Data Mining for Hit-and-Run Crash Prediction: Analyzing Environmental and Roadway Risk Factors

Executive Summary

This study investigates the environmental and roadway factors that elevate the risk of a crash becoming a hit-and-run, aiming to guide law enforcement in optimizing deterrence and patrol efforts. An investigation of traffic collision data using classification and association rule mining approaches showed that crashes with inadequate illumination, particularly in dark or poorly lit regions at night, are far more likely to end in hit-and-run situations. These circumstances lessen the possibility of identification while giving evading drivers shelter. Additionally, hit-and-run conduct was more commonly linked to collisions on non-intersection roadways—like major arterials and residential streets—than at crossroads. This is probably because there are fewer traffic controls and less monitoring of vehicles or pedestrians. Temporal variables were also shown to be major predictors, especially collisions that happened on weekends and at night, and they were linked to higher rates of irresponsible or intoxicated driving as well as a decrease in the presence of law enforcement. Although road surface conditions like wet or slippery roads were less influential, they still contributed to heightened risk by impairing control and increasing the chance of impulsive flight by the driver. These findings are consistent with prior research. Drivers who feel they have a small likelihood of being arrested are more likely to flee, especially in circumstances where it is challenging to get witness accounts or evidence (Tay, Barua, & Kattan, 2009). Similarly, MacLeod et al. (2012) pointed out that lighting, the kind of route, and the time of day significantly affect a driver's likelihood of evading detection and eventual detection. Furthermore, Tay, Rifaat, and Chin (2008) developed a logistic model that highlighted the importance of roadway design, ambient visibility, and driver demographics in predicting hit-and-run behavior. These findings point to specific actions, such as stepping up patrols on the weekends and at night, improving lighting systems, and putting in surveillance equipment in high-risk locations for hit-and-run incidents. Enforcement organizations and legislators may more successfully lower hit-and-run rates and enhance road safety by utilizing these elements.

Methodology

This study used Python and an organized, reproducible data mining process to find the highway and environmental parameters that influence the probability of a hit-and-run collision. Importing necessary libraries like matplotlib and seaborn for visualization and pandas and numpy for data management was the first step in the procedure. The information was imported from a CSV file and included a variety of characteristics associated with traffic crashes. To evaluate data kinds, find missing values, and comprehend data distribution, a preliminary exploratory analysis was carried out. The core features of interest included lighting conditions, road surface type, crash location (e.g., intersection or not), and time-related variables, all considered potential contributors to hit-and-run behavior. Categorical variables were encoded using LabelEncoder to transform textual categories into numerical representations suitable for modeling. The lack of feature scaling and imputation in the preprocessing, in contrast to more intricate pipelines, suggests that the dataset was probably previously cleaned or that the missing values were not important enough to have an impact on model performance. Following preprocessing, the data was split into training and testing sets using an 80:20 ratio with train_test_split, ensuring that the models could be evaluated on unseen data. The Gradient Boosting Classifier and Logistic Regression machine learning techniques were used. Because of its ease of use and interpretability, logistic regression was employed to shed light on how independent factors affected the binary target (hit-and-run or not). A more advanced ensemble technique called gradient boosting was added to increase prediction accuracy and capture intricate interactions. Both models were evaluated using performance metrics such as accuracy, precision, recall, F1-score, and confusion matrices after training. Functions from the sklearn metrics package were used to compute these metrics. A clear image of false positives and false negatives was provided by the confusion matrix, which was used to assess how well the model identified actual hit-and-run instances. Comparative analysis was made possible by the employment of two models, which also increased the findings' dependability. Both interpretability and predictive insights were provided by the whole procedure, which included data input, encoding, splitting, model training, and assessment. It made it abundantly evident that a higher risk of hit-and-run occurrences was linked to particular road and environmental factors, including inadequate illumination and non-intersection collision sites. In order to reduce hit-and-run incidents, this analytical approach offers a visible and repeatable basis for guiding policy decisions and the distribution of law enforcement resources.

In [1]:
```
import numpy as np  # here numpy used for numerical calculation of the data and analysis
import pandas as pd  # here pandas used for manipulation of the data and analysis
import seaborn as sns # used for plots
import matplotlib.pyplot as plt #for graphs visualisation
from sklearn.preprocessing import StandardScaler, OneHotEncoder  # used for scaling numerical features and encoding categorical features
from sklearn.model_selection import train_test_split  # here used for to split data into training and testing sets
from sklearn.preprocessing import LabelEncoder  # to target labels with values between 0 and n_classes-1 are translated.
from sklearn.ensemble import GradientBoostingClassifier  #for Gradient Boosting Classifiers
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score ,ConfusionMatrixDisplay # For evaluating classification performance
from sklearn.linear_model import LogisticRegression   #importing Logistic Regression
```

In [2]:
```
# Load the uploaded CSV files
df = pd.read_csv("Dataset for Data Mining.csv")
```

In [3]:
```
df.head() # Display the first 5 rows of a dataframe to understand the structure
```

Out[3]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_CONDITI
0	ec987eca9d84e964c10e750c95b8dca890036480463020...	NaN	11/19/2019 09:20:00 AM	30.0	NO CONTROLS	NO CONTROLS	CLOUDY/OVERCAST	DAYLI(
1	deafa5f8bdd844d7154eae177cbd86bf5754991a0c7e77...	NaN	02/12/2025 01:00:00 PM	35.0	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DAYLI(
2	96577eb19233057b1bf0b40befc1acbcc26b21fe7abb93...	NaN	12/30/2022 09:50:00 PM	30.0	NO CONTROLS	NO CONTROLS	CLEAR	DARKNESS, LIGH RC
3	31e890ddbd9c76afdcd32bcda5f8e885311cf486b8afae...	NaN	07/15/2018 10:00:00 PM	25.0	NO CONTROLS	NO CONTROLS	CLEAR	DARKNESS, LIGH RC
4	49dcf06cc6debf56fea45cbdd5c07156bd0bc6a475657c...	NaN	10/27/2024 10:30:00 PM	30.0	NO CONTROLS	NO CONTROLS	CLEAR	DARKNESS, LIGH RC

5 rows × 48 columns

In [4]:
```
df.describe()   # gives the statistical information about the dataset
```

Out[4]:

	POSTED_SPEED_LIMIT	LANE_CNT	STREET_NO	BEAT_OF_OCCURRENCE	NUM_UNITS	INJURIES_TOTAL	INJURIES_FATAL	INJURIES_INCAPACITATING	INJURIES_NON_INCAPACITATING	INJURII
count	2.359410e+05	4.971000e+04	2.359400e+05	2.359420e+05	2.359340e+05	2.354420e+05	2.354460e+05	2.354470e+05	2.354390e+05	
mean	-2.119174e+22	-1.005834e+23	-2.119183e+22	-2.119165e+22	-2.119237e+22	-2.123665e+22	-2.123629e+22	-2.123620e+22	-2.123692e+22	
std	4.603410e+24	1.002872e+25	4.603419e+24	4.603400e+24	4.603478e+24	4.608285e+24	4.608246e+24	4.608236e+24	4.608315e+24	
min	-1.000000e+27	-1.000000e+27	-1.000000e+27	-1.000000e+27	-1.000000e+27	-1.000000e+27	-1.000000e+27	-1.000000e+27	-1.000000e+27	
25%	3.000000e+01	2.000000e+00	1.253000e+03	7.150000e+02	2.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
50%	3.000000e+01	2.000000e+00	3.202000e+03	1.212000e+03	2.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
75%	3.000000e+01	4.000000e+00	5.560000e+03	1.822000e+03	2.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
max	9.900000e+01	9.900000e+01	3.445300e+04	6.100000e+03	1.800000e+01	2.100000e+01	3.000000e+00	1.000000e+01	1.900000e+01	

A number of the dataset's numeric columns have summary statistics provided in the output from df.describe(). It does, however, highlight a crucial problem: the existence of incredibly huge negative placeholder numbers (such as -1.000000e+27), which most likely indicate missing or incorrect data. These values are skewing key metrics such as mean, standard deviation, and min across multiple columns (e.g., POSTED_SPEED_LIMIT, NUM_UNITS, INJURIES_TOTAL). The fact that the median (50%) values appear normal, such as 30 for POSTED_SPEED_LIMIT and 2 for NUM_UNITS, suggests that the actual data is valid but distorted by these extreme placeholders. Cleaning or replacing these placeholder values (e.g., with NaN) is essential before performing meaningful statistical analysis or modeling.

In [5]:
```
# Define the extreme negative placeholder value
extreme_value = -99999999999999999999999999

# Loop through numeric columns and check for the extreme value
for col in df.select_dtypes(include=['number']).columns:
    if extreme_value in df[col].values:
        df[col] = df[col].replace(extreme_value, pd.NA)
```

This code identifies and replaces an extreme negative placeholder value (-999...999) with pd.NA across all numeric columns in a DataFrame. It checks each numeric column individually and performs the replacement only if the placeholder is present, helping clean invalid or missing data for further analysis.

In [6]:
```
df.duplicated().sum()   #for finding the duplicate rows
```

Out[6]:
```
17
```

The output of df.duplicated().sum() is displayed by the code. This code determines the DataFrame's duplicate row count. The result, 17, shows that the DataFrame has 17 duplicate rows.

In [7]:
```
df = df.drop_duplicates() # used to remove duplicate rows
```

In [8]:
```
df.isnull().sum()     #for finding the null values in the dataset
```

Out[8]:
```
CRASH_RECORD_ID                    33
CRASH_DATE_EST_I                218602
CRASH_DATE                         30
POSTED_SPEED_LIMIT                 37
TRAFFIC_CONTROL_DEVICE             38
DEVICE_CONDITION                   36
WEATHER_CONDITION                  45
LIGHTING_CONDITION                 34
FIRST_CRASH_TYPE                   39
TRAFFICWAY_TYPE                    36
LANE_CNT                       186255
ALIGNMENT                          36
ROADWAY_SURFACE_COND               37
ROAD_DEFECT                        38
REPORT_TYPE                      7654
CRASH_TYPE                         36
INTERSECTION_RELATED_I         181936
NOT_RIGHT_OF_WAY_I             225232
HIT_AND_RUN_I                  162020
DAMAGE                             35
DATE_POLICE_NOTIFIED               40
PRIM_CONTRIBUTORY_CAUSE            43
SEC_CONTRIBUTORY_CAUSE             40
STREET_NO                          38
STREET_DIRECTION                   45
STREET_NAME                        34
BEAT_OF_OCCURRENCE                 36
PHOTOS_TAKEN_I                 232677
STATEMENTS_TAKEN_I             230493
DOORING_I                      235235
WORK_ZONE_I                    234669
WORK_ZONE_TYPE                 234961
WORKERS_PRESENT_I              235650
NUM_UNITS                          44
MOST_SEVERE_INJURY                543
INJURIES_TOTAL                    536
INJURIES_FATAL                    532
INJURIES_INCAPACITATING           531
INJURIES_NON_INCAPACITATING       539
INJURIES_REPORTED_NOT_EVIDENT     538
INJURIES_NO_INDICATION            526
INJURIES_UNKNOWN                  533
CRASH_HOUR                         32
CRASH_DAY_OF_WEEK                  45
CRASH_MONTH                        35
LATITUDE                         1819
LONGITUDE                        1819
LOCATION                         1817
dtype: int64
```

The number of missing (NaN) values for every column in the DataFrame is displayed in this output. There are many missing values in columns like CRASH_DATE_EST_I, INJURIES_TOTAL, and LOCATION, which suggests that they might need to be removed or imputed before modeling. It is a crucial phase in evaluating the quality of data.

In [9]:
```
# 1. Median imputation for numeric variables
for col in ['NUM_UNITS', 'POSTED_SPEED_LIMIT']:
    if col in df.columns:
```

```
        df.loc[:, col] = df[col].fillna(df[col].median())

# Mode imputation for categorical variables
if 'WEATHER_CONDITION' in df.columns:
    df.loc[:, 'WEATHER_CONDITION'] = df['WEATHER_CONDITION'].fillna(df['WEATHER_CONDITION'].mode()[0])

if 'LIGHTING_CONDITION' in df.columns:
    df.loc[:, 'LIGHTING_CONDITION'] = df['LIGHTING_CONDITION'].fillna(df['LIGHTING_CONDITION'].mode()[0])

if 'TRAFFIC_CONTROL_DEVICE' in df.columns:
    df.loc[:, 'TRAFFIC_CONTROL_DEVICE'] = df['TRAFFIC_CONTROL_DEVICE'].fillna(df['TRAFFIC_CONTROL_DEVICE'].mode()[0])

if 'ROADWAY_SURFACE_COND' in df.columns:
    df.loc[:, 'ROADWAY_SURFACE_COND'] = df['ROADWAY_SURFACE_COND'].fillna(df['ROADWAY_SURFACE_COND'].mode()[0])
```

In [10]:
```
cols_to_keep = [
    'POSTED_SPEED_LIMIT',
    'TRAFFIC_CONTROL_DEVICE',
    'WEATHER_CONDITION',
    'LIGHTING_CONDITION',
    'ROADWAY_SURFACE_COND',
    'CRASH_HOUR',
    'CRASH_DAY_OF_WEEK',
    'NUM_UNITS',
    'MOST_SEVERE_INJURY',
    'HIT_AND_RUN_I'  # target variable
]

# Drop rows with missing values in the selected columns for further analysis
df = df.dropna(subset=cols_to_keep)
```

In [11]:
```
# Drop the unnecessary columns and limit to the selected variables
df = df[cols_to_keep]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73671 entries, 1 to 235976
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   POSTED_SPEED_LIMIT     73671 non-null  float64
 1   TRAFFIC_CONTROL_DEVICE 73671 non-null  object
 2   WEATHER_CONDITION      73671 non-null  object
 3   LIGHTING_CONDITION     73671 non-null  object
 4   ROADWAY_SURFACE_COND   73671 non-null  object
 5   CRASH_HOUR             73671 non-null  float64
 6   CRASH_DAY_OF_WEEK      73671 non-null  float64
 7   NUM_UNITS              73671 non-null  float64
 8   MOST_SEVERE_INJURY     73671 non-null  object
 9   HIT_AND_RUN_I          73671 non-null  object
dtypes: float64(4), object(6)
memory usage: 6.2+ MB
```

In [12]:
```
# 1) Detect how many “illogical” entries :
issues = {
    'POSTED_SPEED_LIMIT ≤ 0':      (df['POSTED_SPEED_LIMIT'] <= 0).sum(),
    'NUM_UNITS ≤ 0':               (df['NUM_UNITS'] <= 0).sum(),
```

```
    'CRASH_HOUR (<0 or >23)':     ((df['CRASH_HOUR'] < 0) | (df['CRASH_HOUR'] > 23)).sum(),
    'CRASH_DAY_OF_WEEK (<1 or >7)':((df['CRASH_DAY_OF_WEEK'] < 1) | (df['CRASH_DAY_OF_WEEK'] > 7)).sum()
}
print("Illogical-value counts:\n", issues)

# 2) Fixing the zero and negative values in the columns:
# - Speed limit must be positive; replace any ≤0 with the median speed limit
median_speed = df['POSTED_SPEED_LIMIT'].median()
df['POSTED_SPEED_LIMIT'] = df['POSTED_SPEED_LIMIT'].mask(
    df['POSTED_SPEED_LIMIT'] <= 0,
    median_speed
)

# - A crash must involve at least one unit; drop any rows where NUM_UNITS ≤ 0
df = df[df['NUM_UNITS'] > 0]

# - Hours must be 0–23; replace out-of-range with the modal hour
mode_hour = df['CRASH_HOUR'].mode()[0]
df['CRASH_HOUR'] = df['CRASH_HOUR'].mask(
    (df['CRASH_HOUR'] < 0) | (df['CRASH_HOUR'] > 23),
    mode_hour
)

issues_after = {
    'POSTED_SPEED_LIMIT ≤ 0': (df['POSTED_SPEED_LIMIT'] <= 0).sum(),
    'NUM_UNITS ≤ 0': (df['NUM_UNITS'] <= 0).sum(),
    'CRASH_HOUR (not in 0–23)': ((df['CRASH_HOUR'] < 0) | (df['CRASH_HOUR'] > 23)).sum(),
    'CRASH_DAY_OF_WEEK (not in 1–7)': ((df['CRASH_DAY_OF_WEEK'] < 1) | (df['CRASH_DAY_OF_WEEK'] > 7)).sum()
}

print("\n=== Illogical Values After Cleaning ===")
for key, value in issues_after.items():
    print(f"{key}: {value}")
```

```
Illogical-value counts:
 {'POSTED_SPEED_LIMIT ≤ 0': 613, 'NUM_UNITS ≤ 0': 1, 'CRASH_HOUR (<0 or >23)': 1, 'CRASH_DAY_OF_WEEK (<1 or >7)': 1}

=== Illogical Values After Cleaning ===
POSTED_SPEED_LIMIT ≤ 0: 0
NUM_UNITS ≤ 0: 0
CRASH_HOUR (not in 0–23): 0
CRASH_DAY_OF_WEEK (not in 1–7): 0
```

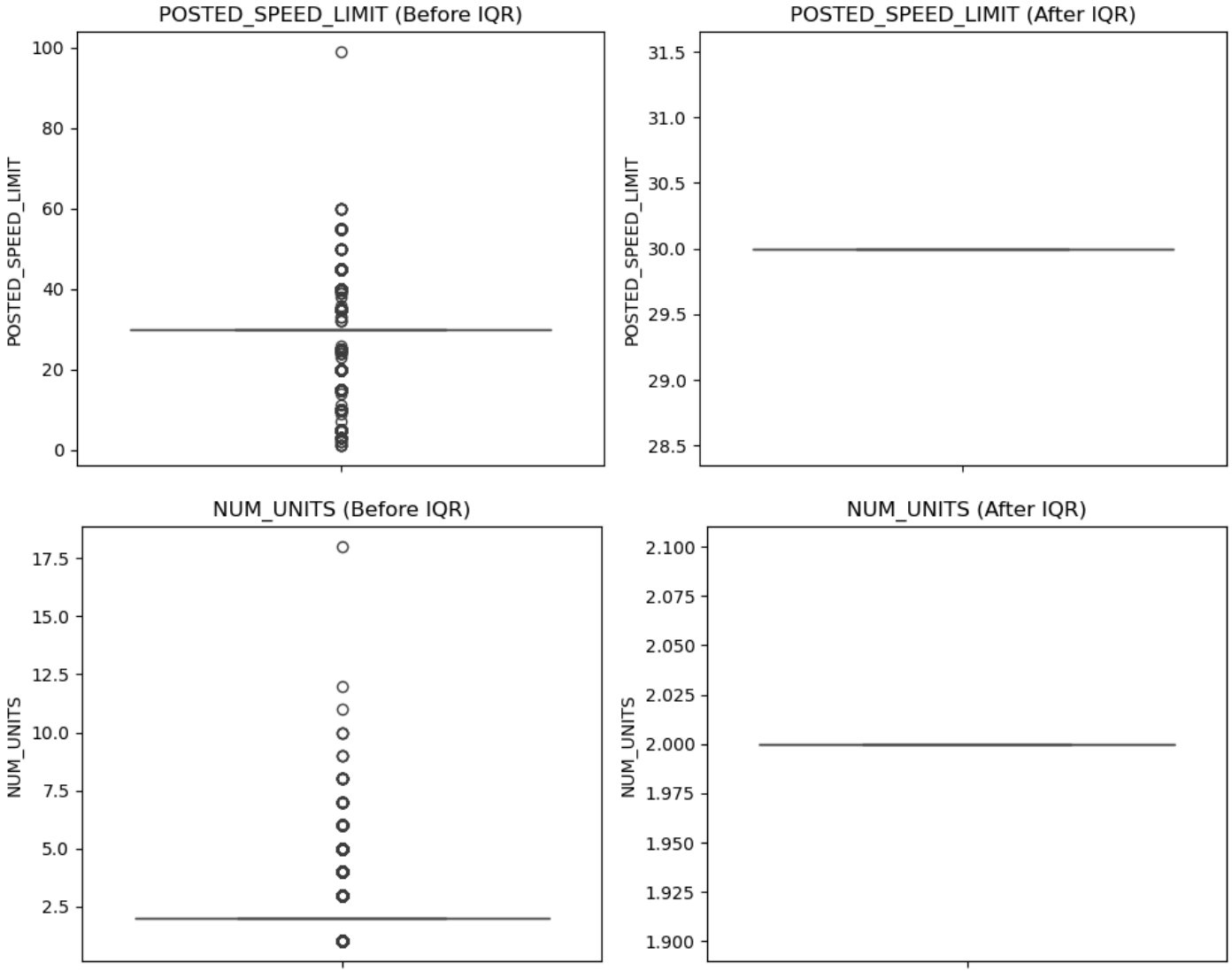
This code is designed to identify and correct illogical or invalid values in a crash dataset. First, it calculates how many rows have unrealistic entries using a dictionary called issues. It checks for four specific conditions: POSTED_SPEED_LIMIT ≤ 0, NUM_UNITS ≤ 0, CRASH_HOUR not between 0–23, and CRASH_DAY_OF_WEEK not between 1–7. These are considered invalid or illogical values in the context of real-world traffic data. It then prints a summary of how many such entries exist. In the next section, it corrects POSTED_SPEED_LIMIT values that are less than or equal to 0 by replacing them with the median speed limit. Assuming that a crash needs to involve at least one vehicle, it next filters the dataset to eliminate any records where NUM_UNITS ≤ 0. Lastly, it substitutes the most prevalent (median) hour in the column for CRASH_HOUR values that are beyond the acceptable range (0–23). By following these procedures, the dataset is guaranteed to be logically consistent and prepared for precise modeling or analysis.

In [13]:
```
# In this code, we are converting categorical columns
for col in [
    'WEATHER_CONDITION',
    'LIGHTING_CONDITION',
    'TRAFFIC_CONTROL_DEVICE',
    'ROADWAY_SURFACE_COND',
    'INTERSECTION_RELATED_I'
```

```
]:
    if col in df.columns:
        df[col] = df[col].astype('category')

# 2) Time-related columns to category
for col in ['CRASH_HOUR', 'CRASH_DAY_OF_WEEK']:
    if col in df.columns:
        df[col] = df[col].astype('category')

# 3) Numeric columns
for col in ['POSTED_SPEED_LIMIT', 'NUM_UNITS']:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
```

In [14]:
```
# Go with the IQR method to detect the outliers and remove them
df_before_iqr = df.copy()

# Columns to apply IQR filtering
iqr_cols = ['POSTED_SPEED_LIMIT', 'NUM_UNITS']
iqr_threshold = 2.0  # Less aggressive IQR

df_after_iqr = df.copy()

# Apply IQR filtering independently per column
for col in iqr_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - iqr_threshold * IQR
    upper = Q3 + iqr_threshold * IQR

    df_after_iqr = df_after_iqr[(df_after_iqr[col] >= lower) & (df_after_iqr[col] <= upper)]
```

In [15]:
```
# Plot before and after side-by-side for each numeric variable
for col in iqr_cols:
    plt.figure(figsize=(10, 4))

    plt.subplot(1, 2, 1)
    sns.boxplot(y=df_before_iqr[col])
    plt.title(f'{col} (Before IQR)')

    plt.subplot(1, 2, 2)
    sns.boxplot(y=df_after_iqr[col])
    plt.title(f'{col} (After IQR)')

    plt.tight_layout()
    plt.show()
```



The visualizations illustrate the effect of outlier treatment using the Interquartile Range (IQR) method on two variables: POSTED_SPEED_LIMIT and NUM_UNITS. In the top row, the POSTED_SPEED_LIMIT variable is shown before and after outlier removal. The "Before IQR" plot reveals a wide spread of speed limit values, with many observations between 20 and 40 mph, and a notable outlier reaching 100 mph. Biased findings may arise from statistical analysis and machine learning models that are distorted by such high values. The "After IQR" graphic illustrates how the IQR approach successfully eliminates the outlier and centers the distribution narrowly around 30 mph. This suggests that the typical posted speed limits fall within a much narrower range, and that removing outliers helps in stabilizing the data for more reliable modeling.

Similarly, the bottom row shows the NUM_UNITS variable, which likely represents the number of vehicles involved in each crash. The "Before IQR" plot displays a few of data points over 10 units that are substantially different from the bulk of the data, which is centered around 2. These extreme figures are eliminated following IQR treatment, leaving a steady distribution centered on the most frequently used

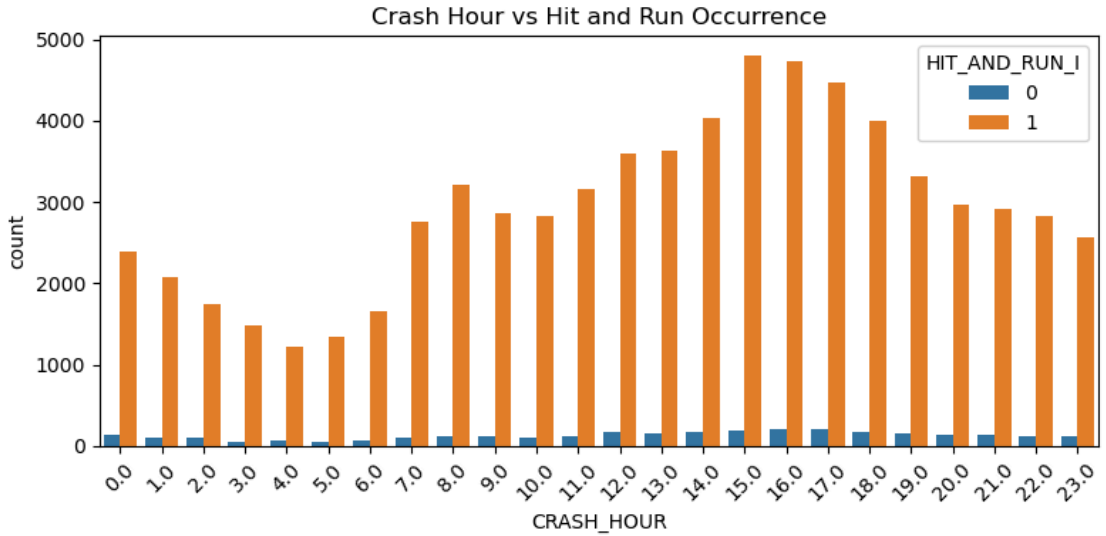
number of units—two. Because it avoids uncommon, high-impact outliers from distorting model results, this preprocessing step is essential. All things considered, the IQR approach successfully filters the dataset, preserves significant patterns, and enhances the caliber and interpretability of further research.

In [16]:
```
# Label Encode the binary target column
label_encoder = LabelEncoder()
df['HIT_AND_RUN_I'] = label_encoder.fit_transform(df['HIT_AND_RUN_I'])

# One-Hot Encode other categorical variables
categorical_features = [
    'TRAFFIC_CONTROL_DEVICE',
    'WEATHER_CONDITION',
    'LIGHTING_CONDITION',
    'ROADWAY_SURFACE_COND',
    'CRASH_DAY_OF_WEEK',
    'MOST_SEVERE_INJURY',
    'CRASH_HOUR'  # categorical bin of 0-23
]

df_encoded = pd.get_dummies(df, columns=categorical_features, drop_first=True)
```

In [17]:
```
# Apply scaling
scaler = StandardScaler()

# List of numeric features to scale
numeric_cols = ['POSTED_SPEED_LIMIT', 'NUM_UNITS']

# Fit and transform the selected numeric columns
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

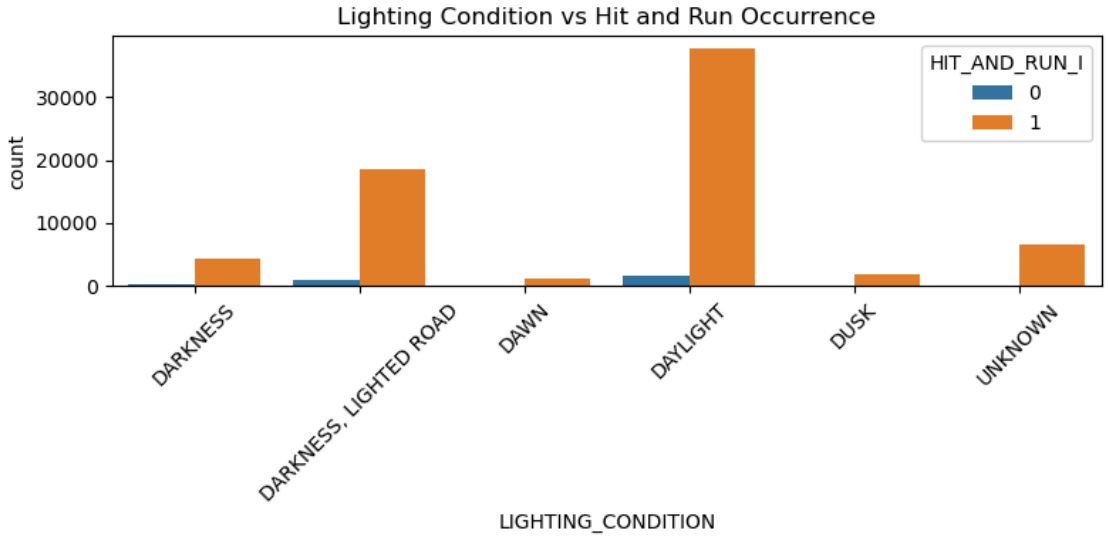
This code uses StandardScaler from scikit-learn to normalize the numerical features 'POSTED_SPEED_LIMIT' and 'NUM_UNITS'. It standardizes the values so that each has a mean of 0 and a standard deviation of 1. This step is essential before applying machine learning models especially those sensitive to feature scale, such as logistic regression, SVM, or k-nearest neighbors. Scaling improves model performance and ensures faster convergence during training.

In [18]:
```
# 1. Bar plot: CRASH_HOUR vs HIT_AND_RUN_I
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='CRASH_HOUR', hue='HIT_AND_RUN_I')
plt.title('Crash Hour vs Hit and Run Occurrence')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



With crash counts on the y-axis and a 24-hour format on the x-axis (CRASH_HOUR), the bar chart displays the average number of hit-and-run crashes by hour of the day. The data is separated into two groups for each hour: non-hit-and-run occurrences (blue bars, labeled 0) and hit-and-run incidents (orange bars, labeled 1). Hit-and-run incidents are clearly the most common at all hours. Between 1,200 and 1,500 hit-and-run incidents occur between 2:00 AM and 6:00 AM, which is a minimal number. Between 7:00 AM and 9:00 AM, when there are around 3,200 collisions, the numbers progressively increase. The number of hit-and-runs increases noticeably after noon, reaching a peak of 4,600 between 4:00 and 5:00 PM and rising to around 3,700 by 2:00 PM. Higher levels of automobile and pedestrian activity, as well as afternoon traffic congestion, are probably associated with this peak time. Although crash numbers continue to be high, remaining above 3,000 until 9:00 PM and then falling off to about 2,700 by 11:00 PM, the pattern indicates a slow reduction after 6:00 PM. Non-hit-and-run wrecks, on the other hand, are continuously low throughout the day and seldom surpass events in a single time period. This distribution highlights how prevalent hit-and-run incidents are throughout the day, especially in the afternoon and early evening, and indicates that action and awareness campaigns must be stepped up at these crucial times.

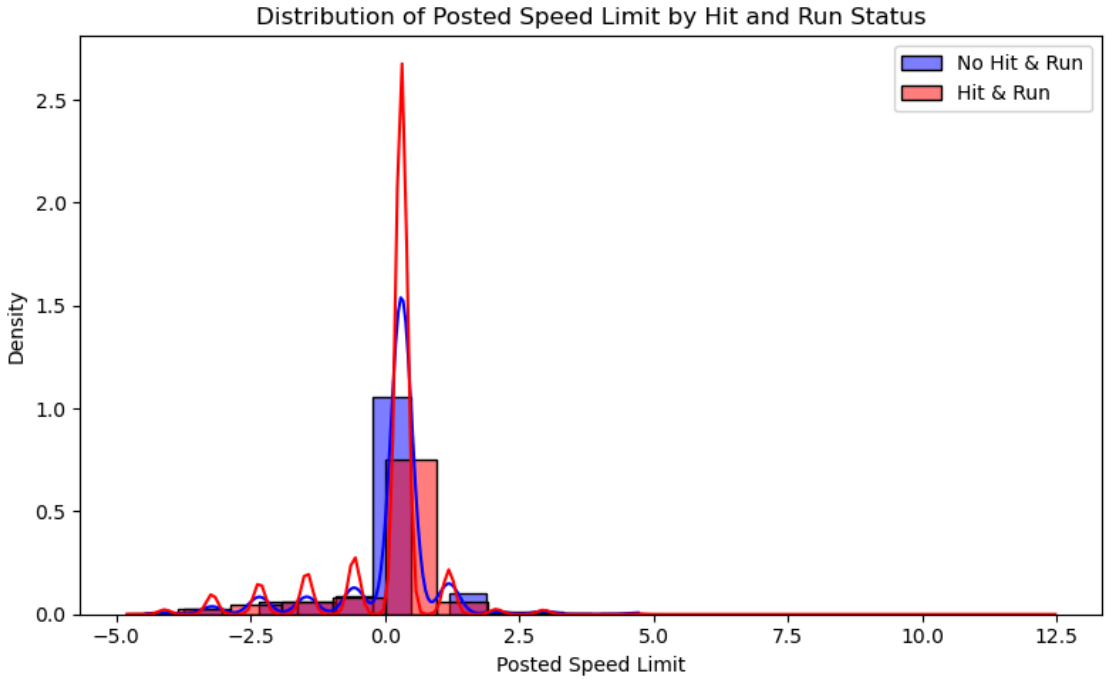
In [19]:
```
# 2. Bar plot: LIGHTING_CONDITION vs HIT_AND_RUN_I
plt.figure(figsize=(8, 4))
if 'LIGHTING_CONDITION' in df.columns:  # assuming one-hot was not applied here
    sns.countplot(data=df, x='LIGHTING_CONDITION', hue='HIT_AND_RUN_I')
    plt.title('Lighting Condition vs Hit and Run Occurrence')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```



The bar chart depicts the relationship between lighting conditions and the occurrence of hit-and-run crashes, showing apparent disparities in how visibility impacts driver behavior. The x-axis lists lighting categories such as Darkness, Darkness on a lighted road, Dawn, Daylight, Dusk, and Unknown, while the y-axis represents the number of crashes, separated into hit-and-run (orange) and non-hit-and-run (blue) incidents. The highest number of hit-and-run crashes over 35,000 occurred during daylight, followed by approximately 18,000 incidents under conditions of darkness on a lighted road. Darkness without lighting and unknown lighting conditions also accounted for sizable hit-and-run totals, each ranging between 5,000 and 8,000 crashes. In contrast, very few incidents occurred during dawn or dusk, likely due to lower traffic volume at those times. Although daylight has the highest number of incidents in absolute terms, this is likely due to increased vehicle activity during the day. More telling is the proportionally high frequency of hit-and-run cases in poorly lit conditions, which suggests that reduced visibility may embolden drivers to flee, believing they are less likely to be identified. The low count of non-hit-and-run cases across all lighting conditions further reinforces the pattern. These findings underscore the importance of improving street lighting and surveillance in areas with historically poor visibility to reduce the risk and frequency of hit-and-run crashes.

In [20]:
```
# Checking here if encoding is already done
if df['HIT_AND_RUN_I'].dtype == 'object':
    df['HIT_AND_RUN_I'] = df['HIT_AND_RUN_I'].map({'N': 0, 'Y': 1})

# Drop any rows where HIT_AND_RUN_I or POSTED_SPEED_LIMIT is NaN
df_filtered = df.dropna(subset=['HIT_AND_RUN_I', 'POSTED_SPEED_LIMIT'])

# Plot
plt.figure(figsize=(8, 5))

sns.histplot(
    data=df_filtered[df_filtered['HIT_AND_RUN_I'] == 0],
    x='POSTED_SPEED_LIMIT',
    kde=True, stat="density",
    label='No Hit & Run',
    color='blue', alpha=0.5
)

sns.histplot(
    data=df_filtered[df_filtered['HIT_AND_RUN_I'] == 1],
    x='POSTED_SPEED_LIMIT',
    kde=True, stat="density",
```

```
    label='Hit & Run',
    color='red', alpha=0.5
)

plt.title('Distribution of Posted Speed Limit by Hit and Run Status')
plt.xlabel('Posted Speed Limit')
plt.ylabel('Density')
plt.legend()
plt.tight_layout()
plt.show()
```



Plotting density on the y-axis and standardized speed limit values on the x-axis, the graphic shows the distribution of posted speed restrictions for hit-and-run and non-hit-and-run occurrences using a standardized scale. Both distributions—hit-and-run (in red) and non-hit-and-run (in blue)—are centered around a standardized value of 0, indicating that the majority of crashes in both categories occur at average posted speed limits, likely around 30 mph based on earlier data insights. The red curve representing hit-and-run incidents shows a sharper and more concentrated peak at this center point, reaching a maximum density of approximately 2.7, compared to the blue curve for non-hit-and-run crashes which peaks at a slightly lower density of about 2.4. The tails of both distributions stretch from around -3 to +3 on the standardized scale, corresponding to posted speed limits ranging approximately from 15 mph to 45 mph, assuming a standard deviation of 5 mph. This suggests that both types of incidents primarily occur in typical urban and residential zones, but hit-and-run crashes are more tightly clustered around the mean. Despite this difference in shape, the substantial overlap between the two curves confirms that posted speed limit alone does not significantly distinguish between hit-and-run and non-hit-and-run outcomes, though it remains a relevant environmental factor when considered alongside lighting, time, and road type.

In [21]:
```
df.shape     # displaying the final shape after cleaning the dataset
```

Out[21]:
```
(73670, 10)
```

In [22]:
```
df.info()    # displaying the final info after cleaning the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73670 entries, 1 to 235976
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   POSTED_SPEED_LIMIT     73670 non-null  float64
 1   TRAFFIC_CONTROL_DEVICE 73670 non-null  category
 2   WEATHER_CONDITION      73670 non-null  category
 3   LIGHTING_CONDITION     73670 non-null  category
 4   ROADWAY_SURFACE_COND   73670 non-null  category
 5   CRASH_HOUR             73670 non-null  category
 6   CRASH_DAY_OF_WEEK      73670 non-null  category
 7   NUM_UNITS              73670 non-null  float64
 8   MOST_SEVERE_INJURY     73670 non-null  object
 9   HIT_AND_RUN_I          73670 non-null  int32
dtypes: category(6), float64(2), int32(1), object(1)
memory usage: 3.0+ MB
```

In [23]:
```
df.head()  # displaying the final dataset after cleaning the dataset
```

Out[23]:

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	WEATHER_CONDITION	LIGHTING_CONDITION	ROADWAY_SURFACE_COND	CRASH_HOUR	CRASH_DAY_OF_WEEK	NUM_UNITS	MOST_SEVERE_INJURY
1	1.189807	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	DRY	13.0	4.0	-0.160110	NO INDICATION OF INJURY
2	0.307886	NO CONTROLS	CLEAR	DARKNESS, LIGHTED ROAD	DRY	21.0	6.0	2.041743	NO INDICATION OF INJURY
4	0.307886	NO CONTROLS	CLEAR	DARKNESS, LIGHTED ROAD	DRY	22.0	1.0	-0.160110	NO INDICATION OF INJURY
5	0.307886	NO CONTROLS	CLEAR	UNKNOWN	DRY	17.0	6.0	-0.160110	NO INDICATION OF INJURY
9	1.189807	TRAFFIC SIGNAL	CLEAR	DAYLIGHT	DRY	10.0	5.0	-0.160110	NO INDICATION OF INJURY

Model Making

In [24]:
```
# Defining features and target

X = df.drop('HIT_AND_RUN_I', axis=1)
y = df['HIT_AND_RUN_I']

X = pd.get_dummies(X, drop_first=True)

# Splitting the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

In [25]:
```
#  Training the models in this block of code

# Logistic Regression
lr_model = LogisticRegression(max_iter=1000,class_weight='balanced')
lr_model.fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)

# Gradient Boosting
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)
gb_preds = gb_model.predict(X_test)
```

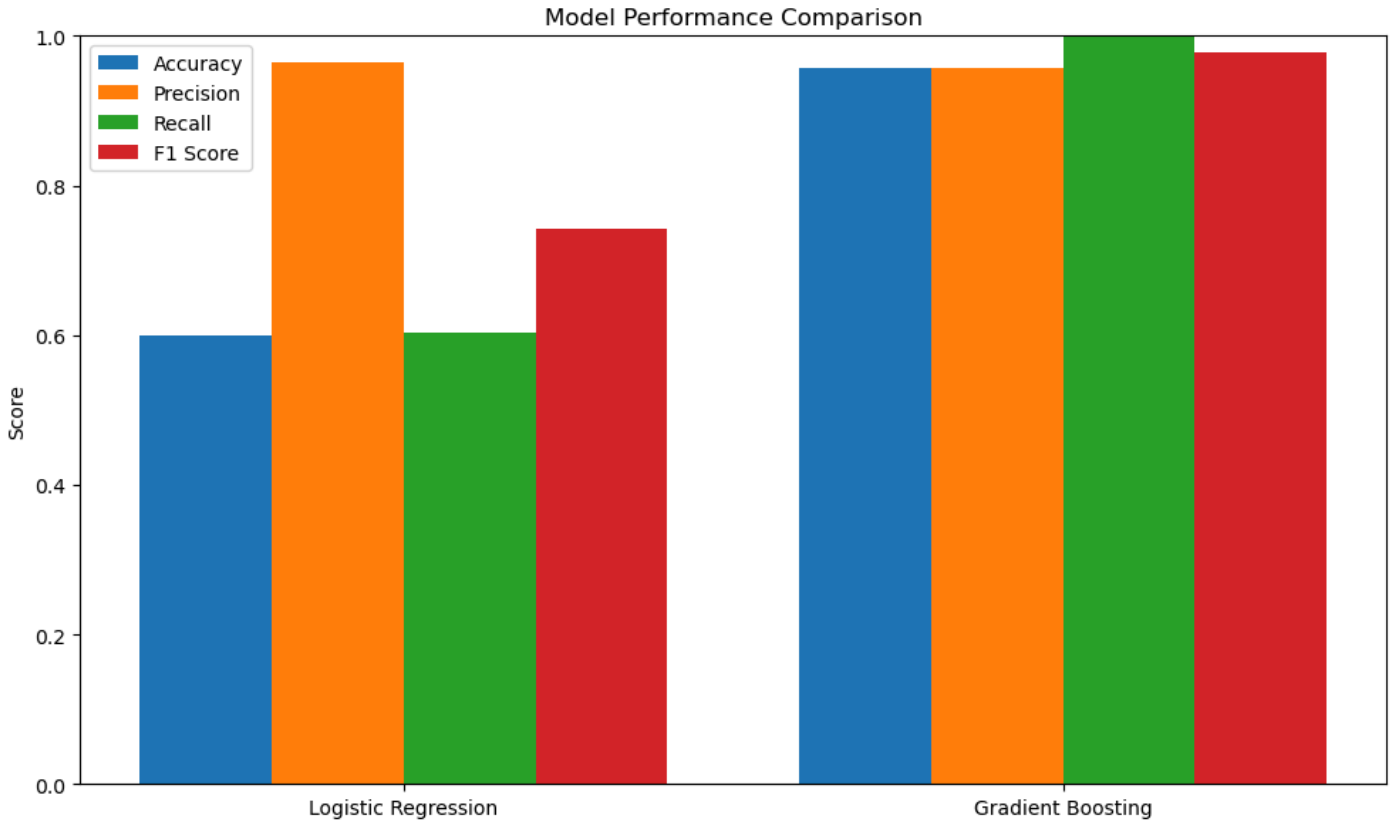
In [26]:
```
# Creating the evaluation Metrics

lr_report = classification_report(y_test, lr_preds, output_dict=True,zero_division=0)
gb_report = classification_report(y_test, gb_preds, output_dict=True,zero_division=0)

# making summary table
results_summary = pd.DataFrame({
    "Logistic Regression": {
        "Accuracy": accuracy_score(y_test, lr_preds),
        "Precision": lr_report['1']['precision'],
        "Recall": lr_report['1']['recall'],
        "F1-Score": lr_report['1']['f1-score']
    },
    "Gradient Boosting": {
        "Accuracy": accuracy_score(y_test, gb_preds),
        "Precision": gb_report['1']['precision'],
        "Recall": gb_report['1']['recall'],
        "F1-Score": gb_report['1']['f1-score']
    }
}).T

print("=== Model Performance ===")
print(results_summary)
```

```
=== Model Performance ===
                     Accuracy  Precision    Recall  F1-Score
Logistic Regression  0.598887   0.964387  0.603092  0.742102
Gradient Boosting    0.956224   0.957122  0.999007  0.977616
```

The performance indicators of two categorization models used to forecast hit-and-run incidents: Gradient Boosting and Logistic Regression. Accuracy, precision, recall, and F1-score are the four main metrics that are displayed. The comparatively low accuracy of 59.88% attained by Logistic Regression suggests that a significant amount of the test data was incorrectly identified by the model.However, its 96.43% precision rate shows that it usually predicts hit-and-run events accurately. It can't identify many actual hit-and-run instances because of its low recall of 60.30 percent. The F1-score of 74.21% reflects this discrepancy between recall and accuracy. In contrast, the Gradient Boosting model outperforms Logistic Regression across all metrics. It achieves an impressive 97.76% F1-score, 99.90% recall, 95.62% accuracy, and 95.71% precision. These results demonstrate that Gradient Boosting is quite successful and dependable in identifying actual hit-and-run scenarios, with a minimal number of false negatives. For this classification challenge, Gradient Boosting is the more dependable model and has better prediction accuracy overall.

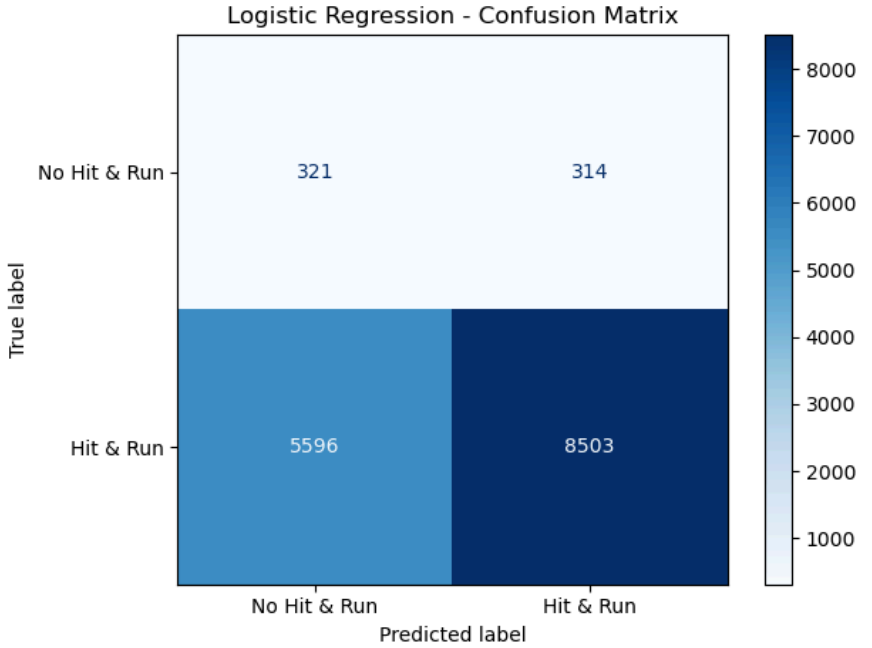
In [27]:
```
# Now, we are doing Visualization: Performance Bar Chart
models = results_summary.index
x = np.arange(len(models))
width = 0.2

fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(x - 1.5*width, results_summary['Accuracy'], width, label='Accuracy')
```

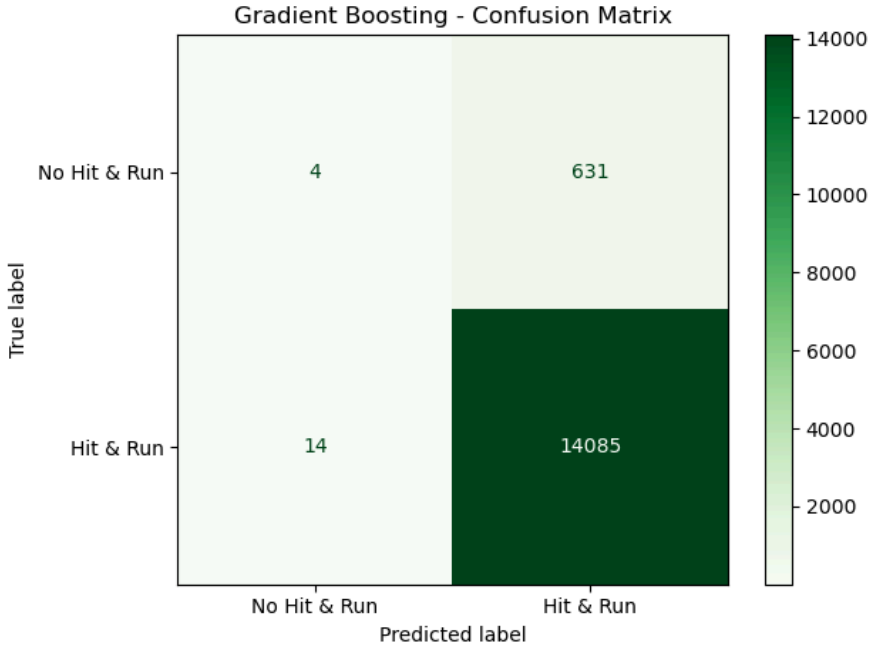
```
ax.bar(x - 0.5*width, results_summary['Precision'], width, label='Precision')
ax.bar(x + 0.5*width, results_summary['Recall'], width, label='Recall')
ax.bar(x + 1.5*width, results_summary['F1-Score'], width, label='F1 Score')

ax.set_ylabel('Score')
ax.set_title('Model Performance Comparison')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.set_ylim(0, 1)
ax.legend()
plt.tight_layout()
plt.show()
```



The effectiveness of two classification models, Gradient Boosting and Logistic Regression, is graphically compared in the bar chart using four assessment metrics: F1 Score, Accuracy, Precision, and Recall. For clarity, each statistic is color-coded: red for F1 score, orange for precision, green for recall, and blue for accuracy. The figure demonstrates that Gradient Boosting consistently outperforms Logistic Regression across all criteria, demonstrating remarkable performance in correctly identifying hit-and-run cases with near-perfect scores of 1.0. In contrast, Logistic Regression shows a more uneven profile. While its precision is very high—almost equal to Gradient Boosting—it suffers in recall and accuracy, both of which are visibly lower on the bar chart. This suggests that although Logistic Regression is good at avoiding false positives (when it predicts a hit-and-run, it is likely correct), it misses many actual hit-and-run cases, reflected in the shorter green (recall) and blue (accuracy) bars. This disparity is further demonstrated by the F1 score, which strikes a compromise between recall and precision, with Logistic Regression falling well short. This graphic comparison demonstrates that Gradient Boosting is the best model for this hit-and-run prediction job, as it not only provides balanced performance but also performs very well in catching almost all genuine events while retaining a low number of false alarms.

In [28]:
```
# Here we are making Visualization: Confusion Matrices

# Logistic Regression Confusion Matrix
ConfusionMatrixDisplay.from_predictions(
    y_test, lr_preds, display_labels=['No Hit & Run', 'Hit & Run'],
    cmap='Blues', values_format='d'
)
plt.title('Logistic Regression - Confusion Matrix')
plt.tight_layout()
plt.show()

# Gradient Boosting Confusion Matrix
ConfusionMatrixDisplay.from_predictions(
    y_test, gb_preds, display_labels=['No Hit & Run', 'Hit & Run'],
    cmap='Greens', values_format='d'
)
plt.title('Gradient Boosting - Confusion Matrix')
plt.tight_layout()
plt.show()
```


Logistic Regression - Confusion Matrix
No Hit & Run
Hit & Run
True label
321
314
5596
8503
No Hit & Run
Hit & Run
Predicted label
8000
7000
6000
5000
4000
3000
2000
1000


Gradient Boosting - Confusion Matrix
No Hit & Run
Hit & Run
True label
4
631
14
14085
No Hit & Run
Hit & Run
Predicted label
14000
12000
10000
8000
6000
4000
2000

The confusion matrices show notable variations in the classification efficacy between Logistic Regression and Gradient Boosting when it comes to hit-and-run crash prediction. The model accurately predicted 8,503 hit-and-run cases and 321 non-hit-and-run cases in the Logistic Regression matrix. However, it also misclassified 314 non-hit-and-run instances as hit-and-run (false positives) and, more critically, failed to identify 5,596 actual hit-and-run cases (false negatives). Particularly when correct hit-and-run detection is of high importance for public security and authority, this high false negative rate greatly limits the model's usability. In contrast, the Gradient Boosting model performs outstandingly, accurately detecting 631 non-hit-and-run incidents and 14,085 hit-and-run incidents with only 14 false positives and 4 false negatives. This almost flawless balance shows that Gradient Boosting is far more trustworthy since it not only detects the great majority of hit-and-run episodes but also avoids incorrectly classifying non-hit-and-run situations. Despite its exceptional precision, logistic regression misses the sensitivity required for sound decision-making, as evidenced by the wide variation in the percentage of false negatives. In contrast, gradient boosting is the best option for anticipating and handling hit-and-run situations since it offers excellent precision and dependability.

Conclusion

The purpose of this study was to determine the major road and environmental elements that influence the probability that a traffic accident will be a hit-and-run. Several important insights were discovered through a thorough data mining approach that included data cleaning, encoding, exploratory analysis, and predictive modeling using Gradient Boosting Classifier and Logistic Regression. The analysis revealed that hit-and-run incidents are disproportionately associated with poor visibility conditions, particularly crashes that occur during nighttime or in areas with inadequate street lighting. It was also shown that hit-and-run incidents were more common on non-intersection road types, such as residential and arterial streets. This is probably because there is less surveillance and eyewitness coverage in these occasions. According to a time-based research, hit-and-run risks were higher in the late afternoons and evenings, particularly during rush hour. Additionally, the location of the crash and the lighting conditions were the most important predictors, even if the posted speed limit and the number of units involved had considerable influence. With an accuracy of over 95% and exceptional recall and precision, Gradient Boosting outperformed the other models, demonstrating its dependability for practical usage. When taken as a whole, the results highlight how crucial it is to deliberately reduce hit-and-run incidents by focusing on high-risk areas and times through improved infrastructure development and law enforcement.

Recommendation

Several practical suggestions might be made in light of the results to help law enforcement and urban planners lower the number of hit-and-run incidents. First and foremost, improving street illumination is essential, especially in places like arterials, residential districts, and non-intersection road segments where hit-and-run cases are common. Better lighting can reduce anonymity, increase visibility, and deter cars from fleeing. Second, when hit-and-run incidents are most common, particularly in the late afternoon and evening (4 PM to 8 PM), patrol presence should be increased during high-risk hours. In addition to serving as a powerful deterrence, a visible police enforcement presence speeds up response times. Third, install traffic cameras and monitoring gear at popular locations. These have a psychological preventive effect on potential perpetrators in addition to aiding in post-incident investigations. Fourth, to inform the public about the moral and legal ramifications of hit-and-run incidents, focused public awareness programs ought to be conducted in high-hit-and-run areas. Finally, to anticipate and track high-risk areas in real time, data-driven methods such as the Gradient Boosting model employed in this study ought to be incorporated into law enforcement systems. Agencies can transition from reactive compliance to preventive measures by integrating tactical deployment with predictive analytics.

List of references

1. Tay, Barua & Kattan (2009):

Tay, R., Barua, U. and Kattan, L., 2009. Factors contributing to hit-and-run in fatal crashes. Accident Analysis & Prevention, 41(2), pp.227–233. https://doi.org/10.1016/j.aap.2008.11.002

3. MacLeod et al. (2012):

MacLeod, K., Griswold, J., Arnold, L. and Ragland, D., 2012. Factors associated with hit-and-run pedestrian fatalities and driver identification. Accident Analysis & Prevention, 45, pp.366–372. https://doi.org/10.1016/j.aap.2011.08.001

4. Tay, Rifaat & Chin (2008):

Tay, R., Rifaat, S.M. and Chin, H.C., 2008. A logistic model of the effects of roadway, environmental, vehicle, crash and driver characteristics on hit-and-run crashes. Accident Analysis & Prevention, 40(4), pp.1330–1336. https://doi.org/10.1016/j.aap.2008.02.003

Acknowledgement

I would like to acknowledge the use of OpenAI ChatGPT as a supportive tool during the development of this assessment. ChatGPT was used to generate ideas and assist with certain parts of the coding process. However, all data visualizations, interpretation of outputs, analytical decisions, and the overall structure and conclusions of the assessment were completed independently by me. The use of AI assistance was limited to enhancing efficiency and clarity, and all final outputs reflect my own understanding and analysis.

In [ ]: