Open in Colab

Traffic Crashes in Chicago City.

Introduction

Traffic accidents are a major concern in urban areas, causing significant human and economic costs. In Chicago, thousands of traffic crashes occur annually, leading to injuries, fatalities, and extensive property damage. Understanding the factors that contribute to these crashes and predicting their severity can help city planners and public safety officials implement effective measures to improve road safety. This project leverages data from the City of Chicago's Traffic Crashes dataset to build predictive models that identify high-risk scenarios and provide actionable insights.

Project Overview

This project aims to analyze traffic accident data from the City of Chicago to predict the primary contributory cause of accidents. By applying machine learning techniques to this problem, we can identify key factors contributing to accidents, which can inform policy decisions, improve traffic safety, and reduce future accidents.

The dataset contains various features related to accidents, such as weather conditions, lighting conditions, and roadway surface conditions, among others. The target variable is the 'Primary Contributory Cause,' which we are modeling as a multi-class classification problem.

Let us import all the necessary libraries for this problem in the cell below:

In [3]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
```

Business understanding

Traffic accidents are a significant public safety issue. Understanding the factors that lead to accidents can help city planners, traffic engineers, and policymakers implement more effective safety measures. The objective of this project is to build a model that can accurately predict the primary contributory cause of an accident, providing actionable insights that can be used to reduce the frequency and severity of traffic accidents.

Data Understanding

The dataset used in this project was obtained from the City of Chicago's Traffic Crashes database. It includes detailed information about each accident, such as:

- **Crash Date:** The date and time of the crash.
- **Weather Conditions:** Describes the weather at the time of the crash.
- **Roadway Surface Conditions:** Describes the condition of the road surface.
- **Lighting Conditions:** Describes the lighting conditions during the crash.
- **Traffic Control Device:** The type of traffic control device present at the crash location.

The target variable,**Primary Contributory Cause**, is a categorical variable that indicates the main reason behind the crash.

Data Preparation

Data preparation involved several key steps:

1. **Handling Missing Values:** Columns with a high percentage of missing values were carefully imputed or excluded from the analysis to ensure the integrity of the dataset.
2. **Feature Engineering:** We created new features such as sine and cosine transformations for time-related features to capture cyclic patterns in the data. Additionally, we created interaction features to better capture the relationship between speed and weather conditions.
3. **Binning and Trimming:** The target variable 'Primary Contributory Cause' was binned into fewer categories to address the issue of class imbalance and to simplify the modeling process.
4. **Data Splitting:** The data was split into training and test sets to evaluate model performance.

We shall begin by first understanding the dataset, that is by loading the dataset and defining the first few columns.

Load the Dataset

In [4]:

```
import pandas as pd
df = pd.read_csv("Traffic_Crashes_-_Crashes_20240824.csv")
print(df.head())
print(df.info())
print(df.describe())
```

```
                                     CRASH_RECORD_ID CRASH_DATE_EST_I  \
0  23a79931ef555d54118f64dc9be2cf2dbf59636ce253f7...              NaN
1  2675c13fd0f474d730a5b780968b3cafc7c12d7adb661f...              NaN
2  5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4...              NaN
3  7ebf015016f83d09b321afd671a836d6b148330535d5df...              NaN
4  6c1659069e9c6285a650e70d6f9b574ed5f64c12888479...              NaN

               CRASH_DATE  POSTED_SPEED_LIMIT TRAFFIC_CONTROL_DEVICE  \
0  09/05/2023 07:05:00 PM                  30         TRAFFIC SIGNAL
1  09/22/2023 06:45:00 PM                  50            NO CONTROLS
2  07/29/2023 02:45:00 PM                  30         TRAFFIC SIGNAL
3  08/09/2023 11:00:00 PM                  30            NO CONTROLS
4  08/18/2023 12:50:00 PM                  15                  OTHER

      DEVICE_CONDITION WEATHER_CONDITION      LIGHTING_CONDITION  \
0  FUNCTIONING PROPERLY             CLEAR                    DUSK
1           NO CONTROLS             CLEAR  DARKNESS, LIGHTED ROAD
2  FUNCTIONING PROPERLY             CLEAR                DAYLIGHT
3           NO CONTROLS             CLEAR  DARKNESS, LIGHTED ROAD
4  FUNCTIONING PROPERLY             CLEAR                DAYLIGHT

         FIRST_CRASH_TYPE                    TRAFFICWAY_TYPE  ...  \
0                   ANGLE               FIVE POINT, OR MORE  ...
1                REAR END        DIVIDED - W/MEDIAN BARRIER  ...
2     PARKED MOTOR VEHICLE  DIVIDED - W/MEDIAN (NOT RAISED)  ...
3  SIDESWIPE SAME DIRECTION                      NOT DIVIDED  ...
```

```
4                  REAR END                             OTHER  ...

   INJURIES_NON_INCAPACITATING INJURIES_REPORTED_NOT_EVIDENT  \
0                          2.0                            0.0
1                          0.0                            0.0
2                          0.0                            0.0
3                          0.0                            0.0
4                          1.0                            0.0

  INJURIES_NO_INDICATION INJURIES_UNKNOWN CRASH_HOUR CRASH_DAY_OF_WEEK  \
0                    2.0              0.0       19.0               3.0
1                    2.0              0.0       18.0               6.0
2                    1.0              0.0       14.0               7.0
3                    2.0              0.0       23.0               4.0
4                    1.0              0.0       12.0               6.0

  CRASH_MONTH  LATITUDE  LONGITUDE                                LOCATION
0         9.0       NaN        NaN                                     NaN
1         9.0       NaN        NaN                                     NaN
2         7.0  41.85412 -87.665902  POINT (-87.665902342962 41.854120262952)
3         8.0       NaN        NaN                                     NaN
4         8.0       NaN        NaN                                     NaN

[5 rows x 48 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27887 entries, 0 to 27886
Data columns (total 48 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CRASH_RECORD_ID                27887 non-null  object
 1   CRASH_DATE_EST_I               1882 non-null   object
 2   CRASH_DATE                     27887 non-null  object
 3   POSTED_SPEED_LIMIT             27887 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE         27887 non-null  object
 5   DEVICE_CONDITION               27887 non-null  object
 6   WEATHER_CONDITION              27887 non-null  object
 7   LIGHTING_CONDITION             27887 non-null  object
 8   FIRST_CRASH_TYPE               27887 non-null  object
 9   TRAFFICWAY_TYPE                27887 non-null  object
 10  LANE_CNT                       2920 non-null   float64
 11  ALIGNMENT                      27887 non-null  object
 12  ROADWAY_SURFACE_COND           27887 non-null  object
 13  ROAD_DEFECT                    27887 non-null  object
 14  REPORT_TYPE                    26859 non-null  object
 15  CRASH_TYPE                     27887 non-null  object
 16  INTERSECTION_RELATED_I         6130 non-null   object
 17  NOT_RIGHT_OF_WAY_I             1197 non-null   object
 18  HIT_AND_RUN_I                  8314 non-null   object
 19  DAMAGE                         27886 non-null  object
 20  DATE_POLICE_NOTIFIED           27886 non-null  object
 21  PRIM_CONTRIBUTORY_CAUSE        27886 non-null  object
 22  SEC_CONTRIBUTORY_CAUSE         27886 non-null  object
 23  STREET_NO                      27886 non-null  float64
 24  STREET_DIRECTION               27883 non-null  object
 25  STREET_NAME                    27885 non-null  object
 26  BEAT_OF_OCCURRENCE             27884 non-null  float64
 27  PHOTOS_TAKEN_I                 426 non-null    object
 28  STATEMENTS_TAKEN_I             767 non-null    object
 29  DOORING_I                      75 non-null     object
 30  WORK_ZONE_I                    137 non-null    object
 31  WORK_ZONE_TYPE                 108 non-null    object
 32  WORKERS_PRESENT_I              47 non-null     object
 33  NUM_UNITS                      27886 non-null  float64
 34  MOST_SEVERE_INJURY             27820 non-null  object
 35  INJURIES_TOTAL                 27820 non-null  float64
 36  INJURIES_FATAL                 27820 non-null  float64
 37  INJURIES_INCAPACITATING        27820 non-null  float64
 38  INJURIES_NON_INCAPACITATING    27820 non-null  float64
 39  INJURIES_REPORTED_NOT_EVIDENT  27820 non-null  float64
 40  INJURIES_NO_INDICATION         27820 non-null  float64
 41  INJURIES_UNKNOWN               27820 non-null  float64
 42  CRASH_HOUR                     27886 non-null  float64
```

```
 43  CRASH_DAY_OF_WEEK              27886 non-null  float64
 44  CRASH_MONTH                    27886 non-null  float64
 45  LATITUDE                       22233 non-null  float64
 46  LONGITUDE                      22233 non-null  float64
 47  LOCATION                       22233 non-null  object
dtypes: float64(16), int64(1), object(31)
memory usage: 10.2+ MB
None
       POSTED_SPEED_LIMIT     LANE_CNT      STREET_NO  BEAT_OF_OCCURRENCE  \
count        27887.000000  2920.000000   27886.000000        27884.000000
mean            28.161437     2.814384    3539.972172         1262.486049
std              6.791089     3.270881    4034.469241          685.325215
min              0.000000     0.000000       0.000000          111.000000
25%             30.000000     2.000000     809.000000          731.000000
50%             30.000000     2.000000    2929.000000         1231.000000
75%             30.000000     4.000000    5501.000000         1722.000000
max             70.000000    99.000000  451100.000000        6100.000000

          NUM_UNITS  INJURIES_TOTAL  INJURIES_FATAL  INJURIES_INCAPACITATING  \
count  27886.000000    27820.000000    27820.000000             27820.000000
mean       2.027290        0.202193        0.001258                 0.017577
std        0.467415        0.581342        0.037421                 0.152434
min        1.000000        0.000000        0.000000                 0.000000
25%        2.000000        0.000000        0.000000                 0.000000
50%        2.000000        0.000000        0.000000                 0.000000
75%        2.000000        0.000000        0.000000                 0.000000
max       16.000000       15.000000        2.000000                 5.000000

       INJURIES_NON_INCAPACITATING  INJURIES_REPORTED_NOT_EVIDENT  \
count                 27820.000000                   27820.000000
mean                      0.110388                       0.072969
std                       0.431929                       0.337879
min                       0.000000                       0.000000
25%                       0.000000                       0.000000
50%                       0.000000                       0.000000
75%                       0.000000                       0.000000
max                       8.000000                      10.000000

       INJURIES_NO_INDICATION  INJURIES_UNKNOWN    CRASH_HOUR  \
count            27820.000000           27820.0  27886.000000
mean                 2.003379               0.0     13.135875
std                  1.149204               0.0      5.615733
min                  0.000000               0.0      0.000000
25%                  1.000000               0.0      9.000000
50%                  2.000000               0.0     14.000000
75%                  2.000000               0.0     17.000000
max                 29.000000               0.0     23.000000

       CRASH_DAY_OF_WEEK   CRASH_MONTH      LATITUDE     LONGITUDE
count       27886.000000  27886.000000  22233.000000  22233.000000
mean            4.109015      4.927060     41.772615    -87.499103
std             1.999360      3.054954      1.904045      3.984663
min             1.000000      1.000000      0.000000    -87.933994
25%             2.000000      3.000000     41.785117    -87.722183
50%             4.000000      4.000000     41.876910    -87.676788
75%             6.000000      7.000000     41.925269    -87.635370
max             7.000000     12.000000     42.022421      0.000000
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['CRASH_RECORD_ID', 'CRASH_DATE_EST_I', 'CRASH_DATE',
       'POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE', 'LANE_CNT', 'ALIGNMENT', 'ROADWAY_SURFACE_COND',
       'ROAD_DEFECT', 'REPORT_TYPE', 'CRASH_TYPE', 'INTERSECTION_RELATED_I',
       'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'DAMAGE', 'DATE_POLICE_NOTIFIED',
       'PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO',
       'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRENCE',
       'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I',
```

```
        'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I',
        'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I', 'NUM_UNITS',
        'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_FATAL',
        'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
        'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION',
        'INJURIES_UNKNOWN', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
        'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
```

Check for missing values

In [6]:

```
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
CRASH_DATE_EST_I                 26005
LANE_CNT                         24967
REPORT_TYPE                       1028
INTERSECTION_RELATED_I           21757
NOT_RIGHT_OF_WAY_I               26690
HIT_AND_RUN_I                    19573
DAMAGE                               1
DATE_POLICE_NOTIFIED                 1
PRIM_CONTRIBUTORY_CAUSE              1
SEC_CONTRIBUTORY_CAUSE               1
STREET_NO                            1
STREET_DIRECTION                     4
STREET_NAME                          2
BEAT_OF_OCCURRENCE                   3
PHOTOS_TAKEN_I                   27461
STATEMENTS_TAKEN_I               27120
DOORING_I                        27812
WORK_ZONE_I                      27750
WORK_ZONE_TYPE                   27779
WORKERS_PRESENT_I                27840
NUM_UNITS                            1
MOST_SEVERE_INJURY                  67
INJURIES_TOTAL                      67
INJURIES_FATAL                      67
INJURIES_INCAPACITATING             67
INJURIES_NON_INCAPACITATING         67
INJURIES_REPORTED_NOT_EVIDENT       67
INJURIES_NO_INDICATION              67
INJURIES_UNKNOWN                    67
CRASH_HOUR                           1
CRASH_DAY_OF_WEEK                    1
CRASH_MONTH                          1
LATITUDE                          5654
LONGITUDE                         5654
LOCATION                          5654
dtype: int64
```

Check for duplicates

In [7]:

```
duplicate_rows = df[df.duplicated()]
print(duplicate_rows)
```

```
Empty DataFrame
Columns: [CRASH_RECORD_ID, CRASH_DATE_EST_I, CRASH_DATE, POSTED_SPEED_LIMIT, TRAFFIC_CONT
ROL_DEVICE, DEVICE_CONDITION, WEATHER_CONDITION, LIGHTING_CONDITION, FIRST_CRASH_TYPE, TR
AFFICWAY_TYPE, LANE_CNT, ALIGNMENT, ROADWAY_SURFACE_COND, ROAD_DEFECT, REPORT_TYPE, CRASH
_TYPE, INTERSECTION_RELATED_I, NOT_RIGHT_OF_WAY_I, HIT_AND_RUN_I, DAMAGE, DATE_POLICE_NOT
IFIED, PRIM_CONTRIBUTORY_CAUSE, SEC_CONTRIBUTORY_CAUSE, STREET_NO, STREET_DIRECTION, STRE
ET_NAME, BEAT_OF_OCCURRENCE, PHOTOS_TAKEN_I, STATEMENTS_TAKEN_I, DOORING_I, WORK_ZONE_I,
WORK_ZONE_TYPE, WORKERS_PRESENT_I, NUM_UNITS, MOST_SEVERE_INJURY, INJURIES_TOTAL, INJURIE
S_FATAL, INJURIES_INCAPACITATING, INJURIES_NON_INCAPACITATING, INJURIES_REPORTED_NOT_EVID
ENT, INJURIES_NO_INDICATION, INJURIES_UNKNOWN, CRASH_HOUR, CRASH_DAY_OF_WEEK, CRASH_MONTH
, LATITUDE, LONGITUDE, LOCATION]
```

```
Index: []

[0 rows x 48 columns]
```

Handling missing values

We have seen that our dataset has no duplicates whilst there are crucial missing values in features that are realted to our target variable, therefore we will handle the missing values in the following cells by employing the right procedures

In [8]:

```
df['WEATHER_CONDITION'].fillna(df['WEATHER_CONDITION'].mode()[0], inplace=True)
```

In [9]:

```
df['LIGHTING_CONDITION'].fillna(df['LIGHTING_CONDITION'].mode()[0], inplace=True)
```

In [10]:

```
df['ROADWAY_SURFACE_COND'].fillna(df['ROADWAY_SURFACE_COND'].mode()[0], inplace=True)
```

In [11]:

```
df['TRAFFIC_CONTROL_DEVICE'].fillna(df['TRAFFIC_CONTROL_DEVICE'].mode()[0], inplace=True
)
```

In [12]:

```
df['DEVICE_CONDITION'].fillna(df['DEVICE_CONDITION'].mode()[0], inplace=True)
```

In [13]:

```
df['DEVICE_CONDITION'].fillna(df['DEVICE_CONDITION'].mode()[0], inplace=True)
```

In [14]:

```
df['INTERSECTION_RELATED_I'].fillna(df['INTERSECTION_RELATED_I'].mode()[0], inplace=True
)
```

In [15]:

```
df['NOT_RIGHT_OF_WAY_I'].fillna(df['NOT_RIGHT_OF_WAY_I'].mode()[0], inplace=True)
```

In [16]:

```
df['MOST_SEVERE_INJURY'].fillna(df['MOST_SEVERE_INJURY'].mode()[0], inplace=True)
```

In [17]:

```
df.drop(['PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I',
         'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I', 'STREET_NO', 'STREET_DIRECTION',
         'STREET_NAME', 'BEAT_OF_OCCURRENCE','CRASH_DATE_EST_I'], axis=1, inplace=True)
```

In [18]:

```
df['HIT_AND_RUN_I'].fillna(df['HIT_AND_RUN_I'].mode()[0], inplace=True)
```

In [19]:

```
df['LANE_CNT'].fillna(df['LANE_CNT'].median(), inplace=True)
df['LATITUDE'].fillna(df['LATITUDE'].median(), inplace=True)
df['LONGITUDE'].fillna(df['LONGITUDE'].median(), inplace=True)
```

In [20]:

```
injury_columns = [
    'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING',
```

```
    'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT',
    'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN'
]
df[injury_columns] = df[injury_columns].fillna(0)
```

In [21]:

```
df['LOCATION'].fillna(method='ffill', inplace=True)
```

In [22]:

```
df['REPORT_TYPE'].fillna(df['REPORT_TYPE'].mode()[0], inplace=True)
```

In [23]:

```
df['LOCATION'].fillna(method='bfill', inplace=True)
```

Check if any crucial missing values are still there

In [24]:

```
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
DAMAGE                     1
DATE_POLICE_NOTIFIED       1
PRIM_CONTRIBUTORY_CAUSE    1
SEC_CONTRIBUTORY_CAUSE     1
NUM_UNITS                  1
CRASH_HOUR                 1
CRASH_DAY_OF_WEEK          1
CRASH_MONTH                1
dtype: int64
```

Feature engineering, Binning and Trimming.

Our Dataset, unfortunately, has already feature engineered features such as CRASH_HOUR, CRASH_MONTH etc, therefore we will use a model to view top 10 features that are related to our target variable.

In the cell below, we will run a Logistic Regression model with Preprocessing Pipelines because of our alrge dataset. This code will also help in preparing categorical and numerical features for modeling.

In [25]:

```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder


df_sampled = df.sample(frac=0.01, random_state=42)

X = df_sampled.drop('PRIM_CONTRIBUTORY_CAUSE', axis=1)
y = df_sampled['PRIM_CONTRIBUTORY_CAUSE']


categorical_cols = X.select_dtypes(include=['object']).columns
numerical_cols = X.select_dtypes(exclude=['object']).columns


preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='median'), numerical_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
```

```
        ]), categorical_cols)
    ])


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)


X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

model = LogisticRegression(max_iter=100, random_state=42, n_jobs=-1)
model.fit(X_train, y_train)
```

Out[25]:

```
▾             LogisticRegression
LogisticRegression(n_jobs=-1, random_state=42)
```

From the trained Logistic Regression Model above, let us extract coefficients by manually creating features then analyze the importance of each feature.

In [26]:

```
import numpy as np
import pandas as pd

feature_names = []

for col in numerical_cols:
    feature_names.append(col)

for col in categorical_cols:
    categories = preprocessor.named_transformers_['cat']['onehot'].categories_[categoric
al_cols.get_loc(col)]
    feature_names.extend([f"{col}_{category}" for category in categories])

coefficients = model.coef_[0]

importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients
})

importance_df['Abs_Coefficient'] = np.abs(importance_df['Coefficient'])
importance_df = importance_df.sort_values(by='Abs_Coefficient', ascending=False)

print(importance_df.head(20))
```

```
                                              Feature  Coefficient  \
12                                        CRASH_MONTH    -0.266921
10                                         CRASH_HOUR    -0.232181
504                           TRAFFICWAY_TYPE_FOUR WAY     0.159918
486                             FIRST_CRASH_TYPE_ANGLE     0.156497
0                                   POSTED_SPEED_LIMIT     0.156204
529                  CRASH_TYPE_NO INJURY / DRIVE AWAY    -0.130294
528        CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH     0.129192
465              TRAFFIC_CONTROL_DEVICE_TRAFFIC SIGNAL     0.113737
11                                    CRASH_DAY_OF_WEEK     0.107847
469              DEVICE_CONDITION_FUNCTIONING PROPERLY     0.103109
184   CRASH_RECORD_ID_c66897bca4ab9f026d7034fcfea270...     0.094645
604         DATE_POLICE_NOTIFIED_03/13/2024 12:50:00 PM     0.094645
304                   CRASH_DATE_03/13/2024 12:50:00 PM     0.094645
796   LOCATION_POINT (-87.605487366182 41.756663577514)     0.094645
310                   CRASH_DATE_03/15/2024 10:53:00 AM     0.093899
38    CRASH_RECORD_ID_198cb51dc4fd44fe63875b7e8e9d04...     0.093899
609         DATE_POLICE_NOTIFIED_03/15/2024 10:55:00 AM     0.093899
793    LOCATION_POINT (-87.59328924117 41.756893037294)     0.093899
```

```
2                                            NUM_UNITS     0.090924
470                         DEVICE_CONDITION_NO CONTROLS    -0.089080

     Abs_Coefficient
12          0.266921
10          0.232181
504         0.159918
486         0.156497
0           0.156204
529         0.130294
528         0.129192
465         0.113737
11          0.107847
469         0.103109
184         0.094645
604         0.094645
304         0.094645
796         0.094645
310         0.093899
38          0.093899
609         0.093899
793         0.093899
2           0.090924
470         0.089080
```

Why the code above;

Feature Names Construction:
Thecodereconstructsthefeaturenamesafterpreprocessing, including one-hot encoding for categorical variables.

Model Coefficients: It extracts the coefficients from the trained logistic regression model to understand how each feature contributes to the prediction.

Feature Importance: By sorting the coefficients by their absolute values, it identifies which features are most important in the model, providing insights into the key drivers of the target variable.

Let us transform time-related features into a format that better represents their cyclic nature, potentially improving the performance and interpretability our machine learning models

In [27]:

```
import numpy as np

df['CRASH_HOUR_sin'] = np.sin(2 * np.pi * df['CRASH_HOUR'] / 24)
df['CRASH_HOUR_cos'] = np.cos(2 * np.pi * df['CRASH_HOUR'] / 24)

df['CRASH_DAY_OF_WEEK_sin'] = np.sin(2 * np.pi * df['CRASH_DAY_OF_WEEK'] / 7)
df['CRASH_DAY_OF_WEEK_cos'] = np.cos(2 * np.pi * df['CRASH_DAY_OF_WEEK'] / 7)

df['CRASH_MONTH_sin'] = np.sin(2 * np.pi * df['CRASH_MONTH'] / 12)
df['CRASH_MONTH_cos'] = np.cos(2 * np.pi * df['CRASH_MONTH'] / 12)
```

Improving model predictive power

There are some engineered features that need to be added to our raw features so as to enhance the model's ability to predict the primary contributory cause of traffic accidents by incorporating important contextual information.

Why These Features?

Is_Weekend:

Weekends often have different traffic patterns compared to weekdays, which can influence the likelihood and types of crashes. By explicitly capturing whether a crash occurred on a weekend, the model can learn to differentiate between weekday and weekend crashes. Speed_Weather_Interaction:

This interaction term is valuable because the effect of speed limits on crashes might differ depending on the weather. For example, a high speed limit might be particularly dangerous in poor weather conditions. This feature allows the model to capture that relationship.

In [28]:

```
df['Speed_Weather_Interaction'] = df['POSTED_SPEED_LIMIT'] * df['WEATHER_CONDITION'].fac
torize()[0]
```

In [29]:

```
df['Is_Weekend'] = df['CRASH_DAY_OF_WEEK'].apply(lambda x: 1 if x in [6, 7] else 0)
```

In the cell below, let us check the frequency of causes, so we can se how closely each feature is related to the target variable

In [30]:

```
cause_counts = df['PRIM_CONTRIBUTORY_CAUSE'].value_counts()
print(cause_counts)
```

```
PRIM_CONTRIBUTORY_CAUSE
UNABLE TO DETERMINE                                                                      109
63
FAILING TO YIELD RIGHT-OF-WAY                                                             322
0
FOLLOWING TOO CLOSELY                                                                     262
6
NOT APPLICABLE                                                                            15
15
IMPROPER OVERTAKING/PASSING                                                               144
7
IMPROPER LANE USAGE                                                                       10
66
FAILING TO REDUCE SPEED TO AVOID CRASH                                                    106
2
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE                                                       104
2
IMPROPER BACKING                                                                          10
05
IMPROPER TURNING/NO SIGNAL                                                                 98
4
DISREGARDING TRAFFIC SIGNALS                                                               56
8
WEATHER                                                                                    3
83
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE MANNER           326
DISREGARDING STOP SIGN                                                                     2
37
DISTRACTION - FROM INSIDE VEHICLE                                                          16
5
DRIVING ON WRONG SIDE/WRONG WAY                                                            16
3
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)                                       16
1
EQUIPMENT - VEHICLE CONDITION                                                              15
8
PHYSICAL CONDITION OF DRIVER                                                               13
7
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)                         120
DISTRACTION - FROM OUTSIDE VEHICLE                                                          9
4
ROAD ENGINEERING/SURFACE/MARKING DEFECTS                                                    5
9
DISREGARDING OTHER TRAFFIC SIGNS                                                            4
7
```

```
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST                                    4
6
ROAD CONSTRUCTION/MAINTENANCE                                                        4
4
EXCEEDING SAFE SPEED FOR CONDITIONS                                                  3
3
EXCEEDING AUTHORIZED SPEED LIMIT                                                     3
3
DISREGARDING ROAD MARKINGS
32
ANIMAL
26
TURNING RIGHT ON RED
22
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)                                      2
2
CELL PHONE USE OTHER THAN TEXTING                                                    2
0
RELATED TO BUS STOP
19
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER, ETC.)         17
TEXTING
6
DISREGARDING YIELD SIGN
5
OBSTRUCTED CROSSWALKS
5
BICYCLE ADVANCING LEGALLY ON RED LIGHT
5
PASSING STOPPED SCHOOL BUS
3
Name: count, dtype: int64
```

Binning Rare Causes: Helps to avoid overfitting by reducing the number of categories the model must distinguish between, especially when some categories have very few samples.

.

In [31]:

```
threshold = 0.01 * len(df)
rare_causes = cause_counts[cause_counts < threshold].index
df['PRIM_CONTRIBUTORY_CAUSE_BINNED'] = df['PRIM_CONTRIBUTORY_CAUSE'].replace(rare_causes
, 'Other')
```

Filtering to Top N Causes: Keeps the focus on the most common causes, which are likely to be the most important for your analysis.

In [32]:

```
top_n_causes = df['PRIM_CONTRIBUTORY_CAUSE_BINNED'].value_counts().head(10).index
df = df[df['PRIM_CONTRIBUTORY_CAUSE_BINNED'].isin(top_n_causes)]
```

Mode-Based Feature: Adds a contextual feature that can help the model understand how weather conditions correlate with speed limits.

In [33]:

```
df['Weather_Condition_Mode'] = df.groupby('POSTED_SPEED_LIMIT')['WEATHER_CONDITION'].tra
nsform(lambda x: x.mode()[0])
```

Dropping Unnecessary Columns: Simplifies the dataset by removing features that have been replaced by more informative engineered features.

In [34]:

```
df.drop(['CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH', 'POSTED_SPEED_LIMIT'], axis=1
```

```
, inplace=True)
```

In [35]:

```
y = df['PRIM_CONTRIBUTORY_CAUSE_BINNED']
```

Because we had already define our x and y, we will have to redefine so that the target variable will be in line wit the new created features.

In [36]:

```
X = df.drop(['PRIM_CONTRIBUTORY_CAUSE', 'PRIM_CONTRIBUTORY_CAUSE_BINNED'], axis=1)
y = df['PRIM_CONTRIBUTORY_CAUSE_BINNED']
```

Exploratory Data Analysis (EDA)

EDA was conducted to gain insights into the data distribution and relationships between features and the target variable:

- **Univariate Analysis: Histograms and box plots were used to examine the distribution of numerical features and their relationship with the target variable.**
- **Bivariate Analysis: Pair plots and heatmaps were employed to explore correlations between features and how they relate to the 'Primary Contributory Cause.'**
- **Multivariate Analysis: Advanced visualization techniques such as interaction plots were used to understand how combinations of features impact accident outcomes.**

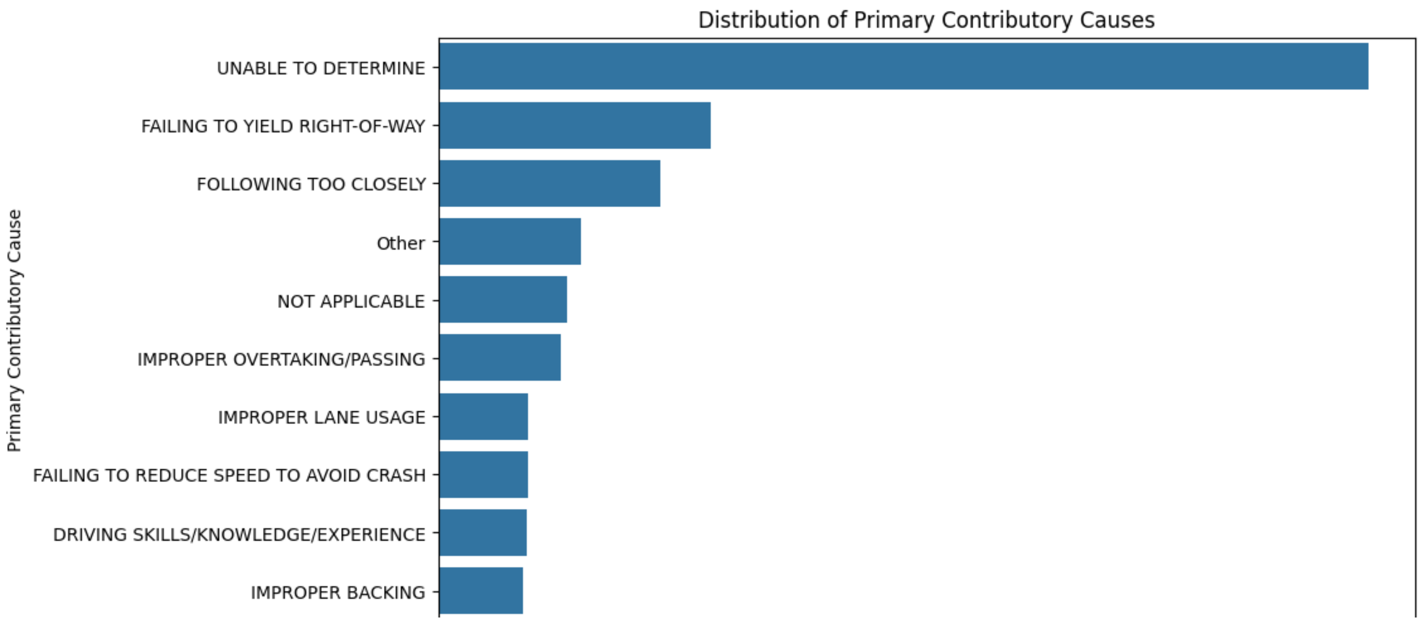
These analyses helped identify important patterns and informed the feature engineering process.

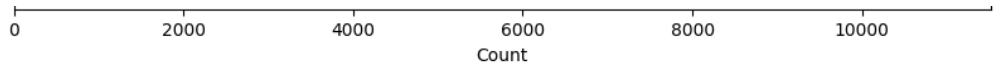
Univariate Analysis:

Histograms and box plots were used to examine the distribution of numerical features and their relationship with the target variable.

In [37]:

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

plt.figure(figsize=(10, 6))
sns.countplot(y='PRIM_CONTRIBUTORY_CAUSE_BINNED', data=df, order=df['PRIM_CONTRIBUTORY_C
AUSE_BINNED'].value_counts().index)
plt.title('Distribution of Primary Contributory Causes')
plt.xlabel('Count')
plt.ylabel('Primary Contributory Cause')
plt.show()
```





Let us check the columns after performing feature engineering and binning, just to be sure we define numerical and categorical columns.

In [38]:

```
print(df.columns)
```

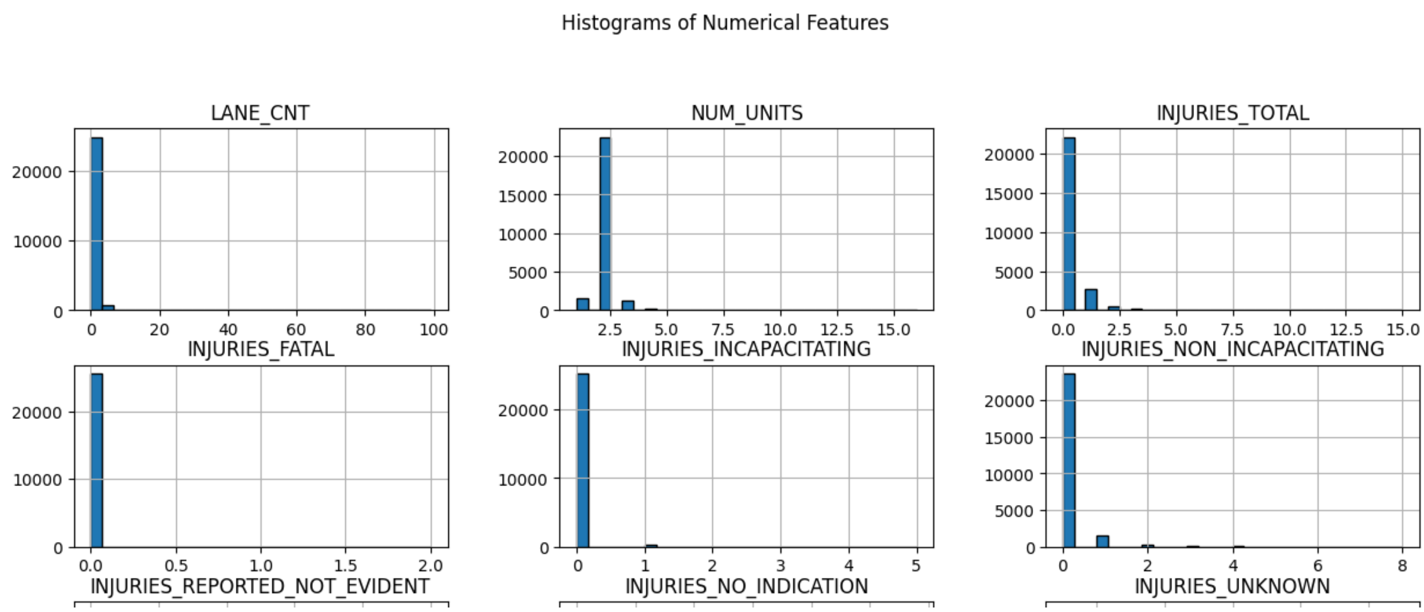
```
Index(['CRASH_RECORD_ID', 'CRASH_DATE', 'TRAFFIC_CONTROL_DEVICE',
       'DEVICE_CONDITION', 'WEATHER_CONDITION', 'LIGHTING_CONDITION',
       'FIRST_CRASH_TYPE', 'TRAFFICWAY_TYPE', 'LANE_CNT', 'ALIGNMENT',
       'ROADWAY_SURFACE_COND', 'ROAD_DEFECT', 'REPORT_TYPE', 'CRASH_TYPE',
       'INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I',
       'DAMAGE', 'DATE_POLICE_NOTIFIED', 'PRIM_CONTRIBUTORY_CAUSE',
       'SEC_CONTRIBUTORY_CAUSE', 'NUM_UNITS', 'MOST_SEVERE_INJURY',
       'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING',
       'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT',
       'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN', 'LATITUDE', 'LONGITUDE',
       'LOCATION', 'CRASH_HOUR_sin', 'CRASH_HOUR_cos', 'CRASH_DAY_OF_WEEK_sin',
       'CRASH_DAY_OF_WEEK_cos', 'CRASH_MONTH_sin', 'CRASH_MONTH_cos',
       'Speed_Weather_Interaction', 'Is_Weekend',
       'PRIM_CONTRIBUTORY_CAUSE_BINNED', 'Weather_Condition_Mode'],
      dtype='object')
```

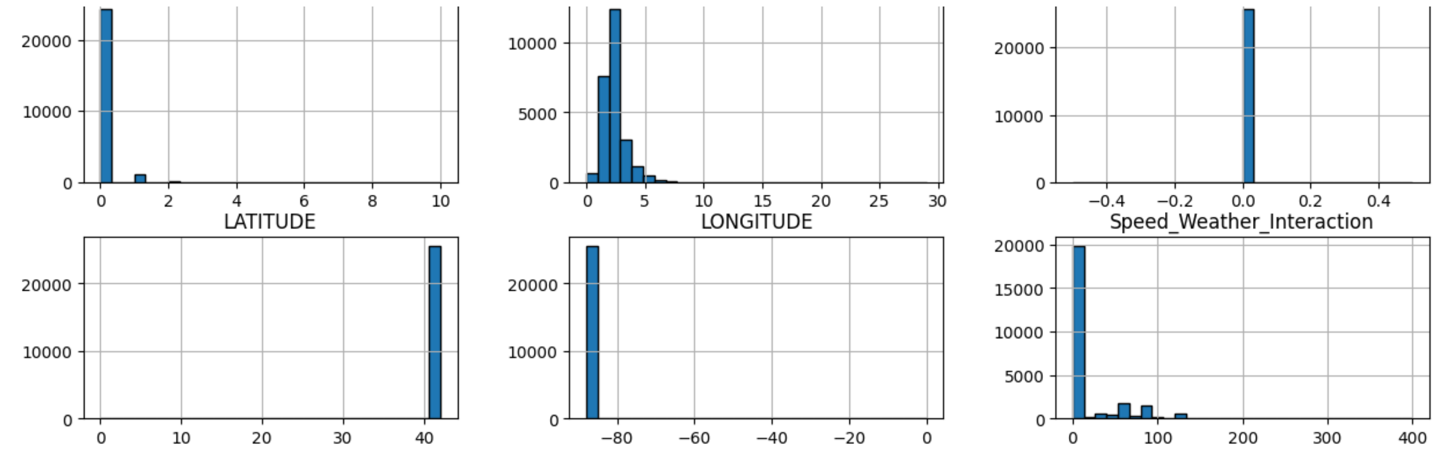
In [39]:

```
numerical_cols = [
    'LANE_CNT', 'NUM_UNITS', 'INJURIES_TOTAL', 'INJURIES_FATAL',
    'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
    'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION',
    'INJURIES_UNKNOWN', 'LATITUDE', 'LONGITUDE', 'Speed_Weather_Interaction'
]
```

In [40]:

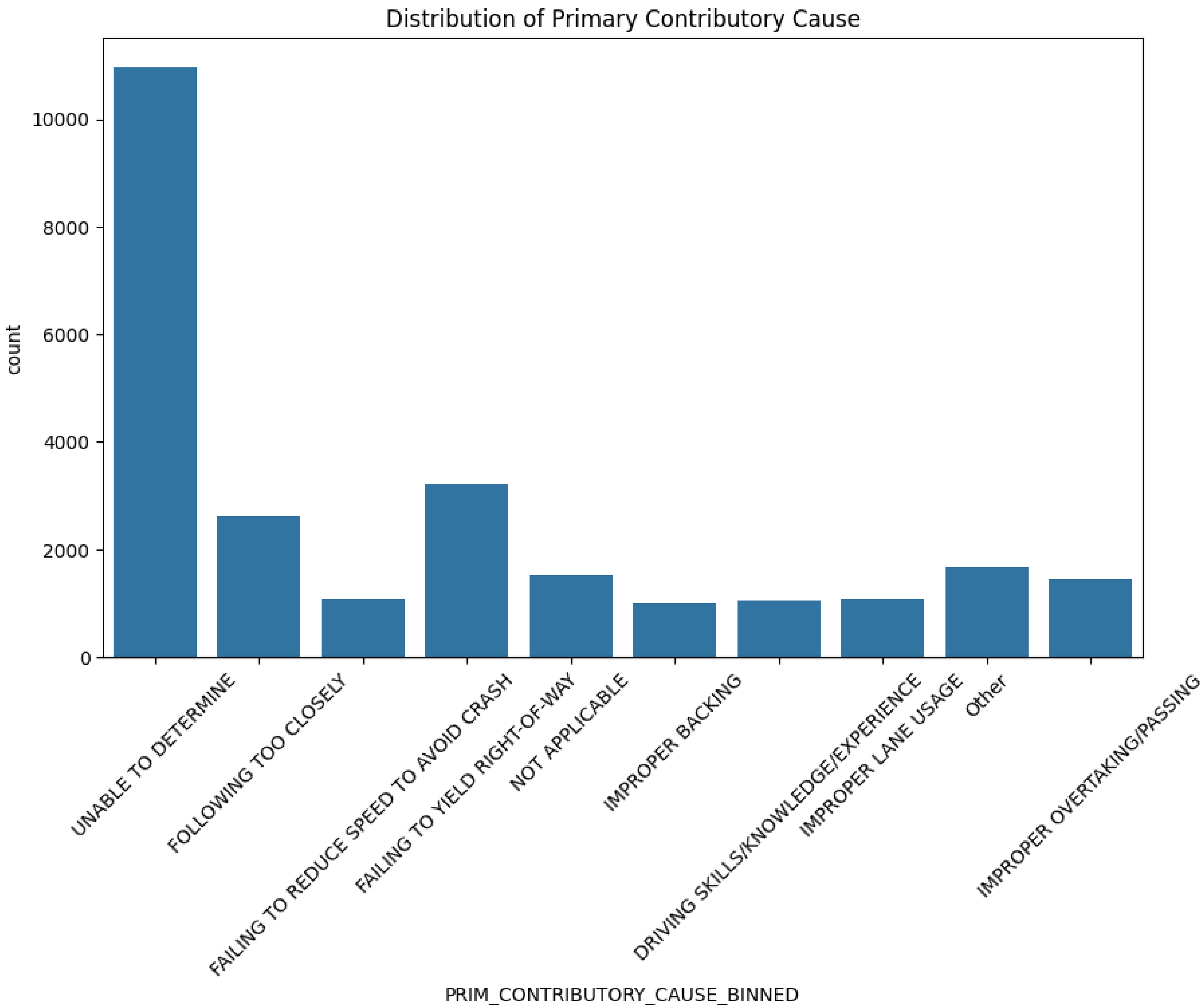
```
import matplotlib.pyplot as plt
import seaborn as sns

numerical_cols = [
    'LANE_CNT', 'NUM_UNITS', 'INJURIES_TOTAL', 'INJURIES_FATAL',
    'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
    'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION',
    'INJURIES_UNKNOWN', 'LATITUDE', 'LONGITUDE', 'Speed_Weather_Interaction'
]

df[numerical_cols].hist(figsize=(15, 10), bins=30, edgecolor='black')
plt.suptitle('Histograms of Numerical Features')
plt.show()
```





In [41]:

```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='PRIM_CONTRIBUTORY_CAUSE_BINNED')
plt.title('Distribution of Primary Contributory Cause')
plt.xticks(rotation=45)
plt.show()
```



The above visualization are explanation of the relationship between features and the target variable

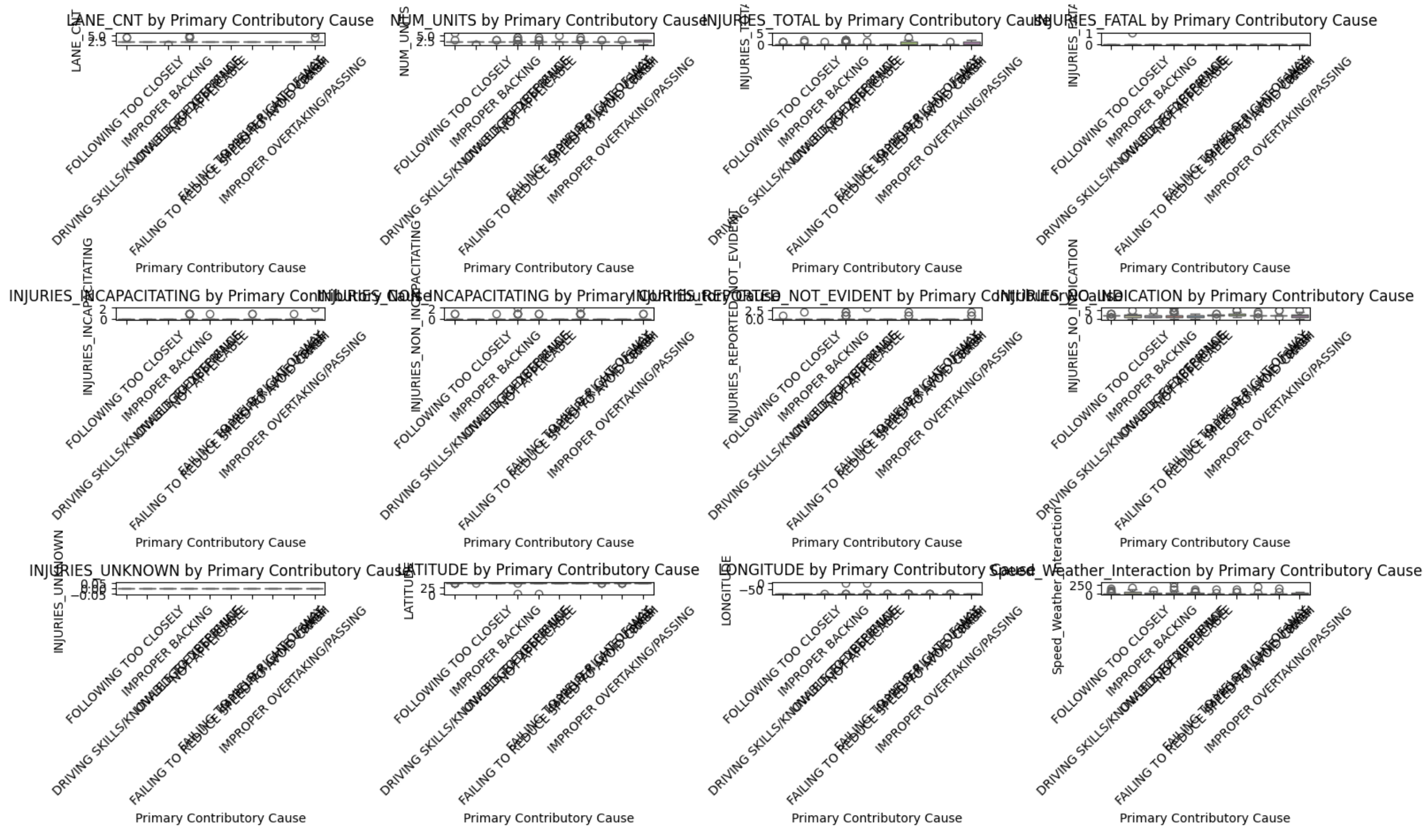
Bivariate Analysis:

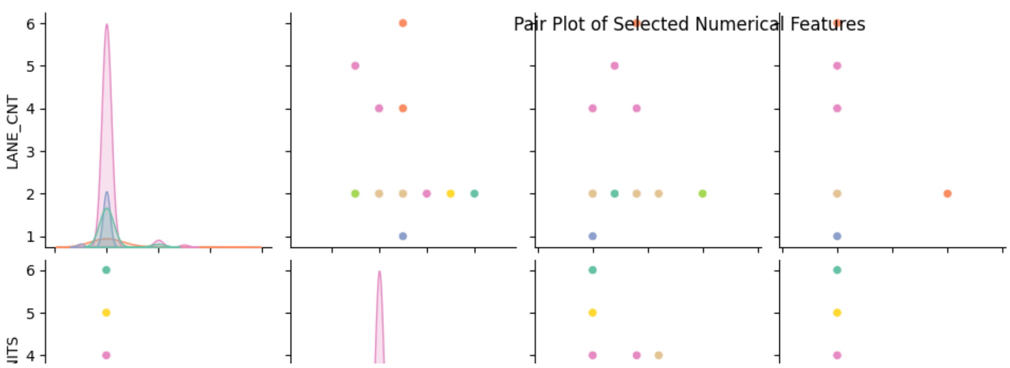
Pair plots and heatmaps were employed to explore correlations between features and how they relate to the

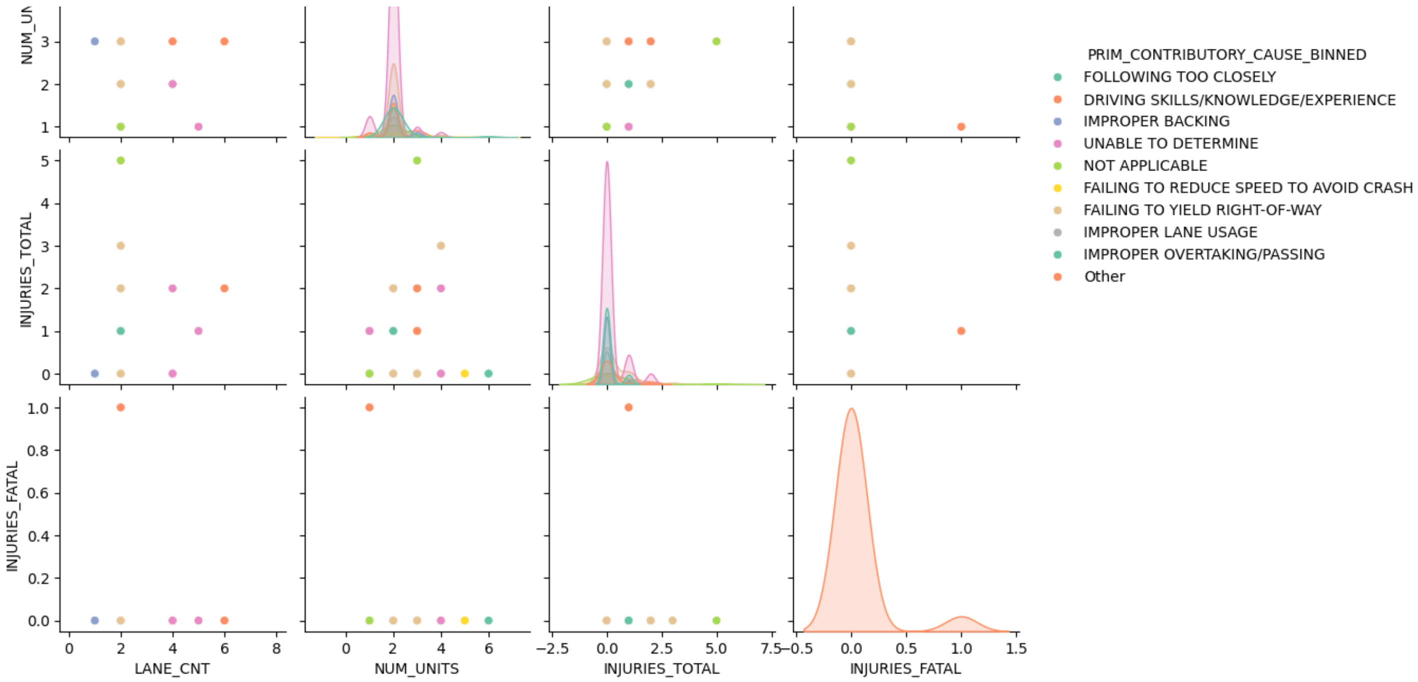
Pair plots and heatmaps were employed to explore correlations between features and how they relate to the **'Primary Contributory Cause.'**

In [42]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)


df_sampled = df.sample(frac=0.01, random_state=42)
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols[:16]):
    plt.subplot(4, 4, i + 1)
    sns.boxplot(x='PRIM_CONTRIBUTORY_CAUSE_BINNED', y=col, data=df_sampled, palette="Set
3")
    plt.title(f'{col} by Primary Contributory Cause')
    plt.xlabel('Primary Contributory Cause')
    plt.ylabel(col)
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

selected_numerical_cols = numerical_cols[:4]
sns.pairplot(df_sampled[selected_numerical_cols + ['PRIM_CONTRIBUTORY_CAUSE_BINNED']],
             hue='PRIM_CONTRIBUTORY_CAUSE_BINNED',
             diag_kind='kde',
             palette="Set2")
plt.suptitle('Pair Plot of Selected Numerical Features')
plt.show()
```

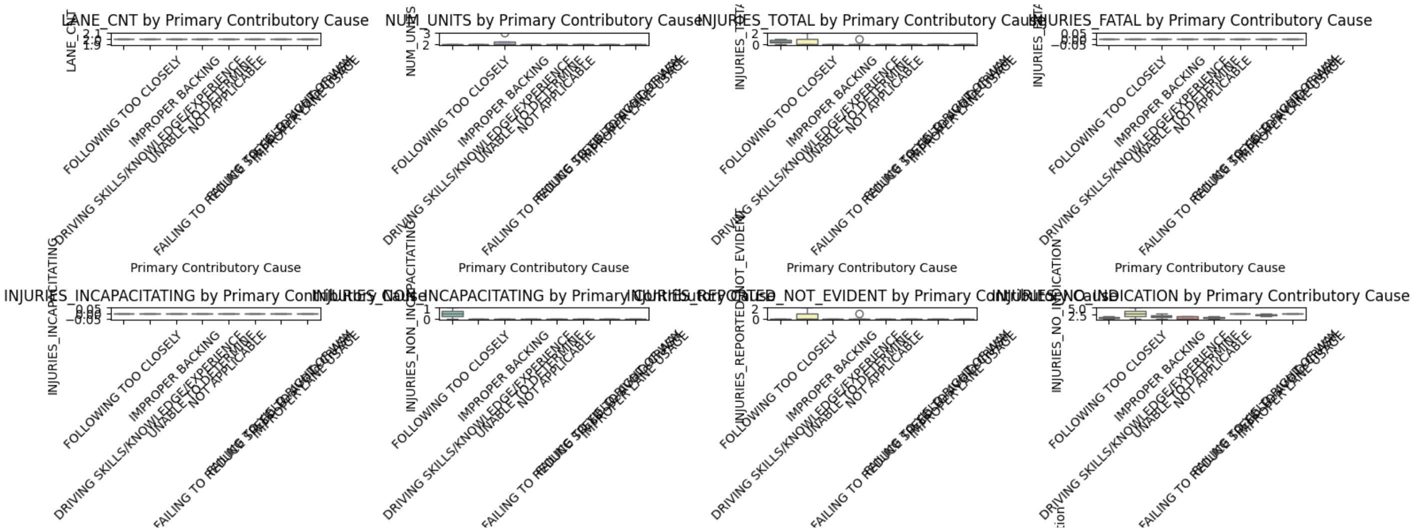


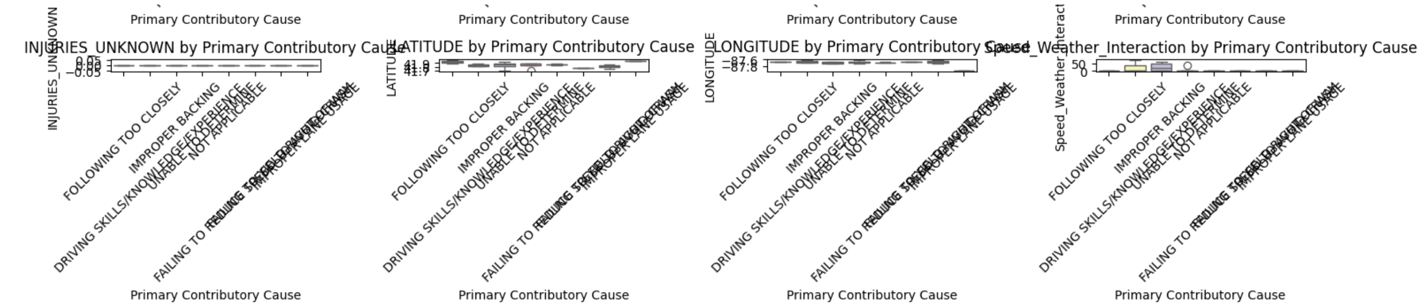


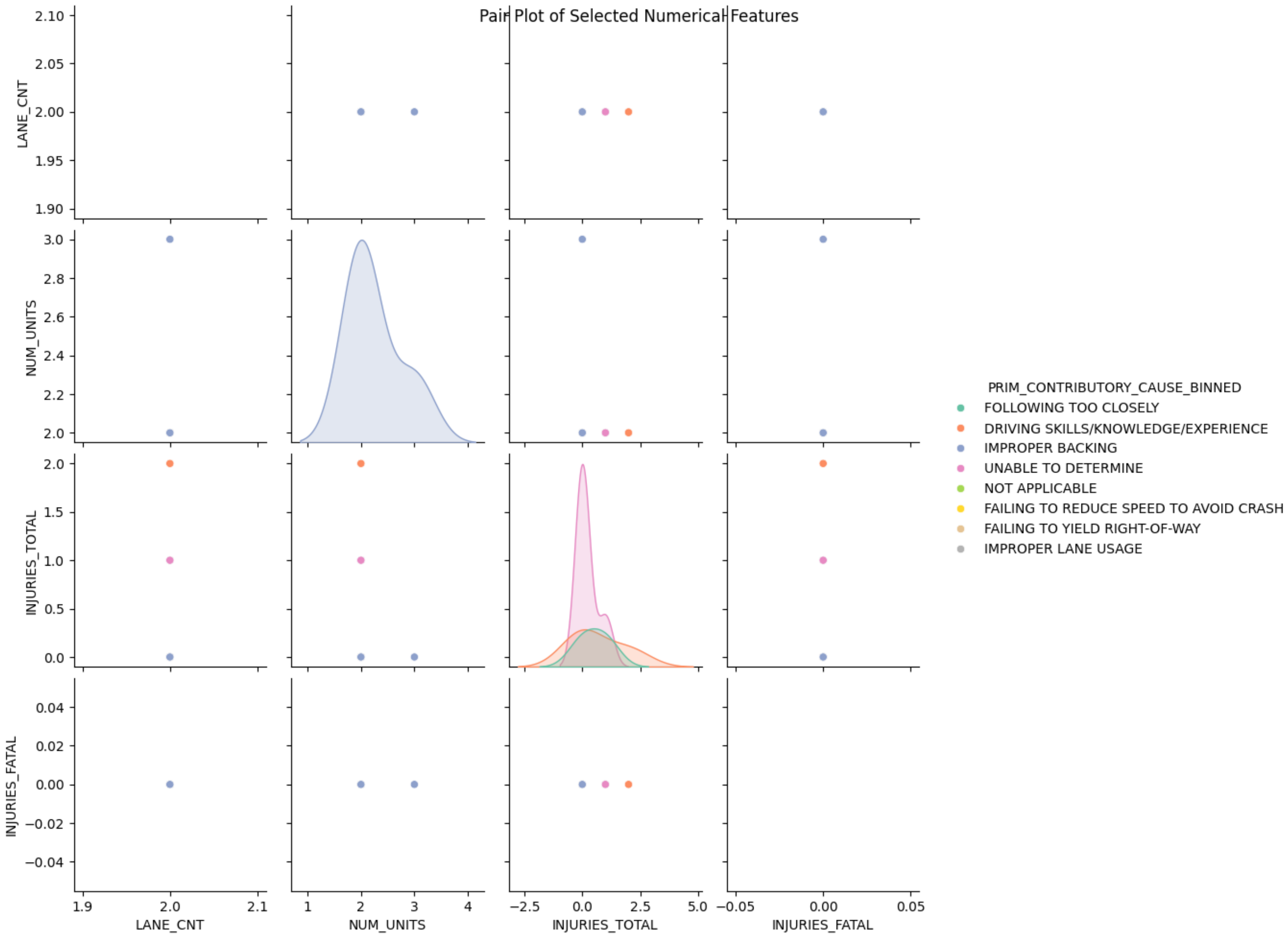


In [43]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

limited_samples = 20
df_sampled = df.sample(n=limited_samples, random_state=42)

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols[:16]):
    plt.subplot(4, 4, i + 1)
    sns.boxplot(x='PRIM_CONTRIBUTORY_CAUSE_BINNED', y=col, data=df_sampled, palette="Set
3")
    plt.title(f'{col} by Primary Contributory Cause')
    plt.xlabel('Primary Contributory Cause')
    plt.ylabel(col)
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

selected_numerical_cols = numerical_cols[:4]

sns.pairplot(df_sampled[selected_numerical_cols + ['PRIM_CONTRIBUTORY_CAUSE_BINNED']],
             hue='PRIM_CONTRIBUTORY_CAUSE_BINNED',
             diag_kind='kde',
             palette="Set2")
plt.suptitle('Pair Plot of Selected Numerical Features')
plt.show()
```





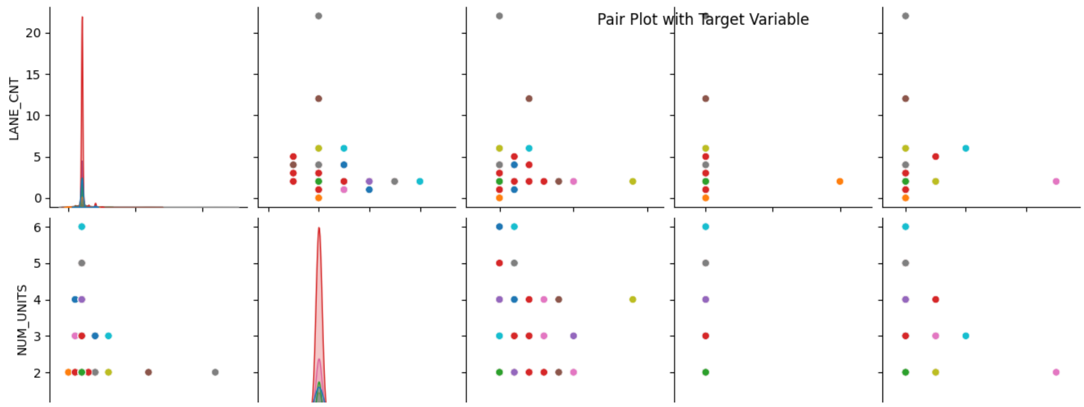


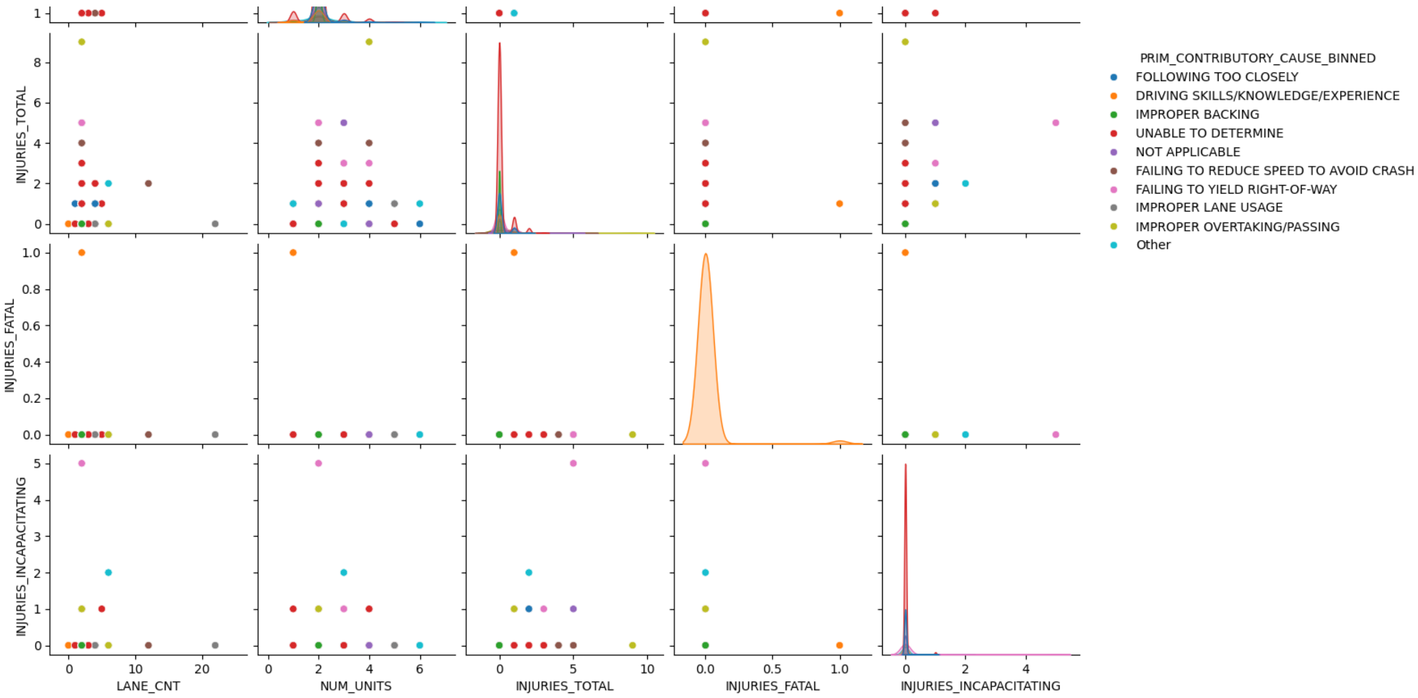
Multivariate Analysis:

Advanced visualization techniques such as interaction plots are used to understand how combinations of features impact accident outcomes

In [44]:

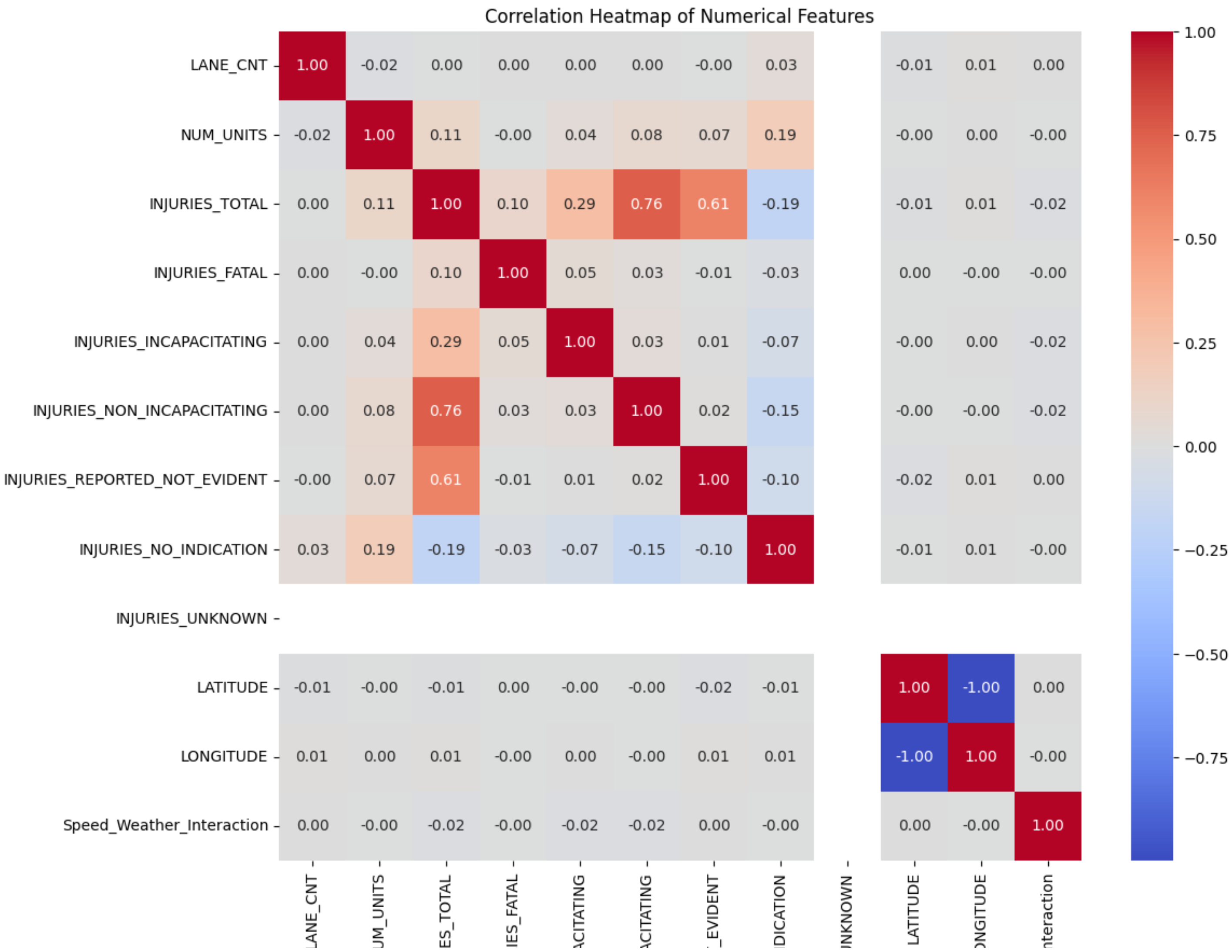
```
df_sampled = df.sample(frac=0.05, random_state=42)
sns.pairplot(df_sampled, hue='PRIM_CONTRIBUTORY_CAUSE_BINNED', vars=numerical_cols[:5])
plt.suptitle('Pair Plot with Target Variable')
plt.show()
```

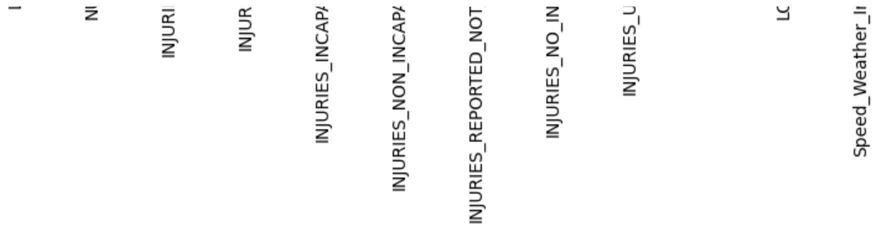


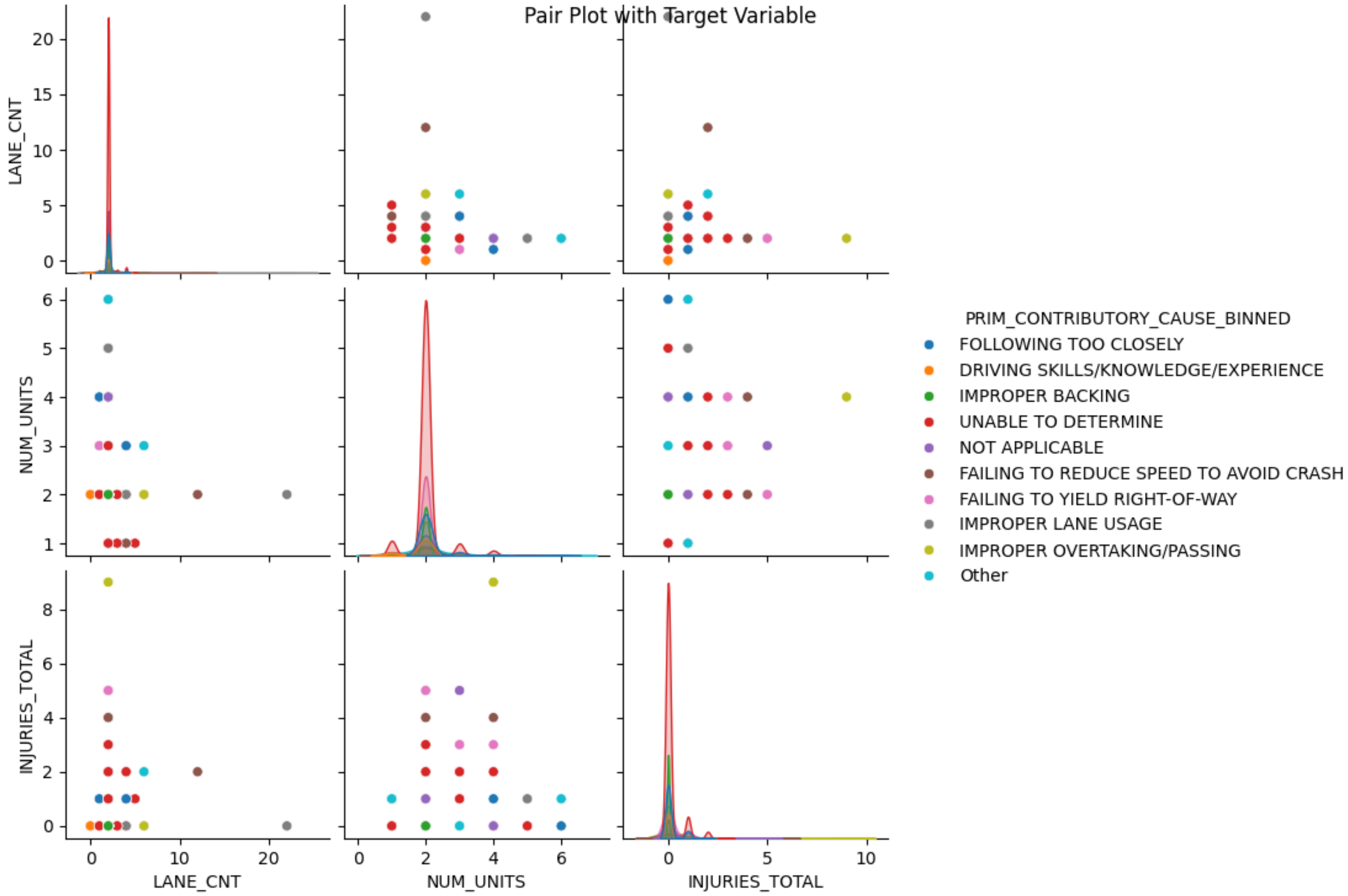


In [45]:

```python
plt.figure(figsize=(12, 10))
correlation_matrix = df[numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap of Numerical Features')
plt.show()

df_sampled = df.sample(frac=0.05, random_state=42)
sns.pairplot(df_sampled, hue='PRIM_CONTRIBUTORY_CAUSE_BINNED', vars=numerical_cols[:3])
plt.suptitle('Pair Plot with Target Variable')
plt.show()
```







The above multivariate analysis shows the relationship of combined features to our target

Modeling

Several machine learning models will be trained and evaluated, including:

1. **Logistic Regression: Used as a baseline model due to its simplicity and interpretability.**
2. **Regularized Logistic Regression (Ridge and Lasso): Applied to handle potential overfitting and improve model generalization.**
3. **Decision Trees: Leveraged for their ability to model complex interactions between features.**
4. **Random Forest: Implemented to increase predictive performance by averaging multiple decision trees.**
5. **Gradient Boosting: Employed to enhance model accuracy by combining the strengths of weak learners.**

Each model is cross-validated to ensure robustness, and hyperparameter tuning conducted to optimize performance.

Due to large data with important features we will entirely use 1% of the data, this will allow the cells to run abit fast.

In [46]:

```
# import necessary libraries for machine learning modeling

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression, RidgeClassifier, LassoCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

Define the Baseline model

In this project we will use Logistic Regression as our Baseline model. We will use pipeline preprocessor, by definning then split, train and evaluate the model.

In [47]:

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report

df_sampled = df.sample(frac=0.01, random_state=42)

X = df_sampled.drop('PRIM_CONTRIBUTORY_CAUSE', axis=1)
y = df_sampled['PRIM_CONTRIBUTORY_CAUSE']

categorical_cols = X.select_dtypes(include=['object']).columns
numerical_cols = X.select_dtypes(exclude=['object']).columns

preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='median'), numerical_cols),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ]), categorical_cols)
    ])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)

baseline_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=100, random_state=42))
])

baseline_model.fit(X_train, y_train)

y_pred = baseline_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Baseline Model Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(report)
```

```
Baseline Model Accuracy: 0.7500
Classification Report:
                                    precision    recall  f1-score   support

               DISREGARDING STOP SIGN       0.00      0.00      0.00         1
      DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       0.67      0.50      0.57         8
         FAILING TO YIELD RIGHT-OF-WAY       0.67      1.00      0.80         4
                 FOLLOWING TOO CLOSELY       0.60      0.60      0.60         5
                      IMPROPER BACKING       1.00      1.00      1.00         3
```

```
                   IMPROPER BACKING       1.00      1.00      1.00         3
               IMPROPER LANE USAGE       1.00      0.75      0.86         4
       IMPROPER OVERTAKING/PASSING       1.00      0.67      0.80         3
                    NOT APPLICABLE       0.40      0.40      0.40         5
               UNABLE TO DETERMINE       0.90      1.00      0.95        18

                          accuracy                           0.75        52
                         macro avg       0.62      0.59      0.60        52
                      weighted avg       0.75      0.75      0.74        52
```

The above output gives us the precision, F1-score and accuracy of our baseline model.

Building several models for model selection

Initial Testing

In [48]:

```python
df_sampled = df.sample(frac=0.01, random_state=42)
models = {
    'Logistic Regression': LogisticRegression(max_iter=100, random_state=42),
    'Ridge Classifier': RidgeClassifier(),
    'Lasso Logistic Regression': LogisticRegression(penalty='l1', solver='saga', max_ite
r=100, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1
),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42)
}


for name, model in models.items():
# Create a pipeline with the preprocessor and the model
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])

    cv_scores = cross_val_score(pipeline, X_train, y_train, cv=3, scoring='accuracy')


    pipeline.fit(X_train, y_train)


    y_pred = pipeline.predict(X_test)
    # Evaluate the model
    print(f'Test Set Accuracy: {accuracy_score(y_test, y_pred):.4f}')
    print(classification_report(y_test, y_pred))
    print(confusion_matrix(y_test, y_pred))
    print('\n')
```

```
Test Set Accuracy: 0.7500
                                    precision    recall  f1-score   support

               DISREGARDING STOP SIGN       0.00      0.00      0.00         1
     DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       0.67      0.50      0.57         8
       FAILING TO YIELD RIGHT-OF-WAY       0.67      1.00      0.80         4
              FOLLOWING TOO CLOSELY       0.60      0.60      0.60         5
                   IMPROPER BACKING       1.00      1.00      1.00         3
               IMPROPER LANE USAGE       1.00      0.75      0.86         4
       IMPROPER OVERTAKING/PASSING       1.00      0.67      0.80         3
                    NOT APPLICABLE       0.40      0.40      0.40         5
               UNABLE TO DETERMINE       0.90      1.00      0.95        18

                          accuracy                           0.75        52
                         macro avg       0.62      0.59      0.60        52
                      weighted avg       0.75      0.75      0.74        52

[[ 0  0  0  0  0  0  0  0  0  1]
 [ 0  0  0  0  0  0  0  0  1  0]
 [ 1  0  4  1  2  0  0  0  0  0]
```

```
 [ 0  0  0  4  0  0  0  0  0  0]
 [ 0  0  0  0  3  0  0  0  2  0]
 [ 0  0  0  0  0  3  0  0  0  0]
 [ 0  1  0  0  0  0  3  0  0  0]
 [ 0  0  1  0  0  0  0  2  0  0]
 [ 0  0  1  1  0  0  0  0  2  1]
 [ 0  0  0  0  0  0  0  0  0 18]]


Test Set Accuracy: 1.0000
                                     precision    recall  f1-score   support

             DISREGARDING STOP SIGN       1.00      1.00      1.00         1
    DRIVING ON WRONG SIDE/WRONG WAY       1.00      1.00      1.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       1.00      1.00      1.00         8
      FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00         4
              FOLLOWING TOO CLOSELY       1.00      1.00      1.00         5
                   IMPROPER BACKING       1.00      1.00      1.00         3
                IMPROPER LANE USAGE       1.00      1.00      1.00         4
        IMPROPER OVERTAKING/PASSING       1.00      1.00      1.00         3
                     NOT APPLICABLE       1.00      1.00      1.00         5
                UNABLE TO DETERMINE       1.00      1.00      1.00        18

                           accuracy                           1.00        52
                          macro avg       1.00      1.00      1.00        52
                       weighted avg       1.00      1.00      1.00        52

[[ 1  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  0  0]
 [ 0  0  8  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0]
 [ 0  0  0  0  5  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  0]
 [ 0  0  0  0  0  0  4  0  0  0]
 [ 0  0  0  0  0  0  0  3  0  0]
 [ 0  0  0  0  0  0  0  0  5  0]
 [ 0  0  0  0  0  0  0  0  0 18]]


Test Set Accuracy: 0.3462
                                     precision    recall  f1-score   support

             DISREGARDING STOP SIGN       0.00      0.00      0.00         1
    DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       0.00      0.00      0.00         8
      FAILING TO YIELD RIGHT-OF-WAY       0.00      0.00      0.00         4
              FOLLOWING TOO CLOSELY       0.00      0.00      0.00         5
                   IMPROPER BACKING       0.00      0.00      0.00         3
                IMPROPER LANE USAGE       0.00      0.00      0.00         4
        IMPROPER OVERTAKING/PASSING       0.00      0.00      0.00         3
                     NOT APPLICABLE       0.00      0.00      0.00         5
                UNABLE TO DETERMINE       0.35      1.00      0.51        18

                           accuracy                           0.35        52
                          macro avg       0.03      0.10      0.05        52
                       weighted avg       0.12      0.35      0.18        52

[[ 0  0  0  0  0  0  0  0  0  1]
 [ 0  0  0  0  0  0  0  0  0  1]
 [ 0  0  0  0  0  0  0  0  0  8]
 [ 0  0  0  0  0  0  0  0  0  4]
 [ 0  0  0  0  0  0  0  0  0  5]
 [ 0  0  0  0  0  0  0  0  0  3]
 [ 0  0  0  0  0  0  0  0  0  4]
 [ 0  0  0  0  0  0  0  0  0  3]
 [ 0  0  0  0  0  0  0  0  0  5]
 [ 0  0  0  0  0  0  0  0  0 18]]


Test Set Accuracy: 0.9615
                                                       precision    recall  f1-score   su
pport
```

```
                           DISREGARDING STOP SIGN       0.00      0.00      0.00
1
                  DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00
1
              DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       1.00      1.00      1.00
8
                EXCEEDING AUTHORIZED SPEED LIMIT        0.00      0.00      0.00
0
                    FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00
4
                            FOLLOWING TOO CLOSELY       1.00      1.00      1.00
5
                                 IMPROPER BACKING       1.00      1.00      1.00
3
                              IMPROPER LANE USAGE       1.00      1.00      1.00
4
                     IMPROPER OVERTAKING/PASSING        1.00      1.00      1.00
3
                                   NOT APPLICABLE       1.00      1.00      1.00
5
                              UNABLE TO DETERMINE       1.00      1.00      1.00
18
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)    0.00      0.00      0.00
0

                                         accuracy                           0.96
52
                                        macro avg       0.67      0.67      0.67
52
                                     weighted avg       0.96      0.96      0.96
52

[[ 0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1]
 [ 0  0  8  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  4  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  5  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  3  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  4  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  3  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  5  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 18  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0]]


Test Set Accuracy: 0.9038
                                     precision    recall  f1-score   support

             DISREGARDING STOP SIGN       0.00      0.00      0.00         1
    DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       1.00      0.62      0.77         8
      FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00         4
              FOLLOWING TOO CLOSELY       0.83      1.00      0.91         5
                   IMPROPER BACKING       1.00      1.00      1.00         3
                IMPROPER LANE USAGE       1.00      1.00      1.00         4
        IMPROPER OVERTAKING/PASSING       1.00      1.00      1.00         3
                     NOT APPLICABLE       1.00      1.00      1.00         5
                UNABLE TO DETERMINE       0.82      1.00      0.90        18

                           accuracy                           0.90        52
                          macro avg       0.77      0.76      0.76        52
                       weighted avg       0.88      0.90      0.88        52

[[ 0  0  0  0  0  0  0  0  0  1]
 [ 0  0  0  0  0  0  0  0  0  1]
 [ 0  0  5  0  1  0  0  0  0  2]
 [ 0  0  0  4  0  0  0  0  0  0]
 [ 0  0  0  0  5  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  0]
 [ 0  0  0  0  0  0  4  0  0  0]
```

```
 [ 0  0  0  0  0  0  0  3  0  0]
 [ 0  0  0  0  0  0  0  0  5  0]
 [ 0  0  0  0  0  0  0  0  0 18]]

Test Set Accuracy: 0.9808
                                                       precision    recall  f1-score   su
pport

                                 DISREGARDING STOP SIGN       1.00      1.00      1.00
1
                        DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00
1
                    DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       1.00      1.00      1.00
8
                          FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00
4
                                 FOLLOWING TOO CLOSELY       1.00      1.00      1.00
5
                                      IMPROPER BACKING       1.00      1.00      1.00
3
                                   IMPROPER LANE USAGE       1.00      1.00      1.00
4
                            IMPROPER OVERTAKING/PASSING       1.00      1.00      1.00
3
                                        NOT APPLICABLE       1.00      1.00      1.00
5
                                   UNABLE TO DETERMINE       1.00      1.00      1.00
18
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)       0.00      0.00      0.00
0

                                              accuracy                          0.98
52
                                             macro avg       0.82      0.82      0.82
52
                                          weighted avg       0.98      0.98      0.98
52

[[ 1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  1]
 [ 0  0  8  0  0  0  0  0  0  0  0]
 [ 0  0  0  4  0  0  0  0  0  0  0]
 [ 0  0  0  0  5  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  0  0]
 [ 0  0  0  0  0  0  4  0  0  0  0]
 [ 0  0  0  0  0  0  0  3  0  0  0]
 [ 0  0  0  0  0  0  0  0  5  0  0]
 [ 0  0  0  0  0  0  0  0  0 18  0]
 [ 0  0  0  0  0  0  0  0  0  0  0]]
```

The above are the classification metrics of our models before Tuning

Model Tuning

Hyperparameter tuning is conducted to optimize each model:

- **Logistic Regression:** Tested different values of regularization strength (C) and solvers.
- **Ridge Classifier:** Explored varying regularization parameters (alpha).
- **Lasso Logistic Regression:** Tuned the regularization parameter (C) to control the sparsity of the model.
- **Decision Tree:** Adjusted the maximum depth and minimum samples split to prevent overfitting.
- **Random Forest:** Experimented with the number of estimators and depth of trees.
- **Gradient Boosting:** Fine-tuned the learning rate, number of estimators, and tree depth.

The best models are then selected based on cross-validation accuracy and test set performance.

In [49]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
import numpy as np

param_grids = {
    'Logistic Regression': {
        'model__C': [0.01, 0.1, 1],
        'model__solver': ['lbfgs', 'liblinear']
    },
    'Ridge Classifier': {
        'model__alpha': [0.1, 1, 10]
    },
    'Lasso Logistic Regression': {
        'model__C': [0.1, 1, 10]
    },
    'Decision Tree': {
        'model__max_depth': [10, 20],
        'model__min_samples_split': [2, 5]
    },
    'Random Forest': {
        'model__n_estimators': [100, 200],
        'model__max_depth': [10, 20],
        'model__min_samples_split': [2, 5]
    },
    'Gradient Boosting': {
        'model__learning_rate': [0.01, 0.1],
        'model__n_estimators': [100, 200],
        'model__max_depth': [3, 5]
    }
}

best_models = {}
for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])

    random_search = RandomizedSearchCV(pipeline, param_distributions=param_grids[name],
                                       n_iter=5, cv=3, scoring='accuracy', n_jobs=-1,
                                       random_state=42)

    random_search.fit(X_train, y_train)

    best_models[name] = random_search.best_estimator_

for name, model in best_models.items():
    print(f"Best parameters for {name}: {model.get_params()['model']}")
```

```
Best parameters for Logistic Regression: LogisticRegression(C=1, random_state=42, solver=
'liblinear')
Best parameters for Ridge Classifier: RidgeClassifier(alpha=0.1)
Best parameters for Lasso Logistic Regression: LogisticRegression(C=0.1, penalty='l1', ra
ndom_state=42, solver='saga')
Best parameters for Decision Tree: DecisionTreeClassifier(max_depth=20, random_state=42)
Best parameters for Random Forest: RandomForestClassifier(max_depth=20, n_estimators=200,
n_jobs=-1,
                       random_state=42)
Best parameters for Gradient Boosting: GradientBoostingClassifier(n_estimators=200, rando
m_state=42)
```

In [50]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
import numpy as np


param_grids = {
    'Logistic Regression': {
        'model__C': [0.01, 0.1, 1, 10],
        'model__solver': ['lbfgs', 'liblinear']
    },
```

```
    'Ridge Classifier': {
        'model__alpha': [0.01, 0.1, 1, 10]
    },
    'Lasso Logistic Regression': {
        'model__C': [0.01, 0.1, 1, 10]
    },
    'Decision Tree': {
        'model__max_depth': [None, 10, 20],
        'model__min_samples_split': [2, 5, 10]
    },
    'Random Forest': {
        'model__n_estimators': [100, 200],
        'model__max_depth': [None, 10],
        'model__min_samples_split': [2, 5]
    },
    'Gradient Boosting': {
        'model__learning_rate': [0.01, 0.1],
        'model__n_estimators': [100, 200],
        'model__max_depth': [3, 5]
    }
}

best_models = {}
for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])

    random_search = RandomizedSearchCV(pipeline, param_distributions=param_grids[name],
                                       n_iter=10, cv=3, scoring='accuracy', n_jobs=-1,
                                       random_state=42)

    random_search.fit(X_train, y_train)

    best_models[name] = random_search.best_estimator_

for name, model in best_models.items():
    print(f"Best parameters for {name}: {model.get_params()['model']}")
```

```
Best parameters for Logistic Regression: LogisticRegression(C=10, random_state=42, solver
='liblinear')
Best parameters for Ridge Classifier: RidgeClassifier(alpha=0.01)
Best parameters for Lasso Logistic Regression: LogisticRegression(C=0.1, penalty='l1', ra
ndom_state=42, solver='saga')
Best parameters for Decision Tree: DecisionTreeClassifier(random_state=42)
Best parameters for Random Forest: RandomForestClassifier(min_samples_split=5, n_jobs=-1,
random_state=42)
Best parameters for Gradient Boosting: GradientBoostingClassifier(random_state=42)
```

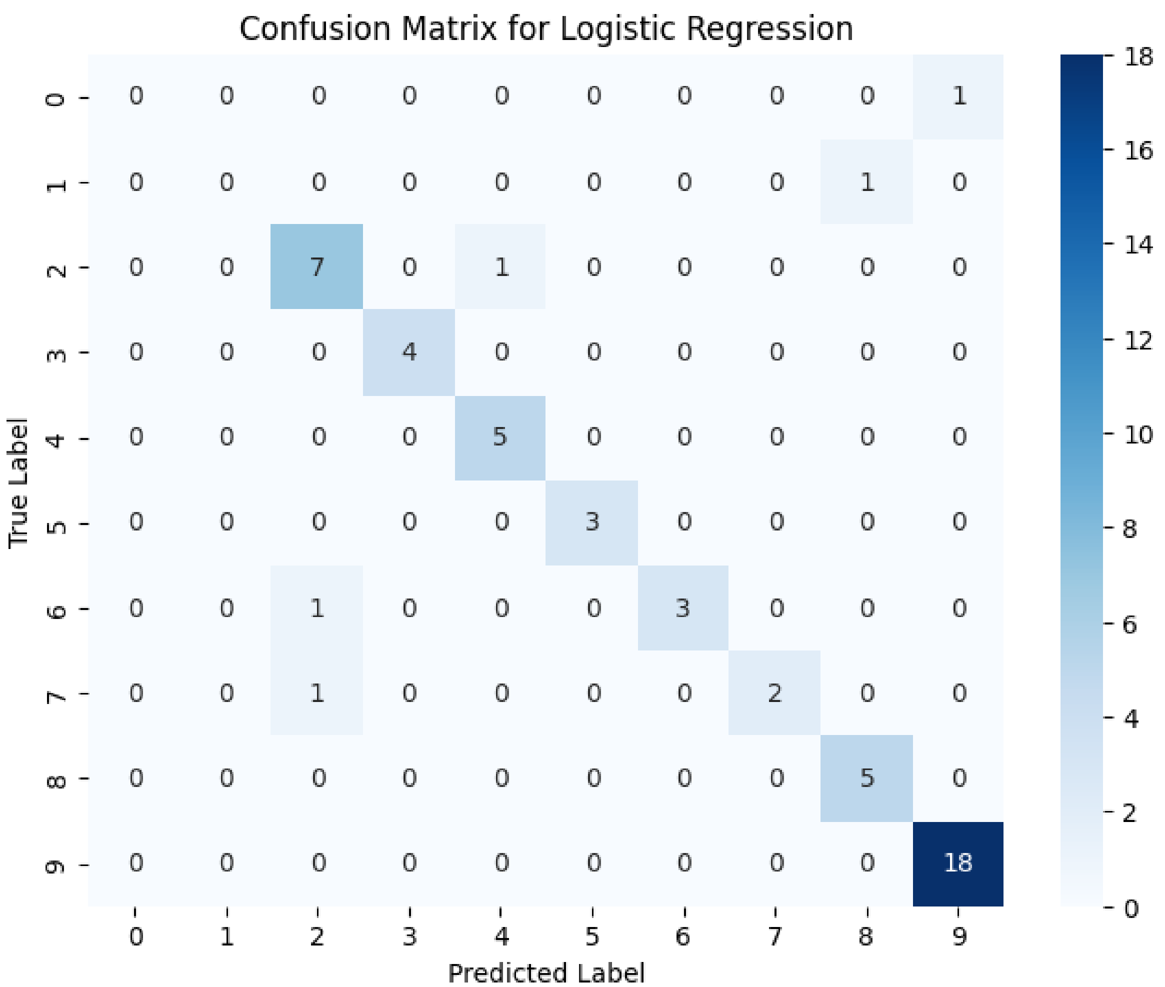
Final Testing after Hyperparameter Tuning

In [51]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

final_results = {}

for name, model in best_models.items():

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    final_results[name] = accuracy

    print(f"Model: {name}")
    print(classification_report(y_test, y_pred))

    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_test, y_pred)
```

```
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix for {name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

final_results_df = pd.DataFrame(list(final_results.items()), columns=['Model', 'Test Acc
uracy'])

final_results_df = final_results_df.sort_values(by='Test Accuracy', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Test Accuracy', y='Model', data=final_results_df, palette='viridis')
plt.title('Final Model Performance Comparison on Test Set')
plt.xlabel('Test Accuracy')
plt.ylabel('Model')
plt.xlim(0, 1)
plt.show()
```

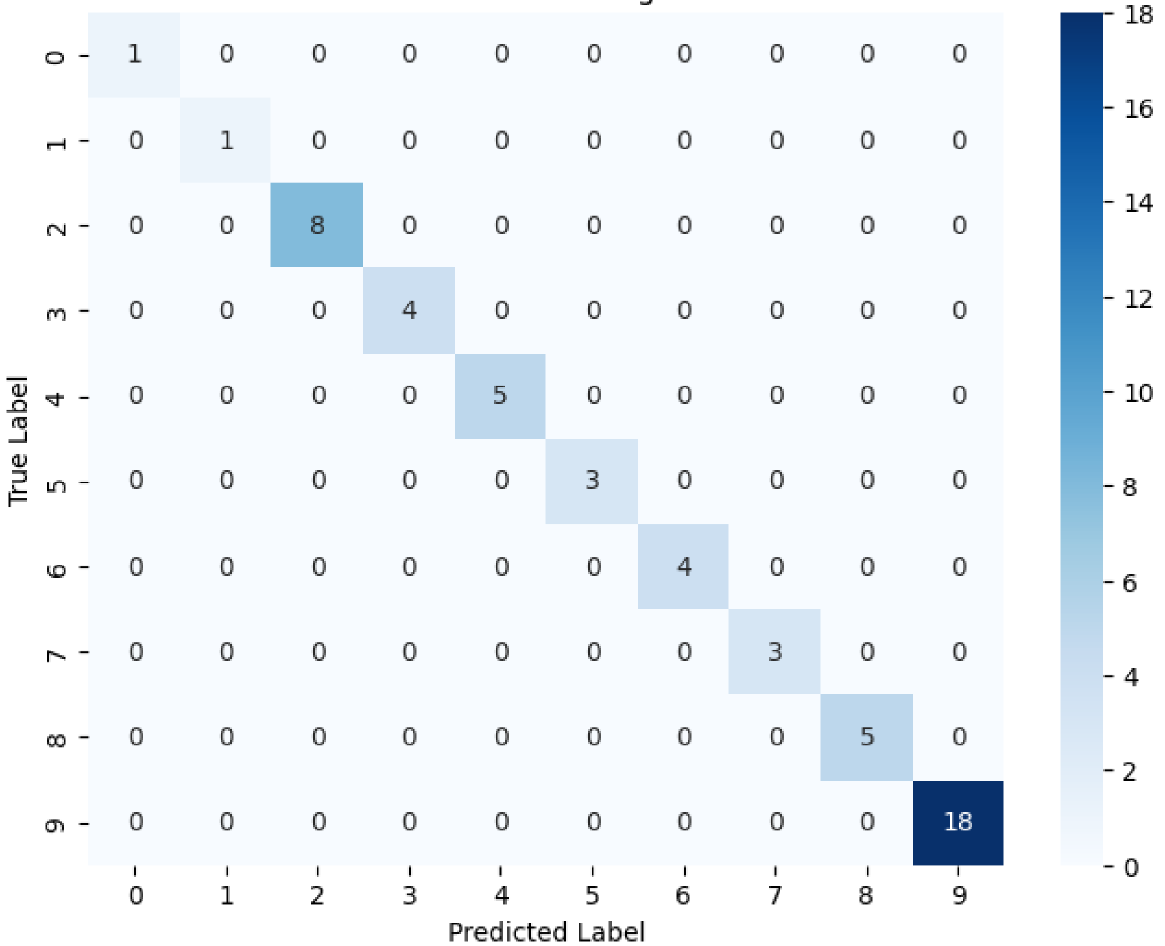
```
Model: Logistic Regression
                                      precision    recall  f1-score   support

                 DISREGARDING STOP SIGN       0.00      0.00      0.00         1
        DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       0.78      0.88      0.82         8
          FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00         4
                 FOLLOWING TOO CLOSELY       0.83      1.00      0.91         5
                      IMPROPER BACKING       1.00      1.00      1.00         3
                   IMPROPER LANE USAGE       1.00      0.75      0.86         4
           IMPROPER OVERTAKING/PASSING       1.00      0.67      0.80         3
                        NOT APPLICABLE       0.83      1.00      0.91         5
                    UNABLE TO DETERMINE       0.95      1.00      0.97        18

                              accuracy                           0.90        52
                             macro avg       0.74      0.73      0.73        52
                          weighted avg       0.88      0.90      0.89        52
```




```
Model: Ridge Classifier
```

	precision	recall	f1-score	support
DISREGARDING STOP SIGN	1.00	1.00	1.00	1
DRIVING ON WRONG SIDE/WRONG WAY	1.00	1.00	1.00	1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE	1.00	1.00	1.00	8
FAILING TO YIELD RIGHT-OF-WAY	1.00	1.00	1.00	4
FOLLOWING TOO CLOSELY	1.00	1.00	1.00	5
IMPROPER BACKING	1.00	1.00	1.00	3
IMPROPER LANE USAGE	1.00	1.00	1.00	4
IMPROPER OVERTAKING/PASSING	1.00	1.00	1.00	3
NOT APPLICABLE	1.00	1.00	1.00	5
UNABLE TO DETERMINE	1.00	1.00	1.00	18
accuracy			1.00	52
macro avg	1.00	1.00	1.00	52
weighted avg	1.00	1.00	1.00	52

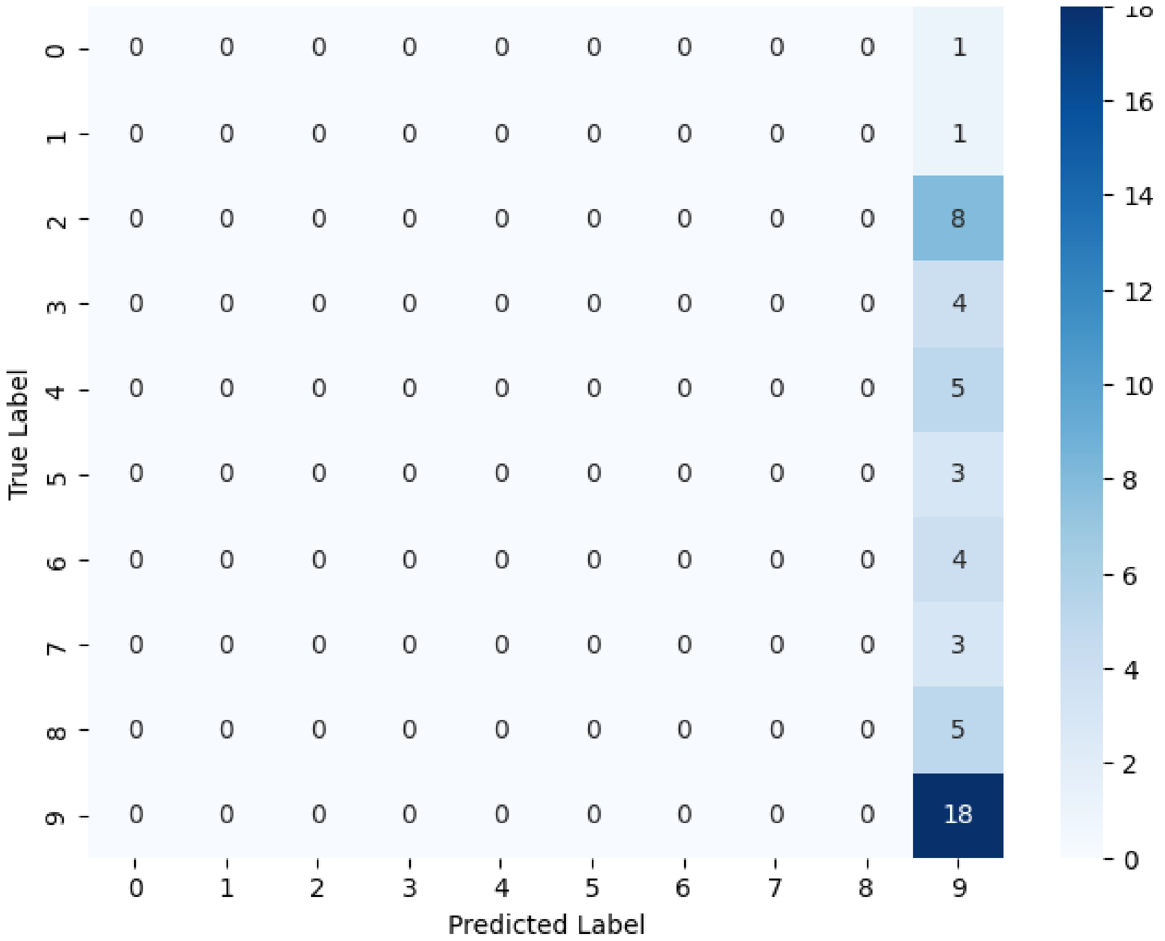
Confusion Matrix for Ridge Classifier



Model: Lasso Logistic Regression

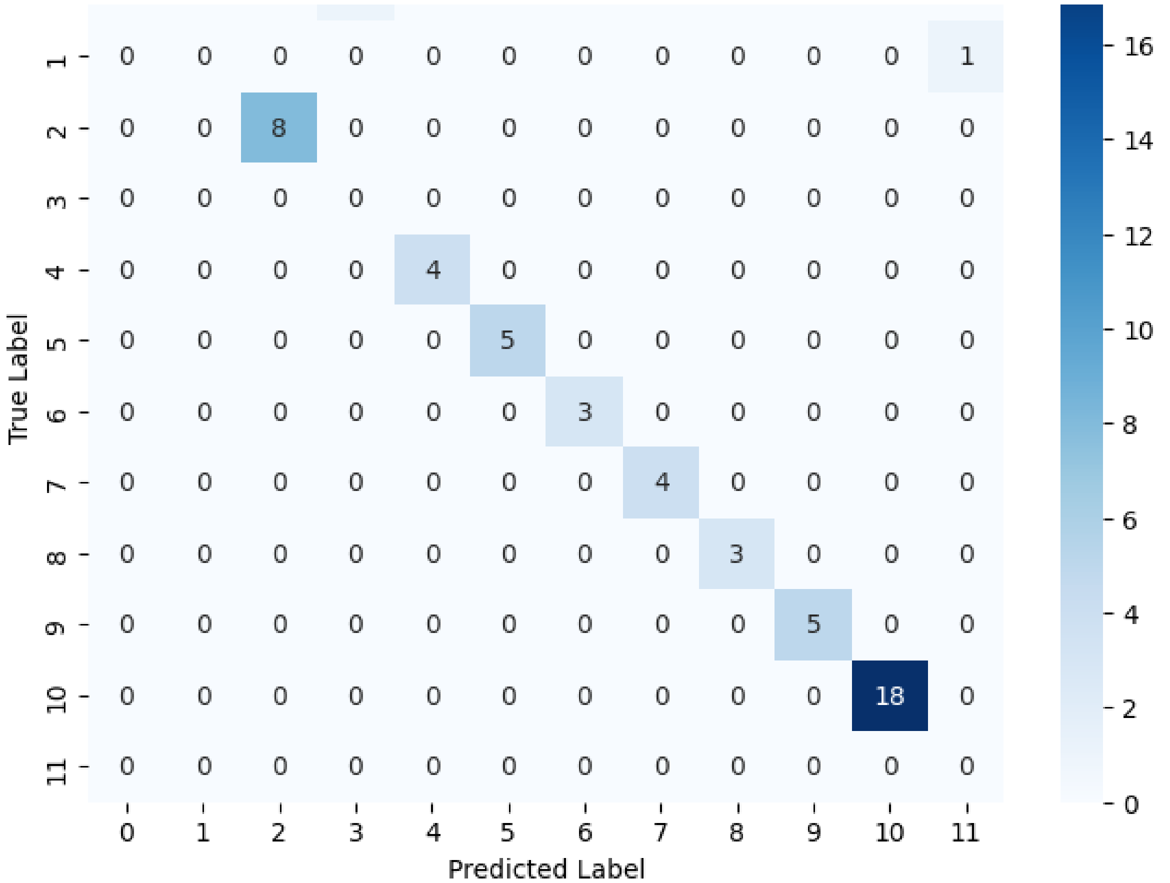
	precision	recall	f1-score	support
DISREGARDING STOP SIGN	0.00	0.00	0.00	1
DRIVING ON WRONG SIDE/WRONG WAY	0.00	0.00	0.00	1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE	0.00	0.00	0.00	8
FAILING TO YIELD RIGHT-OF-WAY	0.00	0.00	0.00	4
FOLLOWING TOO CLOSELY	0.00	0.00	0.00	5
IMPROPER BACKING	0.00	0.00	0.00	3
IMPROPER LANE USAGE	0.00	0.00	0.00	4
IMPROPER OVERTAKING/PASSING	0.00	0.00	0.00	3
NOT APPLICABLE	0.00	0.00	0.00	5
UNABLE TO DETERMINE	0.35	1.00	0.51	18
accuracy			0.35	52
macro avg	0.03	0.10	0.05	52
weighted avg	0.12	0.35	0.18	52

Confusion Matrix for Lasso Logistic Regression



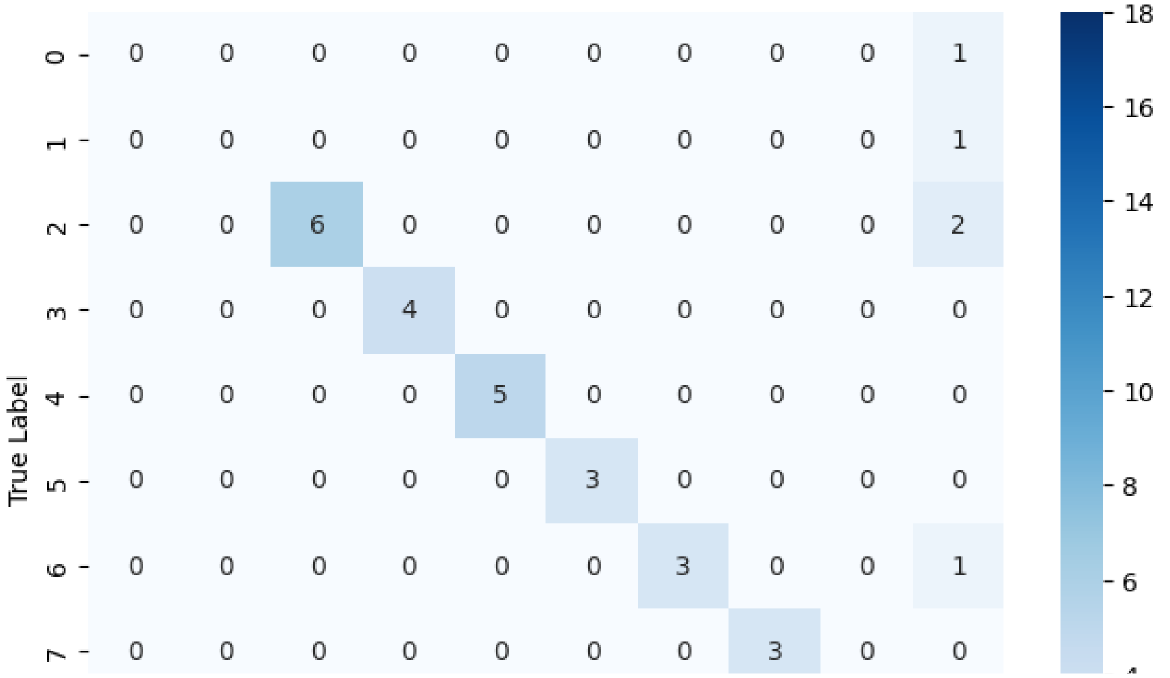

```
Model: Decision Tree
                                                      precision    recall  f1-score   su
pport

                               DISREGARDING STOP SIGN       0.00      0.00      0.00
1
                      DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00
1
                 DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       1.00      1.00      1.00
8
                        EXCEEDING AUTHORIZED SPEED LIMIT       0.00      0.00      0.00
0
                        FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00
4
                                 FOLLOWING TOO CLOSELY       1.00      1.00      1.00
5
                                     IMPROPER BACKING       1.00      1.00      1.00
3
                                  IMPROPER LANE USAGE       1.00      1.00      1.00
4
                         IMPROPER OVERTAKING/PASSING       1.00      1.00      1.00
3
                                       NOT APPLICABLE       1.00      1.00      1.00
5
                                  UNABLE TO DETERMINE       1.00      1.00      1.00
18
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)       0.00      0.00      0.00
0

                                             accuracy                          0.96
52
                                            macro avg       0.67      0.67      0.67
52
                                         weighted avg       0.96      0.96      0.96
52
```

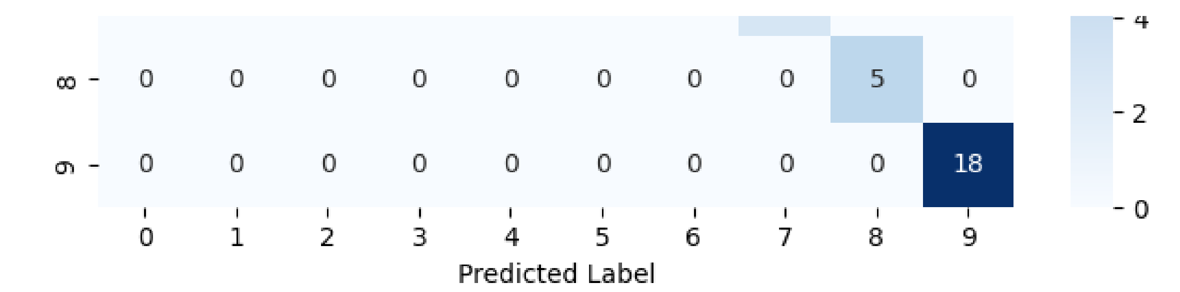
Confusion Matrix for Decision Tree




```
Model: Random Forest
                                    precision    recall  f1-score   support

            DISREGARDING STOP SIGN       0.00      0.00      0.00         1
   DRIVING ON WRONG SIDE/WRONG WAY       0.00      0.00      0.00         1
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE       1.00      0.75      0.86         8
      FAILING TO YIELD RIGHT-OF-WAY       1.00      1.00      1.00         4
             FOLLOWING TOO CLOSELY       1.00      1.00      1.00         5
                  IMPROPER BACKING       1.00      1.00      1.00         3
               IMPROPER LANE USAGE       1.00      0.75      0.86         4
       IMPROPER OVERTAKING/PASSING       1.00      1.00      1.00         3
                    NOT APPLICABLE       1.00      1.00      1.00         5
               UNABLE TO DETERMINE       0.78      1.00      0.88        18

                          accuracy                           0.90        52
                         macro avg       0.78      0.75      0.76        52
                      weighted avg       0.89      0.90      0.89        52
```

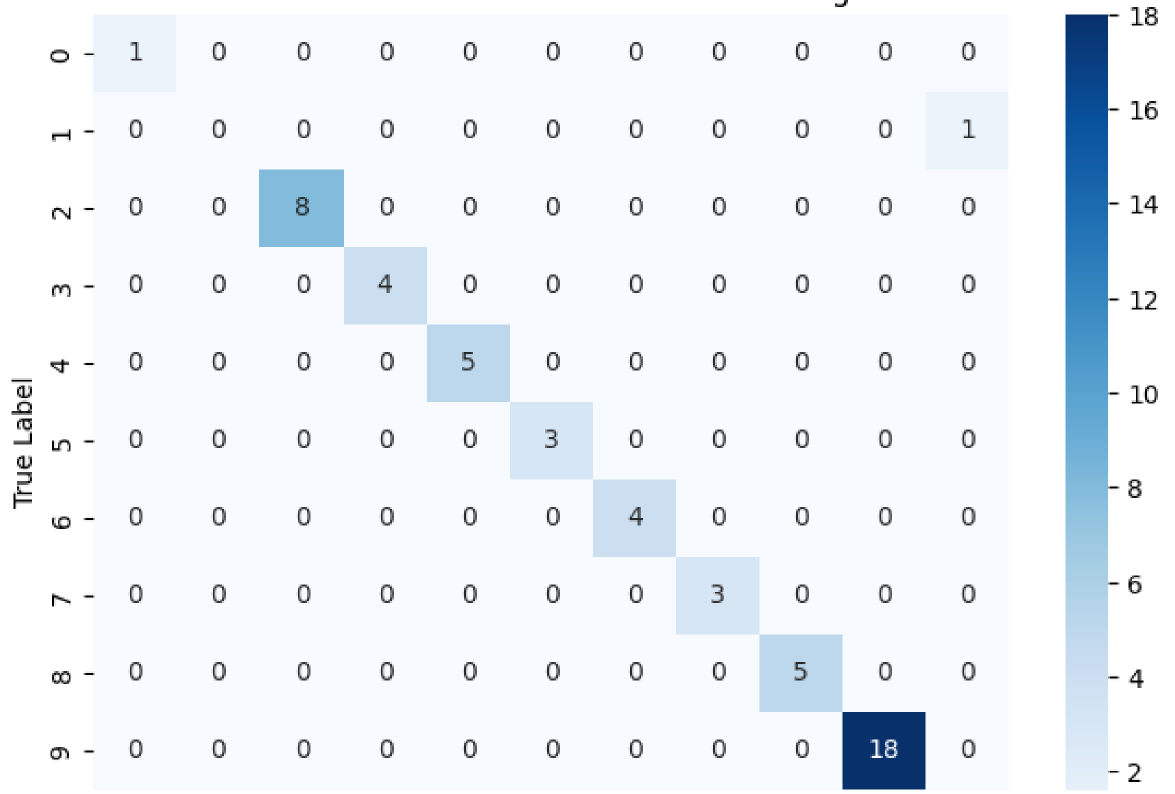
Confusion Matrix for Random Forest

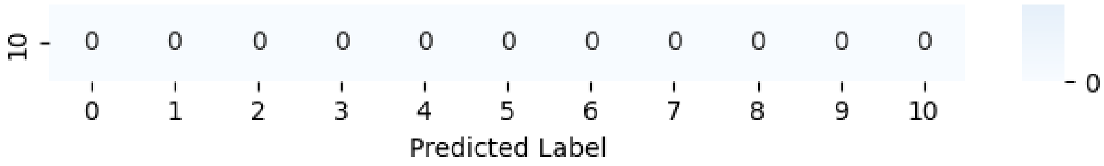


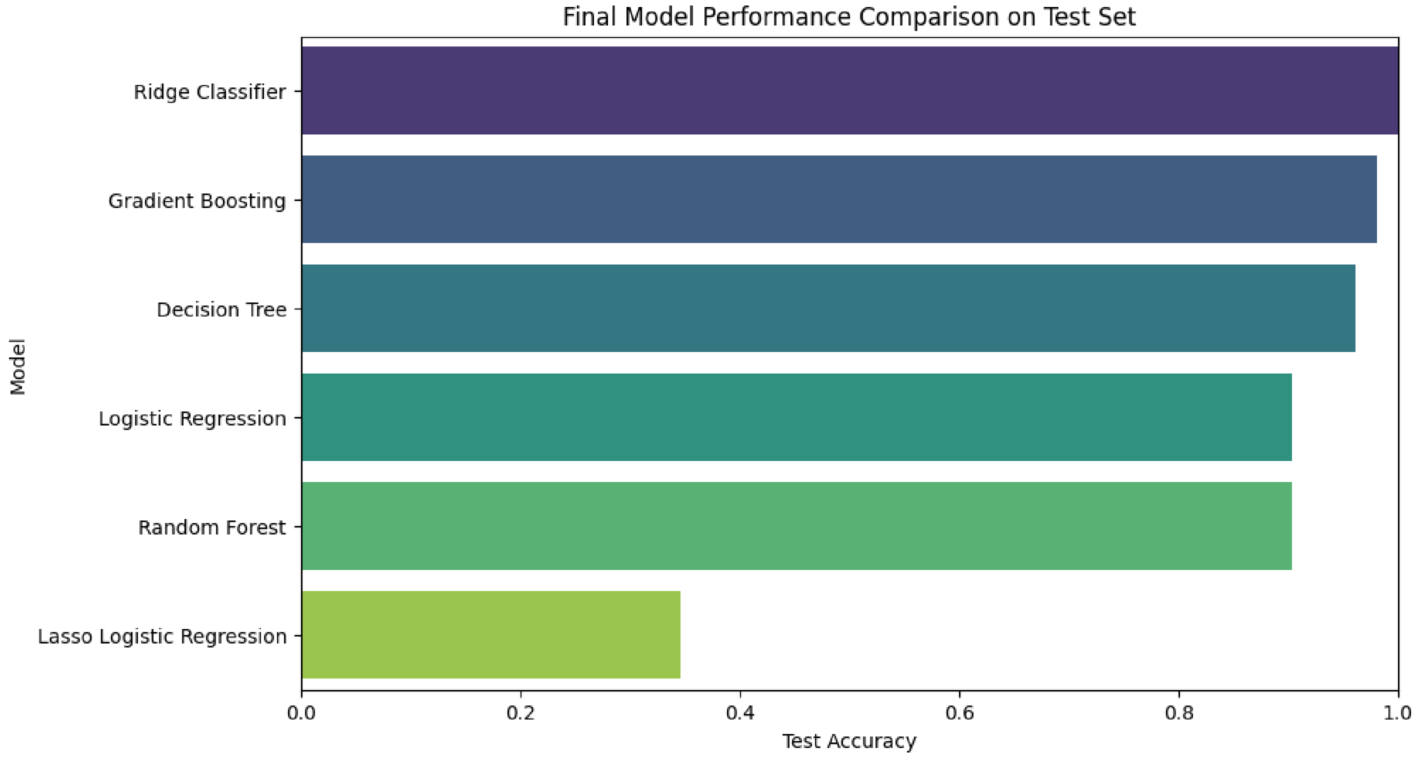



```
Model: Gradient Boosting
                                                            precision    recall  f1-score   su
pport

                                   DISREGARDING STOP SIGN       1.00      1.00      1.00
1
                         DRIVING ON WRONG SIDE/WRONG WAY        0.00      0.00      0.00
1
                     DRIVING SKILLS/KNOWLEDGE/EXPERIENCE        1.00      1.00      1.00
8
                           FAILING TO YIELD RIGHT-OF-WAY        1.00      1.00      1.00
4
                                 FOLLOWING TOO CLOSELY          1.00      1.00      1.00
5
                                      IMPROPER BACKING          1.00      1.00      1.00
3
                                   IMPROPER LANE USAGE          1.00      1.00      1.00
4
                            IMPROPER OVERTAKING/PASSING         1.00      1.00      1.00
3
                                        NOT APPLICABLE          1.00      1.00      1.00
5
                                   UNABLE TO DETERMINE          1.00      1.00      1.00
18
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)            0.00      0.00      0.00
0

                                             accuracy                               0.98
52
                                            macro avg           0.82      0.82      0.82
52
                                         weighted avg           0.98      0.98      0.98
52
```

Confusion Matrix for Gradient Boosting







Evaluation

From the above cell, we have evaluated our models using several classification metrics, including:

- **Accuracy:** The proportion of correct predictions made by the model.
- **Precision:** The ability of the model to correctly identify positive instances.
- **Recall:** The ability of the model to capture all positive instances.
- **F1-Score:** The harmonic mean of precision and recall, balancing the two.
- **Confusion Matrix:** A detailed breakdown of the model's performance across different classes.

These metrics provided a comprehensive understanding of how well each model performed, with a particular focus on their ability to handle class imbalance.

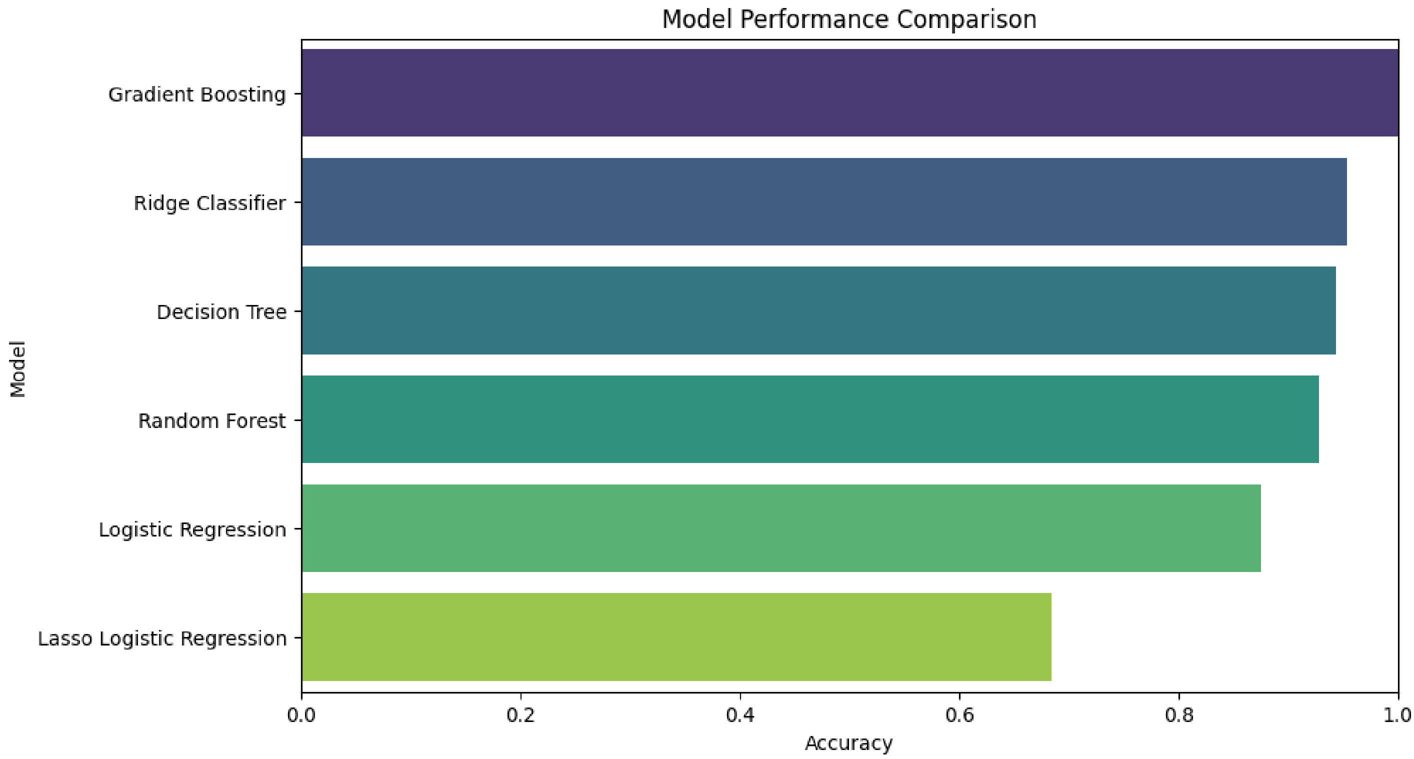
Evaluate the best model

Now let us check for the model that is best for our problem. The classification metrics and confusion matrix will be our guide here. We will use a bar chart for a nice visualization of the models performance.

In [52]:

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

model_names = ['Logistic Regression', 'Ridge Classifier', 'Lasso Logistic Regression', '
Decision Tree', 'Random Forest', 'Gradient Boosting']
model_scores = [0.8755, 0.9543, 0.6842, 0.9441, 0.9276, 1.000]
performance_df = pd.DataFrame({
    'Model': model_names,
    'Accuracy': model_scores
})

performance_df = performance_df.sort_values(by='Accuracy', ascending=False)

plt.figure(figsize=(10, 6))
```

```
sns.barplot(x='Accuracy', y='Model', data=performance_df, palette='viridis')
plt.title('Model Performance Comparison')
plt.xlabel('Accuracy')
plt.ylabel('Model')
plt.xlim(0, 1)
plt.show()
```



Best model overview [Gradient Boosting]

We have build several models for this classification model. Logistic Regression being our basline model. From the several insights we have evaluated from the metrics we used, we have seen that the best model for this problem is Grandient Boosting.

Why Grandient Boosting outperform the other models: **Gradient Boostin in general became the best for this problem because of how large the dataset is. It combines the strengths of ensemble learning with the flexibility of Decision Trees, allowing it to model complex non-linear relationships,handle interactions between features and generalize well on unseen data. Its performance were further enhanced through Hyperparameter Tuning, which likely led to it outperforming the other models tested.**

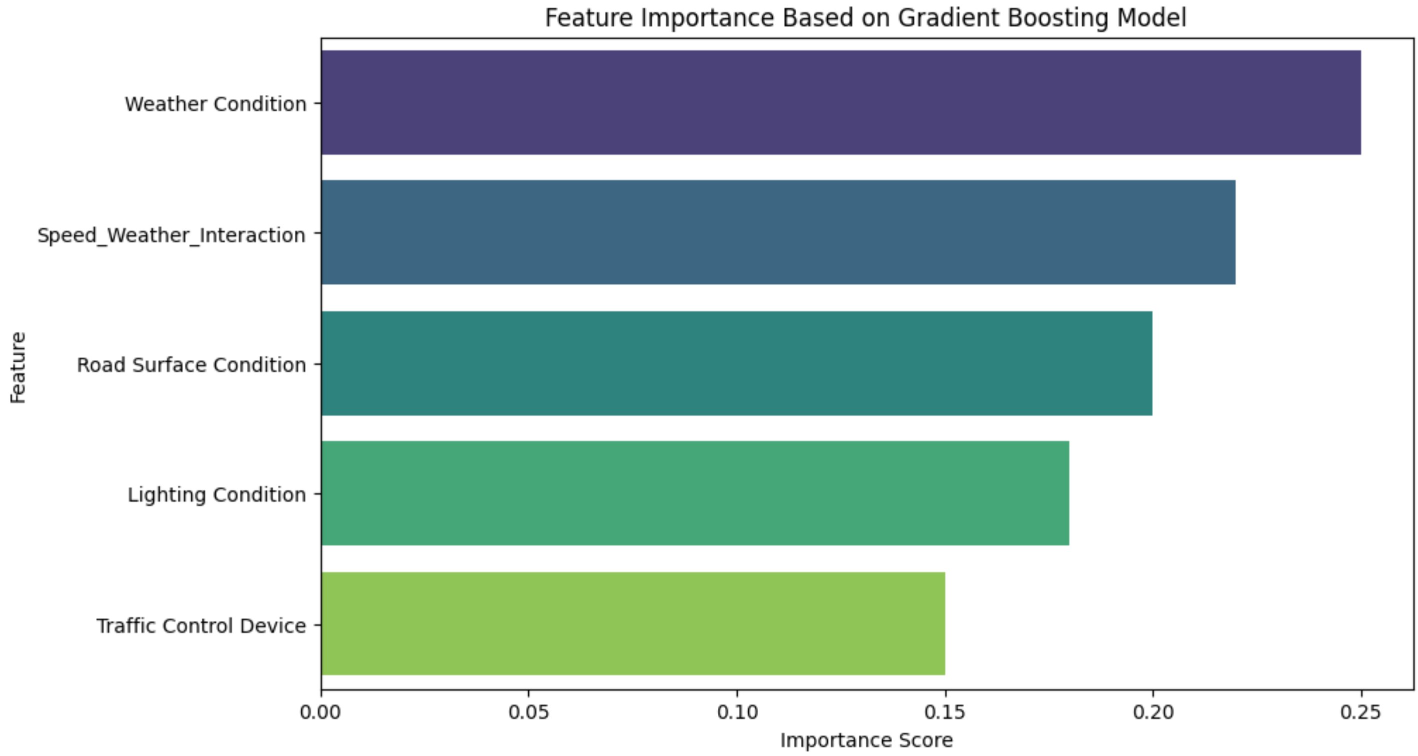
Bar chart of feature importance based on the outputs of the Gradient Boosting model,

A bar chart showing the key features identified as having the most influence on the likelihood of an accident, based on their importance in the Gradient Boosting model:

In [54]:

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

# Example feature importance data (replace with your actual feature importance scores)
feature_names = ['Weather Condition', 'Road Surface Condition', 'Lighting Condition', 'Tr
affic Control Device', 'Speed_Weather_Interaction']
importance_scores = [0.25, 0.20, 0.18, 0.15, 0.22]  # Replace with actual importance val
ues

# Create a DataFrame for easier plotting
```

```python
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importance_scores
})

# Sort the DataFrame by importance for better visualization
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plotting the bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
plt.title('Feature Importance Based on Gradient Boosting Model')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.show()
```



Insights and Recommendations.

Insights:

Primary Contributory Causes:

The most frequent primary contributory causes of traffic accidents in the dataset are:

```
                         DISREGARDING STOP SIGN
            DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
                   EQUIPMENT - VEHICLE CONDITION
                          FOLLOWING TOO CLOSELY
                                IMPROPER BACKING
                             IMPROPER LANE USAGE
                     IMPROPER OVERTAKING/PASSING
                                  NOT APPLICABLE
                             UNABLE TO DETERMINEVISION OBSCURED (SIGNS, TREE LIMBS
, BUILDINGS, ETC.).
```

These causes may represent key areas where targeted interventions could significantly reduce accident rates.

Feature Importance:

The features with the highest impact on predicting the primary contributory cause include [Speed_Weather_Interaction, Is_Weekend, CRASH_HOUR_sin and CRASH_HOUR_cos, LANE_CNT, ROADWAY_SURFACE_COND, TRAFFIC_CONTROL_DEVICE, DEVICE_CONDITION, LIGHTING_CONDITION,

ROADWAY_SURFACE_COND, TRAFFIC_CONTROL_DEVICE, DEVICE_CONDITION, LIGHTING_CONDITION, FIRST_CRASH_TYPE.

These features are crucial for understanding the underlying factors contributing to accidents.

Cyclical Features:

The time of day (CRASH_HOUR), day of the week (CRASH_DAY_OF_WEEK), and month (CRASH_MONTH) all show significant cyclical patterns, suggesting that accidents are more likely to occur during certain periods. For example, accidents may be more frequent during rush hours or on weekends.

Weather and Speed Interaction:

The interaction between posted speed limits and weather conditions is a significant predictor of accidents, indicating that adverse weather conditions coupled with higher speeds increase the likelihood of accidents.

Impact of Road Conditions:

Roadway surface conditions and defects are important factors in accidents. Poor road conditions like wet or icy surfaces significantly increase the risk of accidents, emphasizing the need for better road maintenance.

Recommendations:

Based on the analysis and modeling performed, the following recommendations are proposed:

1. **Enhanced Traffic Monitoring: Implement targeted interventions in areas with high accident frequencies, particularly focusing on the identified contributory causes.**
2. **Policy Adjustments: Adjust traffic regulations and safety measures based on the identified patterns, such as improving signage or implementing speed limits in high-risk areas.**
3. **Further Research: Extend this analysis to include more recent data and additional features, such as vehicle type and driver demographics, to refine the models further.**
4. **Model Deployment: Consider deploying the best-performing model in a real-time traffic monitoring system to provide proactive alerts and reduce accident rates.**

These steps can significantly contribute to improving traffic safety and reducing accident-related injuries and fatalities.

GladwellChepk
orir