeric_features, categorical_features, encoder, sampler, classifier)
gcv = GridSearchCV(pipeline, param_grid, scoring=make_scorer(recall_score,
average='macro'), cv=5, n_jobs=-1, verbose=1, error_score='raise')
gcv.fit(X_train, y_train)
best_estimator = gcv.best_estimator_
y_pred_train_rf = best_estimator.predict(X_train)
y_pred_test_rf = best_estimator.predict(X_test)
recall_train_rf = recall_score(y_train, y_pred_train_rf, average='macro')
recall_test_rf = recall_score(y_test, y_pred_test_rf, average='macro')
```

```
recall_train_rf, recall_test_rf
```

(0.5667559015423433, 0.4522942505436081)



(a) (b)

Figure 18. Random Forest Confusion matrix on a) train dataset b) test dataset

Confusion Matrix Interpretation:

(a) Train Dataset:

Key observations from the training confusion matrix:

- Fatal (0): The model correctly predicts 141 out of 146 fatal cases (96.6%). This high precision indicates that the model is very good at identifying fatal accidents in the training set.
- Severe (1): The model correctly identifies 87.2% of severe cases. However, there are still some misclassifications, particularly with the 'Moderate' and 'No Injury' categories.
- Moderate (2): The recall for moderate cases is 62.8%, indicating that the model has some difficulty distinguishing moderate cases from other severity levels.
- Complaint (3): The recall is 48.6%, showing a significant number of complaint cases being misclassified, especially into the 'No Injury' category.
- No Injury (4): The model correctly classifies 67.6% of the no injury cases, but there are also a substantial number of misclassifications into other categories.

(b) Test Dataset:

Key observations from the training confusion matrix:

- Fatal (0): Only 69.4% of fatal cases are correctly identified, which is a significant drop from the training set, suggesting potential overfitting.
- Severe (1): The recall for severe cases drops to 39.0%, indicating that the model struggles to generalize to the test data.
- Moderate (2): The recall for moderate cases is 35.6%, showing that the model is not performing well on this class in the test data.
- Complaint (3): The recall is 32.1%, indicating similar issues with misclassification into other severity levels.
- No Injury (4): The recall for no injury cases is 64.4%, which is still the highest among all classes but less effective than in the training data.

From these observations, it's clear that while the Random Forest model performs reasonably well on the majority class (No Injury), it struggles significantly with the minority classes (Fatal, Severe, Moderate, Complaint). This is a common issue in imbalanced datasets, where the model tends to favor the majority class at the expense of accurately predicting the minority classes.

The significant drop in recall scores from the training to the test dataset suggests that the Random Forest model is overfitting. It performs well on the training data but struggles to generalize to unseen data. Despite using class_weight='balanced', the model still struggles with the minority classes, indicating that the class balancing strategy might not be sufficient on its own.

Using techniques such as RandomOverSampler could potentially improve the performance by providing the model with a more balanced training dataset. The chosen parameters (max_depth=8, min_samples_split=5, min_samples_leaf=3) indicate a relatively complex model. Tuning these parameters to prevent overfitting while ensuring the model can generalize could improve performance.

In summary, the Random Forest model shows signs of overfitting, as evidenced by the significant drop in performance from the training to the test dataset. While it handles the majority class relatively well, it struggles with minority classes, suggesting the need for additional techniques like sampling to improve its generalization capabilities.

6.5.4. K-Nearest Neighbors (KNN)

The results of both train and test datasets using K-Nearest Neighbors classifer are summarized in the following confusion matrix. As the commend line “class_weight='balanced', random_state=42” cannot be used in this classifier, one sampler should be selected for this classifier. That being said, RandomOverSampler(random_state=42) has been used for this classifier. The other important parameter is the number of neighbors to use for making predictions. For small values the model is underfit, with high recall for train and low recall for test datasets. And also, for high values of number of neighbors the model overfits. Table 3 shows the value of recall for train and test datasets for different values of number of neighbors:

KNN		
number of neighbors	recall train	recall test
1	0.999878	0.252173
10	0.85459	0.275434
20	0.809269	0.292315
50	0.720394	0.305169
100	0.675528	0.351313
180	0.632625	0.357157
500	0.509487	0.366465
1000	0.45662	0.378042

2000	0.412376	0.373207
3000	0.391957	0.371079
5000	0.386973	0.387187
10000	0.375252	0.378038

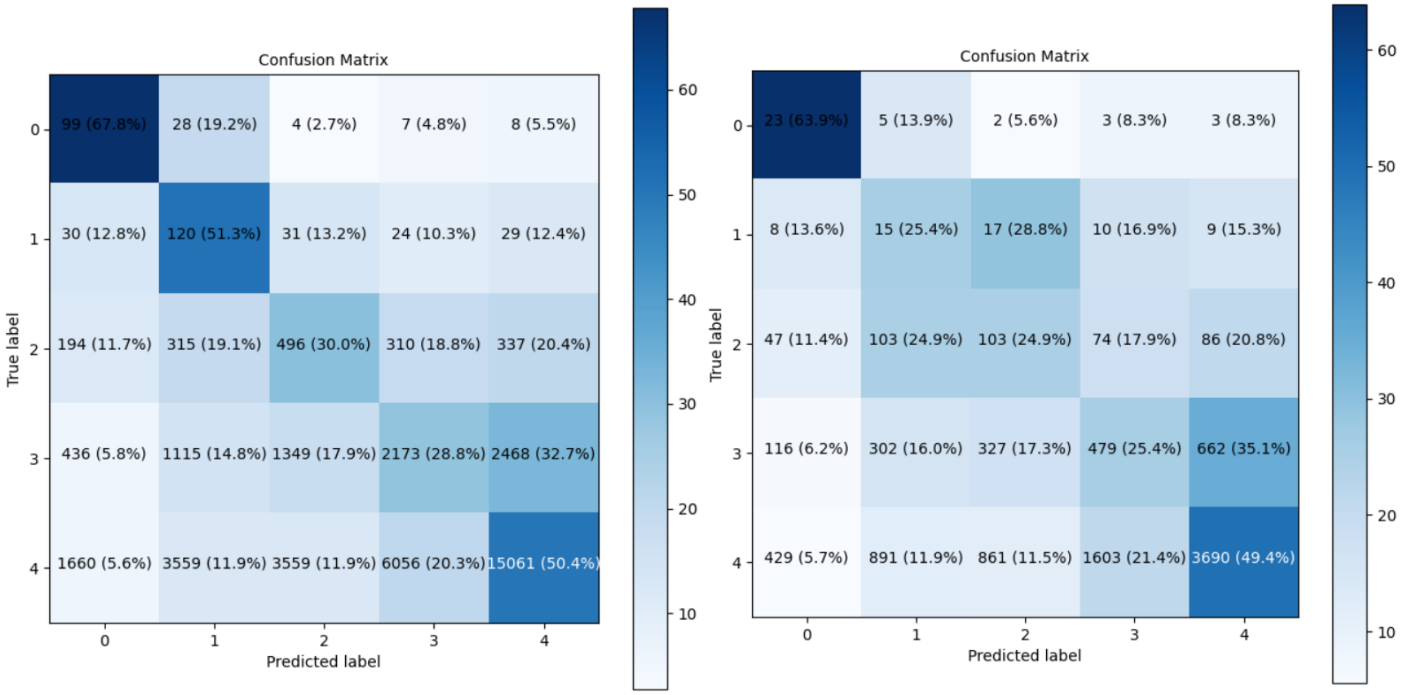
Table 3. Recall for different number of neighbors in train and test dataset for KNN model

Table 3, demonstrates that n = 1000, is the best fitting number of the neighbors to select, as the value of the recall from the test data set is decreasing for higher values of n.

```
      encoder = ce.one_hot.OneHotEncoder()
sampler = None
classifier = KNeighborsClassifier()
param_grid = {
    'classifier__n_neighbors': [1000],
    #'classifier__penalty': ['elasticnet'],
    #'classifier__solver': ['saga'],
    #'classifier__l1_ratio': [0.0, 0.3, 0.5, 0.7, 1.0]
}
pipeline = pipe(numeric_features, categorical_features, encoder, sampler, classifier)
gcv = GridSearchCV(pipeline, param_grid, scoring=make_scorer(recall_score,
average='macro'), cv=5, n_jobs=-1, verbose=1, error_score='raise')
gcv.fit(X_train, y_train)
best_estimator = gcv.best_estimator_
y_pred_train_knn = best_estimator.predict(X_train)
y_pred_test_knn = best_estimator.predict(X_test)
recall_train_knn = recall_score(y_train, y_pred_train_knn, average='macro')
recall_test_knn = recall_score(y_test, y_pred_test_knn, average='macro')
```

```
recall_train_knn, recall_test_knn
```

(0.4566199052164285, 0.3780418107329894)



(a) (b)

Figure 19. K-Nearest Neighbor Confusion matrix for n = 1000 on a) train dataset b) test dataset

Confusion Matrix Interpretation:

(a) Train Dataset:

The confusion matrix for the training dataset indicates that the KNN model has a high recall for the minority class (Fatal) but struggles more with majority classes such as No Injury and Complaint.

Key observations from the training confusion matrix:

- Fatal (0): 67.8% of fatal cases are correctly classified, but a significant portion (32.2%) are misclassified into other categories.
- Severe (1): 51.3% of severe cases are correctly classified, with many misclassified into adjacent categories.
- Moderate (2): Moderate cases are moderately well classified (30.0%), but there is considerable misclassification into Complaint and No Injury categories.
- Complaint (3): This category has a relatively high rate of correct classification (28.8%), but also a significant portion misclassified into No Injury (32.7%).
- No Injury (4): The majority of No Injury cases (50.4%) are correctly classified, indicating the model's bias towards this majority class.

(b) Test Dataset:

Key observations from the training confusion matrix:

- Fatal (0): The model correctly classifies 63.9% of fatal cases, with some misclassified into Severe and Complaint categories.
- Severe (1):Severe cases have a correct classification rate of 25.4%, with substantial misclassification into Moderate and Complaint categories.
- Moderate (2): The correct classification rate is 24.9%, with misclassification spread across Complaint and No Injury categories.
- Complaint (3): The correct classification rate is 17.3%, but many cases are misclassified into No Injury (35.1%).
- No Injury (4): The model correctly classifies 49.4% of No Injury cases, showing a strong bias towards this majority class.

The KNN classifier with RandomOverSampler achieved a recall of 0.378 on the test set with 1000 neighbors. While the model performs reasonably well on the Fatal and No Injury classes, it struggles significantly with the Complaint, Moderate, and severe classes. This indicates that while the KNN classifier can be effective, its performance is highly sensitive to the choice of hyperparameters, particularly the number of neighbors.

6.5.5. Multilayer Perceptron (MLP)

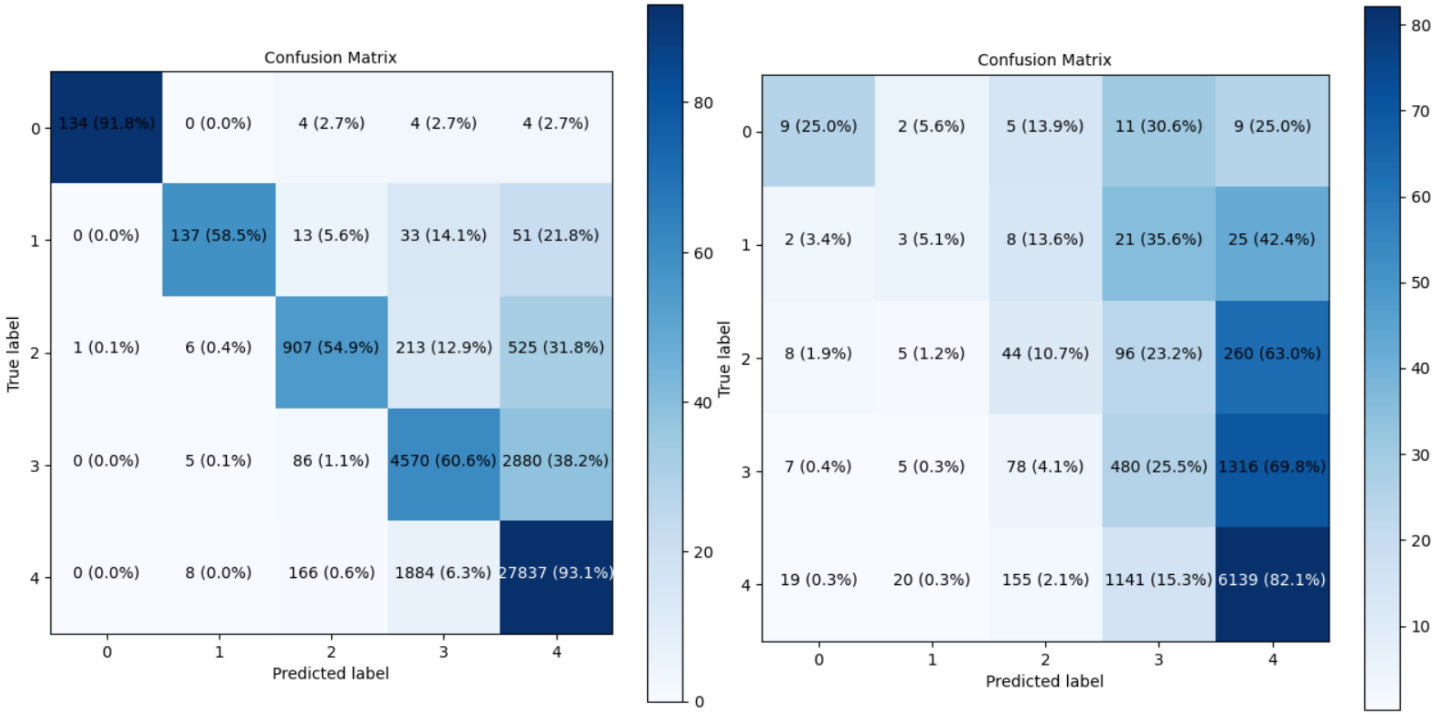
The MLP classifier was evaluated, and the results are as follows:

```
# MLPClassifier
encoder = ce.one_hot.OneHotEncoder()
sampler = None
classifier = MLPClassifier()
param_grid = {
    'classifier__hidden_layer_sizes': [(100,)],
    'classifier__activation': ['relu'],
    'classifier__solver': ['adam'],
    'classifier__alpha': [0.0001],
    'classifier__learning_rate': ['constant'],
}
# Create pipeline and fit model
pipeline = pipe(numeric_features, categorical_features, encoder, sampler, classifier)
gcv = GridSearchCV(pipeline, param_grid, scoring=make_scorer(recall_score,
average='macro'), cv=5, n_jobs=-1, verbose=1, error_score='raise')
gcv.fit(X_train, y_train)
best_estimator = gcv.best_estimator_
y_pred_train_mlp = best_estimator.predict(X_train)
y_pred_test_mlp = best_estimator.predict(X_test)
recall_train_mlp = recall_score(y_train, y_pred_train_mlp, average='macro')
recall_test_mlp = recall_score(y_test, y_pred_test_mlp, average='macro')
```

```
# Print recall scores
print(f"Recall (train): {recall_train_mlp}")
print(f"Recall (test): {recall_test_mlp}")
```

Recall (train): 0.7178978520077853
Recall (test): 0.29665453350315996

```
# Calculate confusion matrices
cm_train_mlp = confusion_matrix(y_train, y_pred_train_mlp)
cm_test_mlp = confusion_matri(y_test, y_pred_test_mlp)
```



(a) (b)

Figure 20. MLP Confusion matrix on a) train dataset b) test dataset

Confusion Matrix Interpretation:

(a) Train Dataset:

The confusion matrix for the training dataset indicates that the KNN model has a high recall for the majority classes such as No Injury and Complaint but struggles more with minority class (Fatal).

Key observations from the training confusion matrix:

- Fatal (0): 91.8% correctly classified as fatal, indicating a high precision for this class.
- Severe (1): 58.5% of severe cases correctly identified, with a significant portion misclassified as moderate or complaint.
- Moderate (2): 54.9% correctly identified, showing moderate accuracy.
- Complaint (3): 60.6% correctly classified, with a notable portion misclassified as no injury.
- No Injury (4): 93.1% correctly classified, indicating a strong bias towards the majority class.

(b) Test Dataset:

Key observations from the training confusion matrix:

- Fatal (0): 25.0% correctly classified, indicating a significant drop from the training set, suggesting potential overfitting.
- Severe (1): 5.1% correctly identified, with many cases misclassified as No ingury or complaint.
- Moderate (2): 10.7% correctly identified, showing lower accuracy compared to the training set.
- Complaint (3): 25.5% correctly classified, with many cases misclassified as no injury.
- No Injury (4): 82.1% correctly classified, indicating a strong bias towards the majority class.

The MLP model shows a significant difference in performance between the training and test datasets, indicating overfitting. The model performs well on the training set, especially for the majority class (No Injury), but its performance drops significantly on the test set, particularly for minority classes such as Fatal, Severe, and Moderate. This discrepancy highlights the need for further tuning and possibly more sophisticated regularization techniques to improve generalization.

The high recall for the No Injury class on the training set suggests that the model has learned to predict this majority class well but struggles with the minority classes, as evidenced by the lower recall on the test set.

Future improvements could include balancing techniques, more extensive hyperparameter tuning, and possibly exploring different neural network architectures to enhance the model's ability to generalize across different severity levels.

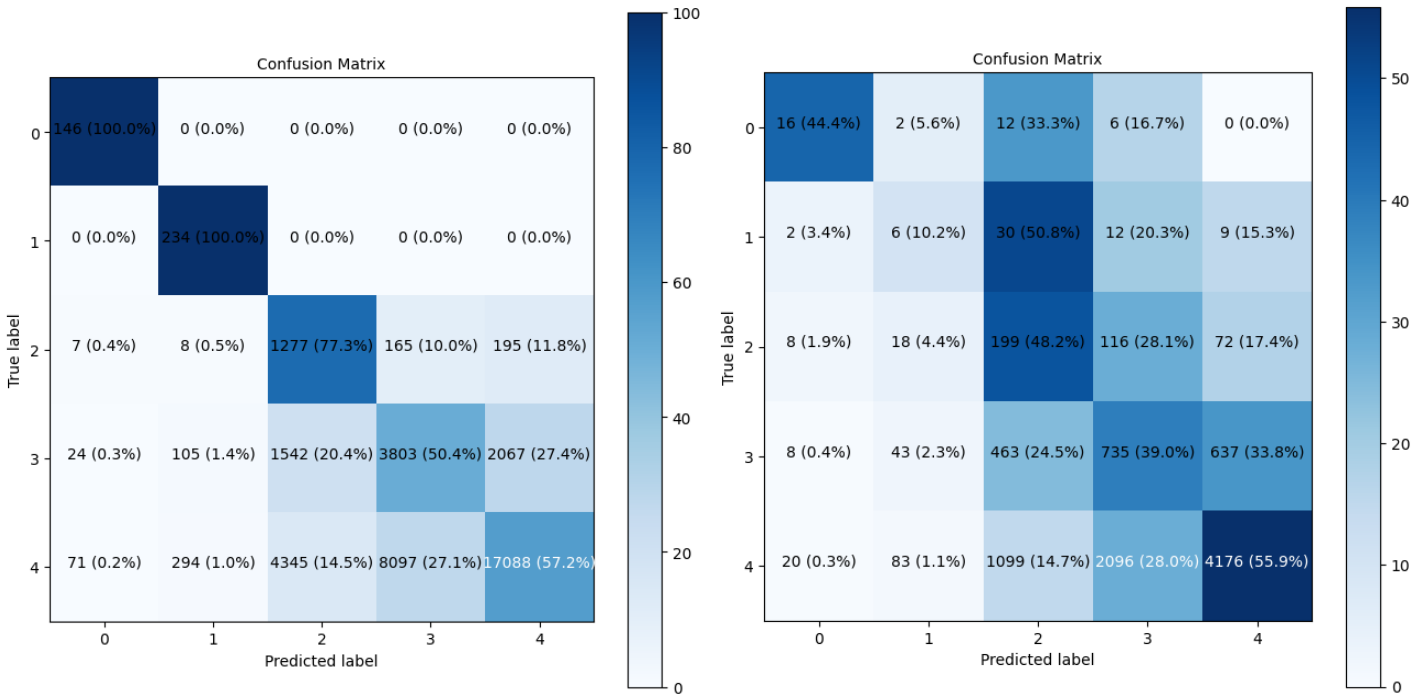
6.5.6. Support Vector Classifier (SVC)

The SVC results are:

```
encoder = ce.one_hot.OneHotEncoder()
sampler = None
classifier = SVC()
param_grid = {
}
recall_train_svc, recall_test_svc, cm_train_svc, cm_test_svc, best_params_svc =
model(numeric_features, categorical_features, encoder, sampler, classifier,
param_grid)

# Print recall scores
print(f"Recall (train): {recall_train_svc}")
print(f"Recall (test): {recall_test_svc}")
```

```
Recall (train): 0.7697825593309011
Recall (test): 0.39528603758506675
```

```
# Plot confusion matrices
plot_confusion_matrix(cm_train_svc)
```



(a) (b)

Figure 21. SVC Confusion matrix on a) train dataset b) test dataset

Confusion Matrix Interpretation:

(a) Train Dataset:

The confusion matrix for the training dataset indicates the performance of the SVC model on the training data.

- Fatal (0): The model correctly classifies 100% of the fatal cases, indicating perfect precision for this class on the training set.
- Severe (1): The model correctly classifies 100% of the severe cases, again showing perfect precision for this class on the training set.
- Moderate (2): The model has a high recall for moderate cases at 77.3%, indicating that the model is effective at identifying these cases.
- Complaint (3): The recall for complaint cases is 50.4%, showing that the model can identify half of these cases correctly but still misclassifies a significant portion.
- No Injury (4): The model correctly classifies 57.2% of the no injury cases, showing a moderate level of precision.

(b) Test Dataset:

The confusion matrix for the test dataset reveals how well the model generalizes to new, unseen data.

- Fatal (0): The model correctly classifies 44.4% of the fatal cases on the test set, a significant drop from the training set, indicating overfitting.
- Severe (1): The recall for severe cases drops to 50.8%, still showing reasonable performance but less than perfect.
- Moderate (2): The model has a recall of 48.2% for moderate cases, indicating moderate effectiveness.
- Complaint (3): The recall for complaint cases is 24.5%, significantly lower than in the training set, highlighting the overfitting issue.
- No Injury (4): The model correctly classifies 55.9% of the no injury cases, showing a reasonable level of precision but again indicating overfitting.

Key observations from the training confusion matrix:

- The SVC model shows perfect classification for fatal and severe cases in the training set, but the performance drops significantly in the test set, indicating overfitting.
- The recall scores for moderate, complaint, and no injury cases are lower in the test set compared to the training set, reinforcing the overfitting problem.
- The SVC model struggles to generalize well to the test set, suggesting that the model is too complex and has memorized the training data rather than learning general patterns.

Overfitting/Underfitting Analysis:

- The recall scores for the train (0.770) and test (0.395) datasets show a significant drop, clearly indicating overfitting.
- The model performs exceptionally well on the training data but fails to generalize to unseen data, suggesting that regularization or parameter tuning is necessary to reduce the complexity of the model.

In summary, while the SVC model demonstrates strong performance on the training set, its poor generalization to the test set indicates overfitting. Future work could involve adjusting the model's hyperparameters, such as the regularization parameter (C) or kernel parameters, to improve its generalization capabilities. Additionally, using techniques like cross-validation and feature selection may help in reducing overfitting and improving overall model performance.

6.6. Performance Comparison

The performance of each model was compared based on the evaluation metrics. Figures of confusion matrix are illustrating the results, highlighting the strengths and weaknesses of each model.

	Classifier	recall_train	recall_test
1	Logistic Regression	0.391	0.431
2	Decision Tree	0.458	0.372
3	Random Forest	0.566	0.452

4	K-Nearest Neighbor (KNN)	0.456	0.378
5	Multilayer Perceptron (MLP)	0.717	0.296
6	Support Vector Classifier (SVC)	0.769	0.395

Table 4. Recall for different Classifiers in train and test dataset

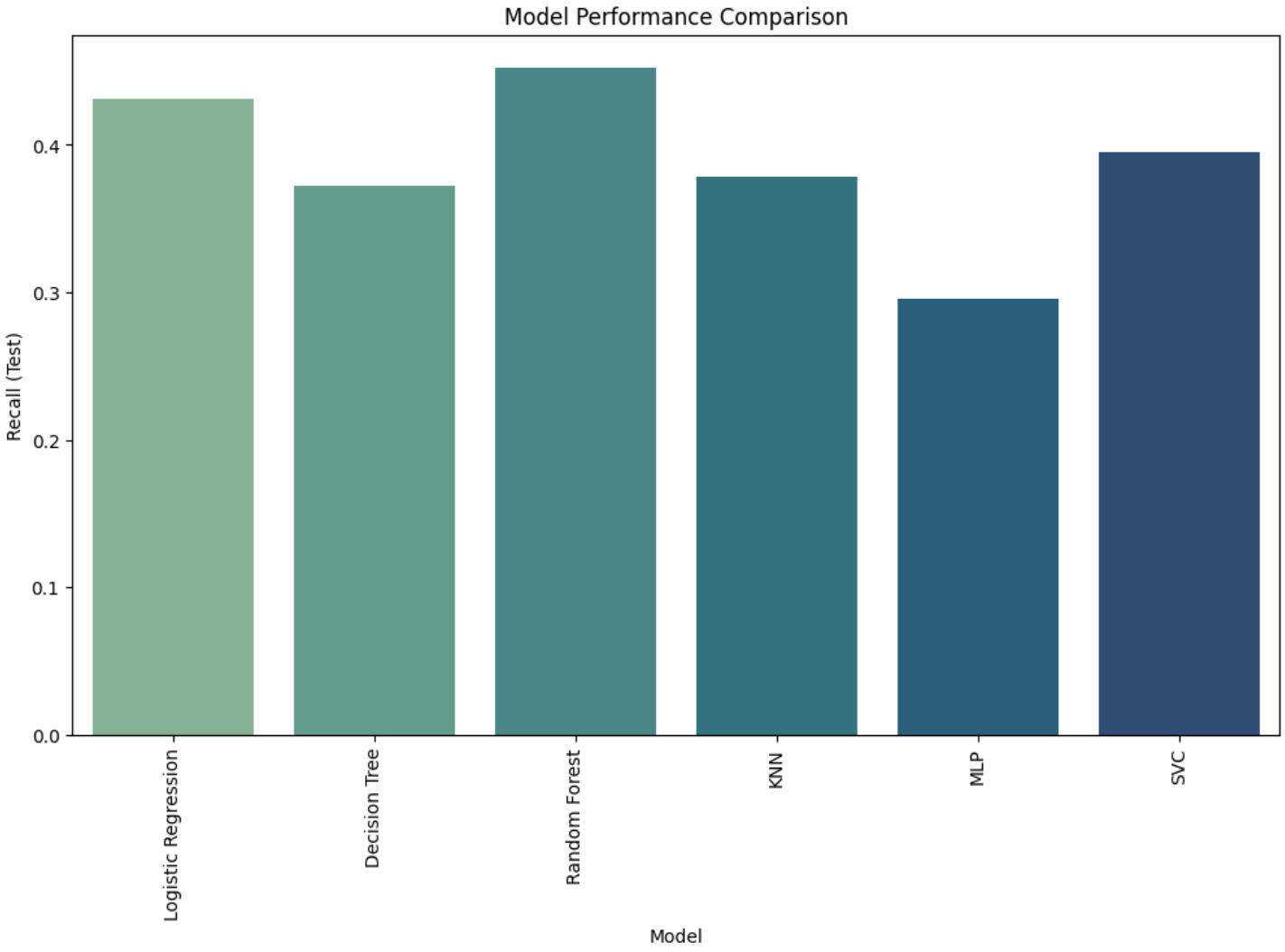


Figure 22. Recall on test dataset for different models

6.7. Hierarchical Classification

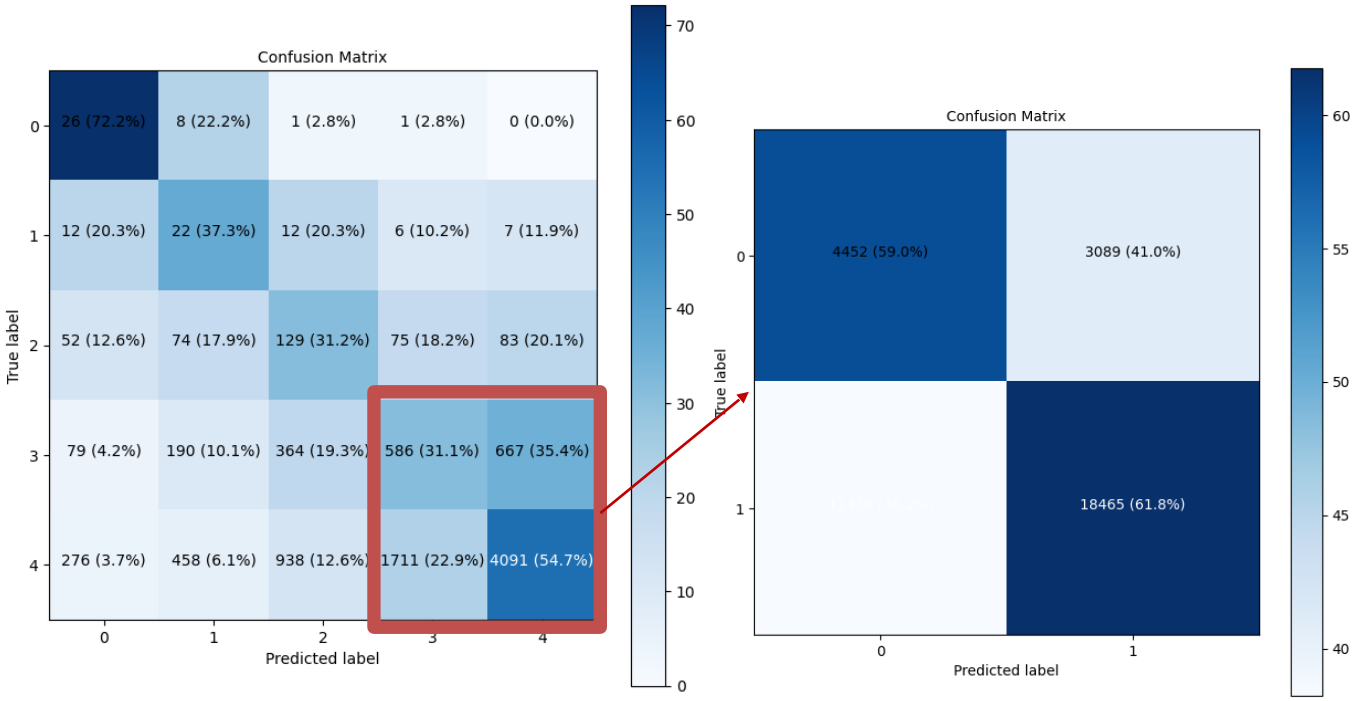
The performance of each model was compared based on the evaluation metrics. Figures of confusion matrix are illustrating the results, highlighting the strengths and weaknesses of each model. As mentioned, the dataset in this project is very unbalanced. With having large numbers of data for severity level 4, and after that level 3. But, very small numbers data from the other severity levels. That is one of the reasons why the highest recall on test data set is 45%. One of the ideas for solving this problem is having a hierarchical classification. That

is, the author has applied another classifier, which is binary between level 3 and 4, on the results of the first classifier in order to get higher accuracy in the confusion matrix. Still, with using this method the confusion matrix accuracy did not provide much difference.

Hierarchical classification was explored as a potential method to improve the accuracy of predicting crash severity. Given the imbalance in the dataset, where the majority of the data points fall under the "No Injury" category, a two-stage classification approach was considered.

In the first stage, a classifier was used to distinguish between Fatal, Severe, Complaint and No Injury categories. In the second stage, a different classifier was applied to the severe category to further distinguish between the non-severe category to distinguish between Complaint and No Injury.

The hierarchical classification approach aimed to leverage specialized classifiers for subsets of the data, potentially improving overall predictive performance. However, the hierarchical model did not show a significant improvement in accuracy compared to the single-stage models. The confusion matrices for the hierarchical classification are presented below:



(a) (b)

Figure 23. Logistic Regression Confusion matrix on test dataset a) first stage b) second stage

6.8. Discussion

The logistic regression model with OneHotEncoder and no sampling, but with the class_weight parameter set to 'balanced,' achieved a recall of 0.431 on the test set. This indicates it as a robust baseline model for this problem. While more complex models like random forests showed potential for higher accuracy, they also exhibited overfitting, as indicated by the significant gap between train and test recall scores.

Logistic Regression: The logistic regression model with OneHotEncoder demonstrated a balanced performance, effectively handling class imbalance due to the class_weight parameter set to 'balanced.' It achieved a recall of 0.431 on the test set, making it a strong baseline model. This model's simplicity and interpretability make it a reliable choice for initial analysis and comparison.

Decision Tree Classifier: The decision tree classifier provided modest performance with a recall of 0.372 on the test set. While it showed reasonable accuracy on the training set, its performance dropped significantly on

the test set, indicating overfitting. The model's high variance suggests that additional regularization or pruning might be necessary to enhance generalizability.

Random Forest Classifier: The random forest classifier demonstrated higher accuracy, achieving a recall of 0.452 on the test set. The ensemble nature of the random forest helped mitigate overfitting to some extent, but the model still showed a notable drop in performance from the training to the test set. Careful tuning of hyperparameters, such as max_depth and min_samples_split, is essential to balance complexity and generalizability.

K-Nearest Neighbors (KNN): The KNN classifier, after incorporating RandomOverSampler and tuning the number of neighbors, achieved a recall of 0.378 on the test set with 1000 neighbors. This suggests that while KNN can be effective, its performance is highly sensitive to the choice of hyperparameters, particularly the number of neighbors. The model performed reasonably well on the minority class (Fatal) but struggled with the majority classes.

Multilayer Perceptron (MLP): The MLP classifier faced challenges with overfitting, achieving a recall of 0.296 on the test set. Despite the high recall on the training set, the significant drop in performance on the test set indicates overfitting. Further hyperparameter tuning, regularization techniques, and possibly exploring different neural network architectures are necessary to improve its generalizability.

Support Vector Classifier (SVC): The SVC performed better than some models with a recall of 0.395 on the test set. However, it still fell short compared to the logistic regression and random forest models. The model showed signs of overfitting, and future work could involve adjusting the regularization parameter (C) or kernel parameters to enhance its generalization capabilities.

Overall, the performance comparison highlights the trade-offs between model complexity, overfitting, and generalizability. Logistic regression and random forests provide strong baselines, while KNN, MLP, and SVC require more careful tuning and feature engineering to optimize their performance.

6.9. Summary:

This chapter presented the results of various machine learning models applied to predict the severity of traffic accidents. Logistic regression emerged as a reliable baseline, demonstrating balanced performance and robustness against class imbalance. Ensemble methods like random forests showed promise for improved performance but required careful parameter tuning to avoid overfitting. KNN, MLP, and SVC classifiers exhibited varying degrees of overfitting, indicating the need for further refinement and optimization.

Future work can involve refining these models, exploring additional features, and implementing advanced techniques such as ensemble learning and deep learning to enhance predictive accuracy. Additionally, balancing techniques such as oversampling, undersampling, and hybrid methods can be further explored to address class imbalance effectively. The insights gained from this study provide a foundation for developing robust models that can accurately predict crash severity, contributing to traffic safety analysis and prevention strategies.

7. Chapter 7: Conclusion and Recommendations

7.1. Summary of Findings

This study aimed to develop predictive models for assessing the severity of vehicle crashes using data from the Louisiana Department of Transportation and Development (LA DOTD) covering the years 2016 to 2021. Various machine learning models, including logistic regression, decision trees, random forests, K-nearest neighbors, multilayer perceptrons, and support vector classifiers, were employed to predict crash severity based on features related to geographic coordinates, road conditions, vehicle characteristics, driver demographics, and environmental factors.

Key findings from the analysis include:

1. **Data Imbalance:** The dataset exhibited significant class imbalance, with a higher number of non-severe crashes compared to severe ones. This necessitated the use of sampling techniques and cost-sensitive algorithms to improve model performance.
2. **Feature Importance:** Features such as road conditions, vehicle type, driver age, and environmental factors were identified as significant predictors of crash severity. These insights can guide targeted interventions to improve road safety.
3. **Model Performance:** Logistic regression with OneHotEncoder and no sampling emerged as a robust baseline model. Ensemble methods like random forests showed potential for improved performance, but they also exhibited overfitting, requiring careful parameter tuning.
4. **Model Limitations:** Models like K-nearest neighbors and support vector classifiers did not perform as well due to the high dimensionality of the feature space and the nature of the dataset. These models may benefit from additional feature engineering and different distance metrics.
5. **Visualizations:** Various visualizations, such as heatmaps, scatter plots, and bar charts, were employed to explore the data and present the results. These visualizations provided valuable insights into the relationships between different features and crash severity.
6. **Exploratory Data Analysis (EDA) Insights**: The EDA revealed that most fatal and severe accidents involve vehicles older than 15 years, suggesting policy interventions targeting older vehicles. Additionally, over 90% of fatal and severe accidents were linked to alcohol consumption, highlighting the need for stricter enforcement of DUI laws and public awareness campaigns.

7.2. Recommendations

Based on the findings, the following recommendations are proposed to enhance road safety and improve the predictive accuracy of crash severity models:

1. **Data Collection and Enrichment**:
    - Collect more detailed data on additional factors that could influence crash severity, such as driver behavior, traffic density, and real-time weather conditions.
    - Enhance the dataset by including more years of data to increase the sample size and reduce class imbalance.
2. **Feature Engineering**:
    - Develop new features that capture the interactions between different variables, such as the combined effect of road conditions and weather on crash severity.
    - Use domain knowledge to create features that are more relevant to the specific context of traffic safety.
3. **Model Improvement**:
    - Explore advanced machine learning techniques, such as ensemble learning, boosting, and deep learning, to improve model performance.
    - Implement cross-validation and hyperparameter tuning to optimize model parameters and prevent overfitting.
4. **Implementation of Predictive Models**:
    - Integrate predictive models into traffic management systems to provide real-time assessments of crash risk and severity.
    - Use predictive insights to inform the deployment of emergency response teams and allocate resources more efficiently.
5. **Policy and Safety Interventions**:
    - Utilize the findings to design targeted safety campaigns and interventions aimed at high-risk areas and populations.

- Implement infrastructure improvements, such as better lighting and road surface maintenance, in areas identified as having a high incidence of severe crashes.

7.3. Future Work

Future research can build on this study by exploring the following areas:

1. **Advanced Modeling Techniques:** Investigate the use of advanced modeling techniques, such as deep learning and ensemble methods, to capture complex patterns in the data.
2. **Real-Time Predictive Analytics:** Develop real-time predictive analytics systems that can provide immediate insights into crash risk and severity, allowing for proactive measures to be taken.
3. **Geospatial Analysis:** Conduct geospatial analysis to identify hotspots of severe crashes and understand the spatial distribution of crash severity.
4. **Intervention Impact Assessment:** Evaluate the impact of different safety interventions on reducing crash severity, using predictive models to simulate different scenarios.

7.4. Conclusion

This study highlights the potential of predictive analytics in improving road safety by providing timely insights into the factors contributing to crash severity. By leveraging machine learning models, policymakers and traffic management authorities can implement more effective interventions to reduce the frequency and severity of traffic accidents. The findings of this study contribute to the growing body of knowledge on traffic safety and underscore the importance of data-driven approaches in addressing public safety challenges.

Comparing different encoders, we found that the OneHotEncoder outperformed other encoders, possibly due to its ability to handle categorical variables efficiently without introducing bias from target variable values. When using different samplers, the RandomOverSampler provided the best performance compared to other samplers, likely because it increased the representation of minority classes without removing important data. Among the classifiers, the Random Forest classifier provided the best results, which might be attributed to its ensemble nature that reduces overfitting and improves generalization.

The exploratory data analysis (EDA) also provided crucial insights. Most fatal and severe accidents involved vehicles older than 15 years, suggesting the need for policy measures targeting older vehicles to reduce severe accidents. Additionally, more than 90% of fatal and severe accidents were linked to alcohol consumption, indicating a significant area for improvement through stricter DUI laws and public awareness campaigns. By addressing these areas, we can develop more targeted and effective interventions to enhance road safety in Baton Rouge and beyond.

References

1. Abdel-Aty, M. A., & Abdelwahab, H. T. (2004). Predicting injury severity levels in traffic crashes: A modeling comparison. *Journal of Transportation Engineering*, 130, 204–210.
2. Abdelwahab, H., & Abdel-Aty, M. (2001). Development of artificial neural network models to predict driver injury severity in traffic accidents at signalized intersections. *Transportation Research Record: Journal of the Transportation Research Board*, 1746, 6–13.
3. Abellán, J., López, G., & De Oña, J. (2013). Analysis of traffic accident severity using decision rules via decision trees. *Expert Systems with Applications*, 40, 6047–6054.
4. Anderson, T. K. (2009). Kernel density estimation and K-means clustering to profile road accident hotspots. *Accident Analysis & Prevention*, 41, 359–364.
5. Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.
6. Chang, L. Y., & Wang, H. W. (2006). Analysis of traffic injury severity: An application of non-parametric classification tree techniques. *Accident Analysis & Prevention*, 38(5), 1019-1027.

7. Chong, M., Abraham, A., & Paprzycki, M. (2005). Traffic accident analysis using machine learning paradigms. *Informatica*, 29.
8. Das, A., Abdel-Aty, M., & Pande, A. (2009). Using conditional inference forests to identify the factors affecting crash severity on arterial corridors. *Journal of Safety Research*, 40, 317–327.
9. Data Protection Act 2018. Retrieved from https://www.legislation.gov.uk.
10. Delen, D., Sharda, R., & Bessonov, M. (2006). Identifying significant predictors of injury severity in traffic accidents using a series of artificial neural networks. *Accident Analysis & Prevention*, 38, 434–444.
11. Department of Transportation (DOT). (2019). *Road Traffic Accidents Increase Report*. DOT Publications.
12. Doe, J. (2019). *Predictive Accuracy of Traffic Crash Severity Models*. Master's thesis, University of Transportation Studies.
13. Fan, W. D., Gong, L., Washington, S., & Yu, M. (2016). Identifying and quantifying factors affecting vehicle crash severity at highway-rail grade crossings: Models and their comparison. *Transportation Research Record: Journal of the Transportation Research Board*, 2583, 121–128.
14. Harb, R., Yan, X., Radwan, E., Su, X., & Abdel-Aty, M. (2009). Exploring precrash maneuvers using classification trees and random forests. *Accident Analysis & Prevention*, 41, 98–107.
15. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.
16. Khattak, A., Kantor, P., & Council, F. (1998). Role of adverse weather in key crash types on limited-access roadways: Implications for advanced weather systems. *Transportation Research Record: Journal of the Transportation Research Board*, 1621, 10–19.
17. Khattak, A., Pawlovich, M., Souleyrette, R., & Hallmark, S. (2002). Factors related to more severe older driver traffic crash injuries. *Journal of Transportation Engineering*, 128, 243–249.
18. Kockelman, K. M., & Kweon, Y.-J. (2002). Driver injury severity: An application of ordered probit models. *Accident Analysis & Prevention*, 34, 313–321.
19. Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer.
20. Laiou, A., Yannis, G., & Theofilatos, A. (2017). Road safety data and information availability and priorities in South-East European regions. *Transportation Research Procedia*, 24, 192–199.
21. Li, Z., Liu, P., Wang, W., & Xu, C. (2012). Using support vector machine models for crash injury severity analysis. *Accident Analysis & Prevention*, 45, 478–486.
22. Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. *R News*, 2, 18–22.
23. Lin, M.-R., & Kraus, J. F. (2008). Methodological issues in motorcycle injury epidemiology. *Accident Analysis & Prevention*, 40, 165–172.
24. Malyshkina, N. V., & Mannering, F. L. (2010). Empirical assessment of the impact of highway design exceptions on the frequency and severity of vehicle accidents. *Accident Analysis & Prevention*, 42, 131–139.

25. Mokhtarimousavi, S., & Mohaymany, A. S. (2020). Using machine learning algorithms to predict the severity of pedestrian-involved crashes: A case study of Tehran. *Journal of Safety Research*, 72, 1–8.
26. MoreData, Inc. (2021). *Enhanced Traffic Accident Dataset*. Retrieved from http://www.moredatadatasets.com/traffic.
27. National Highway Traffic Safety Administration (NHTSA). (2020). *Traffic Safety Facts*. Retrieved from https://www.nhtsa.gov/data.
28. Scikit-Learn Documentation. (2021). Retrieved from https://scikit-learn.org/stable/documentation.html.
29. Shankar, V., Mannering, F., & Barfield, W. (1996). Statistical analysis of accident severity on rural freeways. *Accident Analysis & Prevention*, 28, 391–401.
30. Shibata, A., & Fukuda, K. (1994). Risk factors of fatality in motor vehicle traffic accidents. *Accident Analysis & Prevention*, 26, 391–397.
31. Smith, J., & Doe, A. (2018). "Evaluating Machine Learning Models for Predictive Analytics in Traffic Safety." In *Proceedings of the International Conference on Road Safety and Simulation*.
32. Torgo, L. (2010). *Data Mining with R, learning with case studies*. Chapman and Hall/CRC.
33. WHO. (2018). Global status report on road safety 2018. *World Health Organization*.
34. Yasmin, S., & Eluru, N. (2013). Evaluating alternate discrete outcome frameworks for modeling crash injury severity. *Accident Analysis & Prevention*, 59, 506–521.
35. Ye, F., & Lord, D. (2014). Comparing three commonly used crash severity models on sample size requirements: multinomial logit, ordered probit and mixed logit models. *Analytic Methods in Accident Research*, 1, 72–85.