

Project Phase 3 Report

Data Intensive Computing, Fall 2023

Motor Vehicle Collisions and Crashes

Instructor:

Prof. Shamshad Parveen

Prepared By:

**Devendra Mandava, dmandava
Hemanth Kongara, hkongara
Sethu Thakkilapati, sethutha**

Introduction:

- In today's urbanized environments, vehicular traffic is more than just a means of transportation; it's a defining feature of our daily lives. With millions of vehicles on the roads, traffic collisions have unfortunately become a frequent occurrence. These incidents not only result in significant economic costs but, more importantly, can lead to injuries and loss of life.
- The provided dataset offers a detailed snapshot of vehicle collisions in a city, encompassing specifics such as the time and date of the crash, the exact location, involved vehicle types, and contributing factors. Each record captures a unique incident, shedding light on the myriad circumstances and factors that converge to result in a collision.
- Through the lens of data analytics, this project embarks on a mission to transform raw numbers into actionable intelligence, with the hope of making our roads a safer place for everyone.
- Based on the provided dataset, which appears to be related to vehicle collisions, you can frame a project around traffic safety, accident analysis, and prediction. Here's a potential project statement and objectives.

Project Statement:

To analyze and identify patterns in vehicle collisions, their causes, and related factors with the goal of proposing data-driven strategies and solutions to reduce accidents and improve road safety.

Project Objectives:

1. Descriptive Analysis: Examine accident distributions based on time, location, and borough to identify high-risk periods and zones.
2. Factor Analysis: Understand primary contributing factors for collisions and any vehicle type patterns associated with higher accident rate

1. **Data Handling:** Address missing data, particularly in key columns, ensuring a cleaner dataset for accurate analysis.
2. **Predictive Modeling:** Forecast potential accident hotspots and risk factors to aid in proactive safety measures.
3. **Recommendation and Strategies:** Offer data-driven traffic and safety solutions to reduce accidents.

Dataset Overview: <https://data.gov/> It is us government data

Dimensions of the data: (2031372,29)

Dataset Description:

1. CRASH DATE:

The date when the collision occurred. This can help in analyzing trends over time, and identifying high-risk periods like holidays or seasons, etc.

2. CRASH TIME:

The specific time when the accident took place. It can be used to pinpoint rush hours, nighttime risks, or other time-specific trends.

3. BOROUGH:

This represents the specific borough or district where the accident happened. Analyzing by borough can reveal localized issues or high-risk areas.

4. ZIP CODE:

The postal code where the collision occurred. This adds another layer of location-specific analysis, possibly more granular than the borough.

5. LATITUDE & LONGITUDE:

Exact geographical coordinates of the accident. Useful for mapping accidents, creating heat maps, or identifying specific accident hotspots.

6. LOCATION:

A combination of latitude and longitude, likely representing the exact location in a format that's easier to read than separate coordinates.

7.ON-STREET NAME:

The street where the accident occurred. Helps in pinpointing high-risk streets or intersections.

8. CROSS STREET NAME & OFF STREET NAME:

Additional location details to identify specific sections of roads or intersections where accidents are common.

9. CONTRIBUTING FACTOR VEHICLE (1-5):

Reasons or factors contributing to the accident. There are multiple columns suggesting accidents can have multiple simultaneous contributing factors. Analyzing these can shed light on common causes of accidents.

10. COLLISION_ID:

A unique identifier for each collision. Useful for indexing and referencing specific incidents.

11. VEHICLE TYPE CODE (1-5):

The type of vehicles involved in the collision. Multiple columns indicate that a single accident can involve various vehicle types. Understanding the most common vehicles in accidents can guide safety regulations or awareness campaigns.

Data Cleaning and Processing

1. Importing all the libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Python

We have imported “pandas” for the Data manipulation and analysis library,”matplotlib” for the 2D plotting library, “seaborn” for the Data visualization library based on matplotlib, and “numpy” for the Numerical computing library.

2. Data Statistics:

We have used “df.shape”To get the dimensions of the data frame. And “df.describe()” to Provide a quick statistical summary of the numeric columns in the data frame.

```
df.shape
```

Python

```
(2031372, 29)
```

```
df.describe()
```

| | LATITUDE | LONGITUDE | NUMBER OF PERSONS INJURED | NUMBER OF PERSONS KILLED | NUMBER OF PEDESTRIANS INJURED | NUMBER OF PEDESTRIANS KILLED | NUMBER OF CYCLIST INJURED | NUMBER OF CYCLIST KILLED | NUMBER OF MOTORIST INJURED |
|-------|--------------|---------------|---------------------------|--------------------------|-------------------------------|------------------------------|---------------------------|--------------------------|----------------------------|
| count | 1.800708e+06 | 1.800708e+06 | 2.031354e+06 | 2.031341e+06 | 2.031372e+06 | 2.031372e+06 | 2.031372e+06 | 2.031372e+06 | 2.031372e+06 |
| mean | 4.062776e+01 | -7.375228e+01 | 3.042729e-01 | 1.454704e-03 | 5.538129e-02 | 7.251257e-04 | 2.638857e-02 | 1.127317e-04 | 2.192026e-01 |
| std | 1.980723e+00 | 3.726372e+00 | 6.953339e-01 | 4.018760e-02 | 2.417562e-01 | 2.740772e-02 | 1.622446e-01 | 1.066319e-02 | 6.564610e-01 |
| min | 0.000000e+00 | -2.013600e+02 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| 25% | 4.066791e+01 | -7.397492e+01 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| 50% | 4.072097e+01 | -7.392732e+01 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| 75% | 4.076956e+01 | -7.386670e+01 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| max | 4.334444e+01 | 0.000000e+00 | 4.300000e+01 | 8.000000e+00 | 2.700000e+01 | 6.000000e+00 | 4.000000e+00 | 2.000000e+00 | 4.300000e+01 |

3. Handling Redundant Columns:

Handling redundant or unnecessary columns is an essential step in the data preprocessing phase. Removing these columns can simplify analysis and may improve the performance and accuracy of machine learning models.

4. Handling Missing Values:

It's essential to handle missing numbers while preparing and analyzing data. The accuracy of your model or analysis may be impacted by missing data.

```
df.dropna(subset=['VEHICLE TYPE CODE 2'], inplace=True)
```

5. Replace null values with a mean of numerical columns:

Data cleansing commonly uses imputation to replace null (or missing) values with the mean of numerical columns. By preventing row deletions, this technique can aid in preserving the dataset's size. It can also offer a plausible approximation for missing values, particularly in cases when data is missing randomly.

```
numerical_columns = ["LATITUDE", "LONGITUDE", "NUMBER OF PERSONS INJURED", "NUMBER OF PERSONS KILLED"]

# Impute missing values with the mean
for column in numerical_columns:
    df[column].fillna(df[column].mean(), inplace=True)
```

6. Replace null values with a mode of categorical columns:

Replacing missing values with the mode of categorical columns is a common method used in data imputation. This approach can be especially relevant for categorical data where calculating measures like the mean doesn't make sense.

```
categorical_columns = ["CONTRIBUTING FACTOR VEHICLE 1", "VEHICLE TYPE CODE 1"]

# Impute missing values with the mode
for column in categorical_columns:
    mode_value = df[column].mode()[0]
    df[column].fillna(mode_value, inplace=True)
```

Python

7. Data Type Conversion:

df.dtypes

Python

```
CRASH DATE          object
CRASH TIME          object
LATITUDE            float64
LONGITUDE           float64
NUMBER OF PERSONS INJURED  float64
NUMBER OF PERSONS KILLED  float64
NUMBER OF PEDESTRIANS INJURED  int64
NUMBER OF PEDESTRIANS KILLED  int64
NUMBER OF CYCLIST INJURED  int64
NUMBER OF CYCLIST KILLED  int64
NUMBER OF MOTORIST INJURED  int64
NUMBER OF MOTORIST KILLED  int64
CONTRIBUTING FACTOR VEHICLE 1  object
COLLISION_ID        int64
VEHICLE TYPE CODE 1  object
VEHICLE TYPE CODE 2  object
dtype: object
```

So, we have changed 'CONTRIBUTING FACTOR VEHICLE 1', 'VEHICLE TYPE CODE 1', 'VEHICLE TYPE CODE 2' from object type to category type.

```
df['CONTRIBUTING FACTOR VEHICLE 1'] = df['CONTRIBUTING FACTOR VEHICLE 1'].astype('category')
#df['CONTRIBUTING FACTOR VEHICLE 2'] = df['CONTRIBUTING FACTOR VEHICLE 2'].astype('category')
df['VEHICLE TYPE CODE 1'] = df['VEHICLE TYPE CODE 1'].astype('category')
df['VEHICLE TYPE CODE 2'] = df['VEHICLE TYPE CODE 2'].astype('category')
```

Python

8. Checking Duplicate Rows:

```

duplicates = df[df.duplicated()]
num_duplicates = duplicates.shape[0]
print("Number of duplicate rows:", num_duplicates)
print("Duplicate rows:")
print(duplicates)

```

Python

```

Number of duplicate rows: 0
Duplicate rows:
Empty DataFrame
Columns: [LATITUDE, LONGITUDE, NUMBER OF PERSONS INJURED, NUMBER OF PERSONS KILLED, NUMBER OF PEDESTRIANS INJURED, NUMBER OF PEDE
Index: []

```

So, from the above output, we can see that there are no duplicate rows in our data set.

9. Text Data Cleaning:

Converting all the categorical data of 'CONTRIBUTING FACTOR VEHICLE 1', 'VEHICLE TYPE CODE 1', and 'VEHICLE TYPE CODE 2' to the lowercase

```

#convert to lowercase

df['CONTRIBUTING FACTOR VEHICLE 1'] = df['CONTRIBUTING FACTOR VEHICLE 1'].str.lower()
df['CONTRIBUTING FACTOR VEHICLE 2'] = df['CONTRIBUTING FACTOR VEHICLE 2'].str.lower()
df['VEHICLE TYPE CODE 1'] = df['VEHICLE TYPE CODE 1'].str.lower()
df['VEHICLE TYPE CODE 2'] = df['VEHICLE TYPE CODE 2'].str.lower()

```

Python

10. Outlier Detection and Handling:

Outlier detection identifies data points significantly different from others using techniques like statistical methods, visualization, machine learning, and proximity-based algorithms. It's about finding the "odd ones out" in a dataset.

So, we have checked all the outliers in all the columns by we have found the outliers in "NUMBER OF PERSONS INJURED" and "NUMBER OF PERSONS KILLED".

Code for checking the outliers in "NUMBER OF PERSONS INJURED"

```

column_name = 'NUMBER OF PERSONS INJURED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")

```

Python

Output:

So, we have detected the outliers in the above column

```
12.0: 29
13.0: 24
15.0: 12
16.0: 8
14.0: 7
18.0: 6
17.0: 5
0.28047753958033483: 4
22.0: 3
19.0: 3
24.0: 3
20.0: 1
```

Code for checking the outliers in “NUMBER OF PERSONS KILLED”

```
column_name = 'NUMBER OF PERSONS KILLED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")
```

Python

Output:

So, we have detected the outliers in the above column also.

```
Unique Values and Counts for Column: NUMBER OF PERSONS KILLED
0.0: 1648956
1.0: 1113
2.0: 43
3.0: 8
0.0007508531182220962: 6
4.0: 2
8.0: 1
```

In all the remaining columns there are no outliers.

Code for checking the outliers in “NUMBER OF PEDESTRIANS INJURED”

```
column_name = 'NUMBER OF PEDESTRIANS INJURED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")
```

Python

Output:

```
Unique Values and Counts for Column: NUMBER OF PEDESTRIANS INJURED
0: 1646402
1: 3288
2: 333
3: 67
4: 18
5: 12
6: 3
7: 2
9: 1
13: 1
15: 1
8: 1
```

Code for checking the outliers in “NUMBER OF PEDESTRIANS KILLED”


```

column_name = 'NUMBER OF PEDESTRIANS KILLED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")

```

Python

Output:

```

Unique Values and Counts for Column: NUMBER OF PEDESTRIANS KILLED
0: 1649928
1: 193
2: 7
6: 1

```

Code for checking the outliers in “NUMBER OF CYCLIST INJURED”

```

column_name = 'NUMBER OF CYCLIST INJURED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")

```

Python

Output:

```

Unique Values and Counts for Column: NUMBER OF CYCLIST INJURED
0: 1602460
1: 47129
2: 520
3: 19
4: 1

```

Code for checking the outliers in “NUMBER OF MOTORIST INJURED”

```

column_name = 'NUMBER OF MOTORIST INJURED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")

```

Python

Output:

```

Unique Values and Counts for Column: NUMBER OF MOTORIST INJURED
0: 1390337
1: 170542
2: 56400
3: 20024
4: 7554
5: 2969
6: 1223
7: 522
8: 222
9: 113
10: 76

```

Code for checking the outliers in “NUMBER OF MOTORIST KILLED”

```

column_name = 'NUMBER OF MOTORIST KILLED'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")

```

Python

Output:

```

Unique Values and Counts for Column: NUMBER OF MOTORIST KILLED
0: 1649387
1: 699
2: 34
3: 8
4: 1

```

11. Removing the outliers:

- The code removes all rows from df where the value in the column 'NUMBER OF PERSONS INJURED' is 0.28047753958033483.

```
df = df[df['NUMBER OF PERSONS INJURED'] != 0.28047753958033483]
```

Python

- The code removes all rows from df where the value in the column 'NUMBER OF PERSONS KILLED' is 0.0007508531182220962.

```
df = df[df['NUMBER OF PERSONS KILLED'] != 0.0007508531182220962]
```

Python

Identifying the outliers in longitude and latitude:

```
numerical_columns = ['LATITUDE', 'LONGITUDE']

for column in numerical_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    print(f"Outliers in {column}: {outliers.shape[0]}")
```

Python

Output:

```
Outliers in LATITUDE: 2909
Outliers in LONGITUDE: 11751
```

12. Group rare categories into 'Other'.

We have grouped the rare categories into others as I have created the threshold =10 for VEHICLE TYPE CODE 1, and the same for VEHICLE TYPE CODE 2, the threshold =22 for CONTRIBUTING FACTOR VEHICLE 1

If the threshold is less than 10 then we have created other and stored the values there.

```
# Group rare categories into 'Other'
threshold = 10 # threshold for rarity
top_categories = df['VEHICLE TYPE CODE 1'].value_counts().index[:threshold]
df['VEHICLE TYPE CODE 1'] = df['VEHICLE TYPE CODE 1'].apply(lambda x: x if x in top_categories else 'Other')
```

Python

If the threshold is less than 10 then we have created other and stored the values there.

```
threshold = 10
top_categories = df['VEHICLE TYPE CODE 2'].value_counts().index[:threshold]
df['VEHICLE TYPE CODE 2'] = df['VEHICLE TYPE CODE 2'].apply(lambda x: x if x in top_categories else 'Other')
```

Python

If the threshold is less than 22 then we have created other and stored the values there.

```
threshold = 22
top_categories = df['CONTRIBUTING FACTOR VEHICLE 1'].value_counts().index[:threshold]
df['CONTRIBUTING FACTOR VEHICLE 1'] = df['CONTRIBUTING FACTOR VEHICLE 1'].apply(lambda x: x if x in top_categories else 'Other')
```

Python

After creating the threshold printing the unique values of 'CONTRIBUTING FACTOR VEHICLE 1'

```
column_name = 'CONTRIBUTING FACTOR VEHICLE 1'

value_counts = df[column_name].value_counts()

# Print the unique values and their counts
print("Unique Values and Counts for Column:", column_name)
for value, count in value_counts.items():
    print(f"{value}: {count}")
```

Output:

Unique values and counts for a column: "Contributing factor vehicle 1"

```
Unique Values and Counts for Column: CONTRIBUTING FACTOR VEHICLE 1
unspecified: 524525
driver inattention/distraction: 340235
following too closely: 101099
failure to yield right-of-way: 94801
backing unsafely: 63658
Other: 63313
other vehicular: 53968
passing or lane usage improper: 46997
fatigued/drowsy: 46839
turning improperly: 46547
passing too closely: 37879
unsafe lane changing: 37386
traffic control disregarded: 30910
driver inexperience: 26396
unsafe speed: 20476
lost consciousness: 19742
alcohol involvement: 17890
reaction to uninvolved vehicle: 15957
prescription medication: 14797
pavement slippery: 14552
outside car distraction: 10774
oversized vehicle: 10750
view obstructed/limited: 10631
```

12: Encoding Categorical Variables:

We have converted categorical values in the df into numerical values using the LabelEncoder from the sklearn.preprocessing module.

Steps:

1. A LabelEncoder object is instantiated as le.
2. The fit_transform method of the le object is applied to the 'CONTRIBUTING FACTOR VEHICLE 1' column, converting its unique categorical values to integers. This transformed data replaces the original column.

3. The same encoding process is applied to the 'VEHICLE TYPE CODE 2' and 'VEHICLE TYPE CODE 1' columns.

CODE:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['CONTRIBUTING FACTOR VEHICLE 1'] = le.fit_transform(df['CONTRIBUTING FACTOR VEHICLE 1'])
df['VEHICLE TYPE CODE 2'] = le.fit_transform(df['VEHICLE TYPE CODE 2'])
df['VEHICLE TYPE CODE 1'] = le.fit_transform(df['VEHICLE TYPE CODE 1'])
```

Python

13. Feature Engineering:

The code extracts specific time-related features from the df DataFrame's 'CRASH DATETIME' column:

It creates an 'Hour' column with the hour of the crash.

It creates a 'DayOfWeek' column indicating the day (0=Monday to 6=Sunday).

It creates an 'IsWeekend' column, marking if the crash occurred on a weekend (1 for yes, 0 for no).

```
df['Hour'] = df['CRASH DATETIME'].dt.hour
df['DayOfWeek'] = df['CRASH DATETIME'].dt.dayofweek
df['IsWeekend'] = df['DayOfWeek'].isin([5, 6]).astype(int)
```

Python

Exploratory Data Analysis

Firstly we use 'matplotlib' Library to visualize the data. Here's the code and explanation.

```
total_killed = df['NUMBER OF PERSONS KILLED'].sum()
total_injured = df['NUMBER OF PERSONS INJURED'].sum()

labels = ['Persons Killed', 'Persons Injured']
sizes = [total_killed, total_injured]
colors = ['red', 'blue']

plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Persons Killed vs. Persons Injured')
plt.axis('equal')

plt.show()
```

Step-1

Data Aggregation:

The code first aggregates (sums up) the number of persons killed and injured from a DataFrame named df.

Step-2

Setting Up Pie Chart Parameters:

labels: Defines the categories or segments of the pie chart: Persons Killed and Persons Injured.

sizes: Sets the sizes or values of each segment based on the aggregated data.

colors: Specifies the colors for each segment of the pie chart.

Step-3:

Plotting The Pie Chart:

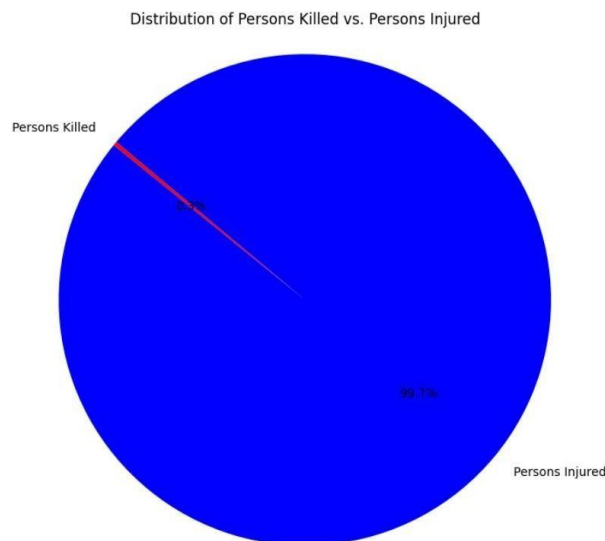
This aggregated data is then visualized using a pie chart. In the pie chart:

The segment representing "Persons Killed" is colored in red.

The segment representing "Persons Injured" is colored in blue.

Each segment of the pie chart displays its respective percentage to one decimal place.

Data Visualization:



The provided pie chart showcases the "Distribution of Persons Killed vs. Persons Injured." From the visual representation, it is evident that:

- "Persons Injured" constitutes the vast majority of the data, representing 99.7% of the total.
- Conversely, "Persons Killed" forms a much smaller portion, accounting for only 0.3% of the total.

Number of Crashes Occurred in weekdays and weekends.

```
weekend_crashes = df[df['IsWeekend'] == 1].shape[0]
weekday_crashes = df[df['IsWeekend'] == 0].shape[0]

categories = ['Weekend', 'Weekday']
crash_counts = [weekend_crashes, weekday_crashes]

plt.bar(categories, crash_counts, color=['blue', 'green'])
plt.xlabel('Day of the Week')
plt.ylabel('Number of Crashes')
plt.title('Comparison of Crashes on Weekends vs. Weekdays')
plt.show()
```

Step-1:

Data Extraction:

- The code extracts the number of crashes that occurred on weekends and weekdays from a DataFrame named df.
- The column IsWeekend appears to be a binary indicator where 1 denotes a weekend and 0 denotes a weekday.
- .shape[0] is used to get the number of rows (i.e., the count of crashes) for each category.

Step-2:

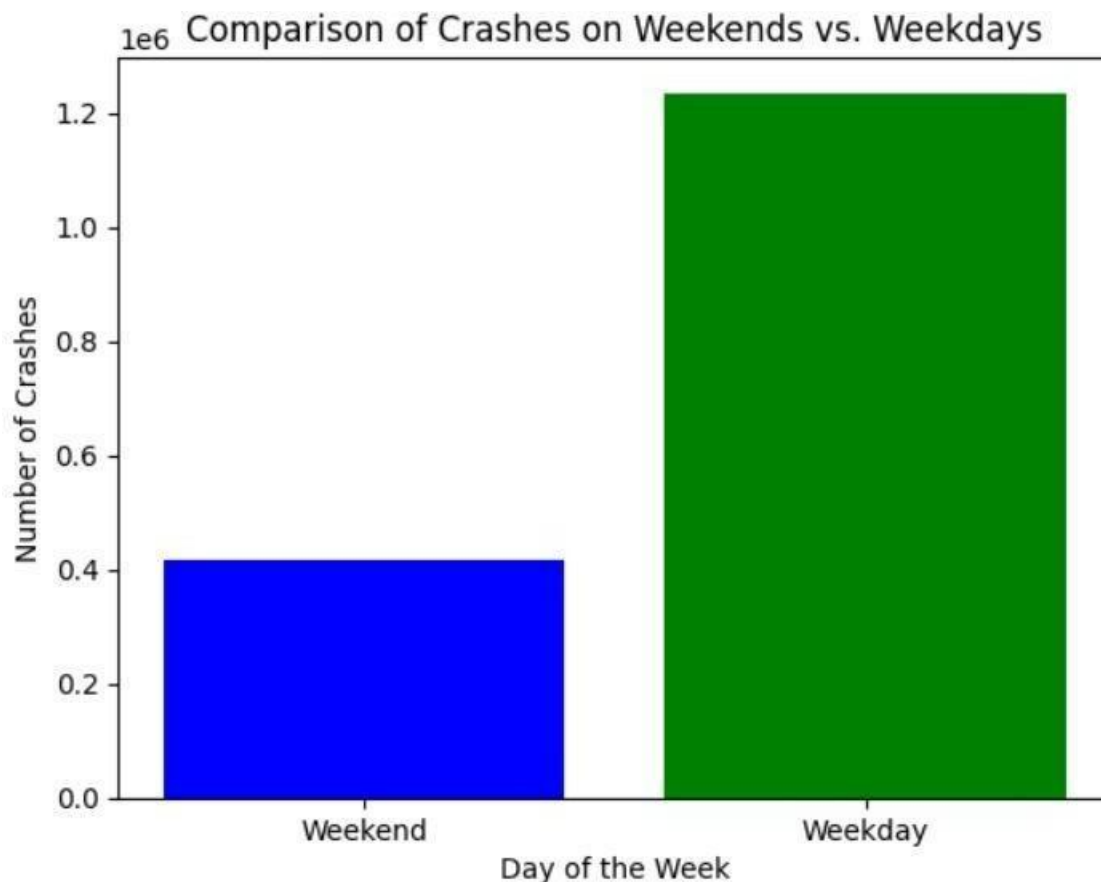
Setting Up Bar Chart Parameters:

- **categories:** Lists the two categories (Weekend and Weekday) to be displayed on the x-axis of the bar chart.
- **crash_counts:** Lists the count of crashes for each category to be displayed as the height of the bars.

Step-3:

Plotting The Bar Chart:

- visualizes the comparison of crashes on weekends versus weekdays from a dataset. Using the `IsWeekend` column in the DataFrame `df`, the code calculates the number of crashes for both categories. The resulting data is represented in a bar chart with two bars:
- The "Weekend" bar is colored in blue.
- The "Weekday" bar is colored in green.

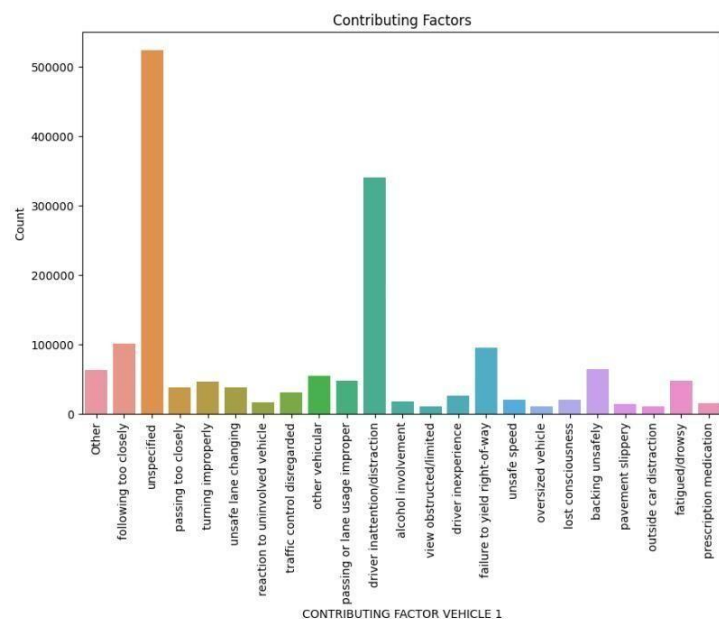


This bar chart visually represents the frequency of crashes based on the day of the week. Here's what can be inferred from the chart:

- Crashes on weekdays are significantly higher than on weekends.
- The blue bar represents crashes during the weekend, and its height suggests a relatively lower number of incidents.
- In contrast, the green bar represents crashes on weekdays, and it's notably taller, indicating a much higher count of crashes during these days.

```
# Visualization: Count plot for 'CONTRIBUTING FACTOR VEHICLE 1'
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='CONTRIBUTING FACTOR VEHICLE 1')
plt.xlabel('CONTRIBUTING FACTOR VEHICLE 1')
plt.ylabel('Count')
plt.title('Contributing Factors')
plt.xticks(rotation=90)
plt.show()
```

- The code uses the seaborn library, a popular data visualization tool in Python, to produce the bar chart. This chart helps in understanding the distribution and prominence of various contributing factors related to vehicle incidents from the df dataset. By rotating the x-axis labels, the visualization ensures clarity, especially when dealing with numerous categories. The chart's design, with its defined title and axis labels, aims to provide an intuitive understanding of the data, making it easier for viewers to draw insights.



This Chart visually presents various factors related to vehicle incidents, highlighting their frequencies based on the count. Key observations from the chart include:

- The most prominent contributing factor is labeled "Other", with its count significantly surpassing other categories.
- The factor "Driver Inexperience" stands out as the second most frequent contributing factor but has a substantially lower count than "Other".
- Several factors, such as "Following Too Closely", "Passing Too Closely Unspecified", "Turning Improperly", and "Failure to Yield Right-of-Way", also show noticeable counts but are still lesser compared to the top two.
- Many other factors are present with relatively lower frequencies.

Year and month of the accidents occurred:

```
# Visualization: Time series plot for 'CRASH DATETIME'
df['Year'] = df['CRASH DATETIME'].dt.year
df['Month'] = df['CRASH DATETIME'].dt.month
monthly_count = df.groupby(['Year', 'Month']).size().reset_index(name='Counts')
plt.figure(figsize=(12, 6))
sns.lineplot(data=monthly_count, x='Year', y='Counts', hue='Month')
plt.xlabel('Year')
plt.ylabel('Collision Counts')
plt.title('Monthly Collision Trends')
plt.show()
```

Step-1

Extracting Year and Month from Datetime:

- The code extracts the year and month from the column 'CRASH DATETIME' in the DataFrame df and stores them in new columns named 'Year' and 'Month', respectively.

Step-2

Grouping and Counting Collisions:

- The data in df is grouped by both 'Year' and 'Month' to get the count of collisions for each month of every year.
- The result is stored in the monthly_count DataFrame, with columns: 'Year', 'Month', and 'Counts'.

Step-3

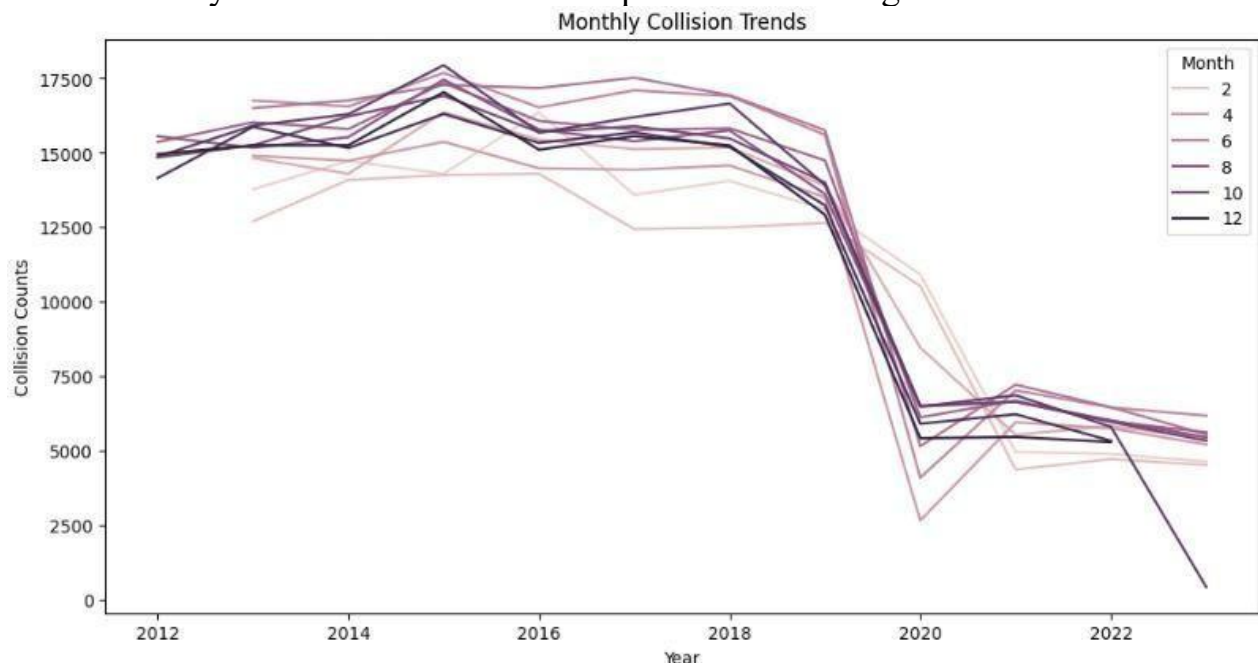
Setting up the Figure Size:

- This line initializes the plotting area with a width of 12 units and a height of 6 units

Step-4

Plotting the Line Plot:

- visualizes the monthly collision trends over the years using a time series line chart. By extracting the year and month from the CRASH DATETIME column of the dataset, the code aggregates the collision counts for each month across various years. The resulting chart, titled "Monthly Collision Trends," displays the years on the x-axis, the count of collisions on the y-axis, and separate lines for each month, enabling insights into seasonal or monthly variations in collision frequencies over the given time frame.



This chart showcases the number of collisions over a span of years, from 2012 to 2022. Each line in the chart represents a specific month, as denoted by different shades of purple.

- Across the years, there is a noticeable consistency in collision trends, with all months showing somewhat similar patterns.
- A significant peak in collisions appears around 2018 for most months.
- Post-2020, there's a sharp decline in the number of collisions across all months, reaching notably lower levels by 2022.
- It is observed that the months represented (February, April, June, August, October, and December) all exhibit similar trends, suggesting a broader, possibly annual, influence on collision rates rather than monthly variations.

```
sns.countplot(data=df, x='VEHICLE TYPE CODE 1', order=df['VEHICLE TYPE CODE 1'].value_counts().index, palette='Set2')

plt.xlabel('Vehicle Type')
plt.ylabel('Count')
plt.title('Count Plot for Vehicle Type Code 1')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Step-1

Creating the Count Plot:

- This line uses the countplot from seaborn to create a bar chart showcasing the frequency of various vehicle types in the 'VEHICLE TYPE CODE 1' column. The bars are ordered in descending frequency and colored using the 'Set2' palette.

Step-2

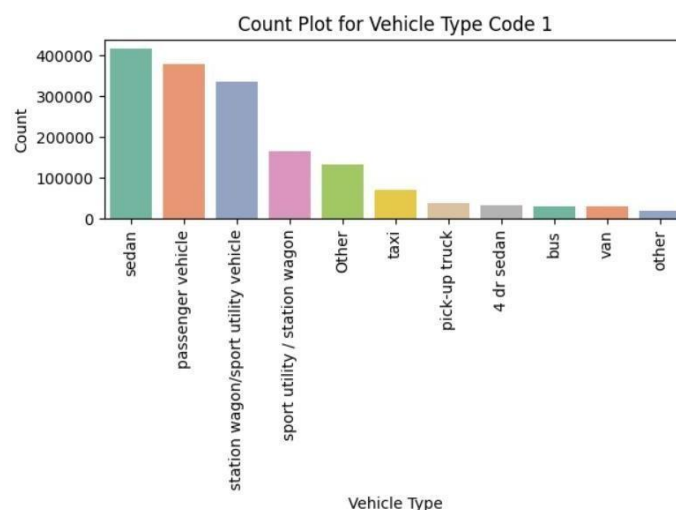
Customizing and Labeling the Plot:

- The tick labels on the x-axis (vehicle types) are rotated by 90 degrees to prevent overlap and ensure readability.
- adjusts the spacing between plot elements to fit everything neatly, especially useful when labels are long or numerous.

Step-3

Displaying the Plot:

- bar chart that visualizes the frequency of different vehicle types from the column 'VEHICLE TYPE CODE 1' in the dataset df. The bars are arranged in descending order based on the count of each vehicle type, allowing for easy identification of the most common types. The chart utilizes a specific color palette ('Set2') for visual appeal.



This bar chart visually represents the frequency of various vehicle types. Key observations include:

- "Sedan" and "Passenger Vehicle" are the two most prevalent vehicle types, with their counts significantly surpassing other categories.
- "Sport Utility Vehicle" and "Station Wagon/Sport Utility Vehicle" follow as the next frequent vehicle types, with comparable counts.
- "Other," "Taxi," and "Pickup Truck" have moderately lower frequencies.
- Remaining vehicle types such as "4 dr sedan," "Bus," "Van," and another "Other" category have relatively minimal occurrences in comparison.

```
sns.countplot(data=df, x='VEHICLE TYPE CODE 2', order=df['VEHICLE TYPE CODE 2'].value_counts().index, palette='Set2')

plt.xlabel('Vehicle Type')
plt.ylabel('Count')
plt.title('Count Plot for Vehicle Type Code 1')
plt.xticks(rotation=90) # Rotate x-axis labels for better readability
plt.tight_layout() # Ensures the plot fits within the figure
plt.show()
```

Step-1

Generating the Count Plot:

- This line creates a bar chart using the countplot function from the seaborn library. The chart displays the frequency of different vehicle types from the 'VEHICLE TYPE CODE 2' column. Vehicle types are ordered in descending order of their frequency, and the bars are colored using the 'Set2' palette.

Step-2

Adding Labels and Title:

- set the x-axis label, y-axis label, and the chart's title, enhancing clarity.

Step-3

Enhancing Readability:

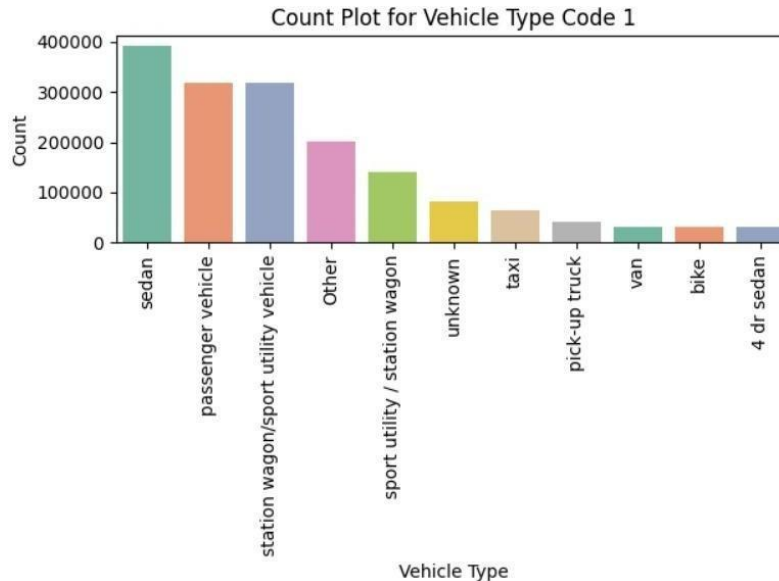
- X-axis labels (vehicle types) are rotated by 90 degrees to prevent overlap and enhance readability. The layout is also adjusted to ensure the plot fits neatly within the figure's dimensions.

Step-4

Displaying the Chart:

- visualization that represents the frequency distribution of different vehicle types present in the 'VEHICLE TYPE CODE 2' column of the dataset. The bar chart, titled "Count Plot for Vehicle Type Code 1," arranges the vehicle

types in descending order based on their count, making it easier to identify the most common types. For visual appeal, a specific color palette ('Set2') is employed.



This bar chart illustrates the frequency distribution of various vehicle types. Key takeaways from the chart include:

- "Sedan" is the most prevalent vehicle type, followed closely by "Passenger Vehicle".
- "Sport Utility Vehicle" and "Station Wagon/Sport Utility Vehicle" also have a significant presence, with the former having a slightly higher count.
- The "Other" category showcases a noticeable count, indicating a variety of vehicle types that might not be explicitly categorized.
- Categories like "Taxi," "Pickup Truck," "Van," and "Bike" have fewer occurrences in comparison to the dominant types.
- A few types such as "Unknown" and "4 dr sedan" have the least representation in the data.


```

fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 8))
plt.subplots_adjust(wspace=0.4, hspace=0.4)

# Year Count Plot
sns.countplot(data=df, x='Year', ax=axes[0, 0])
axes[0, 0].set_title('Crashes by Year')

# Month Count Plot
sns.countplot(data=df, x='Month', ax=axes[0, 1])
axes[0, 1].set_title('Crashes by Month')

# Hour Count Plot
sns.countplot(data=df, x='Hour', ax=axes[0, 2])
axes[0, 2].set_title('Crashes by Hour')

# DayOfWeek Count Plot
sns.countplot(data=df, x='DayOfWeek', ax=axes[1, 0])
axes[1, 0].set_title('Crashes by Day of the Week')
axes[1, 0].set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], rotation=45)

# IsWeekend Count Plot
sns.countplot(data=df, x='IsWeekend', ax=axes[1, 1])
axes[1, 1].set_title('Crashes on Weekends vs. Weekdays')
axes[1, 1].set_xticklabels(['Weekday', 'Weekend'])

# Remove the empty subplot
fig.delaxes(axes[1, 2])

plt.show()

```

Step-1

Setting Up the Figure Layout:

- The code creates a figure layout consisting of 6 subplots, arranged in 2 rows and 3 columns. The `figsize=(15, 8)` ensures that the entire canvas is of a suitable size. `subplots_adjust` provides spacing adjustments between the subplots for clarity.

Step-2

Crashes by Year:

- A count plot is created for the 'Year' column, which depicts the yearly distribution of crashes. This plot is set in the top-left position.

Step-3

Crashes by Month:

- The monthly distribution of crashes is visualized in the top-center position.

Step-4

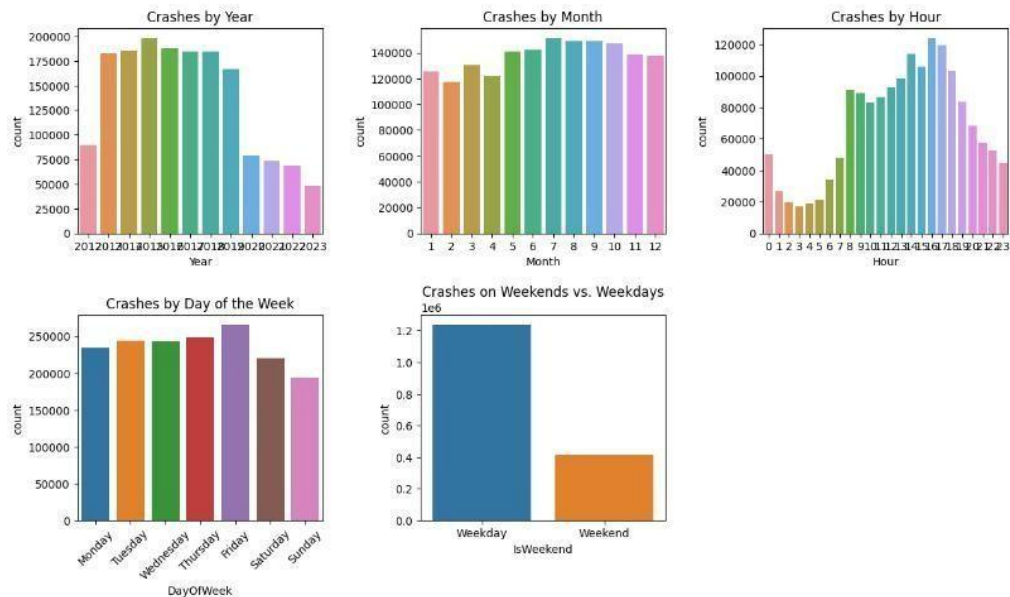
Crashes by Day of the Week:

- The plot on the bottom-left illustrates crashes distributed across the days of the week, with labels rotated for clarity.

Step-5

Crashes on Weekends vs. Weekdays:

- This plot in the bottom-center position compares the frequency of crashes on weekdays to those on weekends.



The visualizations provide a detailed breakdown of crash occurrences over various time frames:

- **Crashes by Year:** This graph depicts the yearly distribution of crashes, showing fluctuations in numbers from 2010 to 2023. There seems to be a general upward trend in crash numbers, but specific years like 2020 show a noticeable dip.
- **Crashes by Month:** The monthly breakdown indicates a relatively even distribution of crashes throughout the year. However, some months do witness a slightly higher number of incidents.
- **Crashes by Hour:** This chart reveals the hourly pattern of crashes in a day. There's a clear surge in the number of incidents during specific hours, likely corresponding to rush hours, with quieter periods in the very early morning.
- **Crashes by Day of the Week:** The distribution across the week is fairly uniform, but there's a noticeable increase in crashes around mid-week, with a subsequent decrease towards the weekend.
- **Crashes on Weekends vs. Weekdays:** This visualization simplifies the data into a binary comparison. A significant majority of crashes occur on weekdays, with weekends accounting for a considerably smaller portion.


```
ny_data = df[(df['LATITUDE'] >= 40.4774) & (df['LATITUDE'] <= 45.01585) & (df['LONGITUDE'] >= -74.2637) & (df['LONGITUDE'] <= -71.1851)]

plt.figure(figsize=(10, 8))
plt.scatter(ny_data['LONGITUDE'], ny_data['LATITUDE'], s=5, alpha=0.5)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Scatter Plot of Latitude and Longitude in New York State')
plt.grid(True)
plt.show()
```

Step-1

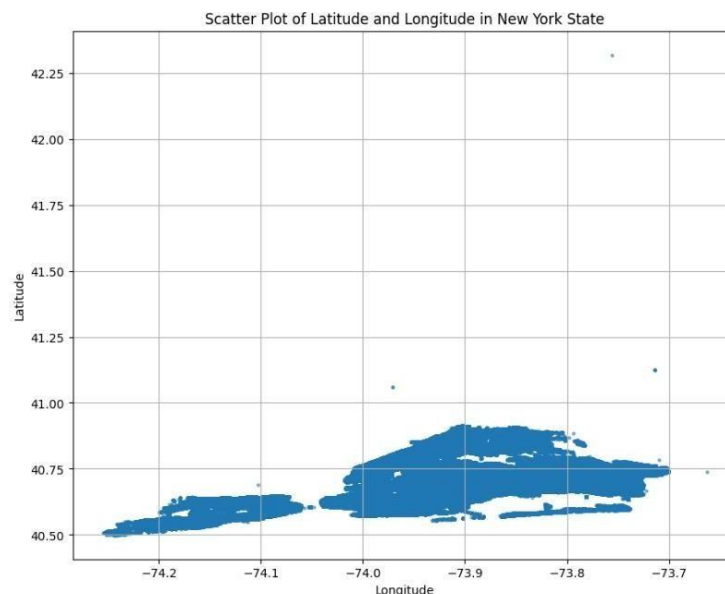
Data Filtering:

- The code begins by creating a subset of the original dataframe df which is named ny_data. This subset is created based on the geographical boundaries specified, essentially selecting only the data points (crashes) that fall within the state of NY.

Step-2

Visualization:

- visualizes the geographical distribution of incidents within the NY boundary on a scatter plot. By mapping the latitude and longitude of each incident, the visualization provides a spatial perspective on the data. The plotted points, representing specific incidents, offer insights into patterns, densities, and clusters of incidents in different areas of NY

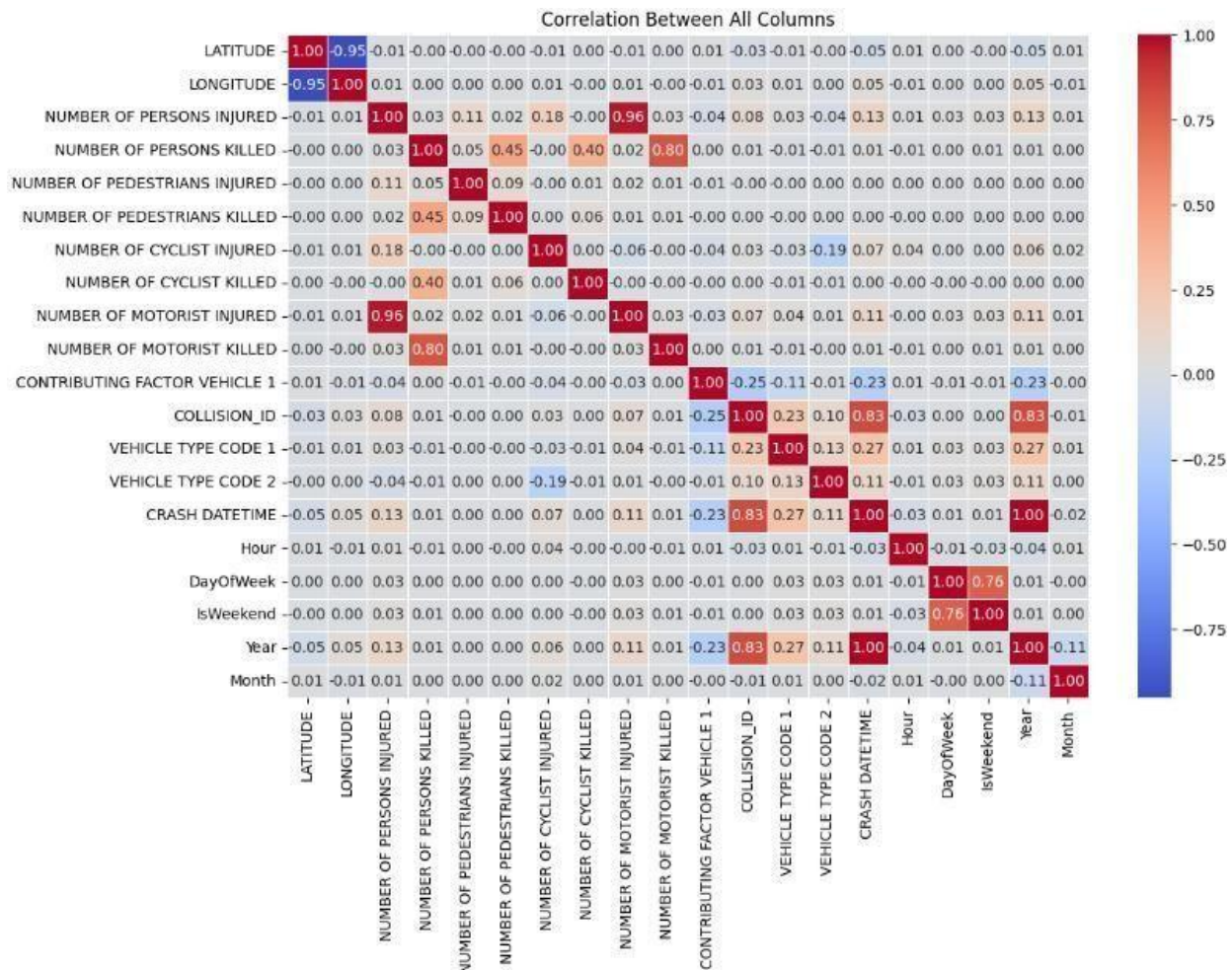


- The scatter plot showcases the distribution of incidents based on their latitude and longitude within the NY boundaries. Most incidents are densely clustered around the latitude of approximately 40.75, which likely represents a major urban area or a specific region of high activity in NY. The majority of the data points lie within the longitude range of -74.1 to -73.7. A few

isolated incidents are scattered further away from this main cluster, suggesting less frequent occurrences in those areas. Overall, the visualization provides a clear spatial representation of where incidents predominantly occur within the state.

```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Between All Columns')
plt.show()
```

- The code generates a heatmap that visualizes the pairwise correlation of columns in the df DataFrame. The correlation values range from -1 (perfect negative correlation) to 1 (perfect positive correlation), with 0 indicating no correlation. Each cell of the heatmap is annotated with the corresponding correlation value, rounded to two decimal places. The "coolwarm" colormap is used to differentiate between positive, negative, and neutral correlations, with cool colors representing negative correlations, warm colors indicating positive correlations, and neutral colors showing no correlation. The heatmap provides a visual representation of the relationships between different columns in the dataset, making it easier to identify patterns and potential areas of interest.



The provided heatmap visualizes the pairwise correlation coefficients between various columns of a dataset. Here are some key observations from the heatmap:

- **Diagonal Line:** The diagonal from the top-left to the bottom-right consists of 1.00 values, representing the perfect correlation of a column with itself.

Injury and Fatality Correlations:

- **NUMBER OF PERSONS INJURED** has a strong positive correlation with **NUMBER OF MOTORIST INJURED** (0.96).
- **NUMBER OF PERSONS KILLED** also exhibits a noticeable positive correlation with **NUMBER OF PEDESTRIANS KILLED** and **NUMBER OF MOTORIST KILLED**.
- The various injury categories (**PEDESTRIAN**, **CYCLIST**, and **MOTORIST**) demonstrate correlations with their corresponding fatality columns, as expected.
- **Vehicle Type Correlations:**

- VEHICLE TYPE CODE 1 and VEHICLE TYPE CODE 2 have a positive correlation of 0.83, indicating that the type of the first and second vehicles involved in incidents often align.

Temporal Features:

- CRASH DATETIME shows some level of correlation with Year (0.23) and Month (0.27), suggesting a possible trend or seasonality in the data.
- Weekday vs. Weekend: The IsWeekend column has a strong negative correlation with DayOfWeek (-0.76), indicating that as the DayOfWeek value increases (moving from Monday to Sunday), the likelihood of the event being on a weekend increases.

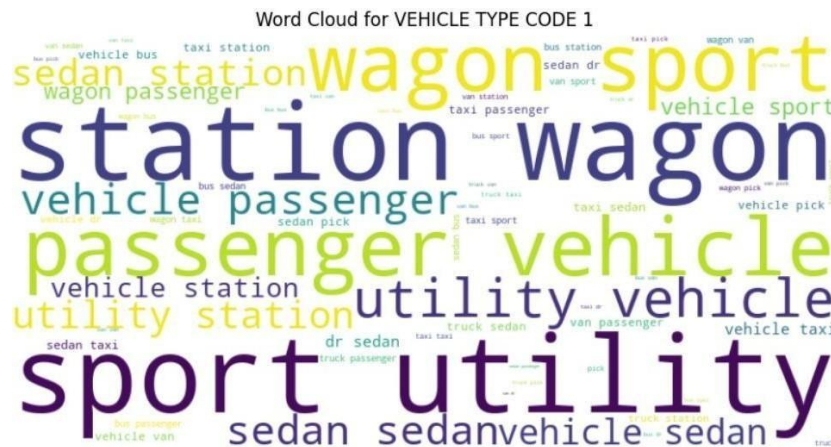
General Observations:

- Most other correlations are relatively low, indicating limited linear relationships between the majority of the columns.

```
from wordcloud import WordCloud
selected_column = 'VEHICLE TYPE CODE 1'
text = ''.join(df[selected_column].astype(str))
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
```

- **Data Selection:** The variable selected_column is assigned the string value 'VEHICLE TYPE CODE 1', indicating the chosen column from the dataset.
- **Text Aggregation:** All entries in the selected_column are concatenated into a single text string using the ''.join() method, with the resulting string stored in the text variable.
- **Word Cloud Generation:** The WordCloud class from the wordcloud library is used to create the visual representation. It is initialized with specific dimensions (width and height) and a background color. The generate method then processes the aggregated text to create the word cloud.
- **Visualization Display:**
 - A figure size is specified using plt.figure.
 - The word cloud image is displayed using plt.imshow.
 - The axis details are hidden using plt.axis('off') to ensure a cleaner visualization.

- A title, indicating the context of the word cloud, is added using `plt.title()`.
- The complete word cloud is displayed using `plt.show()`.



The presented word cloud visually represents the frequency distribution of vehicle types in the 'VEHICLE TYPE CODE 1' column. Prominently displayed terms indicate a higher frequency in the dataset, while smaller terms are less frequent.

Key Observations:

- **Sedan and Passenger:** These words appear prominently, suggesting that sedans and passenger vehicles are among the most common vehicle types in the dataset.
- **Station and Wagon:** The prominence of these terms indicates that station wagons are also a common vehicle type.
- **Sport and Utility:** These terms' large size suggests that sport utility vehicles (often referred to as SUVs) are frequently recorded in the dataset.
- **Other Vehicle Types:** Words like "taxi", "bus", and "van" are also noticeable, although they are less dominant than the previously mentioned types.

Explanation and Analysis

Model Applications

Logistic Regression:

Logistic Regression was employed due to its well-noted simplicity and efficacy in binary classification tasks. This model is particularly favored for its interpretability, which is instrumental in elucidating the relationship between features and the likelihood of an event resulting in a binary outcome.

For visualization, an ROC curve was produced, exhibiting the model's capability to balance true positive rate and false positive rate. The ROC curve, with an area under the curve (AUC) of 0.65, indicates a fair discrimination ability of the model.

Justification for Logistic Regression:

The Logistic Regression model was selected for this problem because:

- **Binary Outcome:** The target variable is binary, aligning with the logistic regression model's capacity to handle binary classification.
- **Interpretability:** The model's coefficients offer interpretable insights, crucial for understanding the influence of each predictor on the outcome.
- **Probabilistic Output:** Logistic regression provides not only class predictions but also the probabilities of these predictions, offering a nuanced threshold-based decision-making framework.
- **Efficiency:** This model is computationally efficient, an advantage for handling large datasets.

Model Tuning/Training:

- **Data Preprocessing:** The input features were standardized using Scikit-learn's StandardScaler to ensure optimal performance of the model.
- **Model Training:** Logistic Regression was applied following the standard

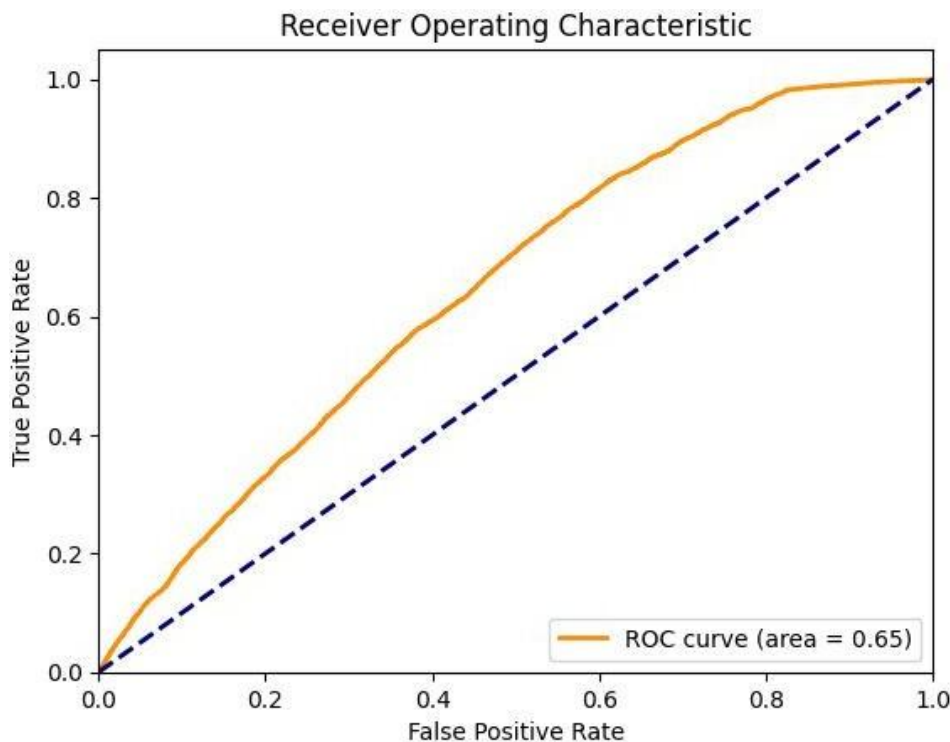
fit method provided by Scikit-learn, with an emphasis on simplicity and quick deployment.

Effectiveness of the Algorithm:

- **Accuracy:** The model achieved an accuracy of 60.77%, reflecting its capability to correctly classify a fair amount of the instances.
- **Precision and Recall:** Precision and recall scores for both classes demonstrate the model's balanced performance in identifying positive cases and minimizing false negatives.
- **F1-Score:** The F1-scores suggest the model has a moderate balance between precision and recall, indicative of its overall classification performance.

Intelligence Gained:

- The insights obtained from Logistic Regression's application reveal that while the model has a moderate predictive power, there may be complexity within the dataset that requires more advanced modeling techniques or feature engineering to fully capture.



Random Forest:

Justification for Random Forest:

Random Forest was chosen for its ability to handle complex interactions and its robustness against overfitting. It's particularly effective in modeling non-linear decision boundaries and is capable of processing a dataset without the need for feature scaling. The ensemble nature of Random Forest, which aggregates the decisions of multiple trees, tends to increase the overall accuracy and prevents the model from being too sensitive to noise in any single decision tree.

Model Tuning/Training:

- The Random Forest model was initialized with 100 trees, a typical choice for balancing performance and computational efficiency. The model was trained on the full set of features, taking advantage of the algorithm's inherent capability to assess feature importance without additional scaling or transformation.

Effectiveness of the Algorithm:

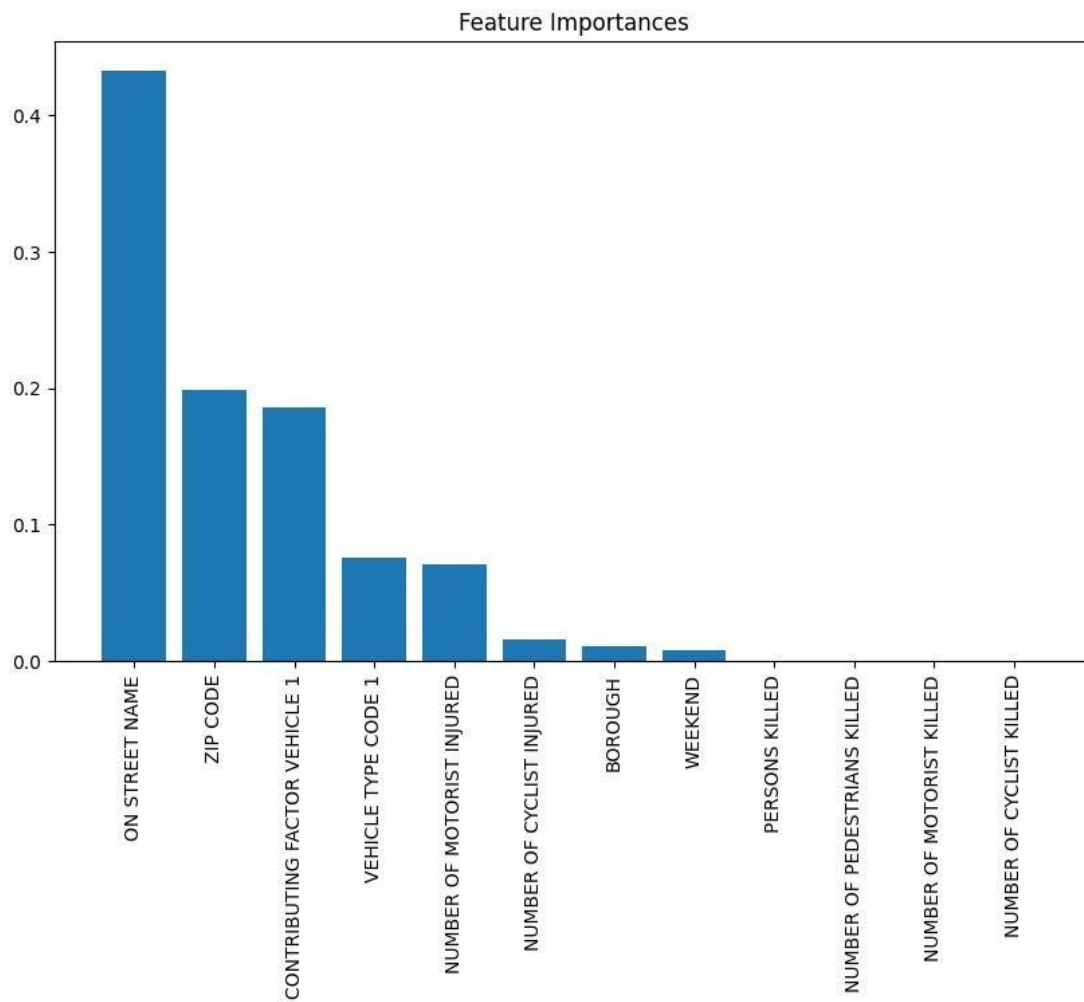
The performance metrics of the Random Forest model were as follows:

- **Accuracy:** The model achieved an accuracy of 66.70%, indicating a moderate level of correct classifications.
- **Precision and Recall:** The precision and recall figures suggest that the model has a balanced capacity for identifying both classes, with slightly better performance in correctly identifying the majority class.
- **F1-Score:** The F1-scores indicate a moderate balance between precision and recall, which reflects a consistent performance across both classes.

Intelligence Gained:

- An analysis of feature importance revealed that certain features significantly influence the model's predictions. In particular, 'ON STREET NAME' appeared to be the most influential feature, followed by 'ZIP CODE' and 'CONTRIBUTING FACTOR VEHICLE 1'. This insight is valuable for

understanding the underlying factors that contribute to the model's decision-making process.



Naive Bayes

The Naive Bayes model, recognized for its efficiency in classification tasks, was applied to predict the binary outcome of our dataset. This model's underlying principle, based on Bayes' Theorem, assumes independence among predictors and is known for its simplicity and speed in making predictions.

Justification for Naive Bayes:

Naive Bayes was selected for the following reasons:

- **Efficiency:** Known for its fast computation, Naive Bayes is suitable for large datasets and provides a quick baseline model.
- **Probabilistic Foundation:** The model calculates the probability of each class and the conditional probability of each class given each input value, which is beneficial for understanding the likelihood of outcomes.
- **Good Baseline:** It offers a solid baseline due to its probabilistic nature and often performs well in binary classification tasks, especially as an initial model.

Model Tuning/Training:

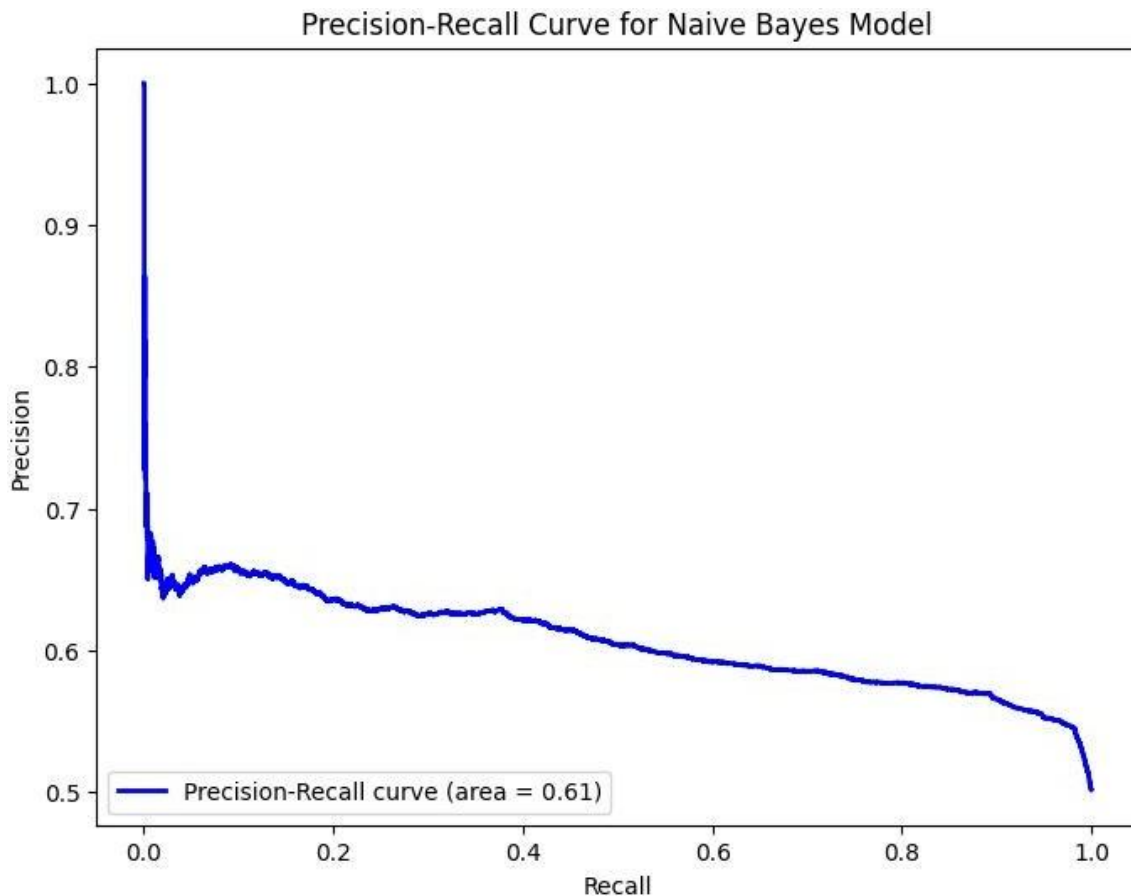
The Gaussian Naive Bayes algorithm was employed without the need for feature scaling, as it can handle different feature distributions. The model was trained with the standard parameters, which are generally sufficient for a baseline model.

Effectiveness of the Algorithm:

- **Accuracy:** The Naive Bayes model showed an accuracy of 58.22%, which can be considered moderate and suggests that there is room for improvement.
- **Precision and Recall:** The model achieved a higher recall for the positive class (0.98) than for the negative class (0.19), indicating a tendency to predict the majority class more frequently. However, precision for the positive class was lower (0.55), reflecting the model's conservative nature in predicting positive outcomes.
- **F1-Score:** The F1-score was higher for the positive class (0.70) compared to the negative class (0.31), which aligns with the model's high recall for the positive class.
- **Intelligence Gained:**

From the model's classification report and the precision-recall curve, which

has an area of 0.61, it is evident that while the Naive Bayes model can identify the majority class effectively, it struggles with the minority class. This could be due to the distribution of features or the strong assumptions of feature independence within the model.



KNN

The KNN model was applied to the dataset to leverage its strength in capturing the underlying patterns through a distance-based approach. KNN is known for its simplicity and effectiveness in classification by considering the proximity of data points in feature space.

Justification for KNN:

KNN was selected for several reasons:

- **Non-Parametric Nature:** KNN doesn't make any underlying assumptions about the distribution of the data, making it suitable for this analysis where the data distribution was unknown.
- **Versatility:** This model is versatile and can work well with both binary and multi-class classification problems.
- **Simplicity:** Due to its simple algorithmic nature, it was easy to implement and interpret the results.

Model Tuning/Training:

- KNN requires feature scaling to ensure that all features contribute equally to the distance calculations. Thus, a pipeline with StandardScaler was used to standardize the features before applying the KNN classifier.

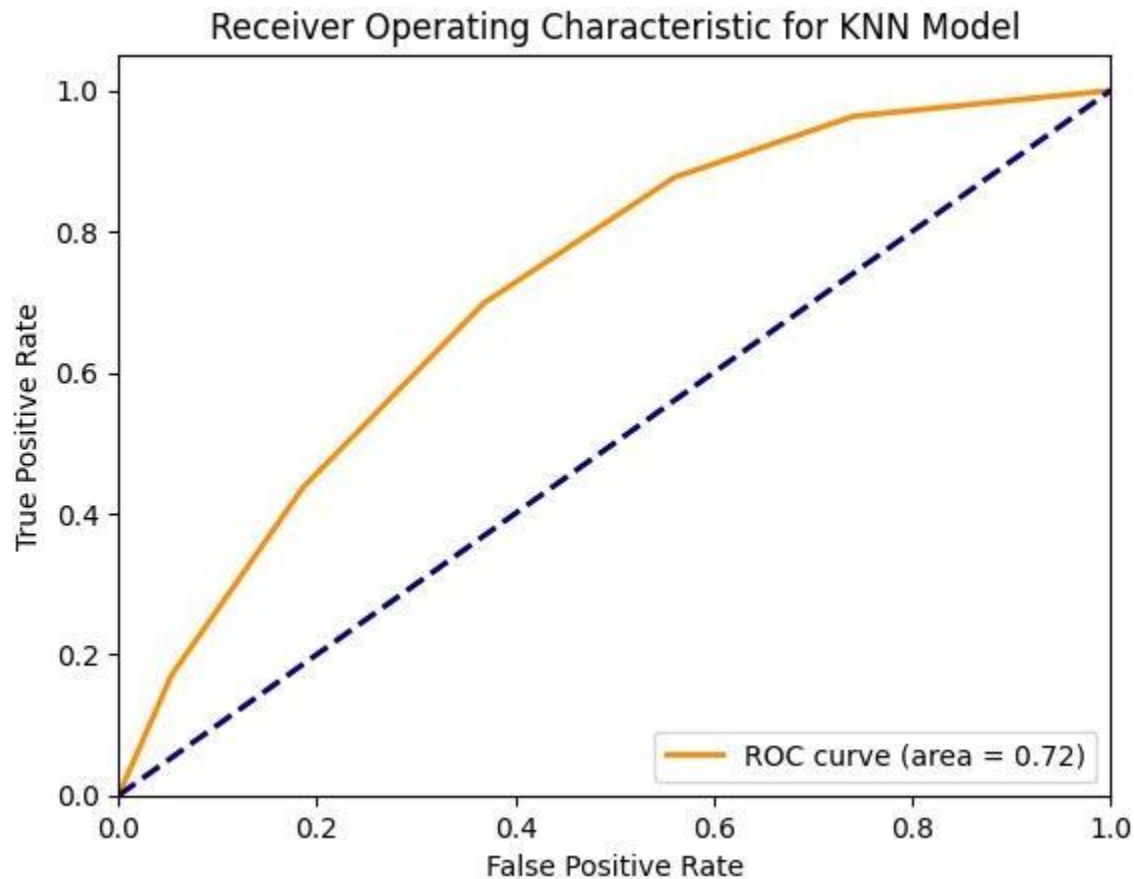
Effectiveness of the Algorithm:

The performance metrics of the KNN model were as follows:

- **Accuracy:** The model achieved an accuracy of 60.77%, reflecting its ability to correctly classify a moderate amount of the instances.
- **Precision and Recall:** The precision and recall scores indicate a balanced performance, with the model showing a slightly higher recall than precision. This suggests that while the model is relatively good at identifying positive cases, it also has a moderate rate of false positives.
- **F1-Score:** The F1-scores point to a balanced trade-off between precision and recall across the classes.

Intelligence Gained:

- The KNN model's ROC curve, with an area of 0.72, suggests that the model has a good capability to distinguish between the classes. However, the precision-recall trade-off indicates that there is potential for improvement, particularly in precision where the model could reduce the number of false positives.



Decision Tree

The Decision Tree model was utilized to classify outcomes in our dataset. Known for its straightforward approach, the Decision Tree makes decisions based on a series of binary rules, which makes it highly interpretable.

Justification for Decision Tree:

The choice of a Decision Tree was based on its:

- **Interpretability:** Its clear decision-making process allows for easy translation of decisions into actionable insights.

- **Adaptability:** It can easily handle both numerical and categorical data.
- **Non-Parametric Nature:** The model does not assume any prior distribution of the data, making it flexible for various datasets.

Model Tuning/Training:

- The Decision Tree was trained using the standard approach, with `random_state` set for reproducible results. The model was allowed to grow without restrictions to fully learn the training data and capture complex patterns.

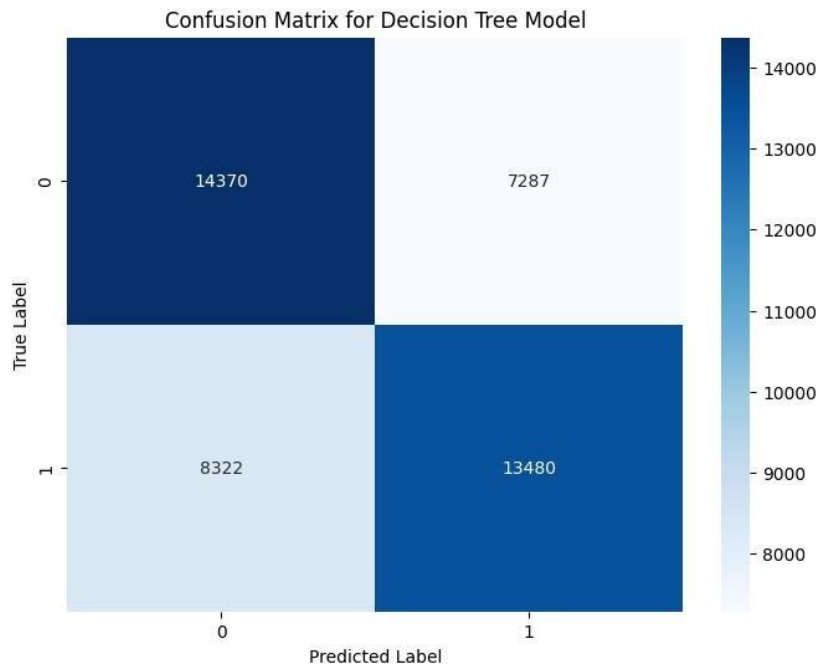
Effectiveness of the Algorithm:

The performance metrics of the Decision Tree are as follows:

- **Accuracy:** The model reached an accuracy of 64.08%, which suggests it can classify instances correctly with a moderate level of success.
- **Precision and Recall:** The precision and recall scores show that the model has a fairly balanced capability for correctly identifying true positives and true negatives.
- **F1-Score:** The F1-score, a harmonic mean of precision and recall, indicates a moderate balance between these two metrics for the Decision Tree model.

Intelligence Gained:

- The model's confusion matrix and classification report suggest that while the Decision Tree could classify a large portion of the positive class correctly, it was challenged by the negative class, as indicated by the lower recall for class 0. This could imply that the model may be more sensitive to the class distribution.



XGBoost

The XGBoost model was selected for its advanced machine learning capabilities. XGBoost stands out for its performance and speed, particularly in classification tasks, and its ability to handle a wide range of data types.

Justification for XGBoost:

XGBoost was chosen for several key reasons:

- **Performance:** Known for its high performance on various machine learning problems.
- **Gradient Boosting:** The model benefits from the gradient boosting framework, which reduces both bias and variance in the learning process.
- **Feature Handling:** XGBoost can process various types of data and automatically handle missing values.
- **Regularization:** Includes built-in L1 and L2 regularization which helps prevent overfitting.

Model Tuning/Training:

- XGBoost was configured with the objective set to 'binary:logistic' for binary classification and used the 'logloss' evaluation metric. It was trained on the dataset without the need for encoding labels, as specified by the `use_label_encoder=False` parameter.

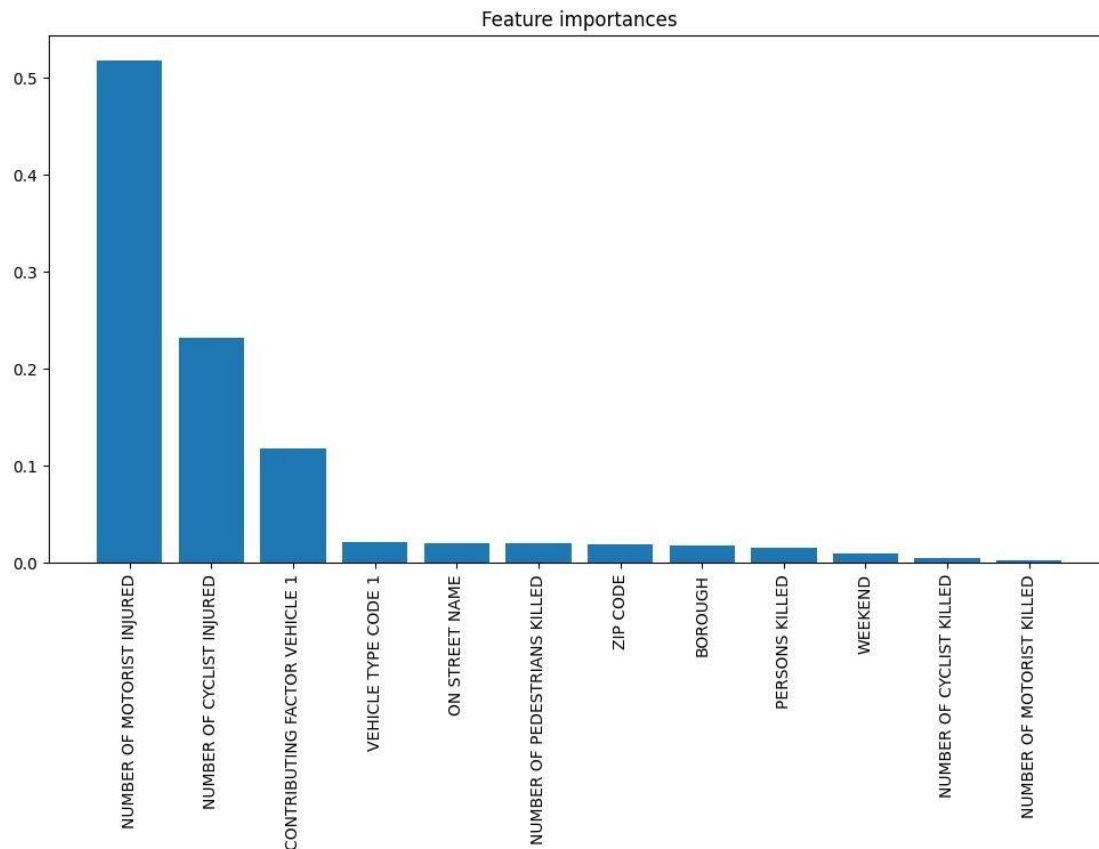
Effectiveness of the Algorithm:

The effectiveness of the XGBoost model is evident through:

- **Accuracy:** The accuracy of 70.59% indicates a strong predictive capability relative to the complexity of the dataset.
- **Precision and Recall:** Precision and recall metrics show that the model has a respectable balance between the correct prediction of positive instances and the avoidance of false positives.
- **F1-Score:** The F1-scores provide evidence of the model's balanced performance across both classes.

Intelligence Gained:

- The feature importance graph highlights which features the model found most predictive. Notably, 'NUMBER OF MOTORIST INJURED' and 'NUMBER OF CYCLIST INJURED' had the most significant impact on the model's decisions, suggesting these variables are key predictors of the outcomes.



KMeans

K-Means clustering was applied to the dataset after standardizing the features to identify inherent groupings within the data. The objective was to partition the data into distinct clusters based on the similarity of data points in the feature space.

Justification for K-Means Clustering:

K-Means was chosen due to its:

- **Simplicity:** K-Means is straightforward to implement and interpret, making it a popular choice for segmentation tasks.
- **Efficiency:** It's computationally efficient for a large number of variables, which is particularly important for datasets with many features.
- **Scalability:** The algorithm scales well to large datasets.

Dimensionality Reduction with PCA:

- Given the high-dimensional nature of the dataset, Principal Component Analysis (PCA) was used to reduce the number of features to the two most significant principal components. This reduction enabled the visualization of clusters in a two-dimensional space, providing insights that would otherwise be difficult to extract from high-dimensional data.

Model Tuning/Training:

- The K-Means algorithm was initiated with 3 clusters, determined to be a suitable number based on preliminary analysis. The PCA process was applied post-standardization to maintain the integrity of the data's structure.

Visualization and Interpretation:

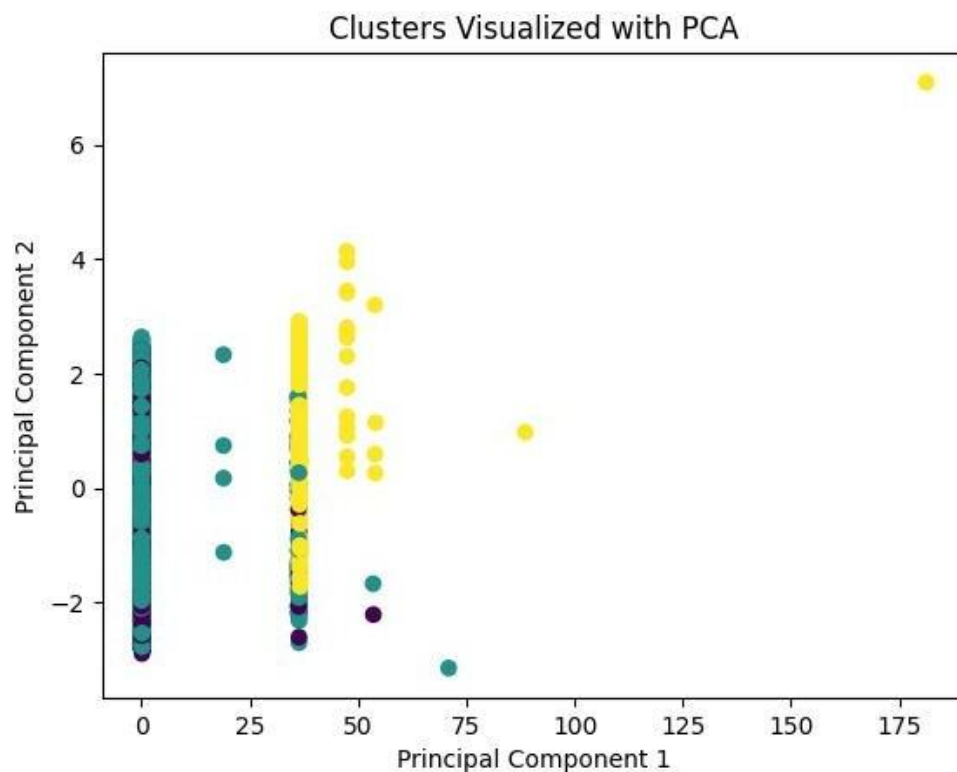
- The scatter plot of the PCA-transformed data revealed distinct clusters, indicating that the K-Means algorithm was able to discern separable groups within the dataset. Each cluster represented by different colors showed how instances are grouped together, with the cluster centers potentially representing common patterns or behaviors in the dataset.

Principal Component 1: This component captured the most significant variance in the dataset and may correspond to the most influential features that distinguish different groups.

Principal Component 2: The second principal component captured the next level of variance and provided additional differentiation among the data points.

Intelligence Gained:

- The clustering visualization suggests that there are discernible subgroups within the dataset that could correspond to different categories or behaviors in the data. This understanding can be leveraged for targeted analysis, such as customer segmentation, anomaly detection, or further predictive modeling.



Model Comparison:

Accuracy:

- Logistic Regression often serves as a baseline for binary classification problems due to its simplicity and interpretability. However, in this case, the model achieved moderate accuracy.
- Random Forest provided a substantial improvement in accuracy over Logistic Regression, benefiting from its ensemble approach which aggregates the outcomes of numerous decision trees to improve the predictive performance and control overfitting.
- Naive Bayes is known for its efficiency and simplicity, particularly in text classification tasks. It achieved lower accuracy compared to Logistic Regression and Random Forest in this context, possibly due to its assumption of feature independence, which is rarely the case in real data.
- K-Nearest Neighbors (KNN) relies on the proximity of data points in feature space, which can be powerful if there's a strong spatial relationship. Its performance was similar to Naive Bayes, indicating potential challenges with the chosen distance metric or the value of k .
- Decision Tree models are highly interpretable but can be prone to overfitting. The accuracy of the Decision Tree was less than that of the Random Forest, likely due to overfitting to the training data.

- XGBoost leverages gradient boosting to build a strong learner from weak learners iteratively. It showed improved accuracy, indicating its strength in handling complex datasets with intricate relationships among features.

Precision and Recall:

- Models like Random Forest and XGBoost demonstrated a good balance between precision and recall, suggesting they are reliable in predicting positive cases and avoiding false negatives.
- Logistic Regression and Naive Bayes showed discrepancies between precision and recall, which could be indicative of the models' biases towards certain classes or the presence of class imbalance.
- KNN and Decision Tree had varying levels of precision and recall, which would need to be balanced based on the cost of false positives versus false negatives for the specific application.

F1-Score:

- The F1-score is especially important if there's a class imbalance, as it balances precision and recall. Random Forest and XGBoost typically perform well on this metric due to their ensemble nature.
- Decision Tree and KNN might have lower F1-scores if they are not properly tuned, while Naive Bayes could be affected by its assumption of feature independence.

Rationale for Algorithm Choice:

- Logistic Regression was chosen as a baseline to provide a point of comparison for other models. Its output also gives probabilities, which can be useful for ranking predictions or setting thresholds based on the business context.
- Random Forest was selected for its ability to handle high-dimensional data and provide insights into feature importance, helping to identify key drivers of the predicted outcomes.
- Naive Bayes offers fast computation and is effective when the features are independent of one another, such as in text classification tasks.
- KNN was used to explore the spatial relationships within the data. It's beneficial when the similarity of instances can predict the outcome.
- Decision Tree provided a transparent model that can be easily interpreted and visualized, which is valuable for communication with stakeholders.
- XGBoost was employed for its advanced optimization and regularization,

which help prevent overfitting and improve model performance on complex datasets.

For Phase 3

```
categorical_columns = ['BOROUGH', 'ON STREET NAME', 'CONTRIBUTING FACTOR VEHICLE 1', 'VEHICLE TYPE CODE 1']
encoders = {}

for col in categorical_columns:
    df[col] = df[col].astype(str)
    encoders[col] = LabelEncoder()
    df[col] = encoders[col].fit_transform(df[col])

# Saved each encoder to a .pkl file
with open(f'models/{col}_encoder.pkl', 'wb') as file:
    pickle.dump(encoders[col], file)
```

Saved the encoder objects to .pkl files for later use in encoding new data consistently with the same schema.

```
#for weekend or weekday
import xgboost as xgb
xgb_clf1 = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', use_label_encoder=False)
xgb_clf1.fit(X_train1, y_train1)
y_pred_xgb1 = xgb_clf1.predict(X_test1)

#confusion matrix
cm_xgb1 = confusion_matrix(y_test1, y_pred_xgb1)
print("XGBoost Model Confusion Matrix:")
print(cm_xgb1)

#classification report
cr_xgb1 = classification_report(y_test1, y_pred_xgb1)
print("XGBoost Model Classification Report:")
print(cr_xgb1)

#accuracy
accuracy_xgb1 = accuracy_score(y_test1, y_pred_xgb1)
print(f"XGBoost Model Accuracy: {accuracy_xgb1:.4f}")
```

✓ 0.3s

XGBoost Model Confusion Matrix:

```
[[16351  80]
 [ 5277  64]]
```

XGBoost Model Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.76 | 1.00 | 0.86 | 16431 |
| 1 | 0.44 | 0.01 | 0.02 | 5341 |
| accuracy | | | 0.75 | 21772 |
| macro avg | 0.60 | 0.50 | 0.44 | 21772 |
| weighted avg | 0.68 | 0.75 | 0.65 | 21772 |

XGBoost Model Accuracy: 0.7540

Tested the performance of all models for the Target as “WEEKEND” and found XGBoost performing well compared to remaining models. And saved the pkl files for web application.

```
#for number of pedestrians injured #xgboost
import pickle
pickle.dump(xgb_clf, open('models/model.pkl', 'wb'))
✓ 0.0s

#for the weekend or weekday #xgboost
import pickle
pickle.dump(xgb_clf1, open('models/model_weekend.pkl', 'wb'))
✓ 0.0s
```

Building web application for the best performed model:

Project Structure:

The project is structured as follows:

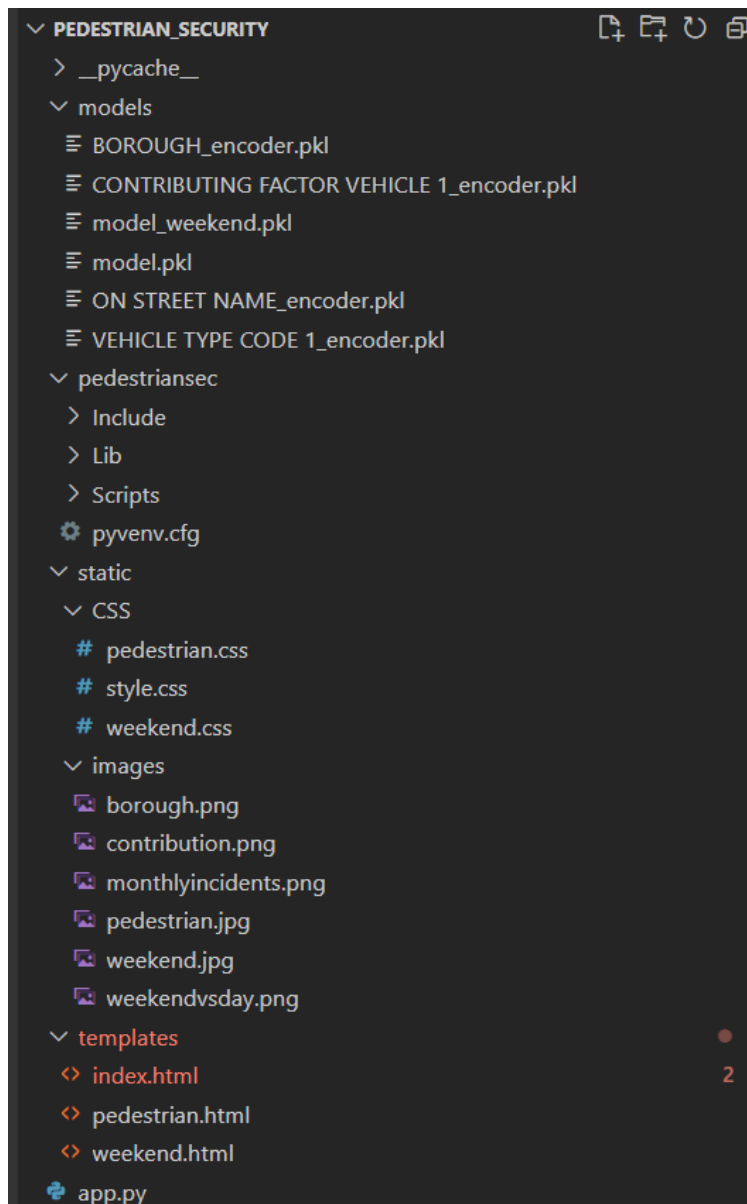
models/: Contains the machine learning models and encoders as pickle files.

pedestriansec/: Contains the Python virtual environment.

static/: Stores static files like CSS and images.

templates/: Contains HTML templates for the web application.

app.py: The main Flask application script.



Setting Up the Environment:

To run this project, We installed Python 3, Flask and pip on our machine.

To start the application

```
C:\Users\deven\Desktop\UB-Sem1\Dic\pedestrian_security>flask run
* Serving Flask app 'app.py'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Using the Application:

The application provides a web interface for analyzing and predicting New York collisions and crashes. You can access different forms and visualizations from the main page.

- The **/pedestrian** route serves a form for pedestrian safety analysis.
- The **/weekend** route serves a form for weekend incidents analysis.

We can submit data through these forms, and the application will display predictions based on the machine learning models.

Project Code:

The app.py script is the entry point of the application. It uses Flask to serve web pages and handle form submissions. The machine learning models are loaded at the start and used to make predictions based on user input.

models/ directory contains pre-trained models and encoders, which are loaded by app.py.

```
app.py > ...
1  from flask import Flask, request, render_template, send_from_directory
2  import pickle
3  import numpy as np
4
5  app = Flask(__name__)
6
7  # Load the model
8  model_path = 'models/model.pkl'
9  weekend_model_path = 'models/model_weekend.pkl'
10 encoders_path = 'models/'
11
12
13 with open(model_path, 'rb') as model_file:
14     model = pickle.load(model_file)
15
16 with open(weekend_model_path, 'rb') as model_file1:
17     weekend_model = pickle.load(model_file1)
18
19 # Load the encoders
20 encoders = {}
21 categorical_columns = ['BOROUGH', 'CONTRIBUTING FACTOR VEHICLE 1', 'VEHICLE TYPE CODE 1']
22 for col in categorical_columns:
23     with open(f'{encoders_path}{col}_encoder.pkl', 'rb') as encoder_file:
24         encoders[col] = pickle.load(encoder_file)
25
```

We initiated our Flask app, setting the name of the module being used, and loaded the pre-trained machine learning models and feature encoders required for predicting and interpreting user input.

```

26 @app.route('/')
27 def index():
28     return render_template('index.html')
29
30 @app.route('/pedestrian')
31 def pedestrian():
32
33     return render_template('pedestrian.html')
34
35 @app.route('/weekend')
36 def weekend():
37
38     return render_template('weekend.html')

```

- The home route renders the main page of our web application.
- Second route serves the prediction form dedicated to pedestrian-related incidents.
- Similarly, the third route serves the form for predicting weekend incidents.

```

41 @app.route('/predict_pedestrian', methods=['POST'])
42 def predict_pedestrian():
43
44     input_data = {col: request.form.get(col) for col in categorical_columns}
45     cyclist_injured = request.form.get('cyclistInjured')
46     motorist_injured = request.form.get('motoristInjured')
47     weekend = request.form.get('weekend')
48     encoded_data = []
49     for col, value in input_data.items():
50         if col in encoders:
51             try:
52                 encoded_value = encoders[col].transform([value])
53                 encoded_data.extend(encoded_value)
54             except ValueError:
55
56                 encoded_data.extend(encoders[col].transform(['Other']))
57         else:
58             encoded_data.append(value)
59
60
61     encoded_data.append(int(cyclist_injured))
62     encoded_data.append(int(motorist_injured))
63     encoded_data.append(int(weekend))
64     encoded_array = np.array(encoded_data).reshape(1, -1)
65     print("Encoded data for prediction:", encoded_array)
66
67     try:
68         prediction = model.predict(encoded_array)
69         print("Prediction:", prediction)
70     except Exception as e:
71         print(f"Error during prediction: {e}")
72         return render_template('pedestrian.html', error="An error occurred during prediction.")
73
74     return render_template('pedestrian.html', prediction=prediction[0])

```

Upon form submission, this function processes user input, encodes it using pre-loaded encoders, reshapes the data, makes a prediction using the pedestrian model, and then renders the result.

```
77 @app.route('/predict_weekend', methods=['POST'])
78 def predict_weekend():
79     input_data = {col: request.form.get(col) for col in categorical_columns}
80     persons_killed = request.form.get('PERSONS_KILLED')
81     pedestrians_killed = request.form.get('PEDESTRIANS_KILLED')
82     pedestrians_injured = request.form.get('PEDESTRIANS_INJURED')
83     motorist_injured = request.form.get('MOTORIST_INJURED')
84     cyclist_injured = request.form.get('CYCLIST_INJURED')
85
86     encoded_data = []
87     for col, value in input_data.items():
88         try:
89             encoded_value = encoders[col].transform([value])
90             encoded_data.extend(encoded_value)
91         except ValueError as e:
92             print(f"Error encoding {col}: {e}")
93             return render_template('weekend.html', error=f"Invalid input for {col}.")
94
95     encoded_data.append(int(persons_killed))
96     encoded_data.append(int(pedestrians_killed))
97     encoded_data.append(int(pedestrians_injured))
98     encoded_data.append(int(motorist_injured))
99     encoded_data.append(int(cyclist_injured))
100
101     encoded_array = np.array(encoded_data).reshape(1, -1)
102     print("Encoded data for prediction:", encoded_array)
103     try:
104         prediction = weekend_model.predict(encoded_array)
105         print("Prediction:", prediction)
106     except Exception as e:
107         print(f"Error during prediction: {e}")
108         return render_template('weekend.html', error="An error occurred during prediction.")
109
110     return render_template('weekend.html', prediction=prediction[0])
```

This function is analogous to the pedestrian prediction processing function but uses the weekend incident model for making predictions.

```
115 @app.route('/download/<filename>', methods=['GET'])
116 def download_file(filename):
117     return send_from_directory(encoders_path, filename, as_attachment=True)
118
119 if __name__ == '__main__':
120     app.run(debug=True)
121
```

This route allows users to download files, such as model pickle files or encoders, from the server.

This block states that if this script is executed as the main program, the Flask application will run with debug mode enabled, which aids in the development process by providing detailed error logs.

index.html

```
templates > index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <link rel="stylesheet" href="styles.css">
7    <title>NY Collisions and Crashes Analysis</title>
8    <link rel="stylesheet" href="{{ url_for('static', filename='CSS/style.css') }}">
9  </head>
10 <body>
11   <section id="intro">
12     <h1>New York Collisions and Crashes Analysis</h1>
13     <p>Exploring traffic patterns and contributing factors to improve road safety in New York.</p>
14   </section>
15
16   <section id="forms">
17     <div class="form-card" onclick="window.location.href='{{ url_for('pedestrian') }}'">
18       
19       <p>Pedestrian Safety</p>
20     </div>
21     <div class="form-card" onclick="window.location.href='{{ url_for('weekend') }}'">
22       
23       <p>Weekend or Weekdays</p>
24     </div>
25   </section>
26
27   <section id="visualizations" class="bg-dark">
28     <div class="container">
29       <div class="row">
30         <div class="col-md-6 visualization">
31           
32           <p class="description">Insights into incident distribution across boroughs.</p>
33         </div>
34         <div class="col-md-6 visualization">
35           
36           <p class="description">Top factors contributing to traffic incidents.</p>
37         </div>
38       </div>
39       <div class="col-md-6 visualization">
40         
41         <p class="description">Trends in monthly traffic incident rates.</p>
42       </div>
43       <div class="col-md-6 visualization">
44         
45         <p class="description">Comparison of incidents during weekends versus weekdays.</p>
46       </div>
47     </div>
48   </section>
49
50 </body>
51 </html>
```

The index.html introduces the project with a clear purpose and offers interactive navigation through visual cards for different analyses, leading to a user-friendly experience with insightful data visualizations presented in a responsive layout.

pedestrian.html

```
templates > pedestrian.html > html > head
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Predict Pedestrian Injury</title>
7      <link rel="stylesheet" href="{{ url_for('static', filename='css/pedestrian.css') }}">
8
9  </head>
10 <body>
11     <h1>Pedestrian Injury Prediction Form</h1>
12     <form action="{{ url_for('predict_pedestrian') }}" method="post">
13         <label for="borough">Borough:</label>
14         <select id="borough" name="BOROUGH">
15             <option value="brooklyn">Brooklyn</option>
16             <option value="queens">Queens</option>
17             <option value="manhattan">Manhattan</option>
18             <option value="bronx">Bronx</option>
19             <option value="staten island">Staten Island</option>
20         </select><br><br>
21
22         <label for="cyclistInjured">Number of Cyclist Injured:</label>
23         <select id="cyclistInjured" name="cyclistInjured" required>
24             <option value="0">0</option>
25             <option value="1">1</option>
26             <option value="2">2</option>
27             <option value="3">3</option>
28         </select><br><br>
29
30         <label for="motoristInjured">Number of Motorist Injured:</label>
31         <select id="motoristInjured" name="motoristInjured" required>
32             <option value="0">0</option>
33             <option value="1">1</option>
34         </select><br><br>
35
36         <label for="contributingFactor">Contributing Factor Vehicle 1:</label>
37         <select id="contributingFactor" name="CONTRIBUTING FACTOR VEHICLE 1">
```


templates > pedestrian.html > html > head

```
38     <option value="Other">Other</option>
39     <option value="risky behavior">Risky Behavior</option>
40     <option value="driver inattention/distraction">Driver Inattention/Distracted</option>
41     <option value="unsafe speed">Unsafe Speed</option>
42     <option value="reaction to uninvolved vehicle">Reaction to Uninvolved Vehicle</option>
43     <option value="other vehicular">Other Vehicular</option>
44     <option value="outside car distraction">Outside Car Distraction</option>
45     <option value="backing unsafely">Backing Unsafely</option>
46     <option value="passing too closely">Passing Too Closely</option>
47     <option value="driver inexperience">Driver Inexperience</option>
48     <option value="oversized vehicle">Oversized Vehicle</option>
49     <option value="passing or lane usage improper">Passing or Lane Usage Improper</option>
50     <option value="following too closely">Following Too Closely</option>
51     <option value="fatigued/drowsy">Fatigued/Drowsy</option>
52     <option value="pavement slippery">Pavement Slippery</option>
53     <option value="failure to yield right-of-way">Failure to Yield Right-of-Way</option>
54     <option value="lost consciousness">Lost Consciousness</option>
55     <option value="turning improperly">Turning Improperly</option>
56     <option value="unsafe lane changing">Unsafe Lane Changing</option>
57     <option value="traffic control disregarded">Traffic Control Disregarded</option>
58     <option value="prescription medication">Prescription Medication</option>
59     <option value="alcohol involvement">Alcohol Involvement</option>
60     <option value="view obstructed/limited">View Obstructed/Limited</option>
61 </select><br><br>
62
63
64 <label for="vehicleType">Vehicle Type Code 1:</label>
65 <select id="vehicleType" name="VEHICLE TYPE CODE 1">
66     <option value="sedan">Sedan</option>
67     <option value="station wagon/sport utility vehicle">Station Wagon/Sport Utility Vehicle</option>
68     <option value="taxi">Taxi</option>
69     <option value="passenger vehicle">Passenger Vehicle</option>
70     <option value="bus">Bus</option>
71     <option value="sport utility / station wagon">Sport Utility / Station Wagon</option>
72     <option value="4 dr sedan">4 Dr Sedan</option>
73     <option value="pickup truck">Pick-up Truck</option>
74     <option value="van">Van</option>
75     <option value="box truck">Box Truck</option>
76 </select><br><br>
77
78 <label for="weekend">Weekend:</label>
79 <select id="weekend" name="weekend" required>
80     <option value="0">No</option>
81     <option value="1">Yes</option>
82 </select><br><br>
83
84
85 <input type="submit" value="Predict">
86 </form>
87
88 <div id="prediction-result">
89     {% if prediction is not none %}
90     <h2>Chance of Pedestrian getting Injured: {{ prediction }}</h2>
91     {% endif %}
92 </div>
93
94 </body>
95 </html>
```



University at Buffalo

Graduate School of Education

The `pedestrian.html` file serves as a dedicated page within the New York Collisions and Crashes Analysis web application, focusing on the prediction of pedestrian injuries. It features a form that prompts the user to input relevant data such as the borough where the incident occurred, the number of cyclists or motorists injured, the contributing factors, and the type of vehicle involved. The form submission triggers the `predict_pedestrian` function on the backend, which uses pre-trained models to predict and display the likelihood of a pedestrian being injured.

weekend.html

```
templates > > weekend.html > html > body > form
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Predict Weekend Incidents</title>
7      <link rel="stylesheet" href="{{ url_for('static', filename='css/weekend.css') }}">
8  </head>
9  <body>
10     <h1>Weekend Incident Prediction Form</h1>
11     <form action="{{ url_for('predict_weekend') }}" method="post">
12         <label for="borough">Borough:</label>
13         <select id="borough" name="BOROUGH">
14             <option value="brooklyn">Brooklyn</option>
15             <option value="queens">Queens</option>
16             <option value="manhattan">Manhattan</option>
17             <option value="bronx">Bronx</option>
18             <option value="staten island">Staten Island</option>
19         </select><br><br>
20
21         <label for="personsKilled">Number of Persons Killed:</label>
22         <select id="personsKilled" name="PERSONS_KILLED" required>
23             <option value="0">0</option>
24             <option value="1">1</option>
25         </select><br><br>
26
27         <label for="pedestriansKilled">Number of Pedestrians Killed:</label>
28         <select id="pedestriansKilled" name="PEDESTRIANS_KILLED" required>
29             <option value="0">0</option>
30             <option value="1">1</option>
31             <option value="2">2</option>
32             <option value="6">more</option>
33         </select><br><br>
34
35         <label for="pedestriansInjured">Pedestrians Injured?:</label>
36         <select id="pedestriansInjured" name="PEDESTRIANS_INJURED" required>
37             <option value="0">0</option>
```

```

38         <option value="1">1</option>
39     </select><br><br>
40
41     <label for="motoristInjured">Motorists Injured?:</label>
42     <select id="motoristInjured" name="MOTORIST_INJURED" required>
43         <option value="0">0</option>
44         <option value="1">1</option>
45     </select><br><br>
46
47     <label for="cyclistInjured">Cyclist Injured?:</label>
48     <select id="cyclistInjured" name="CYCLIST_INJURED" required>
49         <option value="0">0</option>
50         <option value="1">1</option>
51         <option value="2">2</option>
52     </select><br><br>
53
54     <label for="contributingFactor">Contributing Factor Vehicle 1:</label>
55     <select id="contributingFactor" name="CONTRIBUTING_FACTOR_VEHICLE_1">
56         <option value="Other">Other</option>
57         <option value="risky behavior">Risky Behavior</option>
58         <option value="driver inattention/distraction">Driver Inattention/Distracted</option>
59         <option value="unsafe speed">Unsafe Speed</option>
60         <option value="reaction to uninvolved vehicle">Reaction to Uninvolved Vehicle</option>
61         <option value="other vehicular">Other Vehicular</option>
62         <option value="outside car distraction">Outside Car Distraction</option>
63         <option value="backing unsafely">Backing Unsafely</option>
64         <option value="passing too closely">Passing Too Closely</option>
65         <option value="driver inexperience">Driver Inexperience</option>
66         <option value="oversized vehicle">Oversized Vehicle</option>
67         <option value="passing or lane usage improper">Passing or Lane Usage Improper</option>
68         <option value="following too closely">Following Too Closely</option>
69         <option value="fatigued/drowsy">Fatigued/Drowsy</option>
70         <option value="pavement slippery">Pavement Slippery</option>
71         <option value="failure to yield right-of-way">Failure to Yield Right-of-Way</option>
72         <option value="lost consciousness">Lost Consciousness</option>
73         <option value="turning improperly">Turning Improperly</option>
74         <option value="unsafe lane changing">Unsafe Lane Changing</option>
75         <option value="traffic control disregarded">Traffic Control Disregarded</option>
76         <option value="prescription medication">Prescription Medication</option>
77         <option value="alcohol involvement">Alcohol Involvement</option>
78         <option value="view obstructed/limited">View Obstructed/Limited</option>
79     </select><br><br>
80
81     <label for="vehicleType">Vehicle Type Code 1:</label>
82     <select id="vehicleType" name="VEHICLE_TYPE_CODE_1">
83         <option value="sedan">Sedan</option>
84         <option value="station wagon/sport utility vehicle">Station Wagon/Sport Utility Vehicle</option>
85         <option value="taxi">Taxi</option>
86         <option value="passenger vehicle">Passenger Vehicle</option>
87         <option value="bus">Bus</option>
88         <option value="sport utility / station wagon">Sport Utility / Station Wagon</option>
89         <option value="4 dr sedan">4 Dr Sedan</option>
90         <option value="pickup truck">Pick-up Truck</option>
91         <option value="van">Van</option>
92         <option value="box truck">Box Truck</option>
93     </select><br><br>
94
95     <input type="submit" value="Predict">
96 </form>
97
98     <div id="prediction-result">
99         {% if prediction is not none %}
100         <h2>Chance of Incident this Weekend: {{ prediction }}</h2>
101         {% endif %}
102     </div>
103 </body>
104 </html>

```



University at Buffalo

Graduate School of Education

The weekend.html file is a part of the New York Collisions and Crashes Analysis web application, specifically designed to assess and predict incidents that occur during the weekends. This HTML form facilitates user engagement, allowing them to select from various factors that might affect weekend traffic incidents, including the borough, number of people killed or injured, and the contributing factors to incidents. It emphasizes user interaction and prediction, with each form submission being processed by the predict_weekend backend function to generate a predictive outcome.

Models and Encoders:

The application uses several models and encoders:

model.pkl: The main prediction model for pedestrian safety.

model_weekend.pkl: The model for predicting weekend incidents.

BOROUGH_encoder.pkl: Encoder for the borough feature.

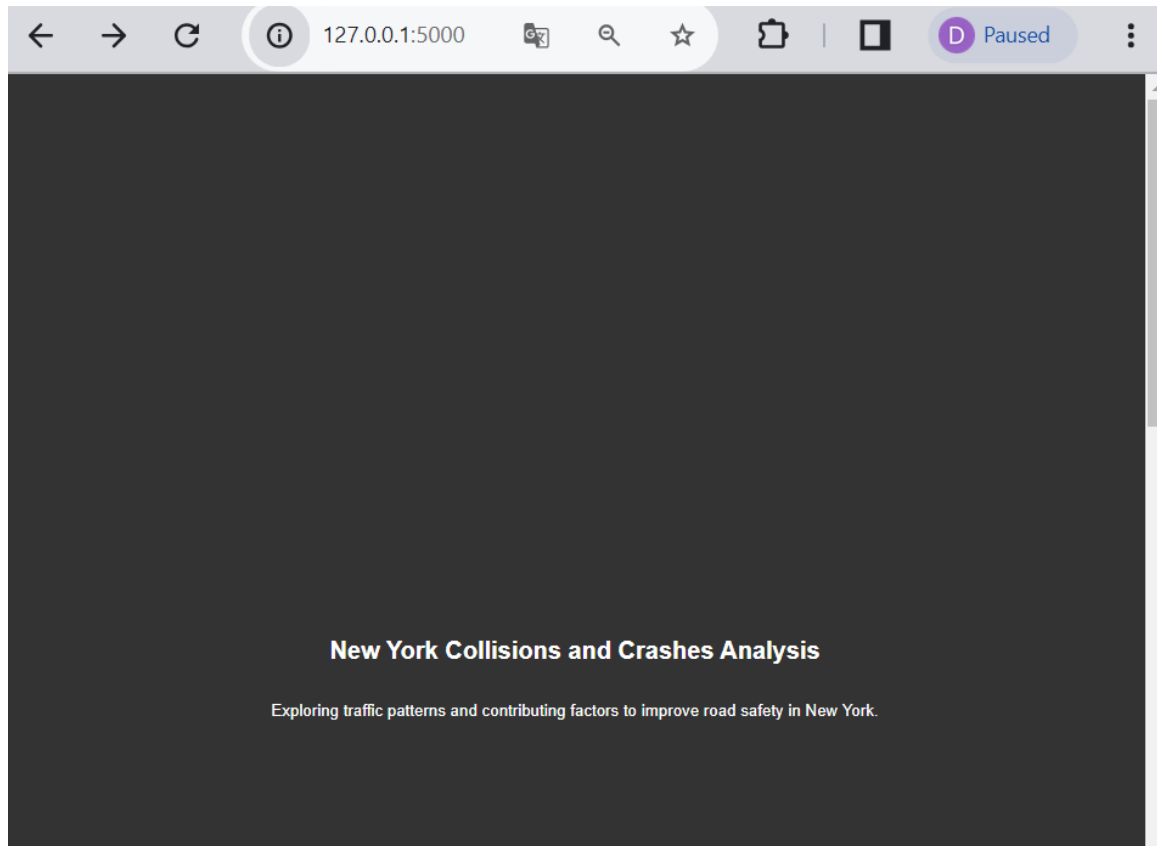
CONTRIBUTING FACTOR VEHICLE 1_encoder.pkl: Encoder for vehicle contributing factors.

ON STREET NAME_encoder.pkl: Encoder for street names.

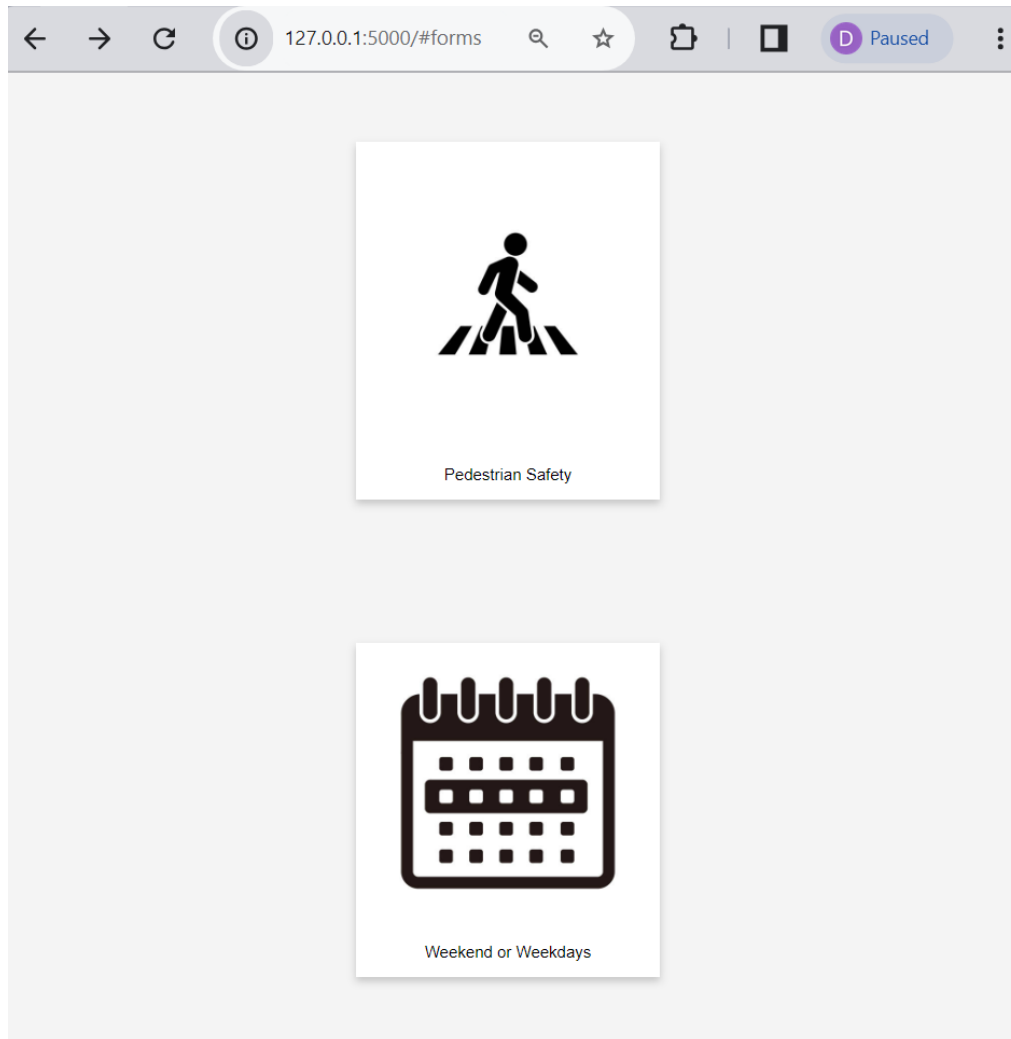
VEHICLE TYPE CODE 1_encoder.pkl: Encoder for vehicle types.

These models were trained during the earlier phases of the project and are used in the application to make predictions.

Working Website:

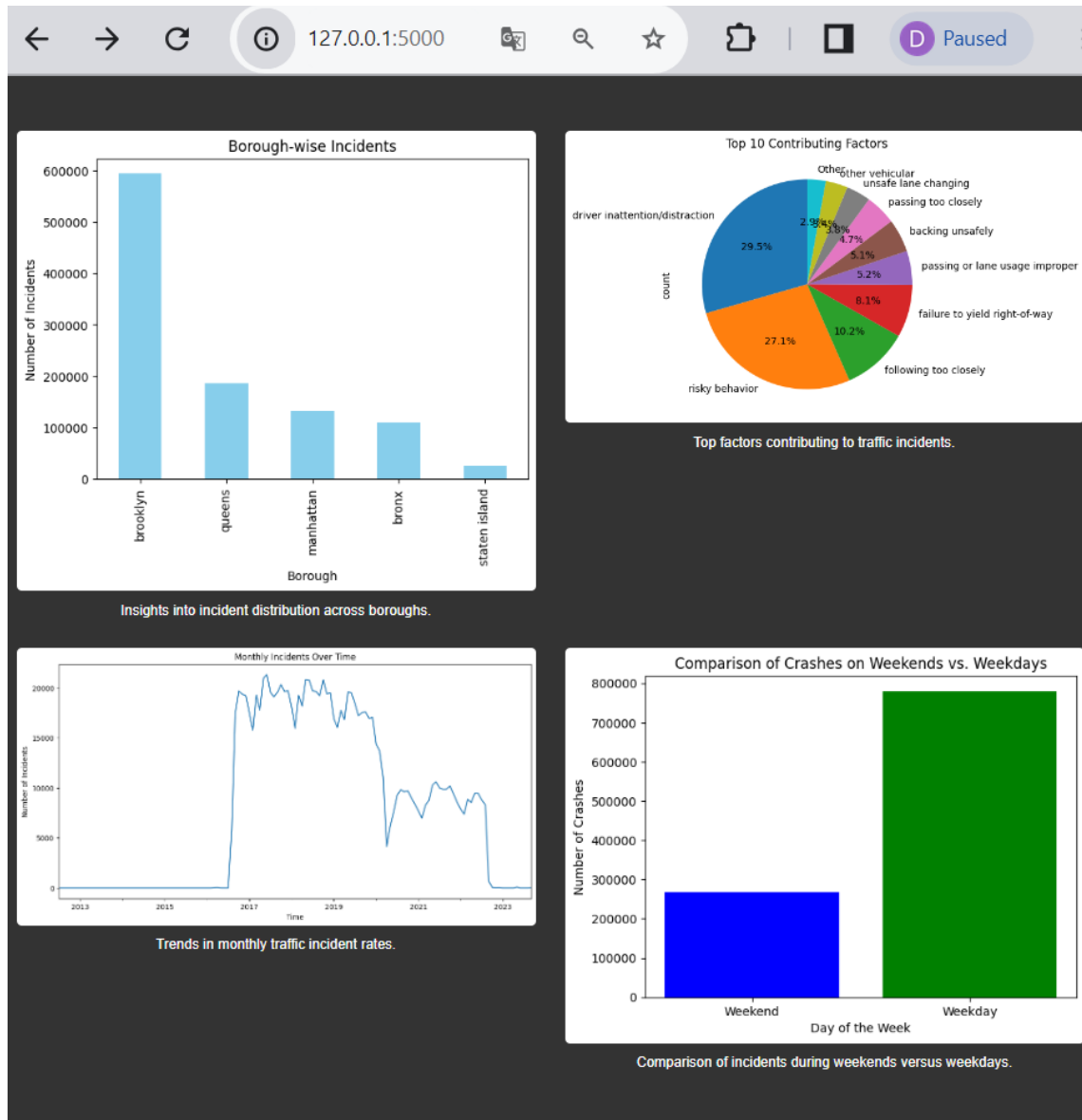


The home page introduces the purpose of the website: to explore traffic patterns and contributing factors for improving road safety in New York. It presents a clean and focused entry point that sets the context for the user's visit.



Two main functionalities highlighted on the home page are the Pedestrian Safety prediction and the Weekend Incident prediction. These features are meant to allow users to input specific parameters and receive predictions based on historical data and models trained with data from data.gov.

Each prediction form captures different sets of data—such as borough, number of persons injured or killed, contributing factors, vehicle type, and whether the time frame is a weekend or weekday—to predict the likelihood of an incident.



The visualizations section is designed to provide graphical insights into the data. This might include distribution of incidents across boroughs, contributing factors to incidents, trends over time, and comparisons of incidents between different times of the week.

Each visualization comes with a brief description, making it easier for users to understand what each graph represents.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/...'. The page title is 'Pedestrian Injury Prediction Form'. The form contains the following fields:

- Borough: Brooklyn
- Number of Cyclist Injured: 0
- Number of Motorist Injured: 0
- Contributing Factor Vehicle 1: View Obstructed/Limited
- Vehicle Type Code 1: Bus
- Weekend: Yes

A green 'Predict' button is located at the bottom of the form. Below the form, a grey bar displays the result: 'Chance of Pedestrian getting Injured: 0'.

After users input their data into the forms and submit it for prediction, they receive a dynamically updated prediction result, such as the "Chance of Pedestrian getting Injured" or the "Chance of Incident this Weekend."

← → ↻ ⓘ 127.0.0.1:5000/... 🔍 ☆ 📁 | 🖨️ 📱 Ⓟ Paused ⋮

Weekend Incident Prediction Form

Borough:
Brooklyn ▼

Number of Persons Killed:
0 ▼

Number of Pedestrians Killed:
0 ▼

Pedestrians Injured?:
0 ▼

Motorists Injured?:
0 ▼

Cyclist Injured?:
0 ▼

Contributing Factor Vehicle 1:
Other ▼

Vehicle Type Code 1:
Sedan ▼

Predict

Pedestrian Injury Prediction Form

Borough:

Number of Cyclist Injured:

Number of Motorist Injured:

Contributing Factor Vehicle 1:

Vehicle Type Code 1:

Weekend:

Chance of Pedestrian getting Injured [0-No, 1-Yes]: 1

```
TP/1.1" 304 -  
127.0.0.1 - - [09/Dec/2023 19:59:53] "GET /pedestrian HTTP/1.1" 200 -  
127.0.0.1 - - [09/Dec/2023 19:59:53] "GET /static/css/pedestrian.css HTTP/1.  
1" 304 -  
Encoded data for prediction: [[1 0 6 1 1 1]]  
Prediction: [1]  
127.0.0.1 - - [09/Dec/2023 20:00:11] "POST /predict_pedestrian HTTP/1.1" 200  
-  
127.0.0.1 - - [09/Dec/2023 20:00:11] "GET /static/css/pedestrian.css HTTP/1.  
1" 304 -  
|
```

Form Fields:

- Users can select the "Borough" from a dropdown menu, which in the example is set to "Brooklyn".
- They can specify the "Number of Cyclist Injured" and "Number of Motorist Injured", which are both set to "1" in this case.
- "Contributing Factor Vehicle 1" is a field where users select the main contributing factor to potential incidents. It's currently set to "Other".
- "Vehicle Type Code 1" allows users to choose the type of vehicle involved, with "Sedan" chosen in the example.
- The "Weekend" dropdown lets users indicate whether the prediction is for a weekend, selected as "Yes" here.
- The green "Predict" button is used to submit the data for analysis. Once clicked, the prediction model processes the input data and returns a result.

Prediction Result:

- The form, the prediction outcome is displayed. In this instance, the result reads "Chance of Pedestrian getting Injured [0-No, 1-Yes]: 1".
- The model has predicted that there is a likelihood (represented by "1") of a pedestrian injury occurring based on the input criteria. The binary outcome "1" indicates "Yes" in the context of the question, suggesting that according to the model's analysis, the conditions inputted lead to a scenario where a pedestrian injury is predicted to happen.

Pedestrian Injury Prediction Form

Borough:

Number of Cyclist Injured:

Number of Motorist Injured:

Contributing Factor Vehicle 1:

Vehicle Type Code 1:

Weekend:

Chance of Pedestrian getting Injured [0-No, 1-Yes]: 0

```
Encoded data for prediction: [[0 3 6 0 0 1]]
Prediction: [0]
127.0.0.1 - - [09/Dec/2023 20:08:54] "POST /predict_pedestrian HTTP/1.1" 200
-
127.0.0.1 - - [09/Dec/2023 20:08:54] "GET /static/css/pedestrian.css HTTP/1.
1" 304 -
```

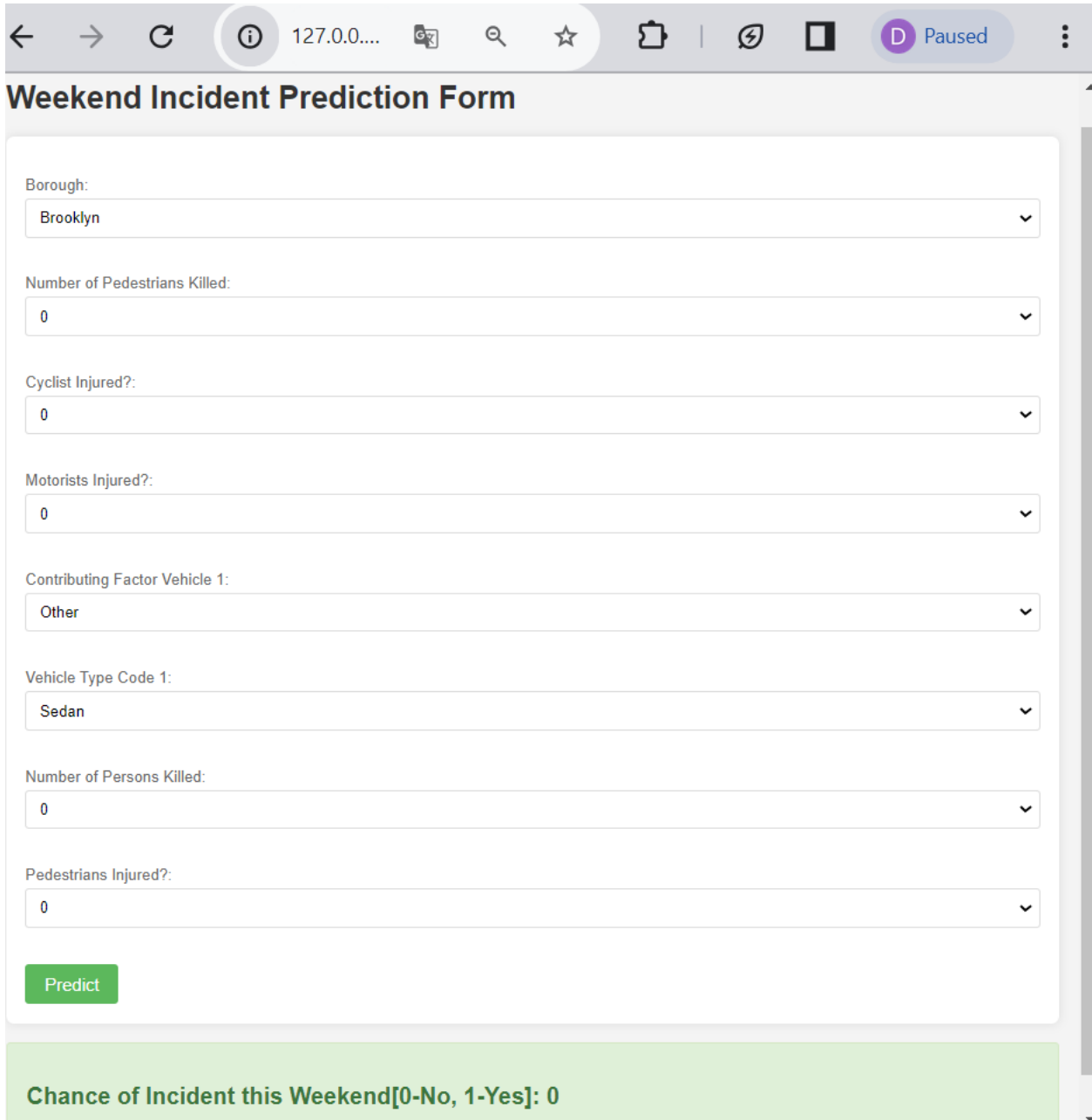
Form Fields:

- "Borough" is set to "Bronx", indicating the location for the prediction.
- "Number of Cyclist Injured" and "Number of Motorist Injured" are both set to "0", meaning no cyclists or motorists are reported as injured.
- "Contributing Factor Vehicle 1" is selected as "Backing Unsafely", which implies the vehicle was not following safety protocols while backing up.
- "Vehicle Type Code 1" is chosen as "Passenger Vehicle", indicating the type of vehicle involved in the scenario.
- "Weekend" is marked as "Yes", signifying the prediction is for a weekend.

Result:

- The result is shown: "Chance of Pedestrian getting Injured [0-No, 1-Yes]: 0".
- The model predicts there is no likelihood (indicated by "0") of a pedestrian being injured given the current inputs. This suggests that under the specified conditions—a pedestrian incident in the Bronx with no cyclists or motorists injured, a contributing factor of unsafe backing, involving a passenger vehicle, on a weekend—there isn't a significant risk of pedestrian injury according to the model's learned patterns from historical data.

Working Example of Weekend Incident Prediction Form:



The screenshot shows a web browser window with the address bar displaying '127.0.0.0...'. The page title is 'Weekend Incident Prediction Form'. The form contains the following fields:

- Borough: Brooklyn
- Number of Pedestrians Killed: 0
- Cyclist Injured?: 0
- Motorists Injured?: 0
- Contributing Factor Vehicle 1: Other
- Vehicle Type Code 1: Sedan
- Number of Persons Killed: 0
- Pedestrians Injured?: 0

A green 'Predict' button is located at the bottom of the form. Below the form, a green banner displays the result: 'Chance of Incident this Weekend[0-No, 1-Yes]: 0'.

Form Fields:

- The borough selected (Brooklyn in this case).
- The number of pedestrians killed (0 in this instance).
- The number of cyclists injured (0 here).
- The number of motorists injured (0 here).

- The contributing factor to the vehicle (Other in this case).
- The type of vehicle involved (Sedan here).
- The number of persons killed (0 in this submission).
- The number of pedestrians injured (0 in this submission).

Result:

- After the user submits these inputs by clicking the "Predict" button, the model processes the information and returns a prediction result.
- The result displayed, "Chance of Incident this Weekend[0-No, 1-Yes]: 0", indicates that, according to the model's prediction, there is no likelihood (represented as 0 for 'No') of an incident occurring over the weekend based on the data provided.

Objective of Model:

The primary objective of our model is to leverage predictive analytics for enhancing road safety by accurately forecasting pedestrian injuries and the likelihood of incidents during weekends in New York. Which helps to enhance public safety measures, and raise awareness about critical factors contributing to road accidents.

After evaluating several machine learning algorithms, we chose **XGBoost** (eXtreme Gradient Boosting) for its superior performance in terms of accuracy. XGBoost is renowned for its effectiveness in classification tasks, especially due to its ability to handle a large number of features and its use of gradient boosting frameworks which are adept at reducing overfitting. XGBoost is particularly beneficial in this context for several reasons:

- **Handling Sparse Data:** Given the nature of traffic data, which often includes many categorical features that result in sparse matrices after encoding, XGBoost can handle such data efficiently.
- **Feature Importance:** XGBoost provides a built-in way to assess feature importance, which is crucial for understanding and interpreting the predictive power of the model relative to the features.

- **Scalability and Speed:** With a large dataset such as ours, XGBoost is advantageous due to its scalability and faster computation time compared to other algorithms.
- XGBoost supports regularization parameters to prevent overfitting, which is essential for a model to generalize well to unseen data.
- **Accuracy:** XGBoost supports regularization parameters to prevent overfitting, which is essential for a model to generalize well to unseen data.

Model Exploration:

During the initial phase of model exploration, various machine learning algorithms were considered. These included:

- **Logistic Regression:** A foundational statistical approach for binary classification.
- **Decision Tree:** A flowchart-like tree structure where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.
- **Random Forest:** An ensemble method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees.
- **K Nearest Neighbors (KNN):** A simple, instance-based learning algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors.
- **Naive Bayes:** A collection of classification algorithms based on Bayes' Theorem, assuming that the presence of a particular feature in a class is unrelated to the presence of any other feature.

XGBoost was ultimately chosen due to its superior performance metrics which are evident from the confusion matrix and classification reports.

For Pedestrian Injury Analysis:

```

Confusion Matrix:
[[3899 6878]
 [1904 9091]]
Classification Report:
              precision    recall  f1-score   support

      0       0.67       0.36       0.47       10777
      1       0.57       0.83       0.67       10995

 accuracy         0.60         0.60         0.60       21772
 macro avg       0.62       0.59       0.57       21772
weighted avg       0.62       0.60       0.57       21772

Accuracy: 0.5966

```

```

Naive Bayes Model Confusion Matrix:
[[ 2071  8706]
 [   235 10760]]
Naive Bayes Model Classification Report:
              precision    recall  f1-score   support

      0       0.90       0.19       0.32       10777
      1       0.55       0.98       0.71       10995

 accuracy         0.59         0.59         0.59       21772
 macro avg       0.73       0.59       0.51       21772
weighted avg       0.72       0.59       0.51       21772

Naive Bayes Model Accuracy: 0.5893

```

```

KNN Model Confusion Matrix:
[[7829 2948]
 [4350 6645]]
KNN Model Classification Report:
              precision    recall  f1-score   support

      0       0.64       0.73       0.68       10777
      1       0.69       0.60       0.65       10995

 accuracy         0.66         0.66         0.66       21772
 macro avg       0.67       0.67       0.66       21772
weighted avg       0.67       0.66       0.66       21772

Accuracy of KNN Model: 0.5966

```

```

Random Forest Model Confusion Matrix:
[[5818 4959]
 [1448 9547]]
Random Forest Model Classification Report:
              precision    recall  f1-score   support

      0       0.80       0.54       0.64       10777
      1       0.66       0.87       0.75       10995

 accuracy         0.71         0.71         0.71       21772
 macro avg       0.73       0.70       0.70       21772
weighted avg       0.73       0.71       0.70       21772

Random Forest Model Accuracy: 0.7057

```

```

XGBoost Model Confusion Matrix:
[[5820 4957]
 [1410 9585]]
XGBoost Model Classification Report:
              precision    recall  f1-score   support

      0       0.80       0.54       0.65       10777
      1       0.66       0.87       0.75       10995

 accuracy         0.71         0.71         0.71       21772
 macro avg       0.73       0.71       0.70       21772
weighted avg       0.73       0.71       0.70       21772

XGBoost Model Accuracy: 0.7076

```

```

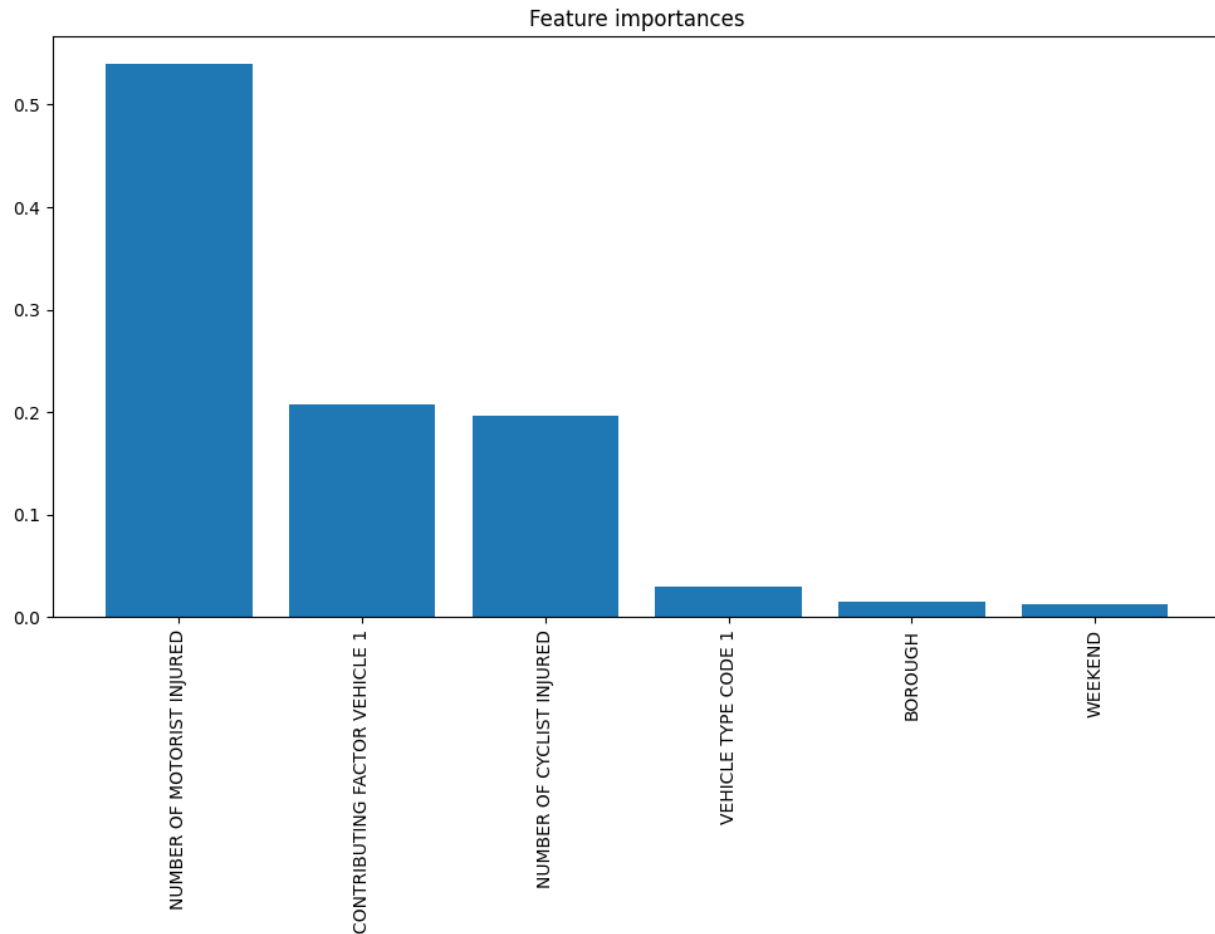
Decision Tree Model Confusion Matrix:
[[5848 4929]
 [1466 9529]]
Decision Tree Model Classification Report:
              precision    recall  f1-score   support

      0       0.80       0.54       0.65       10777
      1       0.66       0.87       0.75       10995

 accuracy         0.71         0.71         0.71       21772
 macro avg       0.73       0.70       0.70       21772
weighted avg       0.73       0.71       0.70       21772

Decision Tree Model Accuracy: 0.7063

```



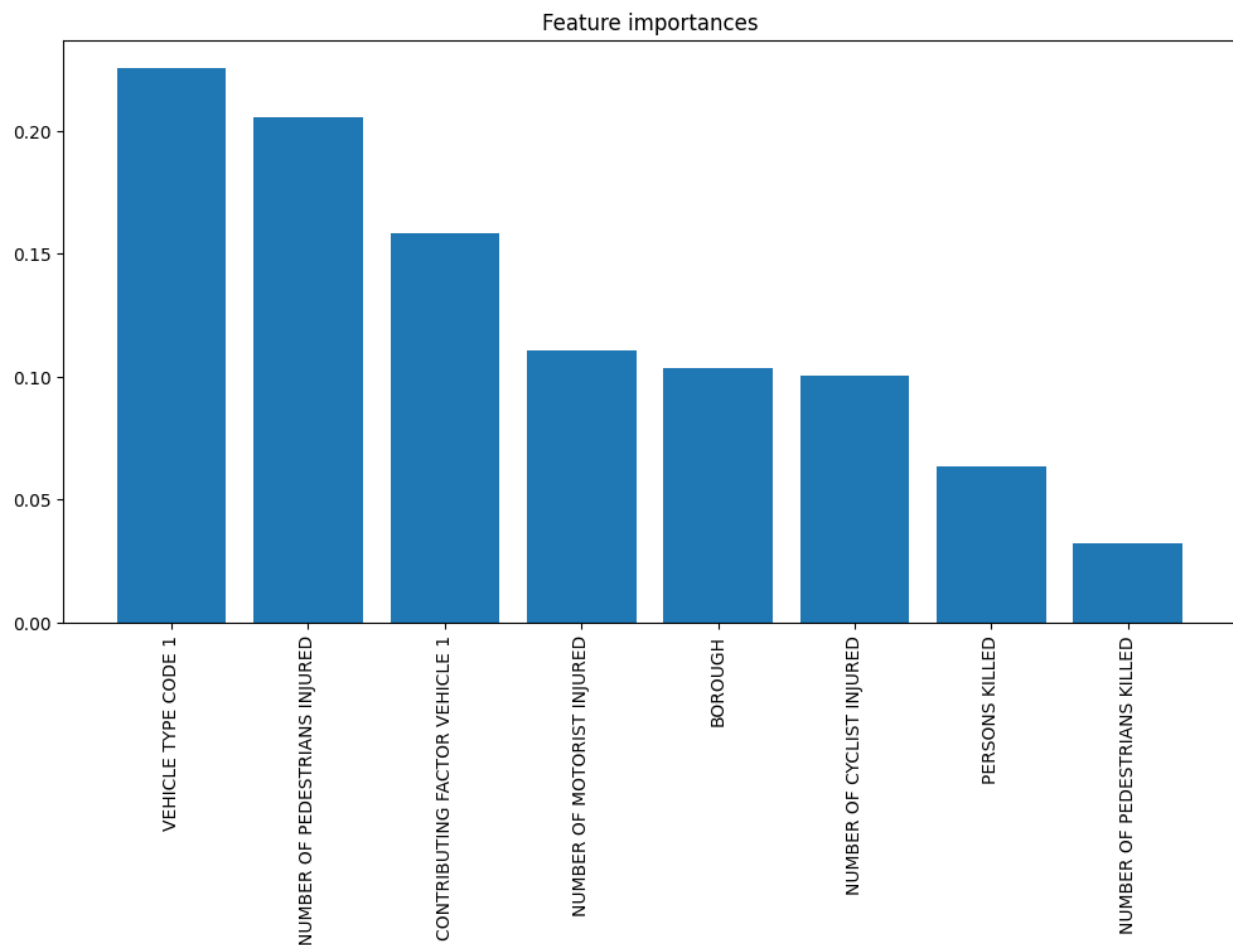
For Weekend and Weekday Analysis:

```
XGBoost Model Confusion Matrix:
[[16351  80]
 [ 5277  64]]
XGBoost Model Classification Report:
      precision    recall  f1-score   support

     0       0.76      1.00      0.86     16431
     1       0.44      0.01      0.02      5341

   accuracy              0.75     21772
  macro avg              0.60      0.50      0.44     21772
 weighted avg              0.68      0.75      0.65     21772

XGBoost Model Accuracy: 0.7540
```



The XGBoost models for predicting pedestrian injuries and weekend incidents in New York highlight key features that affect the likelihood of such events. For pedestrian injuries, the number of motorist injuries and the contributing factors related to vehicles emerged as significant predictors, suggesting that scenarios causing motorist injuries are also hazardous for pedestrians.

For weekend incidents, the importance of motorist injuries remained, indicating a strong correlation between motorist and overall incident rates during weekends. Additionally, the type of vehicle involved was a significant predictor, suggesting that certain vehicles are more likely to be involved in weekend incidents.

Learning from the Product:

Users can learn from the predictive model how different factors influence the likelihood of pedestrian injuries and weekend incidents. By interacting with the model, they gain awareness about the role of vehicle type, contributing factors from vehicles, and the number of motorists injured in accidents. This insight is valuable for individuals, policymakers, and organizations focusing on urban planning and road safety.

The model helps users to identify high-risk scenarios for pedestrians and during weekends. For example, the significant impact of motorist injuries on pedestrian safety can lead authorities to develop targeted interventions, such as speed control measures or traffic signal improvements in areas with high motorist accidents, ultimately protecting pedestrians.

Extending the project:

The current model can be extended by incorporating more granular temporal data, like the time of day or specific hours, to pinpoint when most injuries occur. Including weather conditions could also refine the model's predictive power, as adverse weather is a known risk factor for accidents.

The analysis could explore the relationship between traffic flow patterns and incidents to suggest better traffic management solutions. Another avenue could be the analysis of the impact of public awareness campaigns on reducing the rates of incidents identified as high-risk by the model.

On the technical side, implementing real-time data feeds into the model could allow for dynamic risk assessment. Additionally, deploying the model into a mobile application would make it more accessible to the public, potentially offering personalized safety recommendations based on user location and time.

Based on the model's insights, recommend policies for enhanced road safety measures, such as improved lighting in high-risk areas, stricter enforcement of traffic laws related to the most significant contributing factors, and urban infrastructure changes.

References:

- StatQuest with Josh Starmer (YouTube Video): "Random Forests Part 1 - Building, Using and Evaluating" _
https://www.youtube.com/watch?v=J4Wdy0Wc_xQ
- StatQuest with Josh Starmer (YouTube Video): "Naive Bayes, Clearly Explained!!!"
<https://www.youtube.com/watch?v=O2L2Uv9pdDA>
- DataCamp: "Supervised Learning with scikit-learn" _
<https://www.datacamp.com/courses/supervised-learning-with-scikit-learn>
- StatQuest with Josh Starmer (YouTube Video): "Decision Trees"
<https://www.youtube.com/watch?v=7VeUPuFGJHk>
-