Enhancing Pedestrian Safety in Great Britain:

A Machine Learning Analysis of Collision Severity

Tian Tong, Georgetown University

Abstract

This coding sample analyzes pedestrian-involved road collisions in the United Kingdom using 2023 Department for Transport microdata. The workflow includes structured data cleaning, codebook-based recoding, and feature engineering to identify pedestrian involvement and serious-injury outcomes. Multiple supervised learning models, logistic regression, random forest, and XGBoost, are trained to predict collision risk, evaluated with cross-validation, and interpreted through feature importance metrics. The models highlight the roles of roadway characteristics, lighting conditions, junction features, and temporal patterns in shaping pedestrian riskn under SHAP interpretation.

In parallel, collision coordinates are spatially joined to 2023 Local Authority District boundaries using GeoPandas, enabling district-level aggregation and the construction of choropleth maps that reveal clear geographic variation in pedestrian-related collision shares.

Overall, this coding sample demonstrates an end-to-end analytical pipeline that combines data preprocessing, feature engineering, machine learning, and geospatial analysis within a transparent and reproducible Python workflow. The study illustrates strong technical competency and an applied understanding of how administrative and spatial data can be leveraged to inform policy-relevant insights.

In [3]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from scipy.stats import randint, uniform
import geopandas as gpd

from sklearn import datasets
from sklearn.model_selection import (
    train_test_split, cross_val_score, GridSearchCV, RandomizedSearchCV
)
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_curve, auc, RocCurveDisplay, ConfusionMatrixDisplay,
    precision_recall_curve, roc_auc_score
```

```
)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import mutual_info_classif

from sklearn.svm import SVC
import xgboost as xgb

from sklearn.inspection import permutation_importance

import shap

import lime
import lime.lime_tabular

from joblib import Parallel, delayed

from imblearn.over_sampling import SMOTE

from imblearn.pipeline import Pipeline as ImPipeline

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor
```

In [4]:
```
# Load the road safety dataset
casualty_data = pd.read_csv('dft-road-casualty-statistics-casualty-provisior
```

In [5]:
```
collision_data = pd.read_csv('dft-road-casualty-statistics-collision-provisi
```

In [6]:
```
casualty_data.head()
```

Out[6]:

	collision_index	collision_year	collision_reference	vehicle_reference	casualty_refe
0	2023010419171	2023	010419171	1	
1	2023010419183	2023	010419183	2	
2	2023010419183	2023	010419183	3	
3	2023010419189	2023	010419189	1	
4	2023010419191	2023	010419191	2	

In [7]:
```
collision_data.head()
```

Out[7]:

	collision_index	collision_year	collision_reference	location_easting_osgr	location_
0	2023010419171	2023	010419171	525060.0	
1	2023010419183	2023	010419183	535463.0	
2	2023010419189	2023	010419189	508702.0	
3	2023010419191	2023	010419191	520341.0	
4	2023010419192	2023	010419192	527255.0	

5 rows × 36 columns

In [8]:

```
# Check the shape of the dataset
print(f"Number of Rows of Casualty Data: {casualty_data.shape[0]}, Number of
print(f"Number of Rows of Collision Data: {collision_data.shape[0]}, Number
```

```
Number of Rows of Casualty Data: 62674, Number of Columns: 19
Number of Rows of Collision Data: 49316, Number of Columns: 36
```

In [9]:

```
casualty_data['casualty_pedestrian'] = casualty_data['casualty_class'].apply
```

The STATS19 casualty dataset records the role of each individual involved in a road collision using the variable casualty_class, where:

- 1 = Driver/Rider
- 2 = Passenger
- 3 = Pedestrian

To construct a binary indicator identifying whether a casualty is a pedestrian, I recoded this variable as:

- 0 = not a pedestrian (casualty_class 1 or 2)
- 1 = pedestrian (casualty_class 3)

This transformation simplifies the subsequent analysis by converting multi-category casualty roles into a single interpretable binary measure indicating pedestrian involvement. Because multiple casualties can be linked to the same collision, I then aggregated this variable to the collision level. For each collision, I calculated: casualty_pedestrian_collision=max(casualty_pedestrian)

This value equals 1 if any casualty in the collision was a pedestrian, and 0 otherwise. Merging this collision-level indicator back into the collision dataset provides a consistent measure of pedestrian involvement that can be used for spatial aggregation, modeling, or severity analysis.

In [10]:

```
agg_casualty_data = casualty_data.groupby('collision_reference')['casualty_p

collision_data = collision_data.merge(agg_casualty_data, on='collision_refer
```

```
collision_data['casualty_pedestrian'] = collision_data['casualty_pedestrian'
```

In [13]:
```
print("Total casualties (pedestrians at casualty level):", casualty_data['ca
print("Collisions with pedestrians after merge:", collision_data['casualty_p
```

```
Total casualties (pedestrians at casualty level): 9221
Collisions with pedestrians after merge: 8885
```

In [11]:
```
print(collision_data['casualty_pedestrian'].value_counts())
```

```
casualty_pedestrian
0    40431
1     8885
Name: count, dtype: int64
```

This proportion is consistent with known characteristics of UK STATS19 data, where pedestrian-involved collisions represent a minority of total collisions but often account for a disproportionately high share of severe and fatal injuries. The indicator casualty_pedestrian therefore serves as a meaningful collision-level feature for downstream modeling, severity prediction, and spatial analysis.

In [14]:
```
# Show all columns in collision_data
print(list(collision_data.columns))
```

```
['collision_index', 'collision_year', 'collision_reference', 'location_easti
ng_osgr', 'location_northing_osgr', 'longitude', 'latitude', 'police_force',
'legacy_collision_severity', 'number_of_vehicles', 'number_of_casualties',
'date', 'day_of_week', 'time', 'local_authority_district', 'local_authority_
ons_district', 'local_authority_highway', 'first_road_class', 'first_road_nu
mber', 'road_type', 'speed_limit', 'junction_detail', 'junction_control', 's
econd_road_class', 'second_road_number', 'pedestrian_crossing_human_contro
l', 'pedestrian_crossing_physical_facilities', 'light_conditions', 'weather_
conditions', 'road_surface_conditions', 'special_conditions_at_site', 'carri
ageway_hazards', 'urban_or_rural_area', 'did_police_officer_attend_scene_of_
collision', 'trunk_road_flag', 'lsoa_of_collision_location', 'casualty_pedes
trian']
```

The collision dataset provides a comprehensive record of road collisions in Great Britain, including precise spatial coordinates, administrative geography, road characteristics, environmental conditions, temporal information, collision severity, and response details. These variables support both predictive modeling of collision severity and spatial analysis across local authority districts. A collision-level pedestrian indicator was constructed to identify crashes involving at least one pedestrian, enabling focused analysis of vulnerable road users.

In [16]:
```
# Check nulls
print(collision_data.isnull().sum())
```

```
collision_index                                0
collision_year                                 0
collision_reference                            0
location_easting_osgr                         84
location_northing_osgr                        84
longitude                                     84
latitude                                      84
police_force                                   0
legacy_collision_severity                      0
number_of_vehicles                             0
number_of_casualties                           0
date                                           0
day_of_week                                    0
time                                           0
local_authority_district                       0
local_authority_ons_district                   0
local_authority_highway                        0
first_road_class                               0
first_road_number                              0
road_type                                      0
speed_limit                                    0
junction_detail                                0
junction_control                               0
second_road_class                              0
second_road_number                             0
pedestrian_crossing_human_control              0
pedestrian_crossing_physical_facilities        0
light_conditions                               0
weather_conditions                             0
road_surface_conditions                        0
special_conditions_at_site                     0
carriageway_hazards                            0
urban_or_rural_area                            0
did_police_officer_attend_scene_of_collision   0
trunk_road_flag                                0
lsoa_of_collision_location                     0
casualty_pedestrian                            0
dtype: int64
```

In [17]:

```
# missing values column
columns_to_check = ['location_easting_osgr', 'location_northing_osgr', 'long

# Remove rows where columns have missing values
collision_data = collision_data.dropna(subset=columns_to_check)
```

In [18]:

```
# Convert the 'time' column to datetime
collision_data['time'] = pd.to_datetime(collision_data['time'], format='%H:%

# Round to the nearest hour
collision_data['nearest_hour'] = collision_data['time'].dt.round('H').dt.hou

print(collision_data[['time', 'nearest_hour']].head())
```

```
                 time  nearest_hour
0 1900-01-01 01:24:00             1
1 1900-01-01 02:25:00             2
2 1900-01-01 03:50:00             4
3 1900-01-01 02:13:00             2
4 1900-01-01 01:42:00             2
```

```
/var/folders/jt/3wdp6b7d0mj145sqgh8_fd480000gn/T/ipykernel_3770/561163590.p
y:5: FutureWarning: 'H' is deprecated and will be removed in a future versio
n, please use 'h' instead.
  collision_data['nearest_hour'] = collision_data['time'].dt.round('H').dt.h
our
```

The STATS19 collision dataset records the time of each collision using a string-based "time" variable formatted as "HH:MM". To enable hourly analysis, this field was converted into a proper datetime object using pd.to_datetime. The converted times were then rounded to the nearest hour (e.g., 14:25 → 14:00, 08:50 → 09:00) and the hour component was extracted as an integer. The resulting variable, nearest_hour, provides a clean hourly measure of collision timing that is suitable for temporal pattern analysis and can be used directly as a predictor in the modeling stage.

The "time" variable in the collision dataset contains only clock time ("HH:MM"). When pandas converts these values to datetime objects, it assigns a default placeholder date (1900–01–01) because a full datetime requires both a date and a time. This placeholder does not reflect the actual collision date, which is stored separately in the "date" and "collision_year" fields. The analysis uses only the hour component of the converted time, so the default date has no impact on the results.

In [19]:
```
# Find columns with only one unique value
columns_with_one_unique_value = [col for col in collision_data.columns if co

# Drop these columns
collision_data = collision_data.drop(columns=columns_with_one_unique_value)

# Check results
print("Dropped columns:", columns_with_one_unique_value)
```

```
Dropped columns: ['collision_year', 'local_authority_district', 'urban_or_ru
ral_area', 'trunk_road_flag']
```

Several variables in the provisional collision dataset contained only a single unique value (e.g., all rows recorded the same local authority district, urban/rural indicator, or trunk road flag). Since such variables have no variation and therefore provide no analytical or predictive value, they were removed. The informative geographic identifier (local_authority_ons_district) was retained for spatial aggregation.

In [20]:
```
column_name = [
    'number_of_vehicles',
    'number_of_casualties',
    'day_of_week',
    'nearest_hour',
    'road_type',
```

```
    'speed_limit',
    'junction_control',
    'junction_detail',
    'pedestrian_crossing_human_control',
    'light_conditions',
    'weather_conditions',
    'road_surface_conditions',
    'did_police_officer_attend_scene_of_collision',

    # spatial variables (added)
    'location_easting_osgr',
    'location_northing_osgr',
    'longitude',
    'latitude'
]
```

In [21]:

```
pd.set_option('display.float_format', '{:.2f}'.format)

# Summary statistics of numerical columns
collision_data.describe().T
```

Out[21]:

	count	mean	min
location_easting_osgr	49232.00	457529.78	1393.00
location_northing_osgr	49232.00	275497.73	11566.00
longitude	49232.00	-1.17	-7.55
latitude	49232.00	52.37	49.89
police_force	49232.00	27.15	1.00
legacy_collision_severity	49232.00	2.75	1.00
number_of_vehicles	49232.00	1.81	1.00
number_of_casualties	49232.00	1.27	1.00
day_of_week	49232.00	4.11	1.00
time	49232	1900-01-01 14:11:36.448244992	1900-01-01 00:00:00
first_road_class	49232.00	4.24	1.00
first_road_number	49232.00	774.63	-1.00
road_type	49232.00	5.31	1.00
speed_limit	49232.00	35.67	-1.00
junction_detail	49232.00	4.68	0.00
junction_control	49232.00	1.76	-1.00
second_road_class	49232.00	3.11	-1.00
second_road_number	49232.00	216.67	-1.00
pedestrian_crossing_human_control	49232.00	0.43	-1.00
pedestrian_crossing_physical_facilities	49232.00	1.23	-1.00
light_conditions	49232.00	1.93	-1.00
weather_conditions	49232.00	1.64	-1.00
road_surface_conditions	49232.00	1.37	-1.00
special_conditions_at_site	49232.00	0.30	-1.00
carriageway_hazards	49232.00	0.24	-1.00
did_police_officer_attend_scene_of_collision	49232.00	1.53	-1.00
casualty_pedestrian	49232.00	0.18	0.00
nearest_hour	49232.00	13.95	0.00

Analyze the invalid value of factors

In [22]:
```
print(collision_data['junction_detail'].value_counts())
```

```
junction_detail
0     20206
3     13765
6      4498
1      3602
9      2886
99     1157
8      1069
7       777
2       750
5       522
Name: count, dtype: int64
```

In [23]:
```
print(collision_data['junction_control'].value_counts())
```

```
junction_control
 4    21386
-1    20623
 2     5519
 9     1038
 3      356
 1      310
Name: count, dtype: int64
```

In [24]:
```
print(collision_data['pedestrian_crossing_human_control'].value_counts())
```

```
pedestrian_crossing_human_control
 0    45659
 9     2225
 2      670
-1      449
 1      229
Name: count, dtype: int64
```

In [25]:
```
print(collision_data['light_conditions'].value_counts())
```

```
light_conditions
 1    36276
 4     9621
 6     2357
 7      661
 5      316
-1        1
Name: count, dtype: int64
```

In [26]:
```
print(collision_data['weather_conditions'].value_counts())
```

```
weather_conditions
 1    40241
 2     4700
 9     1501
 8     1486
 5      441
 4      391
 3      290
 7      147
 6       34
-1        1
Name: count, dtype: int64
```

In [27]:
```python
print(collision_data['road_surface_conditions'].value_counts())
```

```
road_surface_conditions
 1    36503
 2    10350
 4      915
 9      684
-1      555
 3      178
 5       47
Name: count, dtype: int64
```

In [28]:
```python
print(collision_data['did_police_officer_attend_scene_of_collision'].value_c
```

```
did_police_officer_attend_scene_of_collision
 1    33374
 3    10125
 2     5727
-1        6
Name: count, dtype: int64
```

In [29]:
```python
# Define the invalid values for each column
invalid_values = {
    'junction_detail': [99],
    'junction_control': [-1, 9],
    'pedestrian_crossing_human_control': [-1, 9],
    'light_conditions': [-1],
    'weather_conditions': [-1],
    'road_surface_conditions': [-1, 9],
    'did_police_officer_attend_scene_of_collision': [-1]
}

# Loop through each column and replace invalid values with the median of val
for column, invalids in invalid_values.items():
    valid_data = collision_data[column][~collision_data[column].isin(invalic
    median_value = valid_data.median()

    # Replace invalid values with median
    collision_data[column] = collision_data[column].replace(invalids, mediar
```

In [30]:
```python
# Summary statistics of numerical columns
collision_data.describe().T
```

Out[30]:

	count	mean	min
location_easting_osgr	49232.00	457529.78	1393.00
location_northing_osgr	49232.00	275497.73	11566.00
longitude	49232.00	-1.17	-7.55
latitude	49232.00	52.37	49.89
police_force	49232.00	27.15	1.00
legacy_collision_severity	49232.00	2.75	1.00
number_of_vehicles	49232.00	1.81	1.00
number_of_casualties	49232.00	1.27	1.00
day_of_week	49232.00	4.11	1.00
time	49232	1900-01-01 14:11:36.448244992	1900-01-01 00:00:00
first_road_class	49232.00	4.24	1.00
first_road_number	49232.00	774.63	-1.00
road_type	49232.00	5.31	1.00
speed_limit	49232.00	35.67	-1.00
junction_detail	49232.00	2.40	0.00
junction_control	49232.00	3.75	1.00
second_road_class	49232.00	3.11	-1.00
second_road_number	49232.00	216.67	-1.00
pedestrian_crossing_human_control	49232.00	0.03	0.00
pedestrian_crossing_physical_facilities	49232.00	1.23	-1.00
light_conditions	49232.00	1.93	1.00
weather_conditions	49232.00	1.64	1.00
road_surface_conditions	49232.00	1.28	1.00
special_conditions_at_site	49232.00	0.30	-1.00
carriageway_hazards	49232.00	0.24	-1.00
did_police_officer_attend_scene_of_collision	49232.00	1.53	1.00
casualty_pedestrian	49232.00	0.18	0.00
nearest_hour	49232.00	13.95	0.00

In [32]:

```
collision_data = (collision_data.assign(
    casualty_over_serious=collision_data['legacy_collision_severity'].apply(
))
```

The STATS19 variable legacy_collision_severity classifies collisions into “Fatal” (1), “Serious” (2), and “Slight” (3). For analysis, I created a binary indicator casualty_over_serious equal to 1 for fatal or serious collisions and 0 for slight collisions. This transformation is common in road safety studies because it groups medically significant injuries into a single meaningful category while maintaining interpretability for statistical or spatial analysis.

In [33]:

```
# Create an interaction term between 'casualty_pedestrian' and 'casualty_ove
collision_data['pedestrian_over_serious'] = (
        collision_data['casualty_pedestrian'] * collision_data['casualty_ove
```

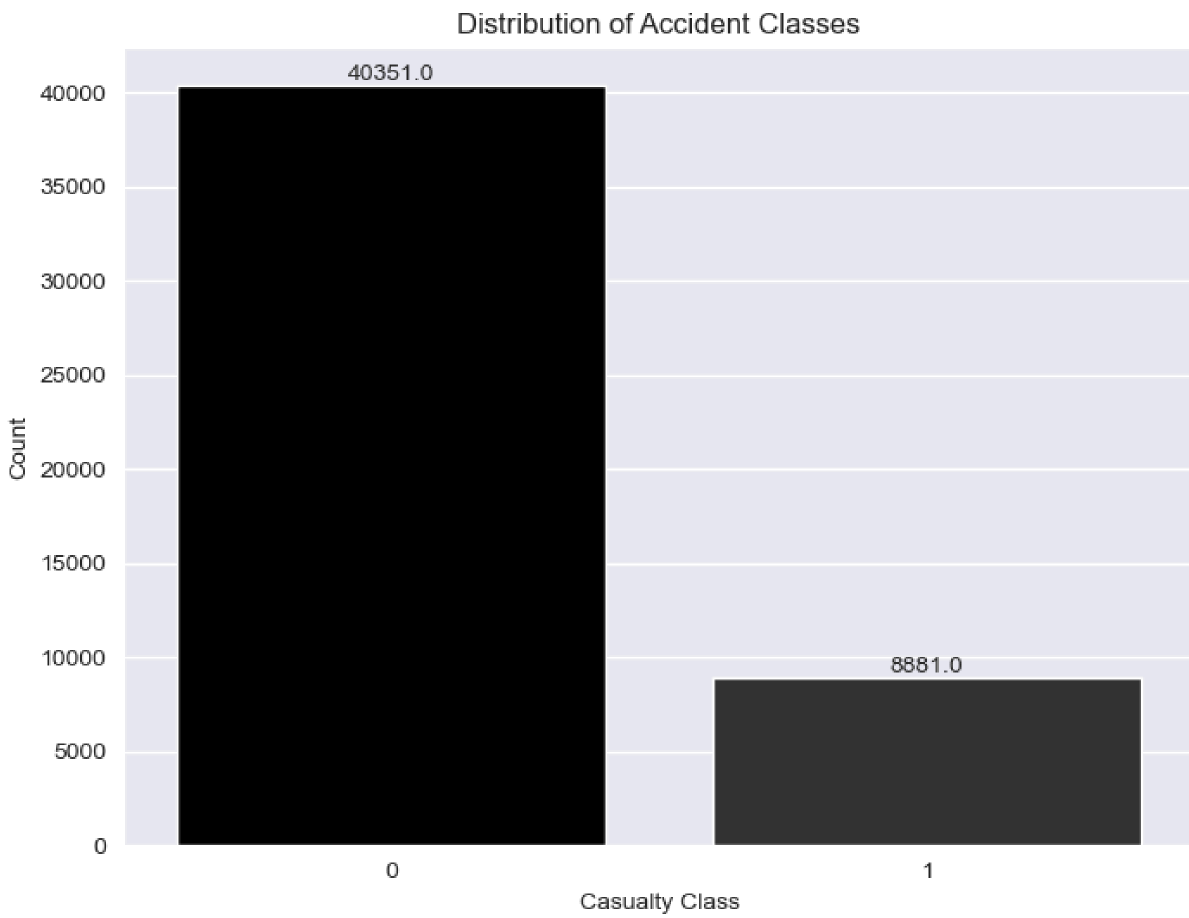
In [34]:

```
# Confusion matrix plotting function
def plot_confusion_matrix(cm, classes, title='Confusion matrix', cmap=plt.cm
    sns.heatmap(cm, annot=True, fmt="d", cmap=cmap)
    plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

EDA

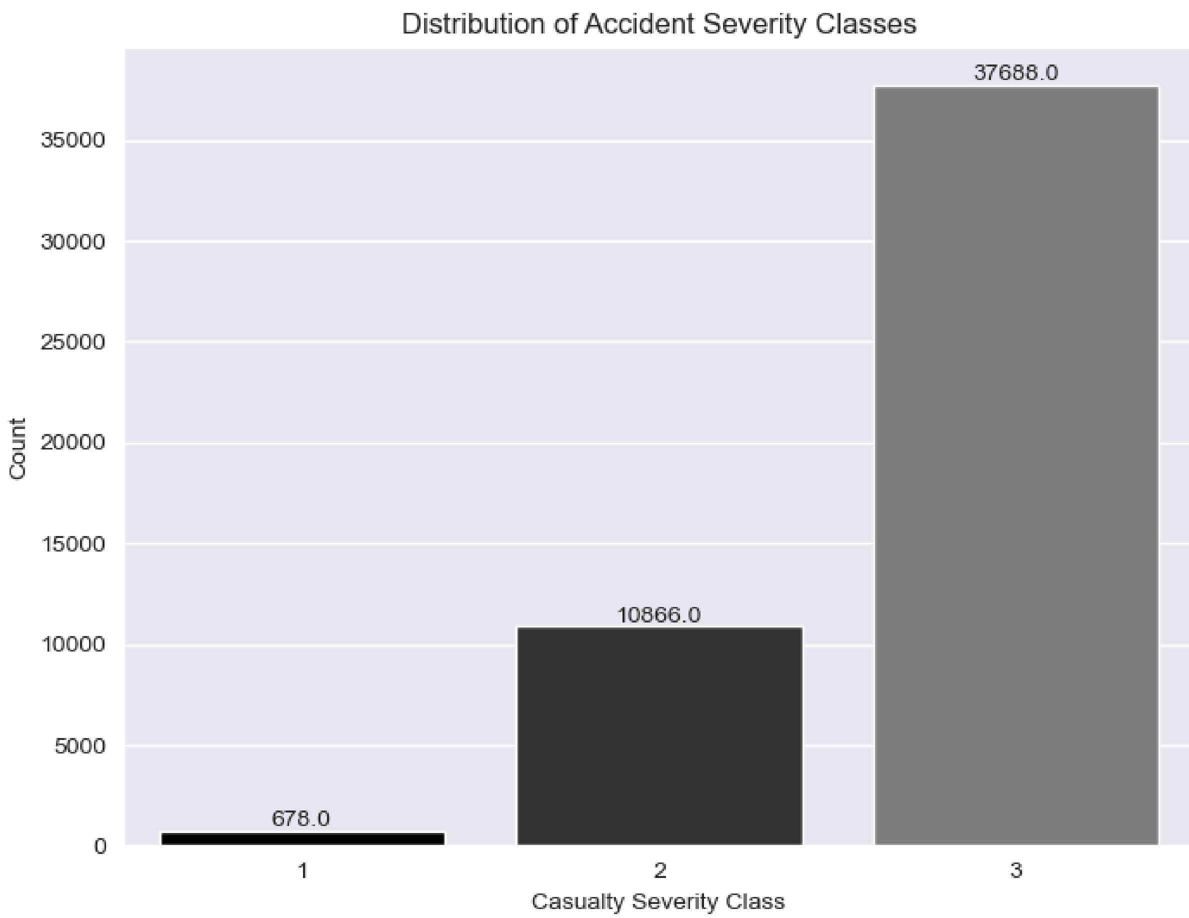
In [19]:

```
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='casualty_pedestrian', data=collision_data, palette=['b

plt.title('Distribution of Accident Classes')
plt.xlabel('Casualty Class')
plt.ylabel('Count')

# Add hover effect
def add_hover_effect(bar, data):
    for rect in bar.patches:
        bar_value = rect.get_height()
        plt.text(rect.get_x() + rect.get_width() / 2, bar_value, f'{bar_valu

add_hover_effect(ax, collision_data)

plt.show()
```



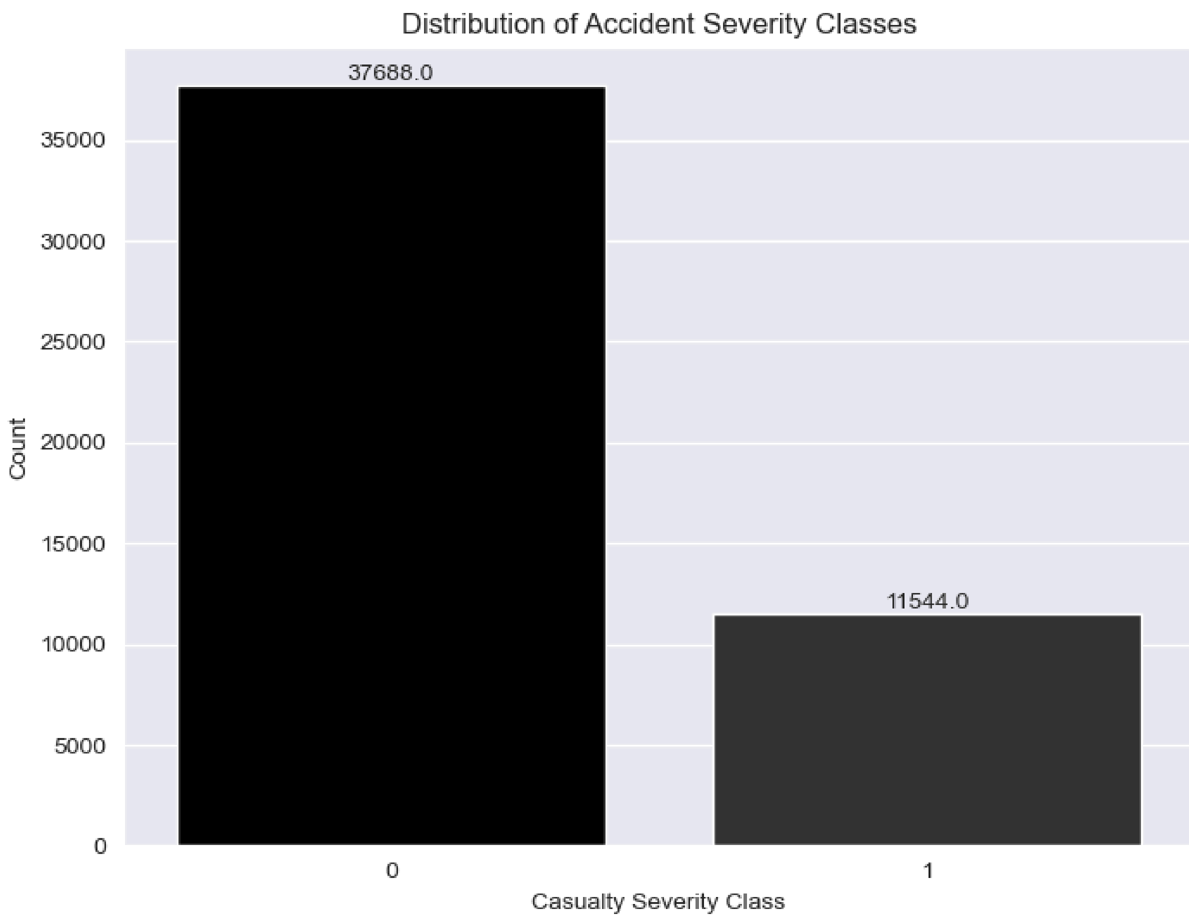
The pedestrian indicator shows a clear imbalance in the collision data. Out of 49,316 merged collision records, 40,431 cases (≈82%) involved no pedestrian, while only 8,885 cases (≈18%) included at least one pedestrian casualty.

In [20]:
```
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='legacy_collision_severity', data=collision_data, palet

plt.title('Distribution of Accident Severity Classes')
plt.xlabel('Casualty Severity Class')
plt.ylabel('Count')

# Add hover effect
def add_hover_effect(bar, data):
    for rect in bar.patches:
        bar_value = rect.get_height()
        plt.text(rect.get_x() + rect.get_width() / 2, bar_value, f'{bar_valu

add_hover_effect(ax, collision_data)

plt.show()
```

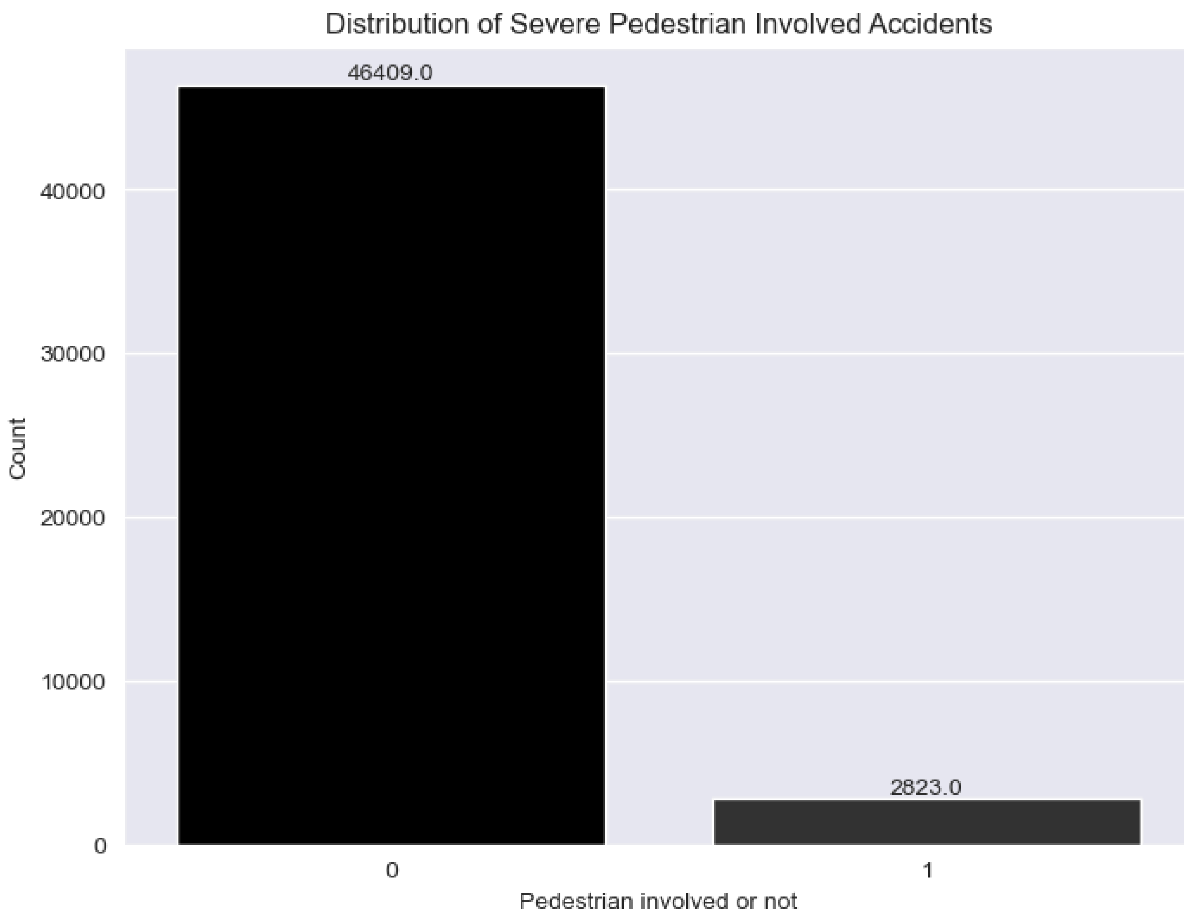


In [21]:

```
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='casualty_over_serious', data=collision_data, palette=[

plt.title('Distribution of Accident Severity Classes')
plt.xlabel('Casualty Severity Class')
plt.ylabel('Count')

# Add hover effect
def add_hover_effect(bar, data):
    for rect in bar.patches:
        bar_value = rect.get_height()
        plt.text(rect.get_x() + rect.get_width() / 2, bar_value, f'{bar_valu

add_hover_effect(ax, collision_data)

plt.show()
```



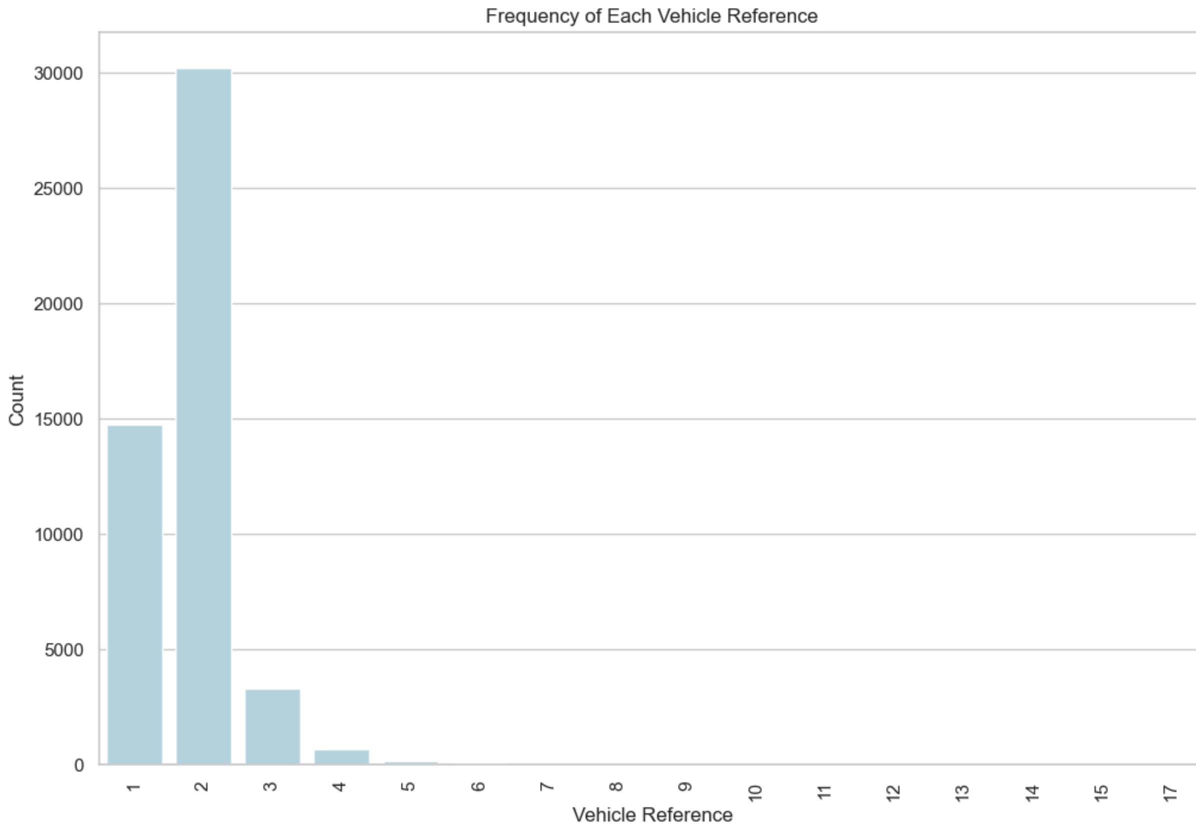
In [22]:
```
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='pedestrian_over_serious', data=collision_data, palette

plt.title('Distribution of Severe Pedestrian Involved Accidents')
plt.xlabel('Pedestrian involved or not')
plt.ylabel('Count')

# Add hover effect
def add_hover_effect(bar, data):
    for rect in bar.patches:
        bar_value = rect.get_height()
        plt.text(rect.get_x() + rect.get_width() / 2, bar_value, f'{bar_valu

add_hover_effect(ax, collision_data)

plt.show()
```



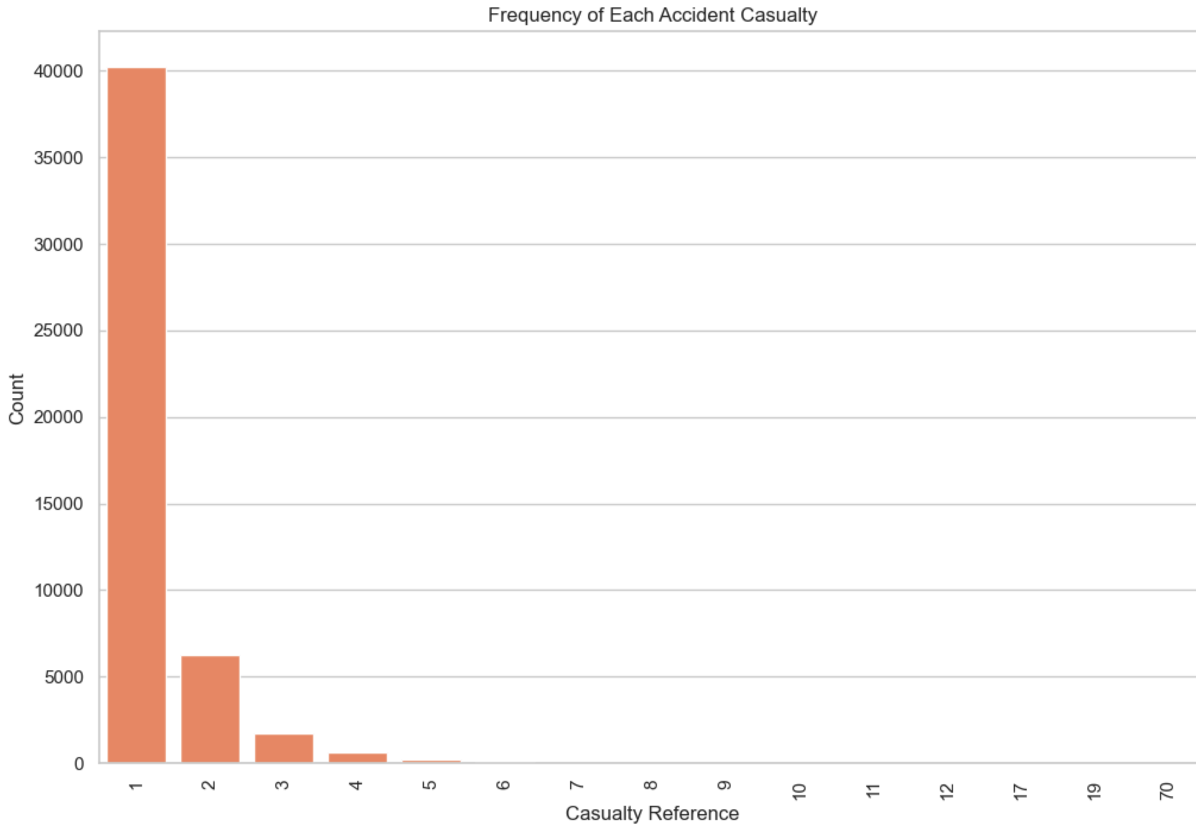
In [23]:
```
column_name = list['number_of_vehicles',
'number_of_casualties',
'day_of_week',
'nearest_hour',
'road_type',
'speed_limit',
'junction_control',
'junction_detail',
'pedestrian_crossing_human_control',
'light_conditions',
'weather_conditions',
'road_surface_conditions',
'did_police_officer_attend_scene_of_collision']
```

In [24]:
```
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Calculate the counts for each unique value in 'casualty_reference'
vehicle_counts = collision_data['number_of_vehicles'].value_counts().reset_i
vehicle_counts.columns = ['Vehicle Reference', 'Count']

plt.figure(figsize=(12, 8))
ax = sns.barplot(x='Vehicle Reference', y='Count', data=vehicle_counts, colo
ax.set_title('Frequency of Each Vehicle Reference')
ax.set_xlabel('Vehicle Reference')
ax.set_ylabel('Count')
```

```
plt.xticks(rotation=90)  # Rotate the labels to avoid overlap if necessary
plt.show()
```



In [25]:
```
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Calculate the counts for each unique value in 'casualty_reference'
vehicle_counts = collision_data['number_of_casualties'].value_counts().reset
vehicle_counts.columns = ['Casualty Reference', 'Count']

plt.figure(figsize=(12, 8))
ax = sns.barplot(x='Casualty Reference', y='Count', data=vehicle_counts, col
ax.set_title('Frequency of Each Accident Casualty')
ax.set_xlabel('Casualty Reference')
ax.set_ylabel('Count')
plt.xticks(rotation=90)  # Rotate the labels to avoid overlap if necessary
plt.show()
```



In [16]:
```
count_greater_than_20 = (collision_data['number_of_casualties'] > 20).sum()
print("Number of entries greater than 20 casualties:", count_greater_than_20
```

```
Number of entries greater than 20 casualties: 1
```

We would consider this point as outlier and remove it.

In [17]:
```
# Filter the dataset to keep only rows where 'number_of_casualties' is 20 or
collision_data = collision_data[collision_data['number_of_casualties'] <= 20
```

In [28]:
```
# Summary statistics of numerical columns
collision_data.describe().T
```

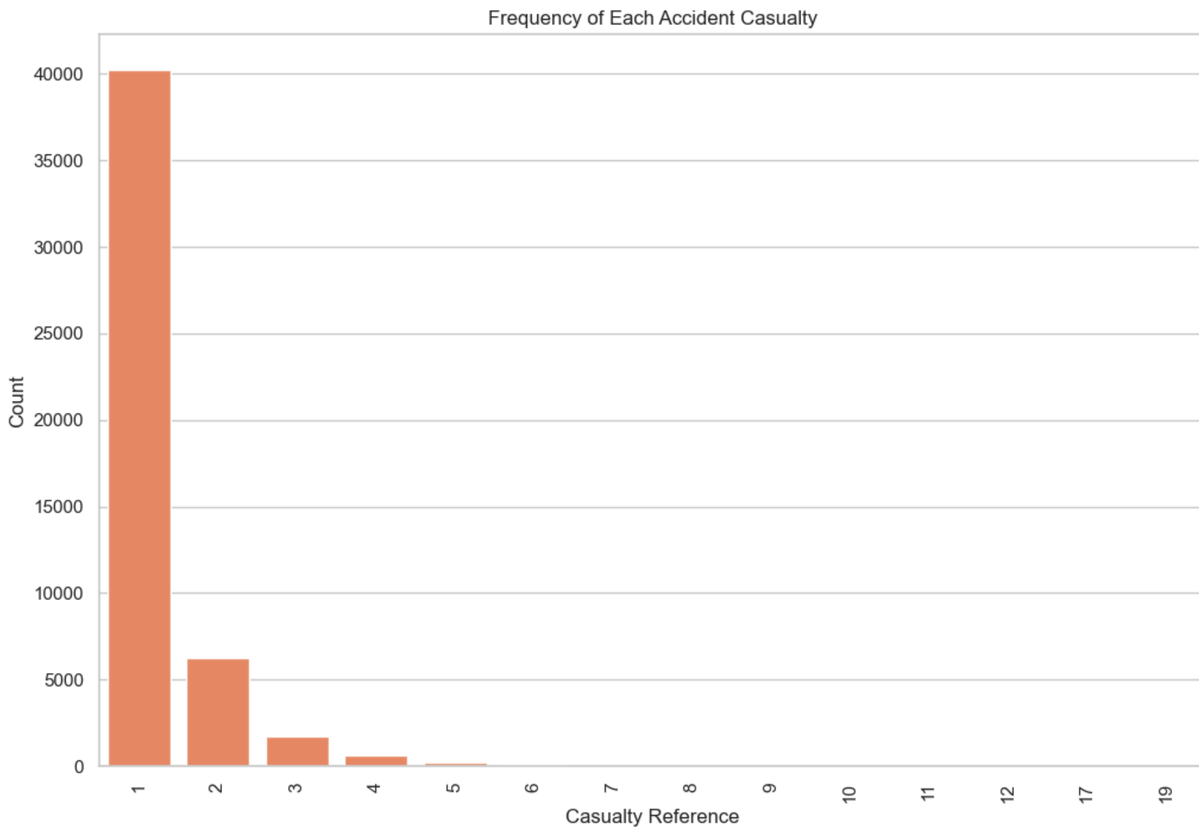
Out [28]:

	count	mean	std	min
location_easting_osgr	49231.00	457532.43	91954.85	1393.00
location_northing_osgr	49231.00	275500.52	146604.05	11566.00
longitude	49231.00	-1.17	1.34	-7.55
latitude	49231.00	52.37	1.32	49.89
police_force	49231.00	27.15	24.36	1.00
legacy_collision_severity	49231.00	2.75	0.46	1.00
number_of_vehicles	49231.00	1.81	0.69	1.00
number_of_casualties	49231.00	1.27	0.69	1.00
day_of_week	49231.00	4.11	1.91	1.00
first_road_class	49231.00	4.24	1.46	1.00
first_road_number	49231.00	774.64	1566.26	-1.00
road_type	49231.00	5.31	1.72	1.00
speed_limit	49231.00	35.67	14.20	-1.00
junction_detail	49231.00	2.40	2.73	0.00
junction_control	49231.00	3.75	0.67	1.00
second_road_class	49231.00	3.11	2.76	-1.00
second_road_number	49231.00	216.67	921.33	-1.00
pedestrian_crossing_human_control	49231.00	0.03	0.24	0.00
pedestrian_crossing_physical_facilities	49231.00	1.23	2.50	-1.00
light_conditions	49231.00	1.93	1.63	1.00
weather_conditions	49231.00	1.64	1.86	1.00
road_surface_conditions	49231.00	1.28	0.57	1.00
special_conditions_at_site	49231.00	0.30	1.53	-1.00
carriageway_hazards	49231.00	0.24	1.40	-1.00
did_police_officer_attend_scene_of_collision	49231.00	1.53	0.81	1.00
casualty_pedestrian	49231.00	0.18	0.38	0.00
nearest_hour	49231.00	13.95	5.28	0.00
casualty_over_serious	49231.00	0.23	0.42	0.00
pedestrian_over_serious	49231.00	0.06	0.23	0.00

In [29]:

```
# Set the aesthetic style of the plots
sns.set(style="whitegrid")
```

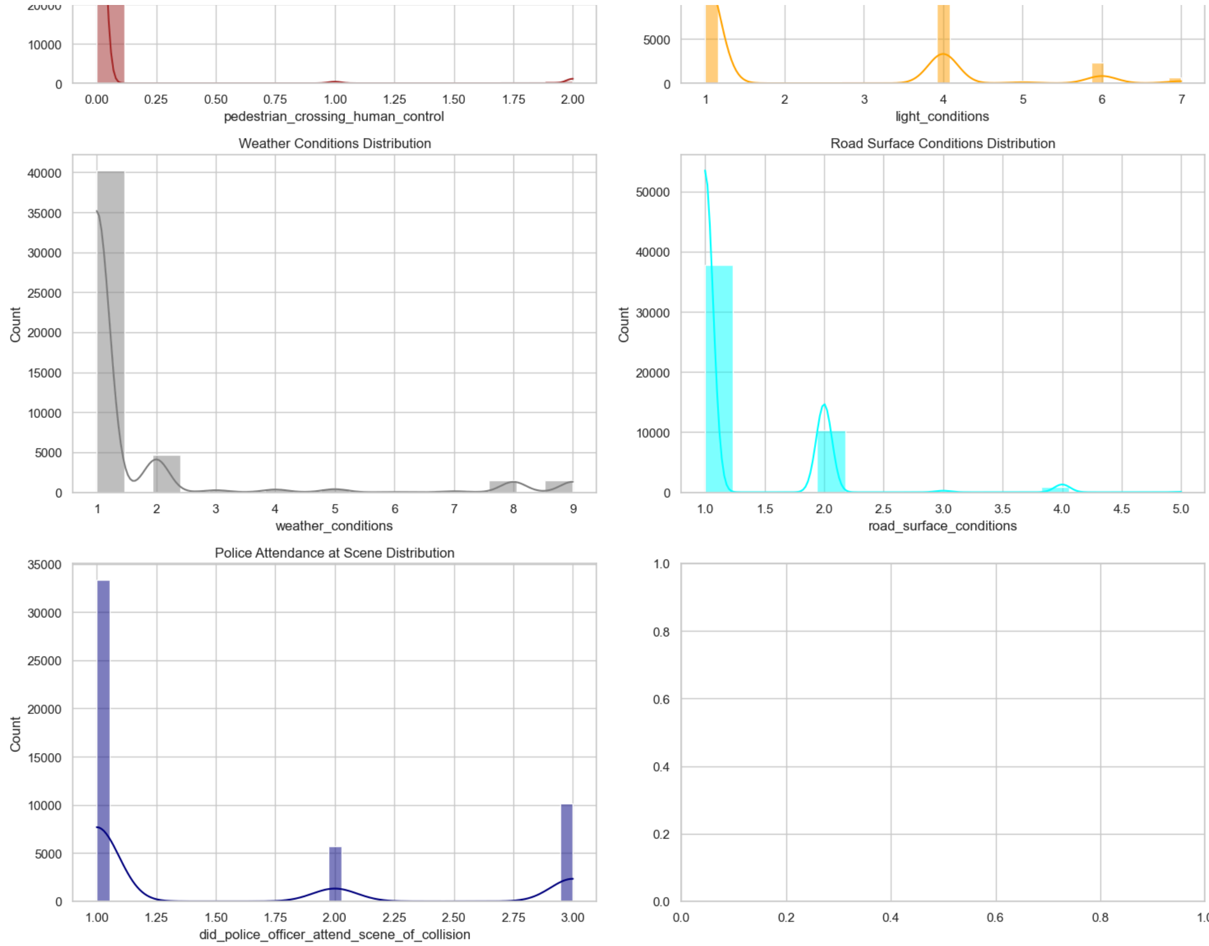
```
# Calculate the counts for each unique value in 'casualty_reference'
vehicle_counts = collision_data['number_of_casualties'].value_counts().reset
vehicle_counts.columns = ['Casualty Reference', 'Count']

plt.figure(figsize=(12, 8))
ax = sns.barplot(x='Casualty Reference', y='Count', data=vehicle_counts, col
ax.set_title('Frequency of Each Accident Casualty')
ax.set_xlabel('Casualty Reference')
ax.set_ylabel('Count')
plt.xticks(rotation=90)  # Rotate the labels to avoid overlap if necessary
plt.show()
```



In [30]:
```
sns.set(style="whitegrid")

fig, axes = plt.subplots(7, 2, figsize=(15, 35))  # Adjust grid size to acco


sns.histplot(collision_data['number_of_vehicles'], kde=True, ax=axes[0, 0],
sns.histplot(collision_data['number_of_casualties'], kde=True, ax=axes[0, 1]
sns.histplot(collision_data['day_of_week'], kde=True, ax=axes[1, 0], color='
sns.histplot(collision_data['nearest_hour'], kde=True, ax=axes[1, 1], color=
sns.histplot(collision_data['road_type'], kde=True, ax=axes[2, 0], color='re
sns.histplot(collision_data['speed_limit'], kde=True, ax=axes[2, 1], color='
sns.histplot(collision_data['junction_control'], kde=True, ax=axes[3, 0], co
sns.histplot(collision_data['junction_detail'], kde=True, ax=axes[3, 1], col
sns.histplot(collision_data['pedestrian_crossing_human_control'], kde=True,
sns.histplot(collision_data['light_conditions'], kde=True, ax=axes[4, 1], co
sns.histplot(collision_data['weather_conditions'], kde=True, ax=axes[5, 0],
sns.histplot(collision_data['road_surface_conditions'], kde=True, ax=axes[5,
sns.histplot(collision_data['did_police_officer_attend_scene_of_collision'],
```

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


Number of Vehicles Distribution
Count
number_of_vehicles
Number of Casualties Distribution
Count
number_of_casualties
Day of Week Distribution
Count
day_of_week
Nearest Hour Distribution
Count
nearest_hour
Road Type Distribution
Count
road_type
Speed Limit Distribution
Count
speed_limit
Junction Control Distribution
Count
junction_control
Junction Detail Distribution
Count
junction_detail
Pedestrian Crossing Human Control Distribution
Count
Light Conditions Distribution
Count



The exploratory analysis examined the distribution of key collision attributes, including temporal variables (day of week, hour of day), environmental conditions (weather, lighting, road surface), and road layout characteristics (junction types, speed limits, road classes). Summary statistics were used to identify dominant categories, assess variation across features, and check for data quality issues such as missing or invalid codes. Pedestrian involvement and collision severity patterns were also reviewed to understand class proportions and overall collision characteristics. This EDA step provided a general overview of the structure and composition of the dataset prior to preprocessing and spatial analysis.

Spatial Analysis

Spatial analysis was included to take advantage of the precise geographic information available in the collision dataset (latitude and longitude). Mapping collision locations and aggregating them to administrative boundaries helps identify whether pedestrian-involved or severe collisions are geographically concentrated, dispersed, or clustered in specific regions. Understanding these spatial patterns provides contextual insight that complements the non-spatial summary statistics and supports more informed interpretation of collision characteristics across different areas.

In [35]:
```
# Turn collision_data into a GeoDataFrame using longitude / latitude
collision_gdf = gpd.GeoDataFrame(
    collision_data,
    geometry=gpd.points_from_xy(collision_data["longitude"], collision_data[
    crs="EPSG:4326"  # WGS84
)
```

In [36]:
```
# Load the LAD 2023 boundaries
lad = gpd.read_file("Local_Authority_Districts_December_2023_Boundaries_UK_E

print(lad.crs)
print(lad.columns)
```

```
EPSG:27700
Index(['LAD23CD', 'LAD23NM', 'LAD23NMW', 'BNG_E', 'BNG_N', 'LONG', 'LAT',
       'GlobalID', 'geometry'],
      dtype='object')
```

CRS: EPSG:27700 → British National Grid (standard for UK spatial data)

Key columns:

- LAD23CD: LAD code (unique ID for each local authority)
- LAD23NM: LAD name (English)
- LAD23NMW: LAD name in Welsh (we might ignore this for our spatial analysis)
- BNG_E, BNG_N: centroid coordinates in British National Grid
- LONG, LAT: centroid coordinates in WGS84
- geometry: LAD polygon geometry (what we use for spatial join)

In [38]:
```
# Make sure both layers use the same CRS
collision_gdf = collision_gdf.to_crs(lad.crs)
```

In [39]:
```
# Keep only the needed LAD columns
lad_small = lad[["LAD23CD", "LAD23NM", "geometry"]]

# Spatial join: assign each collision to a LAD
collision_gdf = gpd.sjoin(
    collision_gdf,
    lad_small,
    how="left",
    predicate="within"
)
```

In [40]:
```
# QC
print(collision_gdf[["collision_reference", "LAD23CD", "LAD23NM"]].head())
print("Share of collisions with no LAD match:",
      collision_gdf["LAD23CD"].isna().mean())
```

```
  collision_reference    LAD23CD      LAD23NM
0           010419171  E09000024       Merton
1           010419183  E09000010      Enfield
2           010419189  E09000017   Hillingdon
3           010419191  E09000003       Barnet
4           010419192  E09000032   Wandsworth
Share of collisions with no LAD match: 0.00018280792980175496
```

99.982% of collisions found a correct LAD. Only about 9 out of ~49,000 collisions landed outside LAD polygons (likely on boundaries or water)

In [41]:
```python
# Aggregate collisions by LAD
lad_summary = (
    collision_gdf
    .groupby("LAD23CD", as_index=False)
    .agg(
        lad_name=("LAD23NM", "first"),
        collisions=("collision_reference", "nunique"),
        ped_collisions=("casualty_pedestrian", "sum"),
        serious_collisions=("casualty_over_serious", "sum")
    )
)

lad_summary["ped_share"] = lad_summary["ped_collisions"] / lad_summary["coll
lad_summary["serious_share"] = lad_summary["serious_collisions"] / lad_summa

lad_summary.head()
```

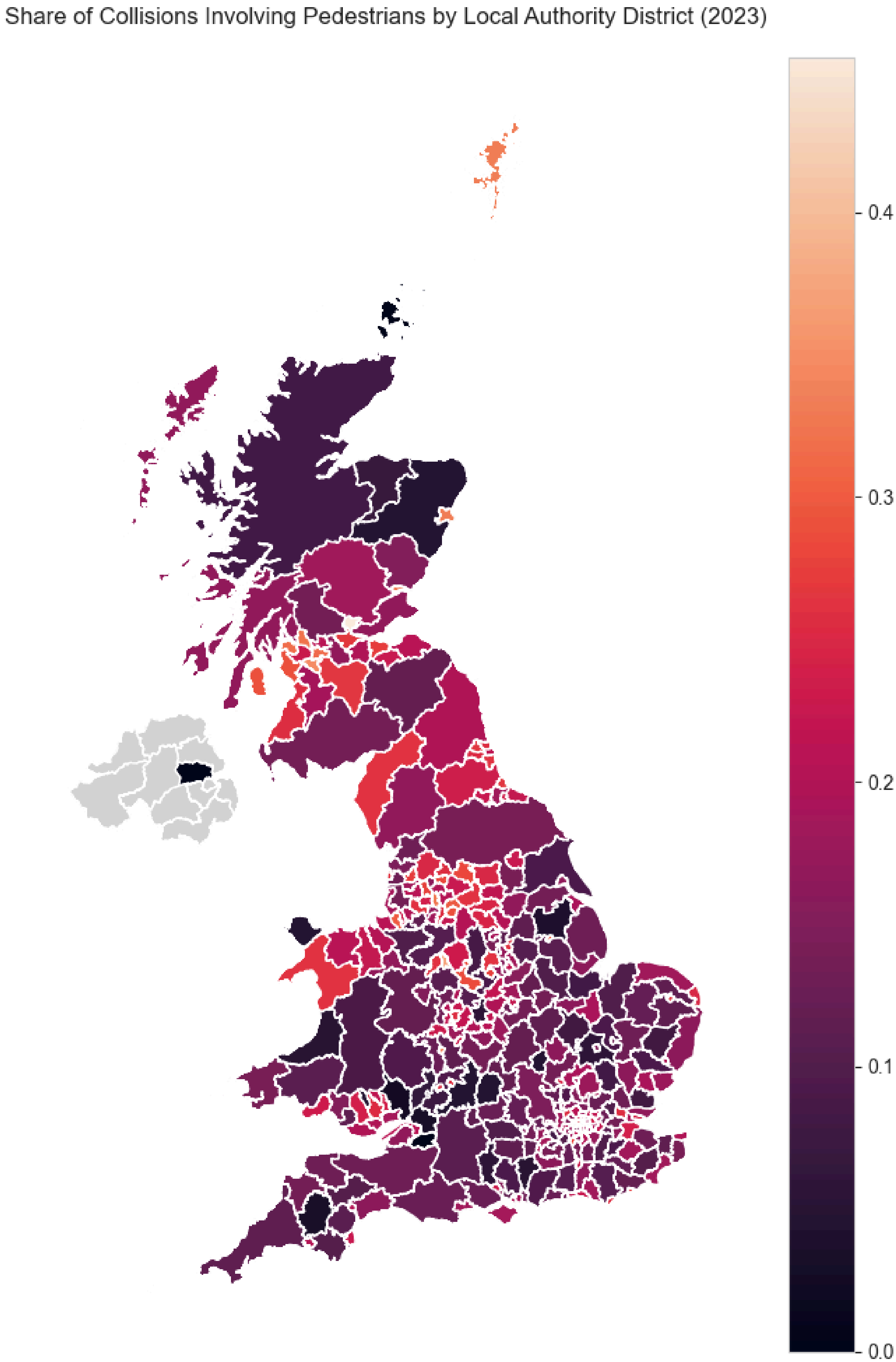
Out[41]:

	LAD23CD	lad_name	collisions	ped_collisions	serious_collisions	ped_share
0	E06000001	Hartlepool	49	13	11	0.27
1	E06000002	Middlesbrough	97	23	18	0.24
2	E06000003	Redcar and Cleveland	63	14	21	0.22
3	E06000004	Stockton-on-Tees	102	19	21	0.19
4	E06000005	Darlington	62	14	20	0.23

In [42]:
```python
# Merge LAD summary into LAD polygons
lad_merged = lad.merge(lad_summary, on="LAD23CD", how="left")
```

In [43]:
```python
# choropleth map
fig, ax = plt.subplots(figsize=(8, 10))

lad_merged.plot(
    column="ped_share",     # could also plot "serious_share"
    legend=True,
    ax=ax,
    missing_kwds={"color": "lightgrey", "label": "No data"}
)

ax.set_axis_off()
```

```
ax.set_title("Share of Collisions Involving Pedestrians by Local Authority D
plt.tight_layout()
plt.show()
```


Share of Collisions Involving Pedestrians by Local Authority District (2023)
0.4
0.3
0.2
0.1
0.0

In [48]:
```
# Reproject to WGS84 for interactive web mapping
lad_merged_web = lad_merged.to_crs(epsg=4326)

m = lad_merged_web.explore(
    column="ped_share",                         # same variable as your static m
    cmap="magma",                               # or whatever colormap you used
    legend=True,
    tooltip=["LAD23NM", "collisions", "ped_collisions", "ped_share"],  # hov
    popup=False,
)

m.save("ped_share_explore.html")
```

The choropleth map shows the proportion of collisions that involved at least one pedestrian in each Local Authority District in the United Kingdom during 2023. Higher pedestrian collision shares appear in several densely populated areas, including many London boroughs, parts of the West Midlands, and portions of central Scotland. Lower shares are more common in rural districts and areas with lower traffic intensity. The pattern suggests that pedestrian exposure and local road environments vary meaningfully across regions, which may help explain differences in collision risk.

Predictive Modeling

Random Forest and XGBoost were used as the primary predictive models because both algorithms handle nonlinear relationships, high-dimensional categorical data, and complex interaction effects common in collision datasets. Random Forest provides a robust baseline due to its stability and resistance to overfitting, while XGBoost offers improved predictive performance through gradient-boosted trees and built-in regularization. Together, these models allow for reliable comparison, strong out-of-sample accuracy, and interpretable feature importance measures relevant for understanding key factors associated with pedestrian or high-severity collisions.

Model Training: Pedestrian

In [31]:
```
# Set the target feature as 'Pdestrian'
X = collision_data[['number_of_vehicles',
                    'number_of_casualties',
                    'day_of_week',
                    'nearest_hour',
                    'road_type',
                    'speed_limit',
                    'junction_control',
                    'junction_detail',
                    'pedestrian_crossing_human_control',
                    'light_conditions',
                    'weather_conditions',
                    'road_surface_conditions',
                    'did_police_officer_attend_scene_of_collision']]
```

```
y = collision_data['casualty_pedestrian']


# Perform an 80-20 training-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Address class imbalance in the training set using SMOTE
print('Original dataset shape %s' % Counter(y_train))

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check whether the imbalance issue has been addressed
print('Resampled dataset shape %s' % Counter(y_train_smote))
```

```
Original dataset shape Counter({0: 32279, 1: 7105})
Resampled dataset shape Counter({0: 32279, 1: 32279})
```

Random Forest Model: Pedestrian Focus

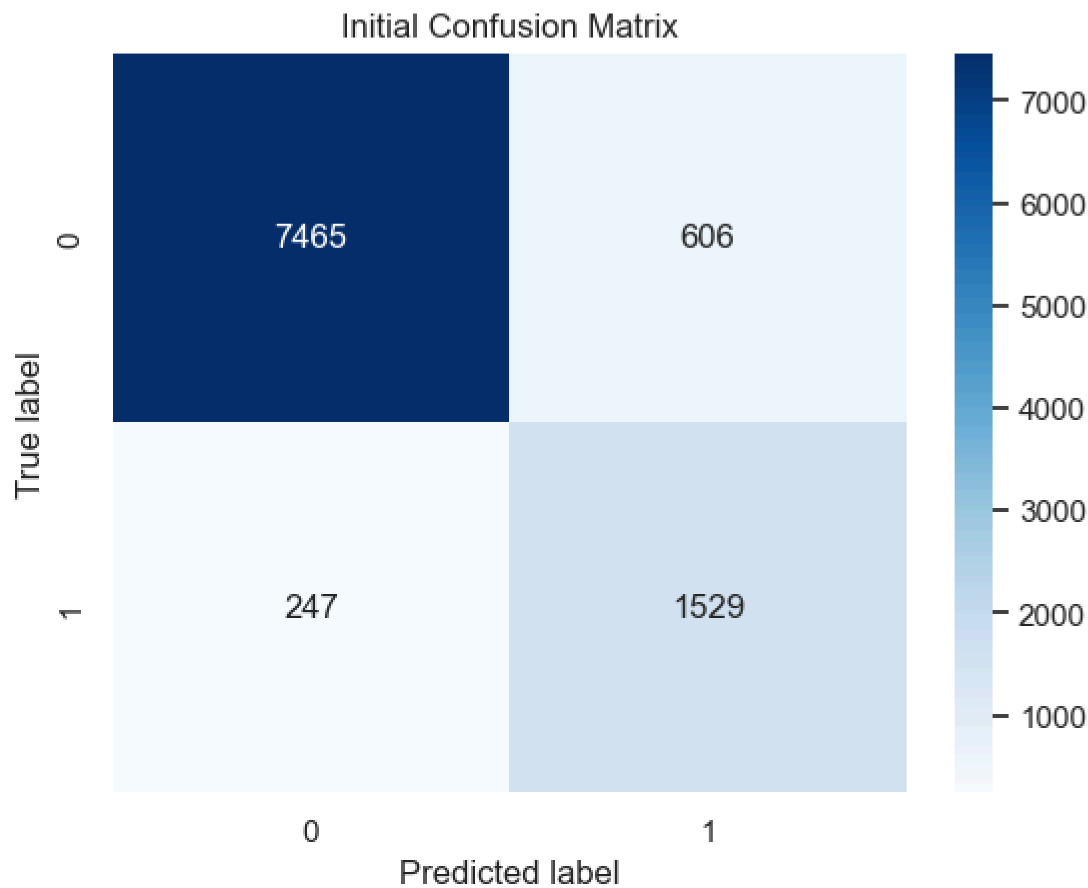
In [28]:
```
# Initialize the Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Train the model
rf.fit(X_train_smote, y_train_smote)
```

Out[28]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [33]:
```
# Make predictions
y_pred_pd = rf.predict(X_test)
y_pred_proba_pd = rf.predict_proba(X_test)[:, 1]  # Probabilities for ROC AU

# Calculate metrics
accuracy_pd = accuracy_score(y_test, y_pred_pd)
roc_auc_pd = roc_auc_score(y_test, y_pred_proba_pd)
report_pd = classification_report(y_test, y_pred_pd)

print(f"ROC AUC: {roc_auc_pd:.5f}")
print(f"Accuracy before Tuning of Pedestrian: {accuracy_pd:.5f}")
print("\n Classification Report before Tuning of Pedestrian:\n", report_pd)
```

```
ROC AUC: 0.93959
Accuracy before Tuning of Pedestrian: 0.91337

 Classification Report before Tuning of Pedestrian:
               precision    recall  f1-score   support

           0       0.97      0.92      0.95      8071
           1       0.72      0.86      0.78      1776

    accuracy                           0.91      9847
   macro avg       0.84      0.89      0.86      9847
weighted avg       0.92      0.91      0.92      9847
```

In [34]:
```
cm_initial_rf= confusion_matrix(y_test, y_pred_pd)
plot_confusion_matrix(cm_initial_rf, classes=['Not Pedestrian', 'Pedestrian'
```



In [35]:
```
# Define the parameter grid
param_grid_pd = {
    'n_estimators': [50,100,200],
    'max_depth': [None, 1,5,10],
    'min_samples_leaf': [1,2,10],
    'min_samples_split': [2,5,10]
}

clf_pd = RandomForestClassifier(random_state=42)

# Set up Grid Search CV
grid_search_pd = GridSearchCV(estimator=clf_pd,
```

```
                              param_grid=param_grid_pd,
                              cv=5,
                              scoring='accuracy',
                              n_jobs=-1,
                              verbose=1)

# Perform grid search
grid_search_pd.fit(X_train_smote, y_train_smote)

# Output the best parameters and the corresponding score
print("Best parameters:", grid_search_pd.best_params_)
print("Best score: {:.5f}".format(grid_search_pd.best_score_))
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_spl
it': 5, 'n_estimators': 200}
Best score: 0.94053
```

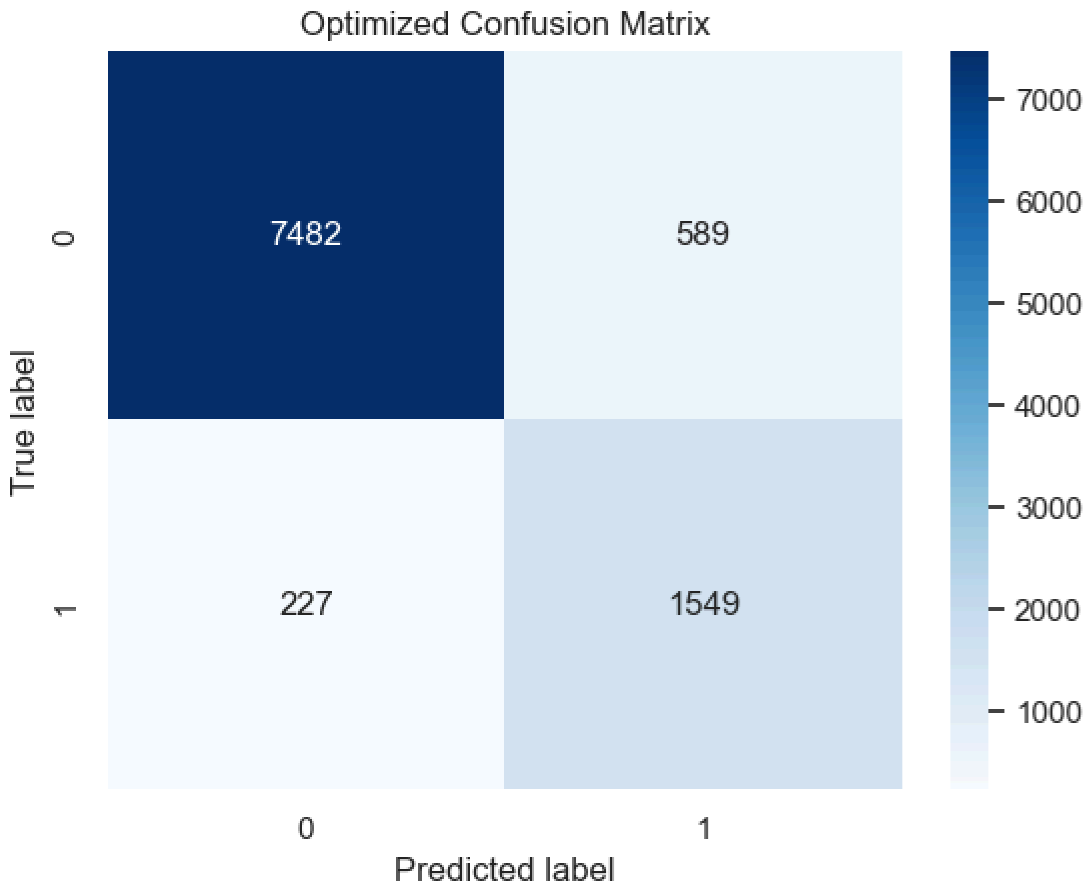
In [36]:

```
# Re-train the model using the best parameters
pd_optimized = RandomForestClassifier(**grid_search_pd.best_params_, random_
pd_optimized.fit(X_train_smote, y_train_smote)
# Re-evaluate the model
y_pred_opt_pd = pd_optimized.predict(X_test)
y_pred_opt_proba_pd = pd_optimized.predict_proba(X_test)[:, 1]
accuracy_opt_pd= accuracy_score(y_test, y_pred_opt_pd)
roc_auc_opt_pd = roc_auc_score(y_test, y_pred_opt_proba_pd)
report_opt_pd = classification_report(y_test, y_pred_opt_pd)

print(f"Optimized Accuracy: {accuracy_opt_pd:.5f}")
print("\n Optimized Classification Report:\n", report_opt_pd)
```

```
Optimized Accuracy: 0.91713

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.93      0.95      8071
           1       0.72      0.87      0.79      1776

    accuracy                           0.92      9847
   macro avg       0.85      0.90      0.87      9847
weighted avg       0.93      0.92      0.92      9847
```

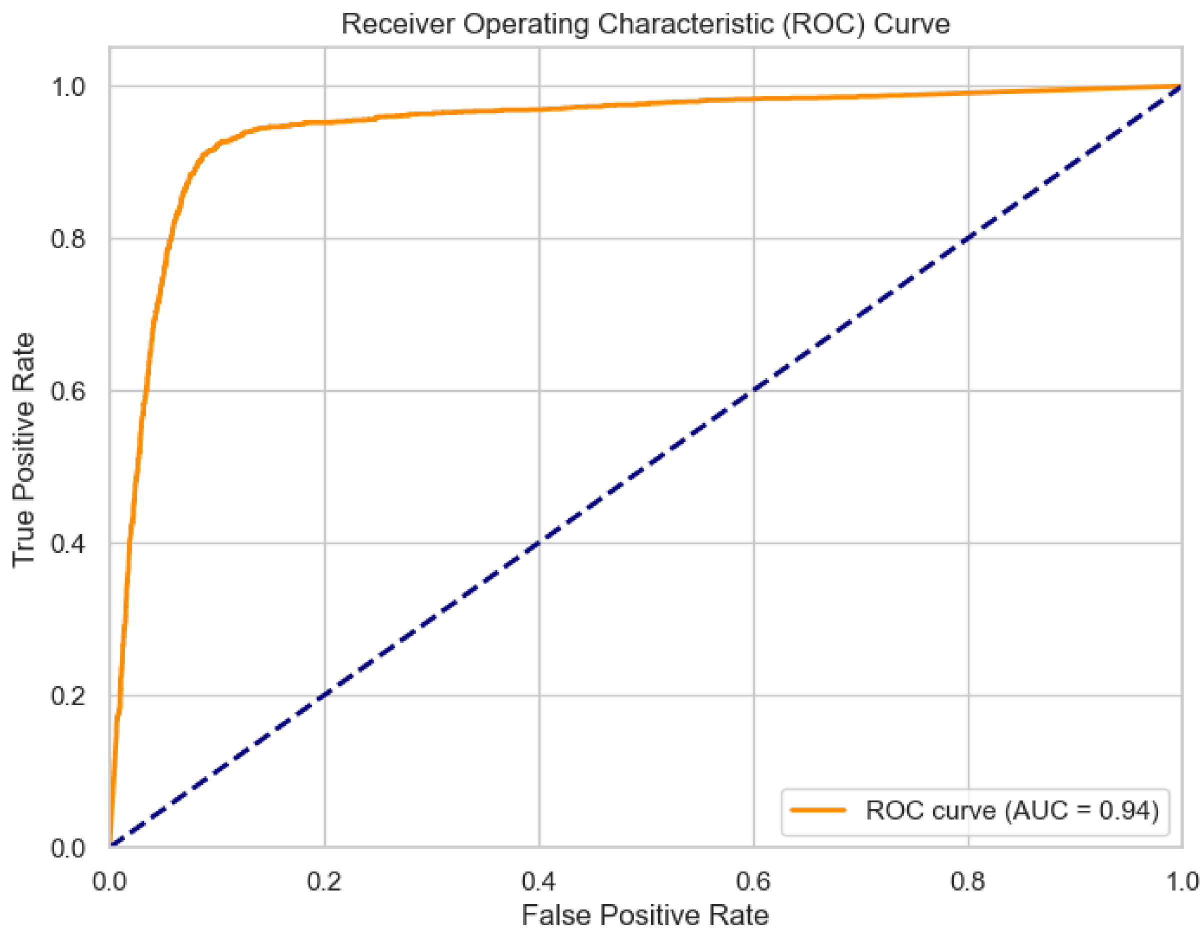
In [37]:

```
cm_optimized_pd= confusion_matrix(y_test, y_pred_opt_pd)
plot_confusion_matrix(cm_optimized_pd, classes=['Not Pedestrian', 'Pedestria
```



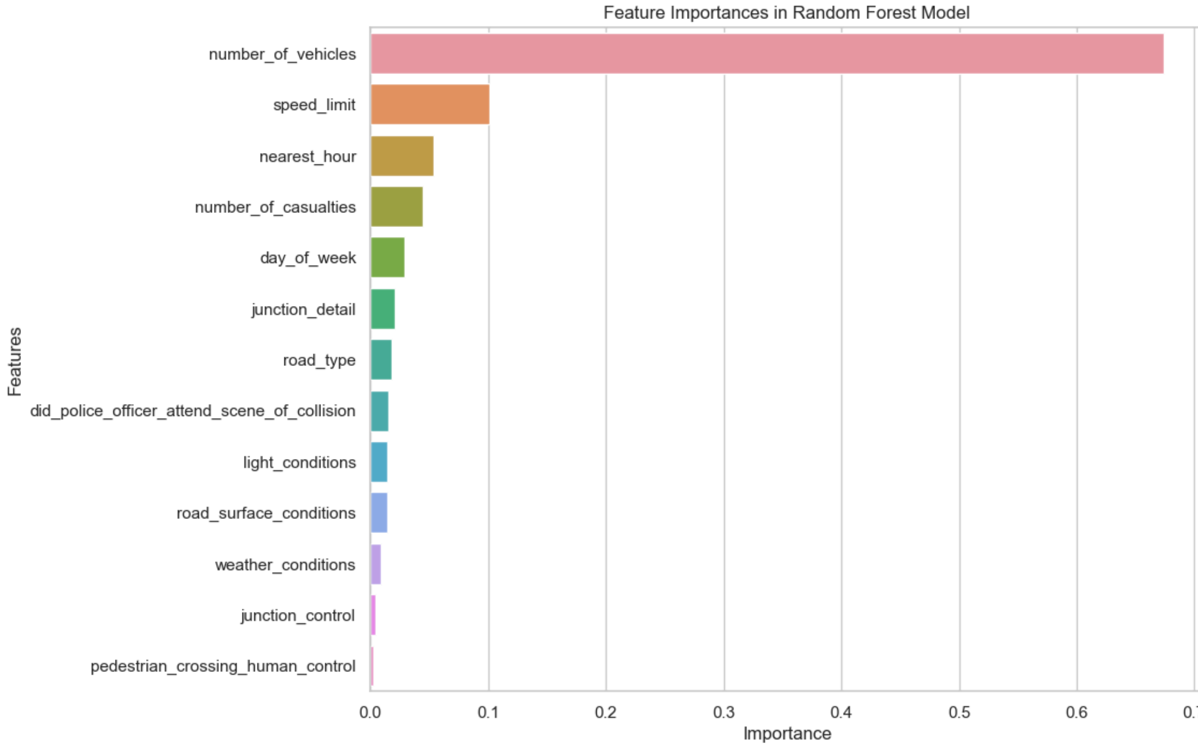
In [38]:

```
# Generate false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_opt_proba_pd)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"Optimized ROC AUC: {roc_auc_opt_pd:.5f}")
```



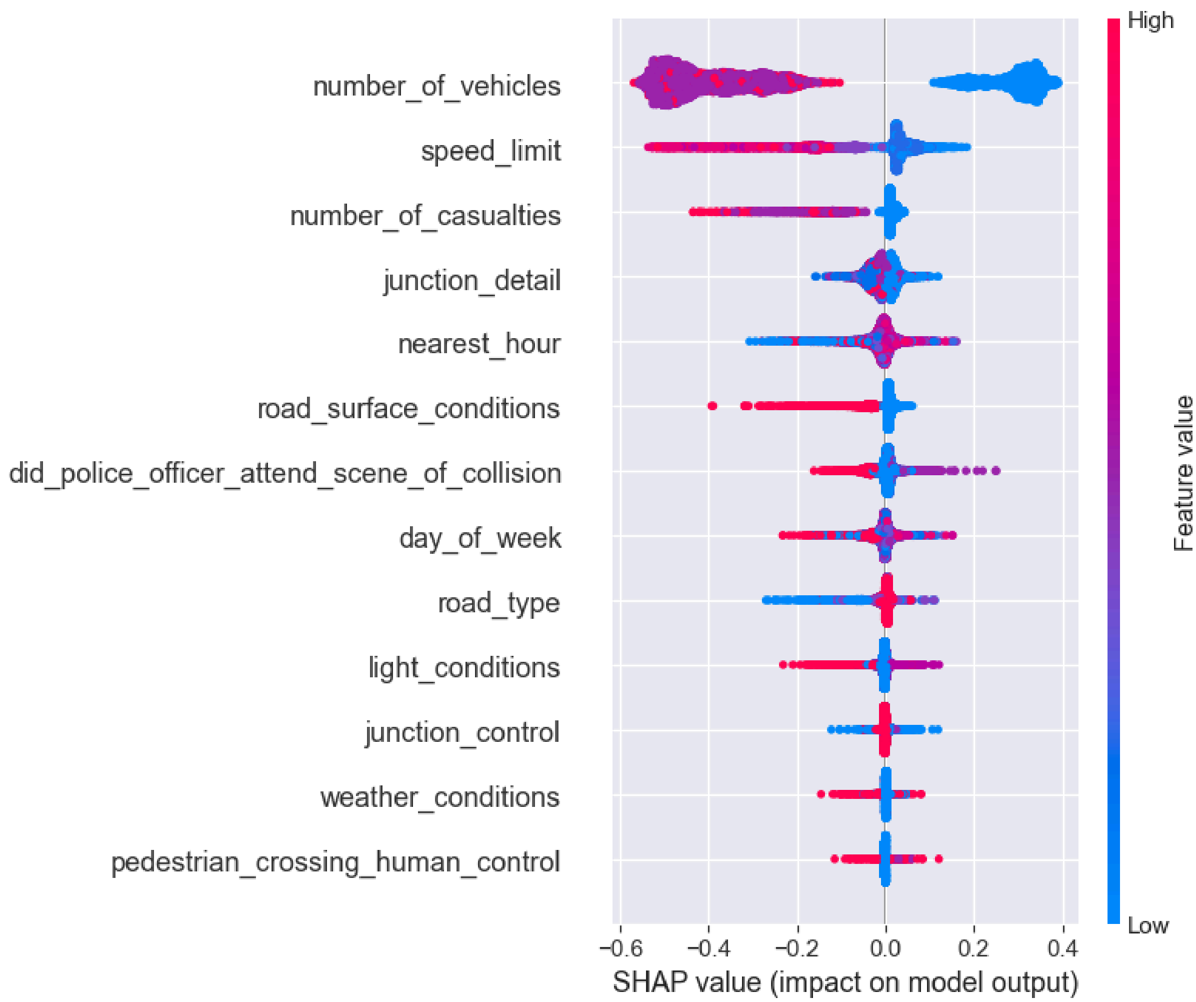

```
Optimized ROC AUC: 0.94396
```

In [39]:
```
best_rf = grid_search_pd.best_estimator_

# Extract feature importances
feature_importances = best_rf.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to hold feature names and their importance
importances_pd = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
})

# Sort the DataFrame by importance in descending order
importances_pd = importances_pd.sort_values(by='Importance', ascending=False

# Visualize the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_pd)
plt.title('Feature Importances in Random Forest Model')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



In [37]:

```
# Initialize the SHAP Explainer with check_additivity set to False
explainer_pd = shap.Explainer(pd_optimized)

# Compute SHAP values for the test set
shap_values_pd = explainer_pd.shap_values(X_test)

shap_values_positive_class_pd = shap_values_pd[:, :, 1]

shap.summary_plot(shap_values_positive_class_pd, X_test)
```



XGBoost

In [32]:
```
# Initialize and train the XGBoost classifier
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'
xgb_model.fit(X_train_smote, y_train_smote)
```

Out[32]:

▾ XGBClassifier

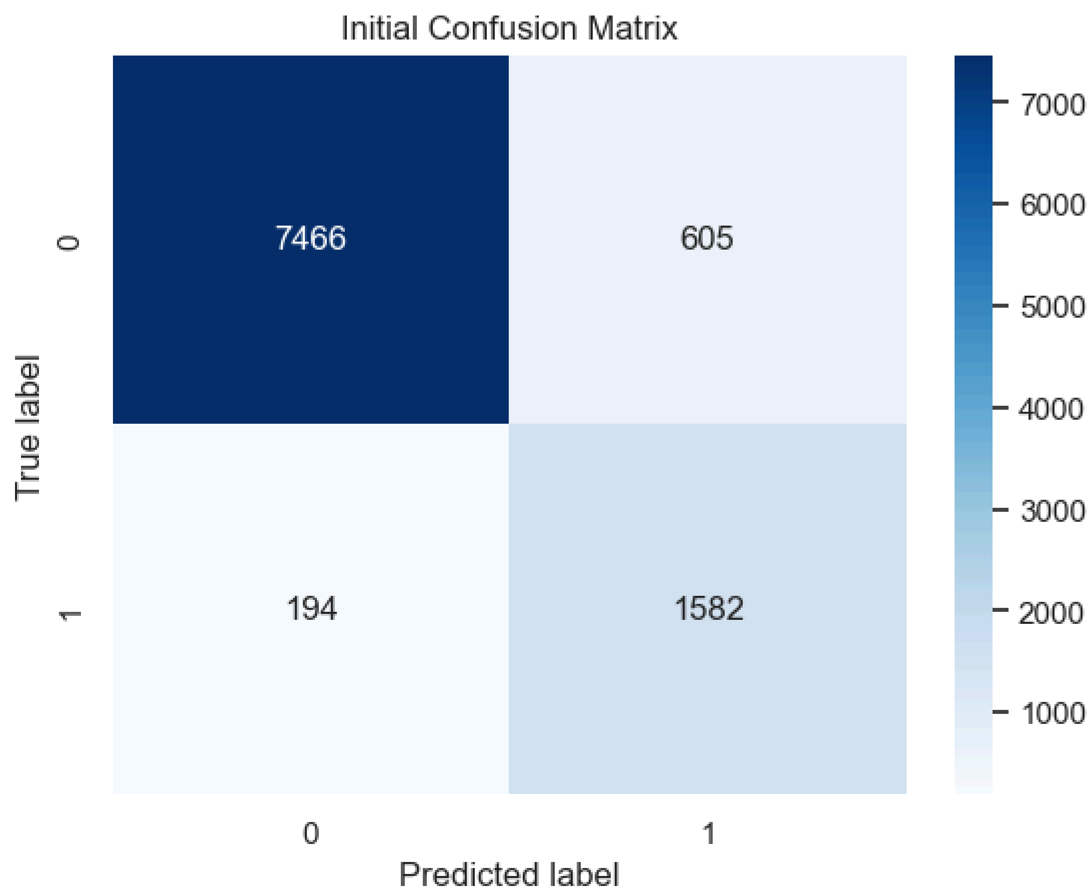
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_ro
unds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=N
one,
```

In [33]:
```
# Make predictions
y_pred_xg = xgb_model.predict(X_test)
y_pred_proba_xg = xgb_model.predict_proba(X_test)[:, 1]  # Probabilities for
```

```python
# Calculate metrics
accuracy_xg = accuracy_score(y_test, y_pred_xg)
roc_auc_xg = roc_auc_score(y_test, y_pred_proba_xg)
report_xg = classification_report(y_test, y_pred_xg)

print(f"Accuracy of Pedestrian before Tuning: {accuracy_xg:.5f}")
print(f"ROC AUC: {roc_auc_xg:.5f}")
print("\nClassification Report of Pedestrian before Tuning:\n", report_xg)
```

```
Accuracy of Pedestrian before Tuning: 0.91886
ROC AUC: 0.95274

Classification Report of Pedestrian before Tuning:
               precision    recall  f1-score   support

           0       0.97      0.93      0.95      8071
           1       0.72      0.89      0.80      1776

    accuracy                           0.92      9847
   macro avg       0.85      0.91      0.87      9847
weighted avg       0.93      0.92      0.92      9847
```

In [34]:
```python
cm_initial_xg= confusion_matrix(y_test, y_pred_xg)
plot_confusion_matrix(cm_initial_xg, classes=['Not Pedestrian', 'Pedestrian'
```



In [38]:
```python
param_grid_xg = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
```

```
    'n_estimators': [100, 200],
    'subsample': [0.8, 0.9, 1],
    'colsample_bytree': [0.3, 0.7],
    'gamma': [0, 0.1, 0.2]
}

xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'

grid_search_xg = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xg,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search_xg.fit(X_train_smote, y_train_smote)

print("Best parameters:", grid_search_xg.best_params_)
print("Best score: {:.5f}".format(grid_search_xg.best_score_))
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best parameters: {'colsample_bytree': 0.7, 'gamma': 0.2, 'learning_rate': 0.
2, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.9}
Best score: 0.93860
```

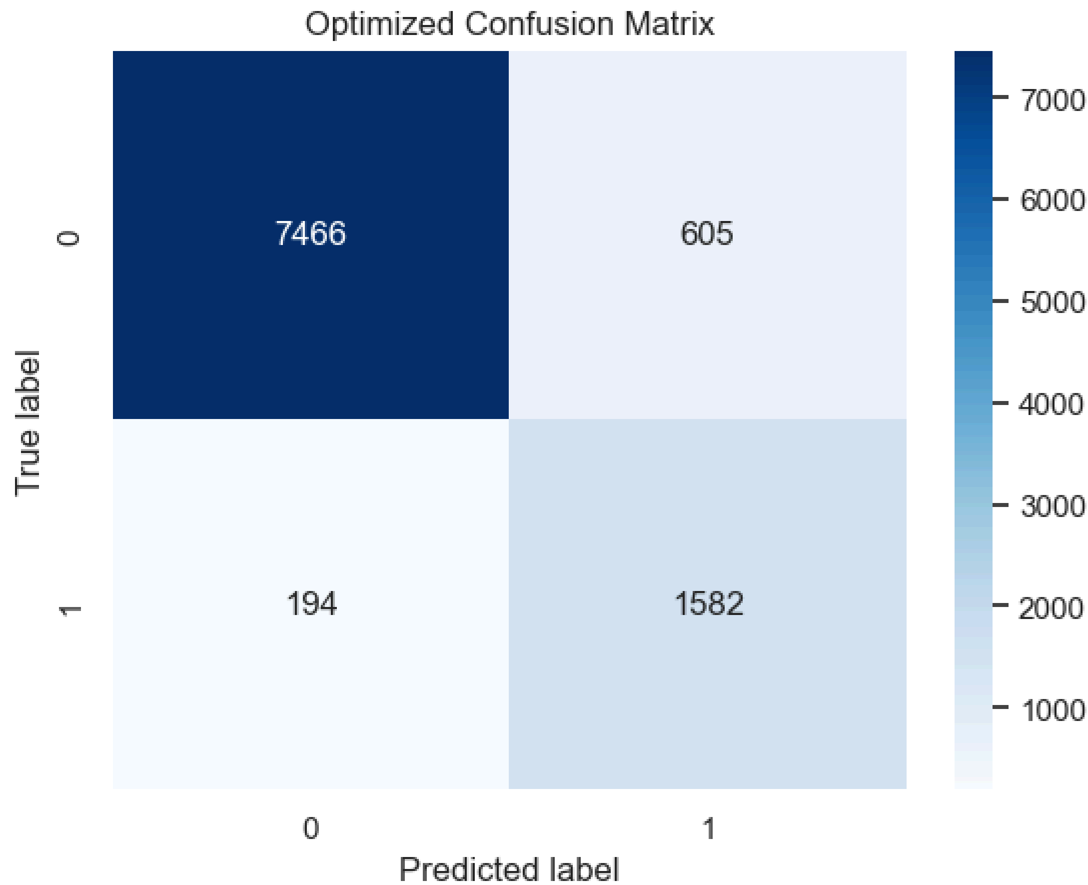
In [39]:

```
# Re-train the model using the best parameters from the correct grid search
xgb_optimized = xgb.XGBClassifier(**grid_search_xg.best_params_, use_label_e
xgb_optimized.fit(X_train_smote, y_train_smote)

# Re-evaluate the model
y_pred_opt_xg = xgb_optimized.predict(X_test)
y_pred_opt_proba_xg = xgb_optimized.predict_proba(X_test)[:, 1]
accuracy_opt_xg = accuracy_score(y_test, y_pred_opt_xg)
roc_auc_opt_xg = roc_auc_score(y_test, y_pred_opt_proba_xg)
report_opt_xg = classification_report(y_test, y_pred_opt_xg)

# Output the optimized accuracy and ROC AUC, along with the classification r
print(f"Optimized Accuracy: {accuracy_opt_xg:.5f}")
print("\n Optimized Classification Report:\n", report_opt_xg)
```

```
Optimized Accuracy: 0.92058

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.93      0.95      8071
           1       0.73      0.90      0.80      1776

    accuracy                           0.92      9847
   macro avg       0.85      0.91      0.88      9847
weighted avg       0.93      0.92      0.92      9847
```

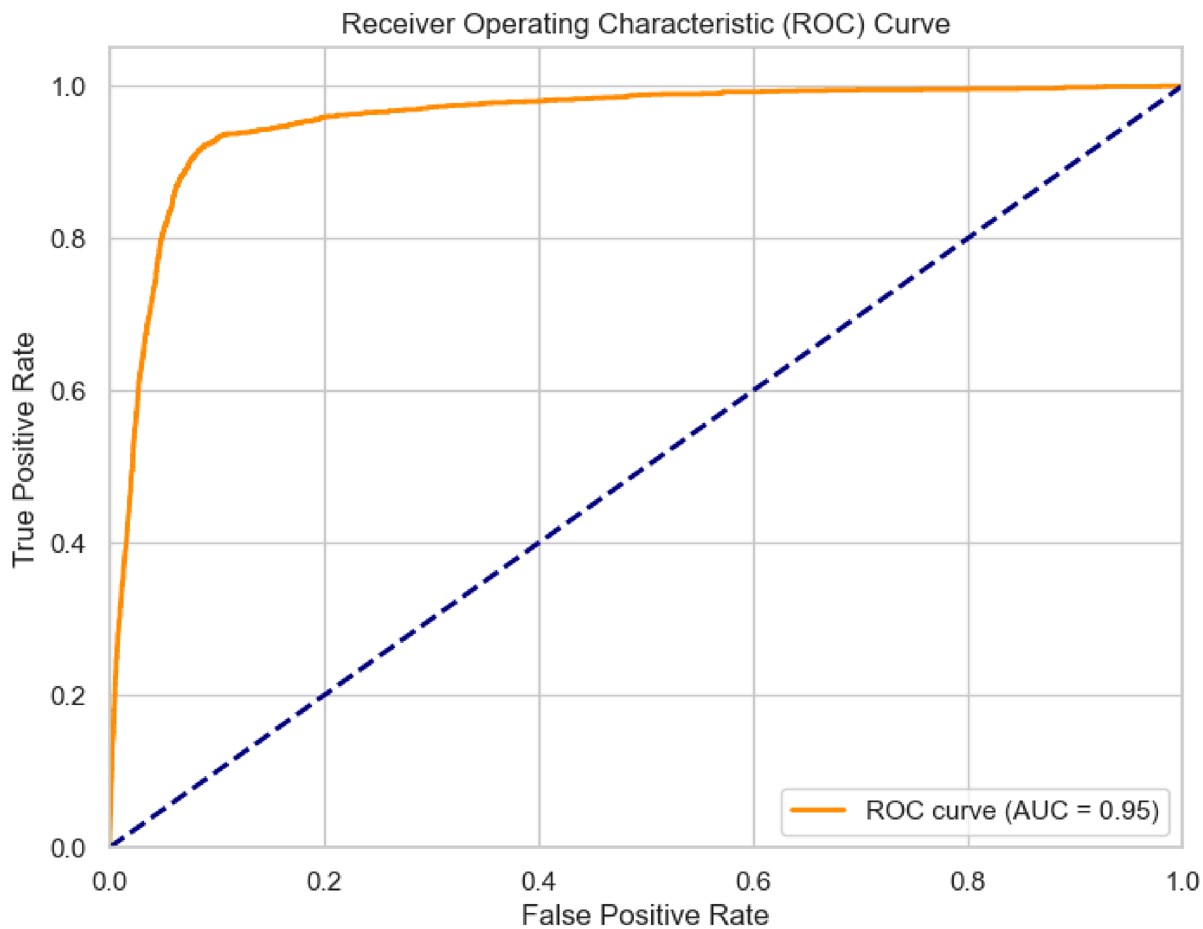
In [40]:

```
cm_optimized_xg= confusion_matrix(y_test, y_pred_xg)
plot_confusion_matrix(cm_optimized_xg, classes=['Not Pedestrian', 'Pedestria
```



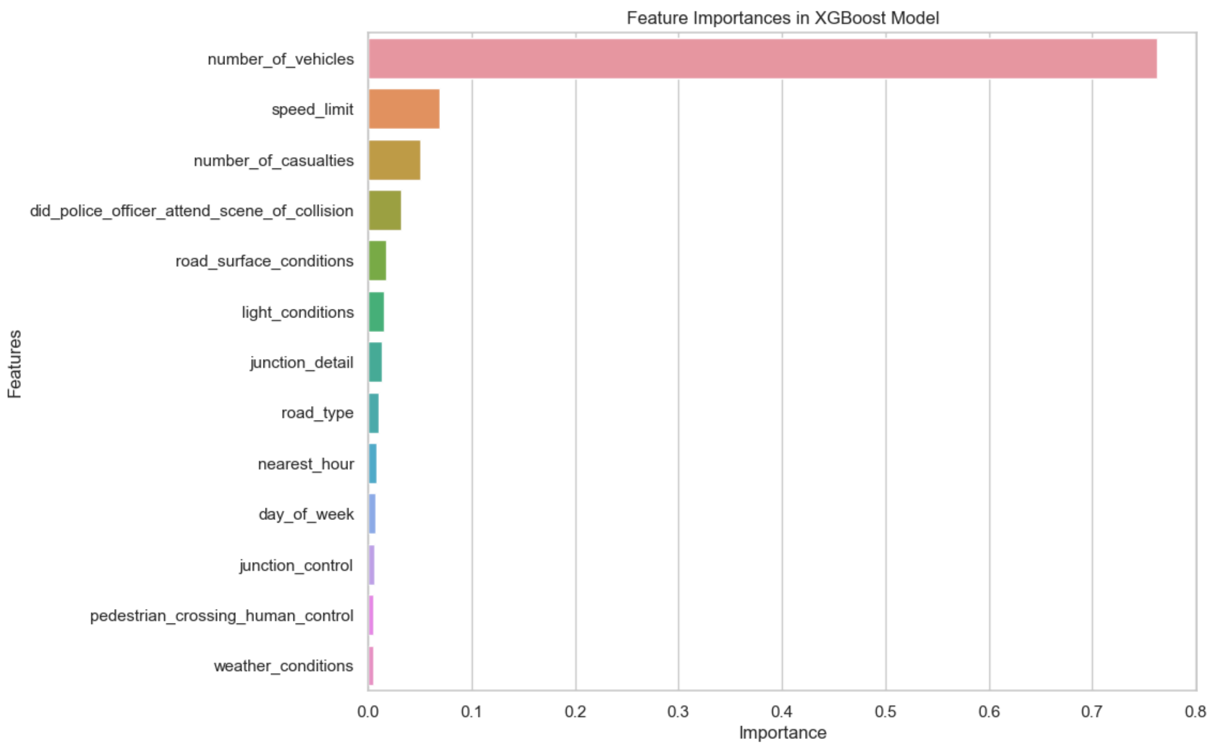
In [47]:

```
# Generate false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_opt_proba_xg)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"Optimized ROC AUC: {roc_auc_opt_xg:.5f}")
```

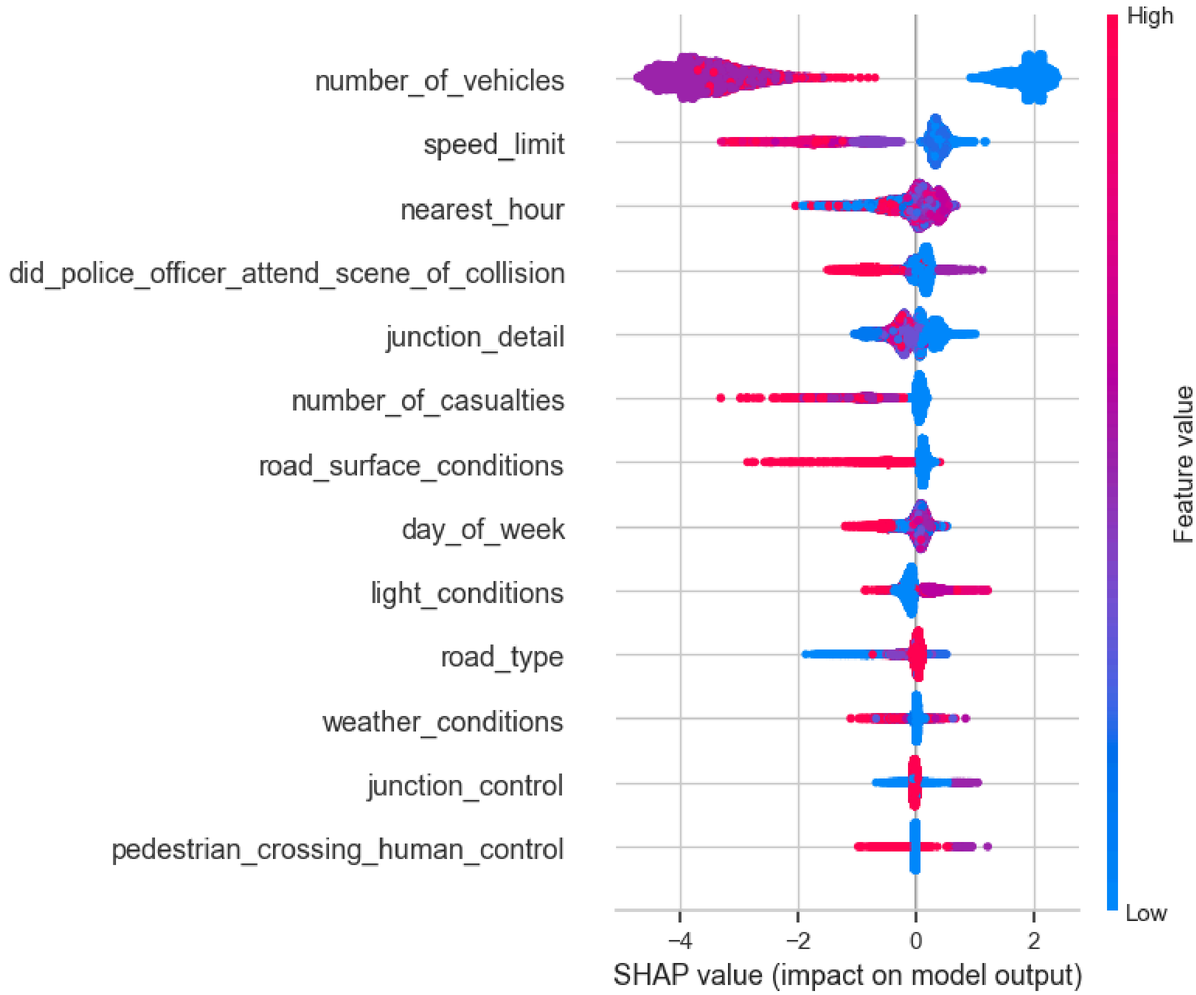



```
Optimized ROC AUC: 0.95497
```

In [41]:
```
# Extract feature importances
importances_xg = xgb_optimized.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to hold feature names and their importance
importances_xg = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances_xg
})

# Sort the DataFrame by importance in descending order
importances_xg = importances_xg.sort_values(by='Importance', ascending=False

# Visualize the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_xg)
plt.title('Feature Importances in XGBoost Model')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



In [49]:
```
# Initialize the SHAP Explainer with your model
explainer = shap.TreeExplainer(xgb_optimized)

# Compute SHAP values for the test set
shap_values = explainer.shap_values(X_test)

# For a detailed summary plot that shows the impact of the top features acro
shap.summary_plot(shap_values, X_test)
```



Model Training: Severity Level

In [20]:

```
# Set the target feature as 'RainTomorrow'
X = collision_data[['number_of_vehicles',
                    'number_of_casualties',
                    'day_of_week',
                    'nearest_hour',
                    'road_type',
                    'speed_limit',
                    'junction_control',
                    'junction_detail',
                    'pedestrian_crossing_human_control',
                    'light_conditions',
                    'weather_conditions',
                    'road_surface_conditions',
                    'did_police_officer_attend_scene_of_collision']]
y = collision_data['casualty_over_serious']


# Perform an 80-20 training-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Address class imbalance in the training set using SMOTE
```

```
print('Original dataset shape %s' % Counter(y_train))

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check whether the imbalance issue has been addressed
print('Resampled dataset shape %s' % Counter(y_train_smote))
```

```
Original dataset shape Counter({0: 30150, 1: 9234})
Resampled dataset shape Counter({0: 30150, 1: 30150})
```

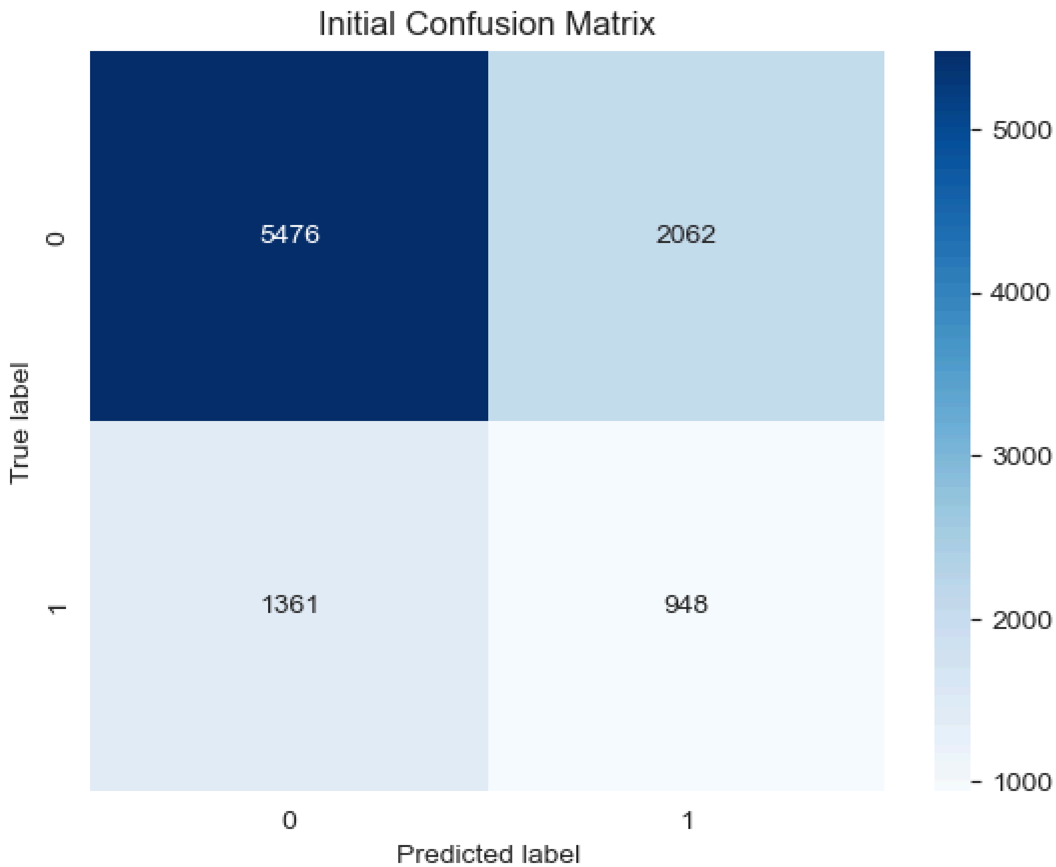
In [21]:
```
# Initialize the Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Train the model
rf.fit(X_train_smote, y_train_smote)
```

Out[21]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [22]:
```
# Make predictions
y_pred_rf = rf.predict(X_test)
y_pred_proba_rf = rf.predict_proba(X_test)[:, 1]  # Probabilities for ROC AU

# Calculate metrics
accuracy_rf = accuracy_score(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, y_pred_proba_rf)
report_rf = classification_report(y_test, y_pred_rf)

print(f"ROC AUC: {roc_auc_rf:.5f}")
print(f"Accuracy before Tuning of Severity Level: {accuracy_rf:.5f}")
print("\n Classification Report before Tuning of Severity Level:\n", report_
```

```
ROC AUC: 0.62095
Accuracy before Tuning of Severity Level: 0.65238

 Classification Report before Tuning of Severity Level:
               precision    recall  f1-score   support

           0       0.80      0.73      0.76      7538
           1       0.31      0.41      0.36      2309

    accuracy                           0.65      9847
   macro avg       0.56      0.57      0.56      9847
weighted avg       0.69      0.65      0.67      9847
```

In [27]:
```
cm_initial_rf= confusion_matrix(y_test, y_pred_rf)
plot_confusion_matrix(cm_initial_rf, classes=['Not Serious', 'Serious'], tit
```



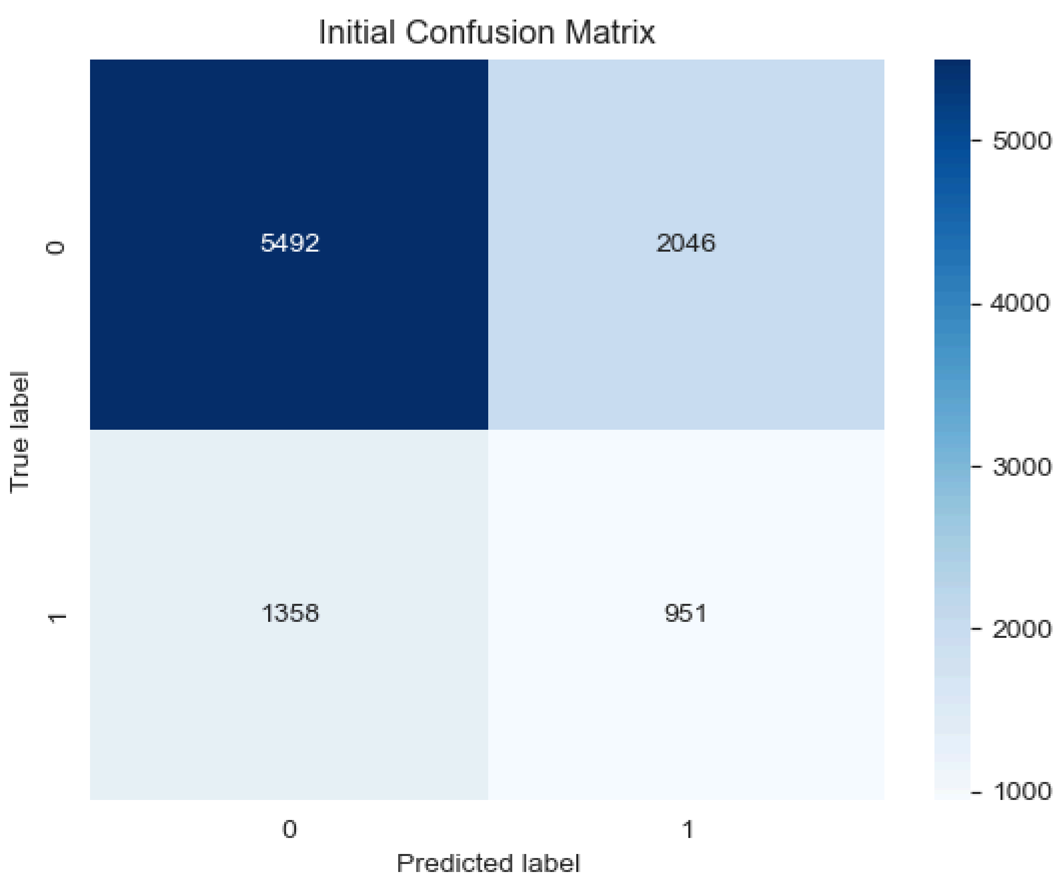
In [23]:
```
# Define the parameter grid
param_grid_rf = {
    'n_estimators': [50,100,200],
    'max_depth': [None, 1,5,10],
    'min_samples_leaf': [1,2,10],
    'min_samples_split': [2,5,10]
}

clf_rf = RandomForestClassifier(random_state=42)

# Set up Grid Search CV
grid_search_rf = GridSearchCV(estimator=clf_rf,
                              param_grid=param_grid_rf,
                              cv=5,
                              scoring='accuracy',
                              n_jobs=-1,
                              verbose=1)

# Perform grid search
grid_search_rf.fit(X_train_smote, y_train_smote)

# Output the best parameters and the corresponding score
print("Best parameters:", grid_search_rf.best_params_)
print("Best score: {:.5f}".format(grid_search_rf.best_score_))
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_spl
it': 2, 'n_estimators': 200}
Best score: 0.75784
```

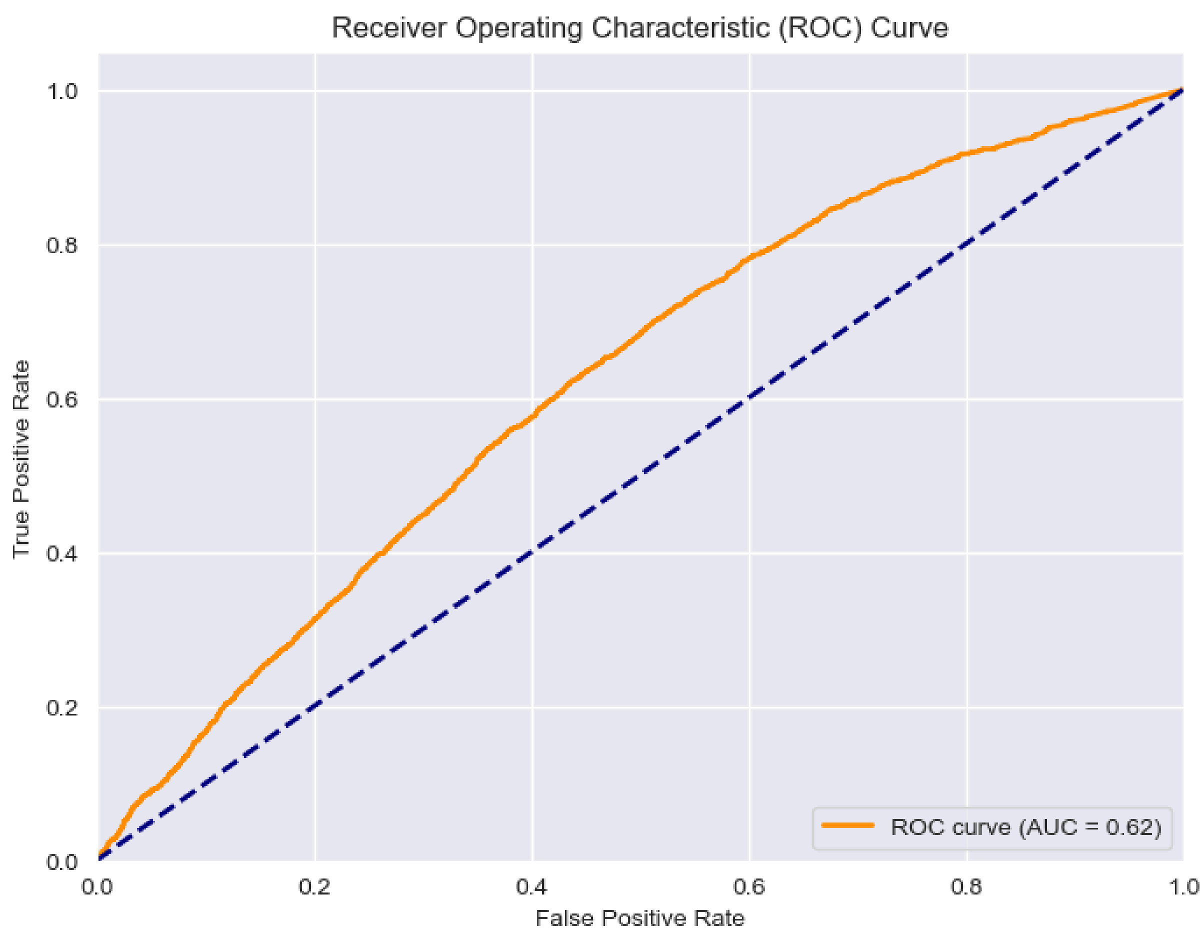
In [24]:
```
# Re-train the model using the best parameters
rf_optimized = RandomForestClassifier(**grid_search_rf.best_params_, random_
rf_optimized.fit(X_train_smote, y_train_smote)
# Re-evaluate the model
y_pred_opt_rf = rf_optimized.predict(X_test)
y_pred_opt_proba_rf = rf_optimized.predict_proba(X_test)[:, 1]
accuracy_opt_rf= accuracy_score(y_test, y_pred_opt_rf)
roc_auc_opt_rf = roc_auc_score(y_test, y_pred_opt_proba_rf)
report_opt_rf = classification_report(y_test, y_pred_opt_rf)

print(f"Optimized Accuracy: {accuracy_opt_rf:.5f}")
print("\n Optimized Classification Report:\n", report_opt_rf)
```

```
Optimized Accuracy: 0.65431

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.73      0.76      7538
           1       0.32      0.41      0.36      2309

    accuracy                           0.65      9847
   macro avg       0.56      0.57      0.56      9847
weighted avg       0.69      0.65      0.67      9847
```

In [25]:
```
cm_optimized_rf= confusion_matrix(y_test, y_pred_opt_rf)
plot_confusion_matrix(cm_optimized_rf, classes=['Not Serious', 'Serious'], t
```



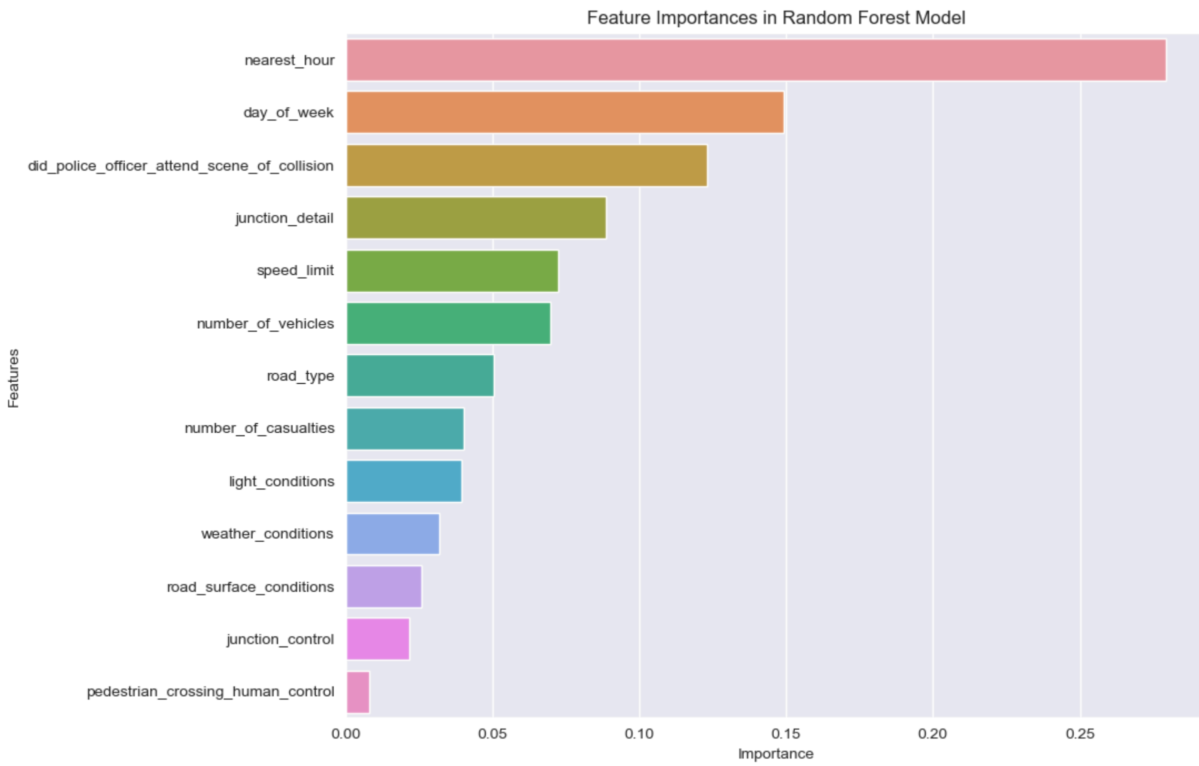
In [26]:
```
# Generate false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_opt_proba_rf)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"Optimized ROC AUC: {roc_auc_opt_rf:.5f}")
```




```
Optimized ROC AUC: 0.62136
```

In [27]:
```
best_rf = grid_search_rf.best_estimator_

# Extract feature importances
feature_importances = best_rf.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to hold feature names and their importance
importances_rf = pd.DataFrame({
    'Feature': feature_names,
```

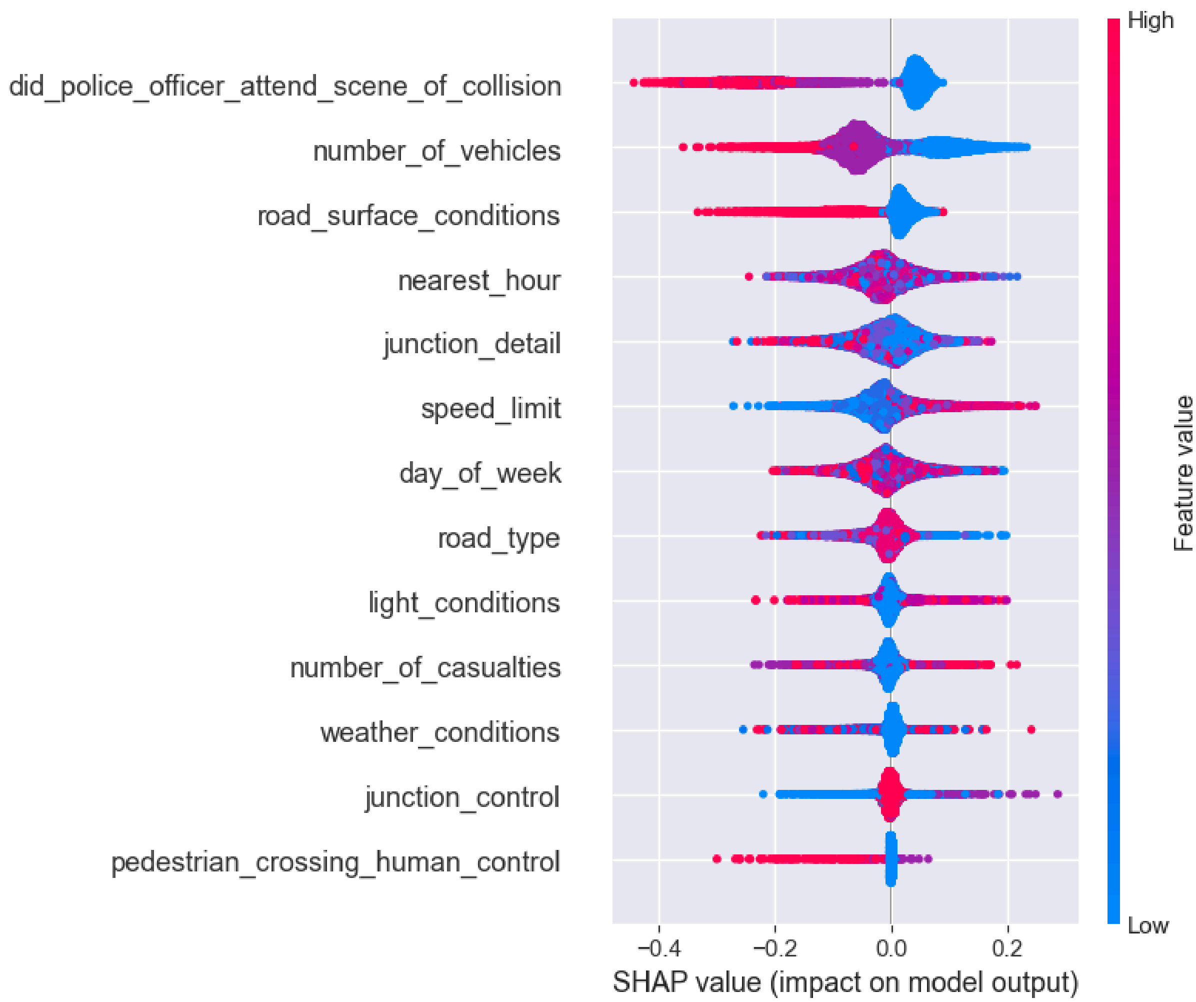
```
    'Importance': feature_importances
})

# Sort the DataFrame by importance in descending order
importances_rf = importances_rf.sort_values(by='Importance', ascending=False

# Visualize the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_rf)
plt.title('Feature Importances in Random Forest Model')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



In [28]:

```
# Initialize the SHAP Explainer using a lambda function to wrap the predict_
explainer_rf = shap.Explainer(lambda x: rf_optimized.predict_proba(x), X_tra

# Compute SHAP values for the test set
shap_values_rf = explainer_rf(X_test)

# Since Random Forest is a binary classifier in this case, shap_values will
shap_values_positive_class_rf = shap_values_rf[..., 1]

# Plotting the SHAP summary plot for the positive class
shap.summary_plot(shap_values_positive_class_rf, X_test)
```

```
PermutationExplainer explainer: 9848it [2:41:57,  1.01it/s]
```



In [29]:

```
# Initialize and train the XGBoost classifier
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'
xgb_model.fit(X_train_smote, y_train_smote)
```

Out[29]:

```
▾                              XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_ro
unds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=N
one,
```

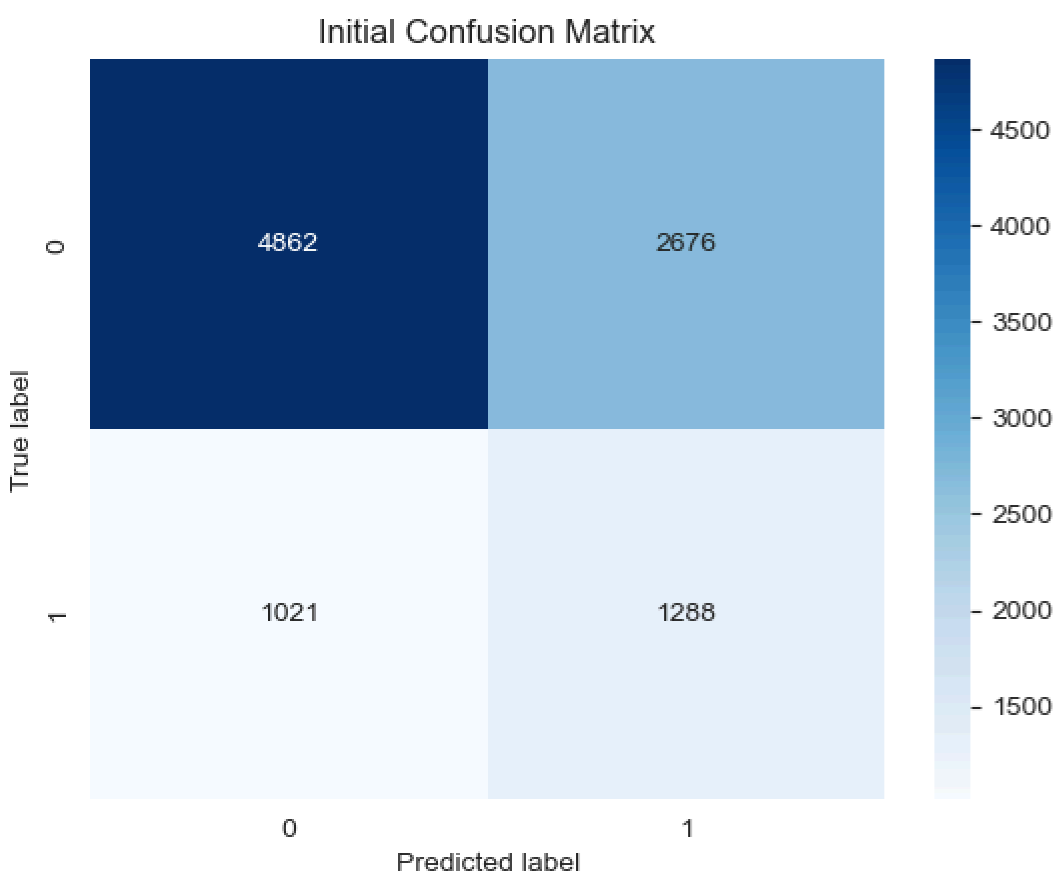
In [30]:

```
# Make predictions
y_pred_xg = xgb_model.predict(X_test)
y_pred_proba_xg = xgb_model.predict_proba(X_test)[:, 1]  # Probabilities for

# Calculate metrics
accuracy_xg = accuracy_score(y_test, y_pred_xg)
```

```
roc_auc_xg = roc_auc_score(y_test, y_pred_proba_xg)
report_xg = classification_report(y_test, y_pred_xg)

print(f"Accuracy before Tuning: {accuracy_xg:.5f}")
print(f"ROC AUC: {roc_auc_xg:.5f}")
print("\nClassification Report before Tuning:\n", report_xg)
```

```
Accuracy before Tuning: 0.62456
ROC AUC: 0.65475

Classification Report before Tuning:
               precision    recall  f1-score   support

           0       0.83      0.64      0.72      7538
           1       0.32      0.56      0.41      2309

    accuracy                           0.62      9847
   macro avg       0.58      0.60      0.57      9847
weighted avg       0.71      0.62      0.65      9847
```

In [36]:
```
cm_initial_xg= confusion_matrix(y_test, y_pred_xg)
plot_confusion_matrix(cm_initial_xg, classes=['Not Pedestrian', 'Pedestrian'
```



In [32]:
```
param_grid_xg = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200],
    'subsample': [0.8, 0.9, 1],
    'colsample_bytree': [0.3, 0.7],
```

```
    'gamma': [0, 0.1, 0.2]
}

xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'

grid_search_xg = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xg,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search_xg.fit(X_train_smote, y_train_smote)

print("Best parameters:", grid_search_xg.best_params_)
print("Best score: {:.5f}".format(grid_search_xg.best_score_))
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best parameters: {'colsample_bytree': 0.7, 'gamma': 0.2, 'learning_rate': 0.
2, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
Best score: 0.70680
```

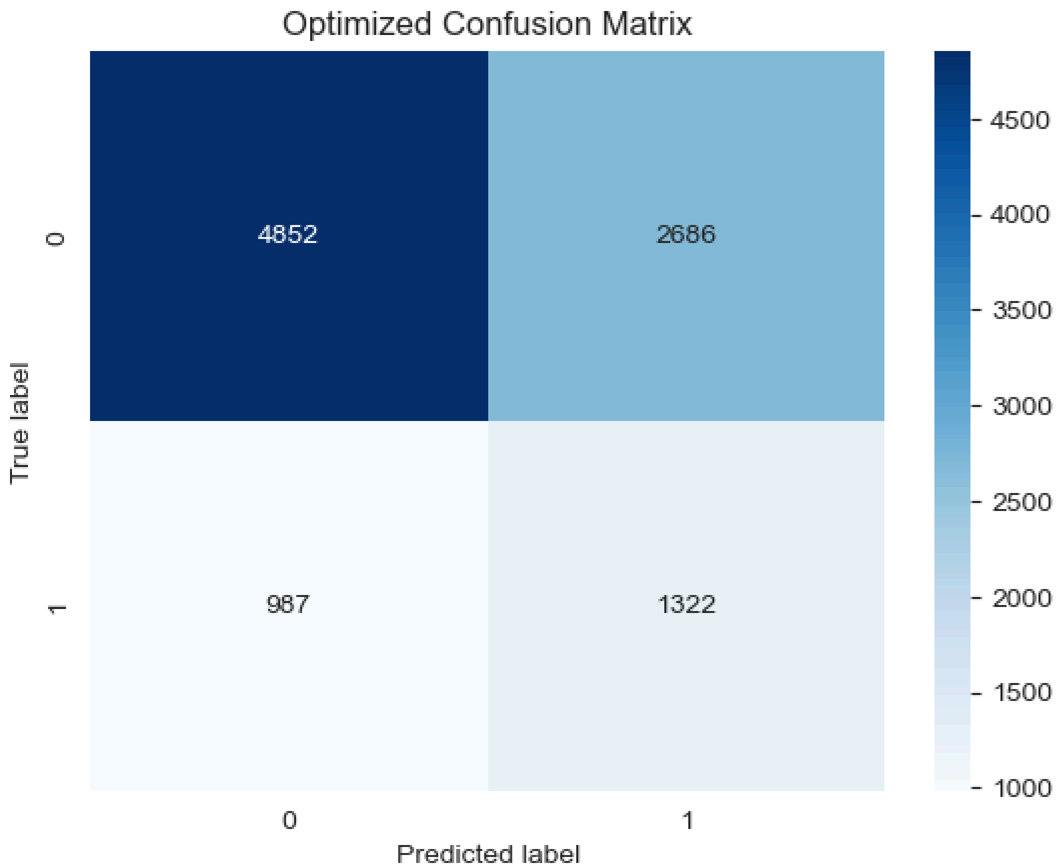
In [33]:

```
# Re-train the model using the best parameters from the correct grid search
xgb_optimized = xgb.XGBClassifier(**grid_search_xg.best_params_, use_label_e
xgb_optimized.fit(X_train_smote, y_train_smote)

# Re-evaluate the model
y_pred_opt_xg = xgb_optimized.predict(X_test)
y_pred_opt_proba_xg = xgb_optimized.predict_proba(X_test)[:, 1]
accuracy_opt_xg = accuracy_score(y_test, y_pred_opt_xg)
roc_auc_opt_xg = roc_auc_score(y_test, y_pred_opt_proba_xg)
report_opt_xg = classification_report(y_test, y_pred_opt_xg)

# Output the optimized accuracy and ROC AUC, along with the classification r
print(f"Optimized Accuracy: {accuracy_opt_xg:.5f}")
print("\n Optimized Classification Report:\n", report_opt_xg)
```

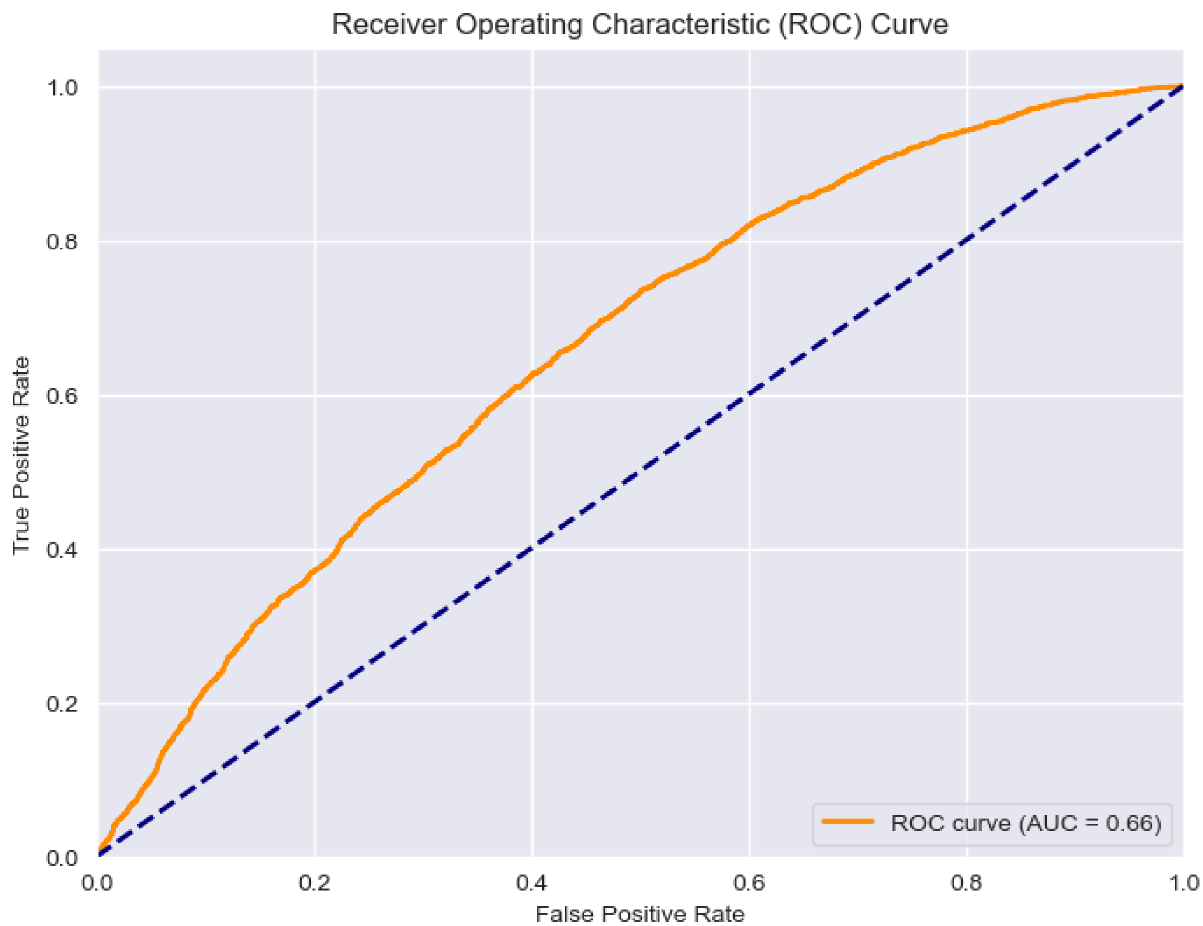
```
Optimized Accuracy: 0.62699

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.64      0.73      7538
           1       0.33      0.57      0.42      2309

    accuracy                           0.63      9847
   macro avg       0.58      0.61      0.57      9847
weighted avg       0.71      0.63      0.65      9847
```

In [35]:

```
cm_optimized_xg= confusion_matrix(y_test, y_pred_opt_xg)
plot_confusion_matrix(cm_optimized_xg, classes=['Not Serious', 'Serious'], t
```



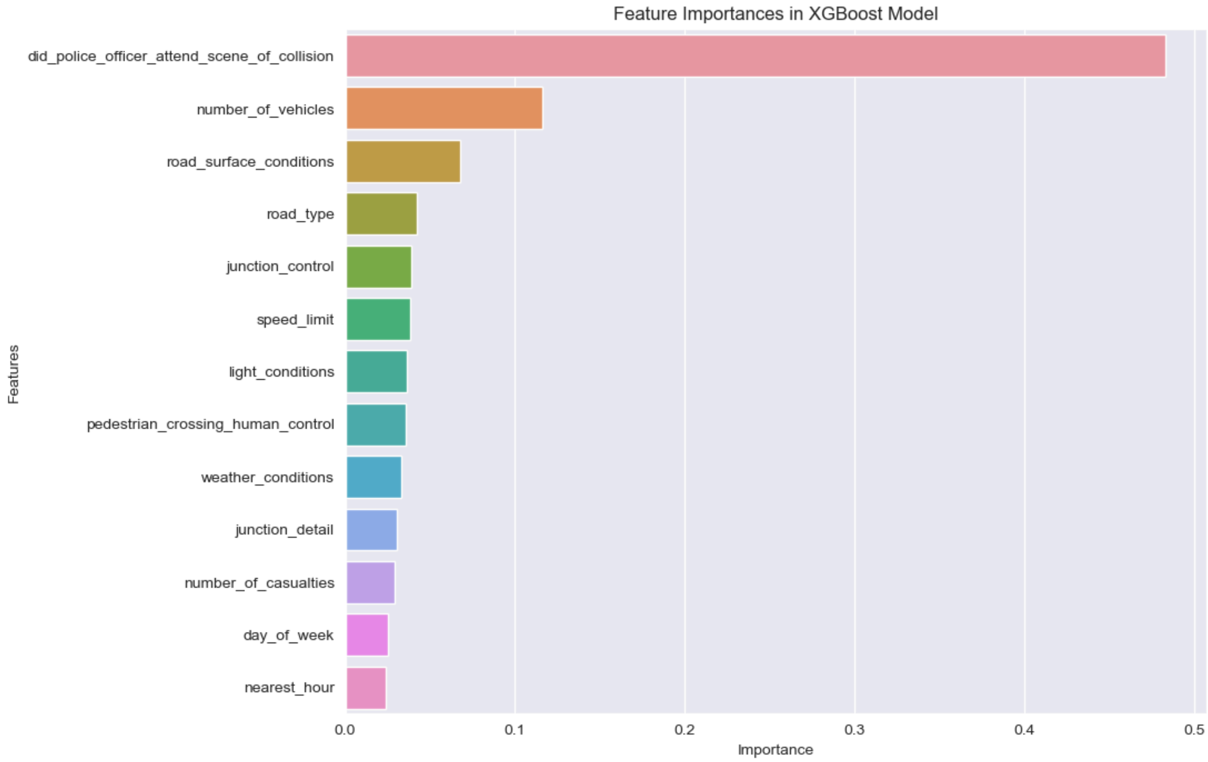
In [37]:
```python
# Generate false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_opt_proba_xg)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"Optimized ROC AUC: {roc_auc_opt_xg:.5f}")
```



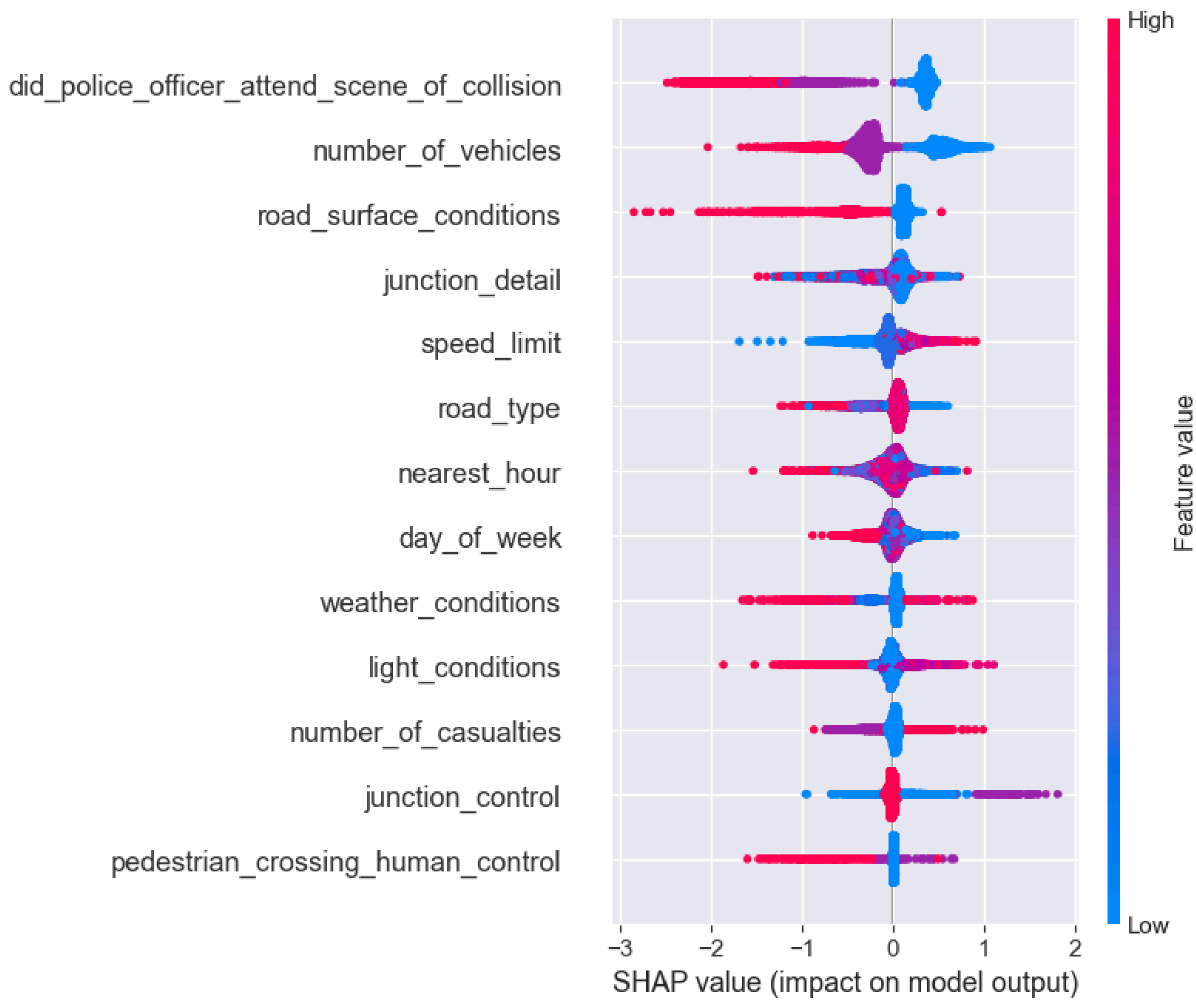

```
Optimized ROC AUC: 0.65843
```

In [38]:

```
# Extract feature importances
importances_xg = xgb_optimized.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to hold feature names and their importance
importances_xg = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances_xg
})

# Sort the DataFrame by importance in descending order
importances_xg = importances_xg.sort_values(by='Importance', ascending=False

# Visualize the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_xg)
plt.title('Feature Importances in XGBoost Model')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



In [39]:
```
# Initialize the SHAP Explainer with your model
explainer = shap.TreeExplainer(xgb_optimized)

# Compute SHAP values for the test set
shap_values = explainer.shap_values(X_test)

# For a detailed summary plot that shows the impact of the top features acro
shap.summary_plot(shap_values, X_test)
```



Model Training: Serious Pedestrian Case Focus

In [18]:

```
# Set the target feature as 'RainTomorrow'
X = collision_data[['number_of_vehicles',
                    'number_of_casualties',
                    'day_of_week',
                    'nearest_hour',
                    'road_type',
                    'speed_limit',
                    'junction_control',
                    'junction_detail',
                    'pedestrian_crossing_human_control',
                    'light_conditions',
                    'weather_conditions',
                    'road_surface_conditions',
                    'did_police_officer_attend_scene_of_collision']]
y = collision_data['pedestrian_over_serious']


# Perform an 80-20 training-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Address class imbalance in the training set using SMOTE
```

```
print('Original dataset shape %s' % Counter(y_train))

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check whether the imbalance issue has been addressed
print('Resampled dataset shape %s' % Counter(y_train_smote))
```

```
Original dataset shape Counter({0: 37126, 1: 2258})
Resampled dataset shape Counter({0: 37126, 1: 37126})
```

In [15]:

```
# Initialize the Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Train the model
rf.fit(X_train_smote, y_train_smote)
```

Out[15]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

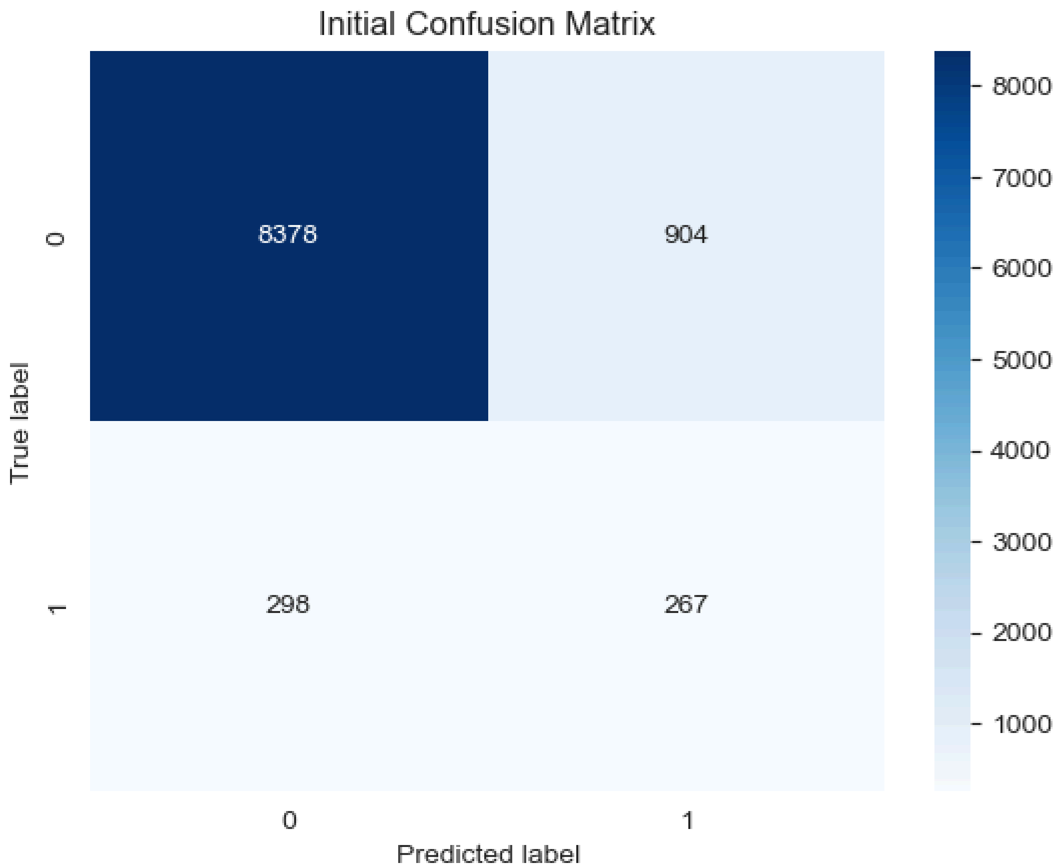
In [16]:

```
# Make predictions
y_pred_rf = rf.predict(X_test)
y_pred_proba_rf = rf.predict_proba(X_test)[:, 1]  # Probabilities for ROC AU

# Calculate metrics
accuracy_rf = accuracy_score(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, y_pred_proba_rf)
report_rf = classification_report(y_test, y_pred_rf)

print(f"ROC AUC: {roc_auc_rf:.5f}")
print(f"Accuracy before Tuning: {accuracy_rf:.5f}")
print("\n Classification Report before Tuning:\n", report_rf)
```

```
ROC AUC: 0.84862
Accuracy before Tuning: 0.87793

 Classification Report before Tuning:
               precision    recall  f1-score   support

           0       0.97      0.90      0.93      9282
           1       0.23      0.47      0.31       565

    accuracy                           0.88      9847
   macro avg       0.60      0.69      0.62      9847
weighted avg       0.92      0.88      0.90      9847
```

In [23]:

```
cm_initial_rf= confusion_matrix(y_test, y_pred_rf)
plot_confusion_matrix(cm_initial_rf, classes=['Not Serious Pedestrian Case',
```



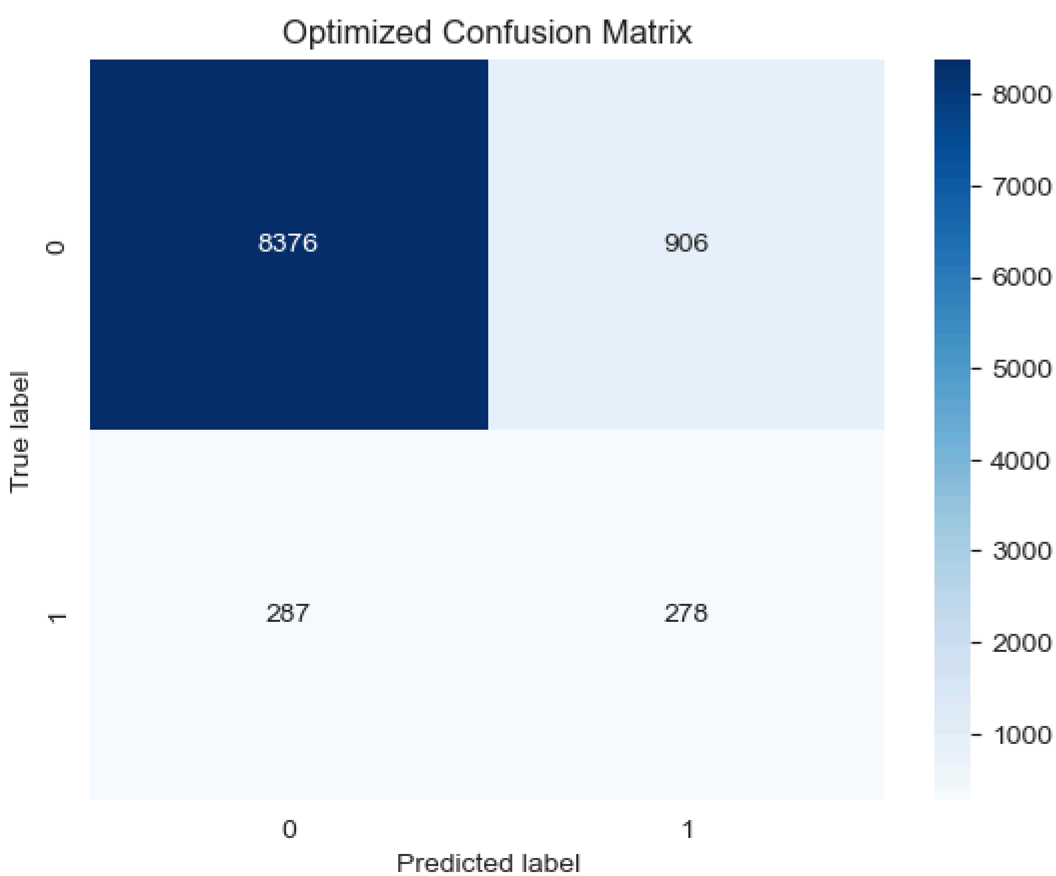
In [18]:
```
# Define the parameter grid
param_grid_cb = {
    'n_estimators': [50,100,200,300],
    'max_depth': [None, 1,5,10],
    'min_samples_leaf': [1,2,10],
    'min_samples_split': [2,5,10]
}

clf_cb = RandomForestClassifier(random_state=42)

# Set up Grid Search CV
grid_search_cb = GridSearchCV(estimator=clf_cb,
                              param_grid=param_grid_cb,
                              cv=5,
                              scoring='accuracy',
                              n_jobs=-1,
                              verbose=1)

# Perform grid search
grid_search_cb.fit(X_train_smote, y_train_smote)

# Output the best parameters and the corresponding score
print("Best parameters:", grid_search_cb.best_params_)
print("Best score: {:.5f}".format(grid_search_cb.best_score_))
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_spl
it': 5, 'n_estimators': 300}
Best score: 0.93228
```

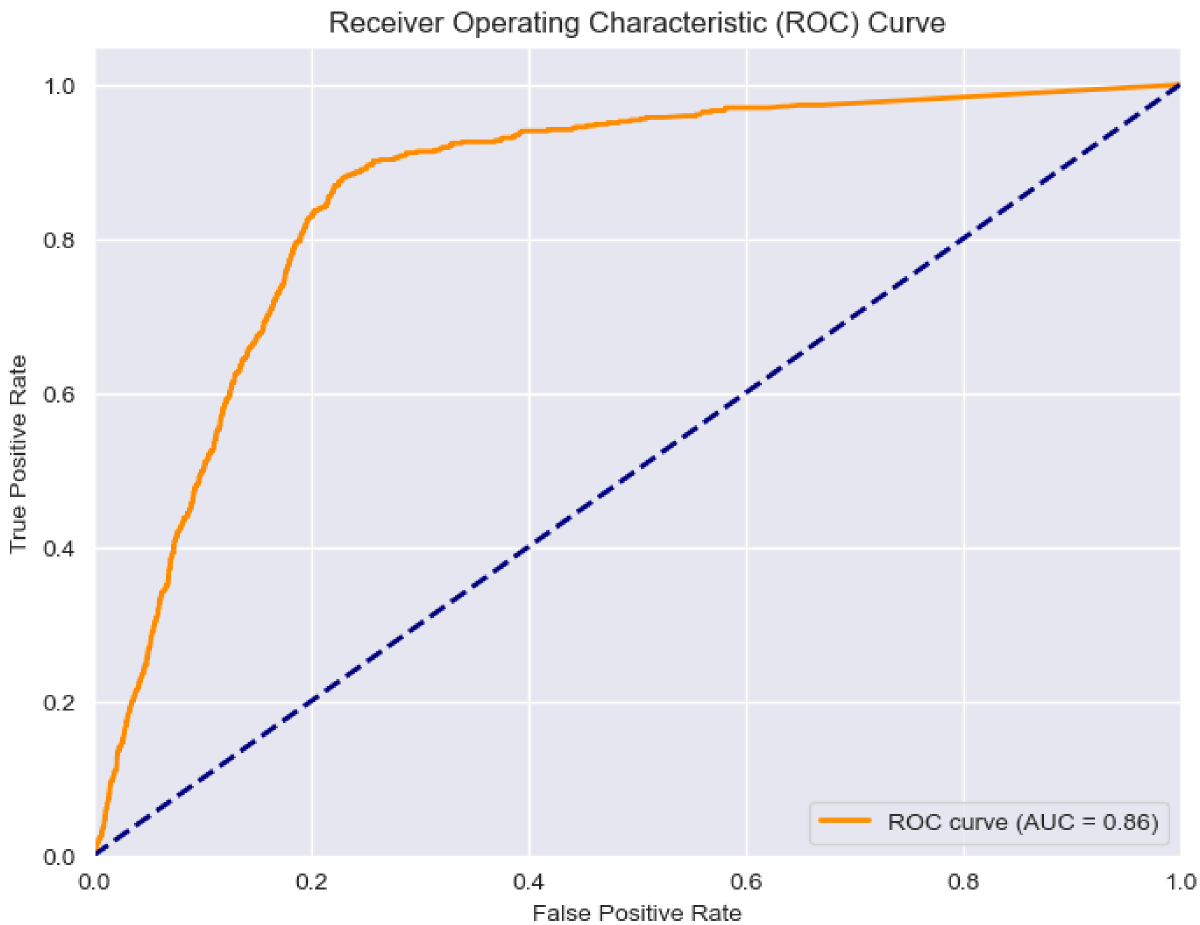
In [21]:
```
# Re-train the model using the best parameters
cb_optimized = RandomForestClassifier(**grid_search_cb.best_params_, random_
cb_optimized.fit(X_train_smote, y_train_smote)

# Re-evaluate the model
y_pred_opt_cb = cb_optimized.predict(X_test)
y_pred_opt_proba_cb = cb_optimized.predict_proba(X_test)[:, 1]
accuracy_opt_cb = accuracy_score(y_test, y_pred_opt_cb)
roc_auc_opt_cb = roc_auc_score(y_test, y_pred_opt_proba_cb)
report_opt_cb = classification_report(y_test, y_pred_opt_cb)

print(f"Optimized Accuracy: {accuracy_opt_cb:.5f}")
print("\n Optimized Classification Report:\n", report_opt_cb)
```

```
Optimized Accuracy: 0.87885

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.90      0.93      9282
           1       0.23      0.49      0.32       565

    accuracy                           0.88      9847
   macro avg       0.60      0.70      0.63      9847
weighted avg       0.92      0.88      0.90      9847
```

In [24]:
```
cm_optimized_cb= confusion_matrix(y_test, y_pred_opt_cb)
plot_confusion_matrix(cm_optimized_cb, classes=['Not Serious Pedestrian Case
```



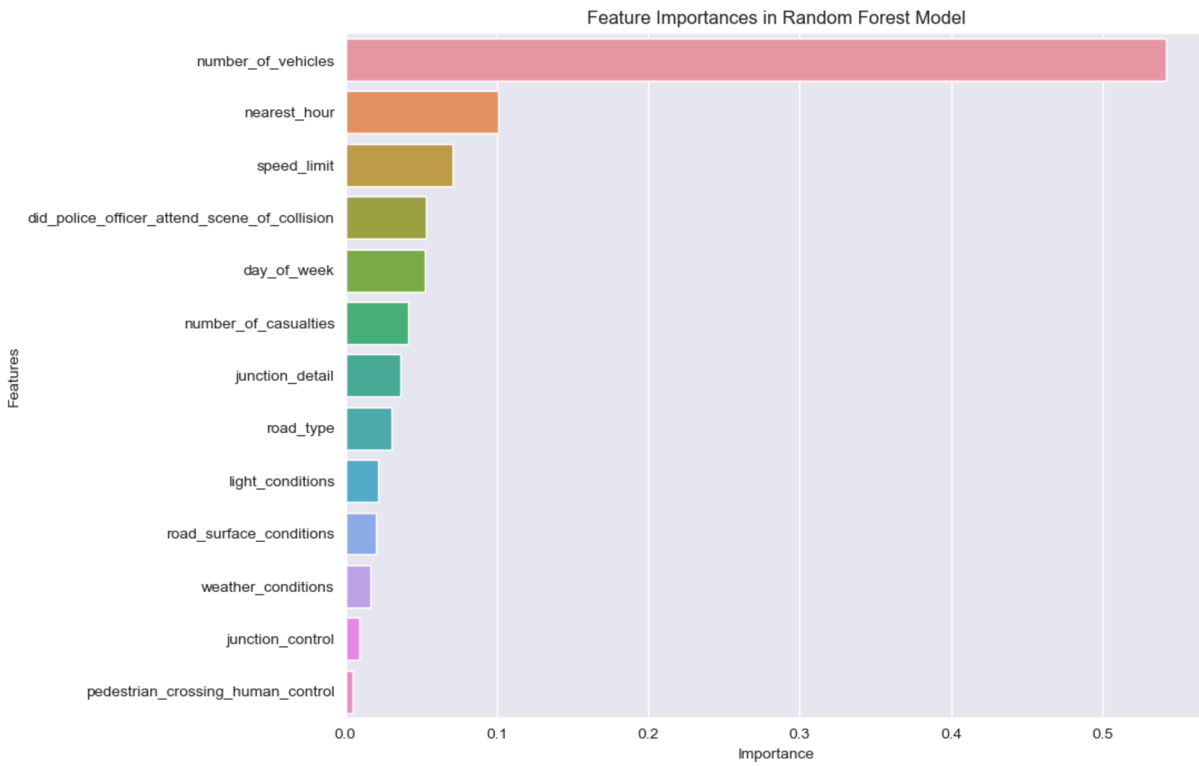
In [25]:
```
# Generate false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_opt_proba_cb)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"Optimized ROC AUC: {roc_auc_opt_cb:.5f}")
```




```
Optimized ROC AUC: 0.85861
```

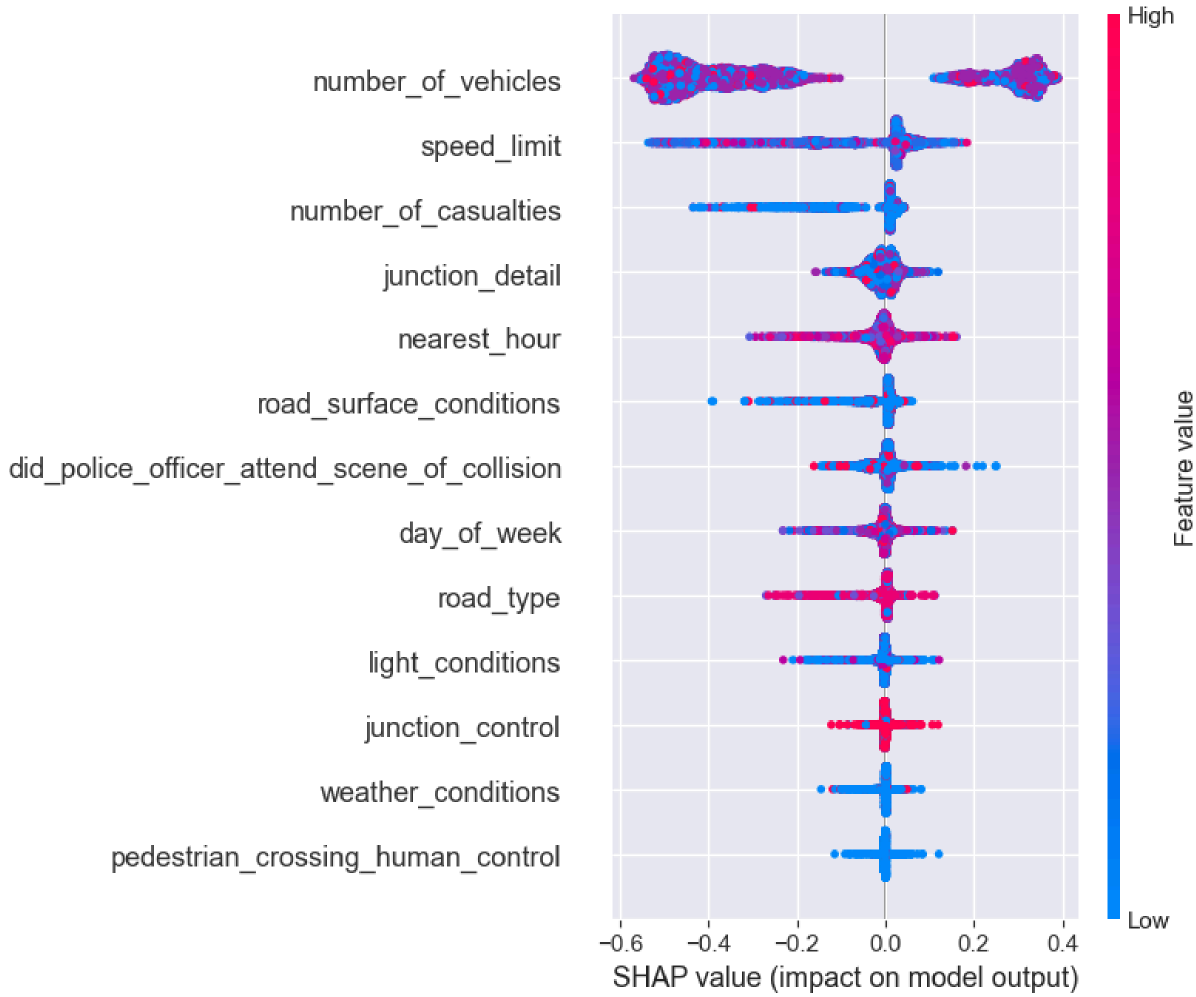
In [26]:
```
best_cb = grid_search_cb.best_estimator_

# Extract feature importances
feature_importances = best_cb.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to hold feature names and their importance
importances_cb = pd.DataFrame({
    'Feature': feature_names,
```

```
    'Importance': feature_importances
})

# Sort the DataFrame by importance in descending order
importances_cb = importances_cb.sort_values(by='Importance', ascending=False

# Visualize the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_cb)
plt.title('Feature Importances in Random Forest Model')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



In [111…

```
# Initialize the SHAP Explainer with check_additivity set to False
explainer_cb = shap.Explainer(cb_optimized)

# Compute SHAP values for the test set
shap_values_cb = explainer_cb.shap_values(X_test)

shap_values_positive_class_cb = shap_values_pd[:, :, 1]

shap.summary_plot(shap_values_positive_class_cb, X_test)
```



In [19]:
```
# Initialize and train the XGBoost classifier
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'
xgb_model.fit(X_train_smote, y_train_smote)
```

Out[19]:
```
▾                              XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_ro
unds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=N
one,
```

In [20]:
```
# Make predictions
y_pred_xg = xgb_model.predict(X_test)
y_pred_proba_xg = xgb_model.predict_proba(X_test)[:, 1]  # Probabilities for

# Calculate metrics
accuracy_xg = accuracy_score(y_test, y_pred_xg)
```

```
roc_auc_xg = roc_auc_score(y_test, y_pred_proba_xg)
report_xg = classification_report(y_test, y_pred_xg)

print(f"Accuracy before Tuning: {accuracy_xg:.5f}")
print(f"ROC AUC: {roc_auc_xg:.5f}")
print("\nClassification Report before Tuning:\n", report_xg)
```

```
Accuracy before Tuning: 0.84929
ROC AUC: 0.86630

Classification Report before Tuning:
               precision    recall  f1-score   support

           0       0.98      0.86      0.91      9282
           1       0.23      0.67      0.34       565

    accuracy                           0.85      9847
   macro avg       0.60      0.76      0.63      9847
weighted avg       0.93      0.85      0.88      9847
```

Run the original hyperparameter setting for comparison:

In [21]:
```
param_grid_xg1 = {
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200],
    'subsample': [0.8, 0.9, 1],
    'colsample_bytree': [0.3, 0.7],
    'gamma': [0, 0.1, 0.2]
}

xgb_model1 = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss

grid_search_xg1 = GridSearchCV(
    estimator=xgb_model1,
    param_grid=param_grid_xg1,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search_xg1.fit(X_train_smote, y_train_smote)

print("Best parameters:", grid_search_xg1.best_params_)
print("Best score: {:.5f}".format(grid_search_xg1.best_score_))
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best parameters: {'colsample_bytree': 0.7, 'gamma': 0, 'learning_rate': 0.2,
'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
Best score: 0.90838
```

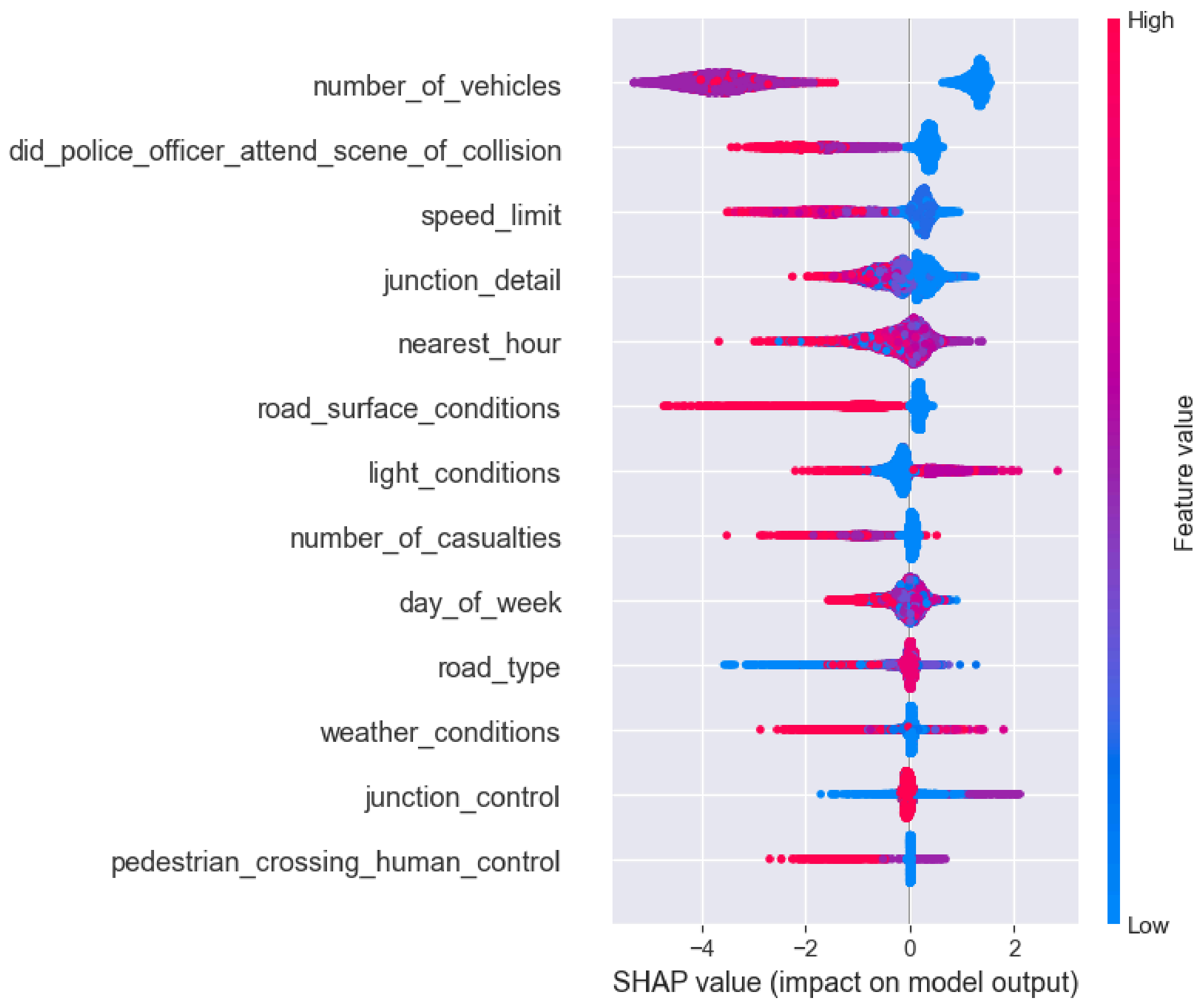
In [23]:
```
# Re-train the model using the best parameters from the correct grid search
xgb_optimized1 = xgb.XGBClassifier(**grid_search_xg1.best_params_, use_label
xgb_optimized1.fit(X_train_smote, y_train_smote)
```

```
# Re-evaluate the model
y_pred_opt_xg1 = xgb_optimized1.predict(X_test)
y_pred_opt_proba_xg1 = xgb_optimized1.predict_proba(X_test)[:, 1]
accuracy_opt_xg1 = accuracy_score(y_test, y_pred_opt_xg1)
roc_auc_opt_xg1 = roc_auc_score(y_test, y_pred_opt_proba_xg1)
report_opt_xg1 = classification_report(y_test, y_pred_opt_xg1)

# Output the optimized accuracy and ROC AUC, along with the classification r
print(f"Optimized Accuracy: {accuracy_opt_xg1:.5f}")
print("\n Optimized Classification Report:\n", report_opt_xg1)
```

```
Optimized Accuracy: 0.84533

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.86      0.91      9282
           1       0.22      0.68      0.33       565

    accuracy                           0.85      9847
   macro avg       0.60      0.77      0.62      9847
weighted avg       0.93      0.85      0.88      9847
```

In [24]:

```
# Initialize the SHAP Explainer with your model
explainer = shap.TreeExplainer(xgb_optimized1)

# Compute SHAP values for the test set
shap_values = explainer.shap_values(X_test)

# For a detailed summary plot that shows the impact of the top features acro
shap.summary_plot(shap_values, X_test)
```



In [38]:

```
param_grid_xg = {
    'max_depth': [5,10,20],
    'learning_rate': [0.1, 0.2,0.3],
    'n_estimators': [100, 200,300],
    'subsample': [0.8, 0.9, 1],
    'colsample_bytree': [0.3, 0.7],
    'gamma': [0, 0.1, 0.2]
}

xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss'

grid_search_xg = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid_xg,
    scoring='accuracy',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search_xg.fit(X_train_smote, y_train_smote)

print("Best parameters:", grid_search_xg.best_params_)
print("Best score: {:.5f}".format(grid_search_xg.best_score_))
```

```
Fitting 5 folds for each of 486 candidates, totalling 2430 fits
Best parameters: {'colsample_bytree': 0.7, 'gamma': 0.1, 'learning_rate': 0.
3, 'max_depth': 20, 'n_estimators': 200, 'subsample': 0.9}
Best score: 0.93378
```

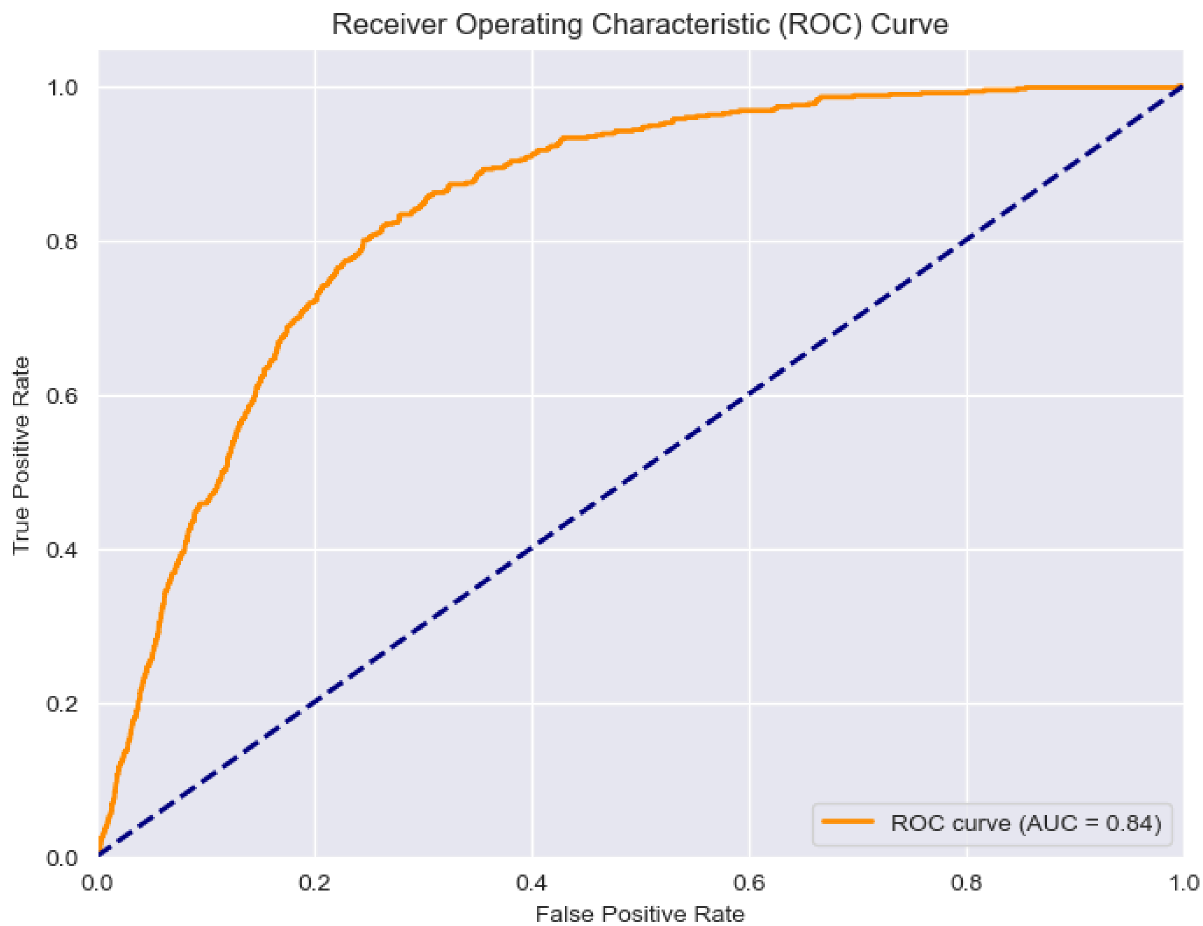
In [40]:

```python
# Re-train the model using the best parameters from the correct grid search
xgb_optimized = xgb.XGBClassifier(**grid_search_xg.best_params_, use_label_e
xgb_optimized.fit(X_train_smote, y_train_smote)

# Re-evaluate the model
y_pred_opt_xg = xgb_optimized.predict(X_test)
y_pred_opt_proba_xg = xgb_optimized.predict_proba(X_test)[:, 1]
accuracy_opt_xg = accuracy_score(y_test, y_pred_opt_xg)
roc_auc_opt_xg = roc_auc_score(y_test, y_pred_opt_proba_xg)
report_opt_xg = classification_report(y_test, y_pred_opt_xg)

# Output the optimized accuracy and ROC AUC, along with the classification r
print(f"Optimized Accuracy: {accuracy_opt_xg:.5f}")
print("\n Optimized Classification Report:\n", report_opt_xg)
```

```
Optimized Accuracy: 0.87814

 Optimized Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.90      0.93      9282
           1       0.22      0.46      0.30       565

    accuracy                           0.88      9847
   macro avg       0.59      0.68      0.62      9847
weighted avg       0.92      0.88      0.90      9847
```

In [41]:

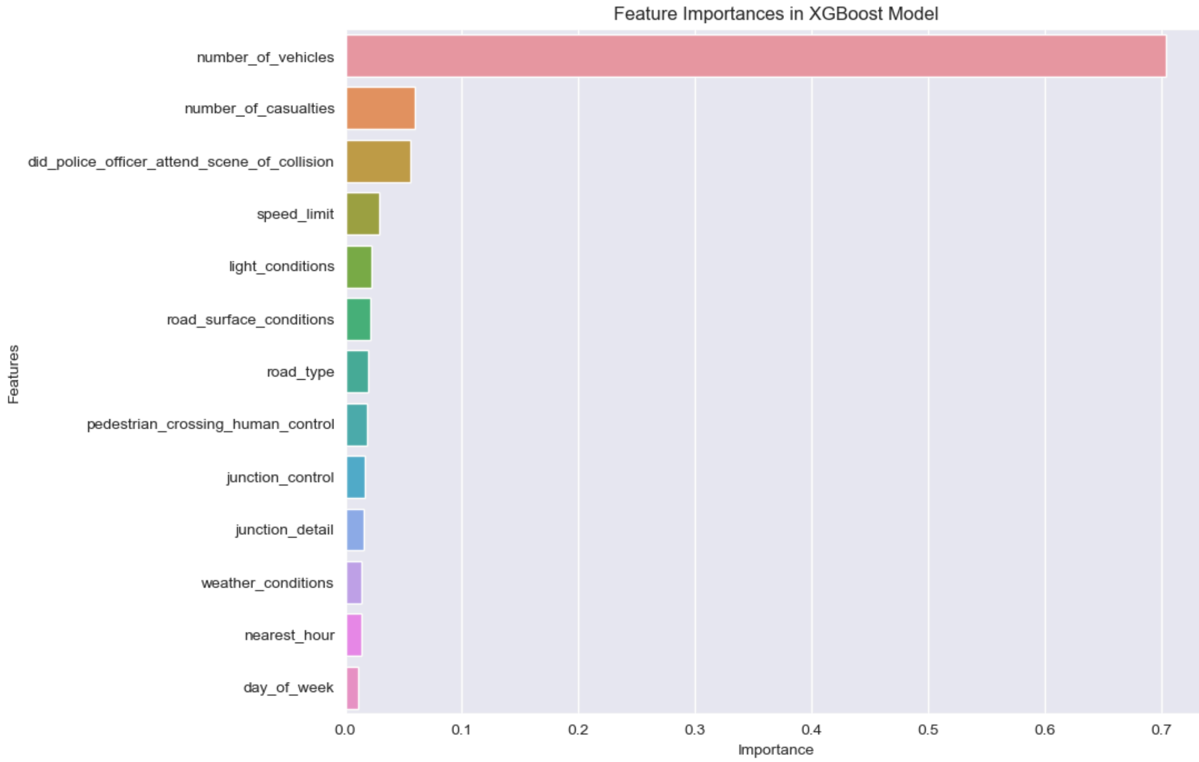
```python
# Generate false positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_pred_opt_proba_xg)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"Optimized ROC AUC: {roc_auc_opt_xg:.5f}")
```




```
Optimized ROC AUC: 0.83819
```

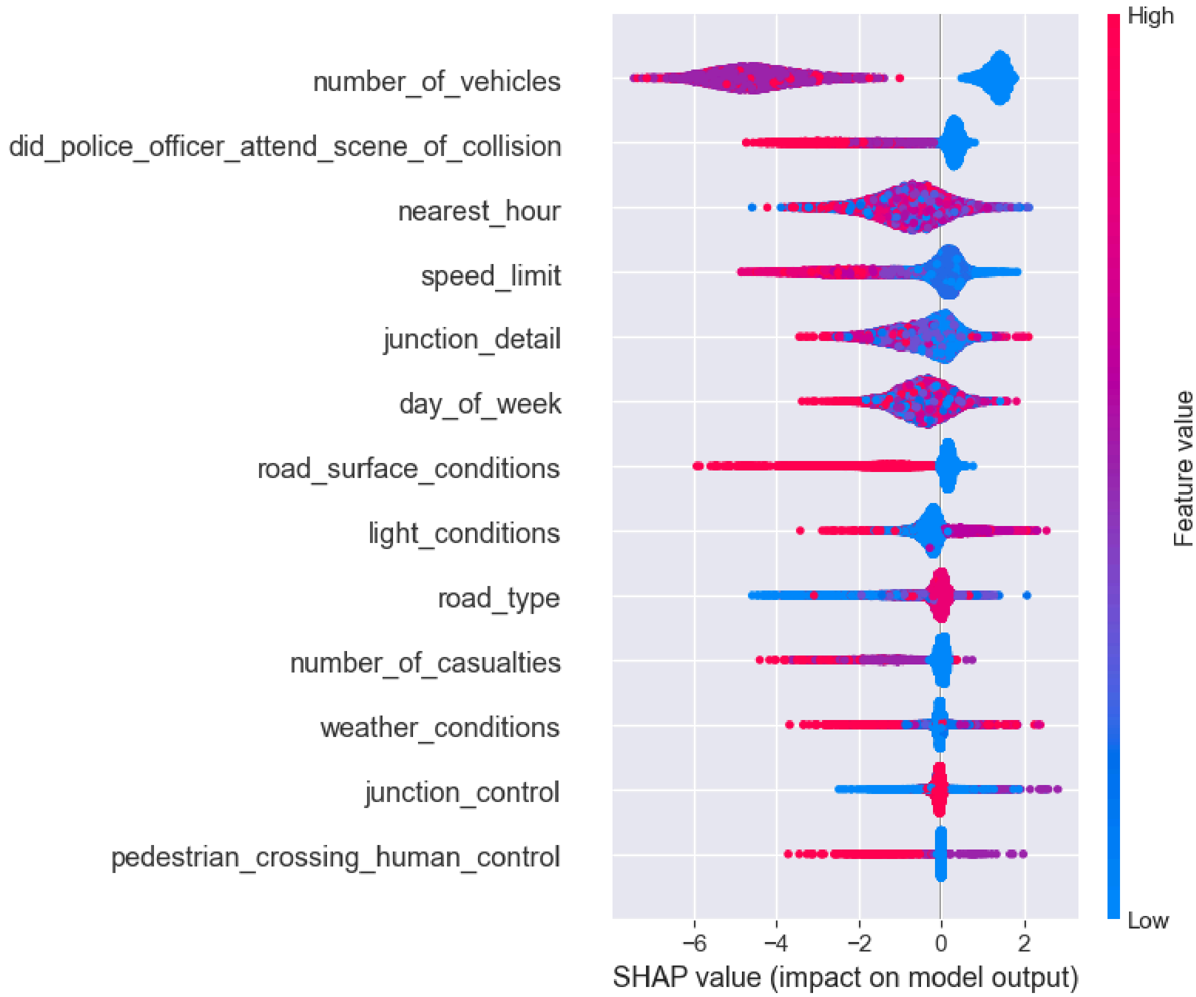
In [42]:

```python
# Extract feature importances
importances_xg = xgb_optimized.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to hold feature names and their importance
importances_xg = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances_xg
})

# Sort the DataFrame by importance in descending order
importances_xg = importances_xg.sort_values(by='Importance', ascending=False

# Visualize the feature importances
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importances_xg)
plt.title('Feature Importances in XGBoost Model')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



In [43]:

```
# Initialize the SHAP Explainer with your model
explainer = shap.TreeExplainer(xgb_optimized)

# Compute SHAP values for the test set
shap_values = explainer.shap_values(X_test)

# For a detailed summary plot that shows the impact of the top features acro
shap.summary_plot(shap_values, X_test)
```


number_of_vehicles
did_police_officer_attend_scene_of_collision
nearest_hour
speed_limit
junction_detail
day_of_week
road_surface_conditions
light_conditions
road_type
number_of_casualties
weather_conditions
junction_control
pedestrian_crossing_human_control
−6
−4
−2
0
2
SHAP value (impact on model output)
High
Low
Feature value