Candidate Numbers:

```
* 640844 - Lynn Komen
* 837128 - Mark Mugo
```

Machine Learning for Business Analytics- UK Road Safety Dataset

This notebook is a Data Preprocessing and Explorary Data Analysis report on the UK Road Safety.

Table of Contents

0.0 - Importing Libraries and Preparing Environment

In [1]:
```
import time
# We will monitor the time it takes to run the notebook
startnb = time.time()
```

In [3]:
```
#Base Libraries
import re
import warnings
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

#Library for Plotting
import seaborn as sns
sns.set(style="darkgrid")
import matplotlib.pyplot as plt
%matplotlib inline

#Library for Data Preprocessing and Cleaning
from sklearn.exceptions import ConvergenceWarning

# Importing train_test_split function from scikit-learn to split data into
from sklearn.model_selection import train_test_split
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.base import TransformerMixin, BaseEstimator
from phik.report import plot_correlation_matrix
from scipy.stats import chi2_contingency

# Suppressing warnings to keep the output clean and more readable
import warnings
warnings.filterwarnings('ignore')
```

1.0 - Business Objective

Improving Casualty Severity Prediction and Response in the Health Sector

In the health sector, one of the primary objectives is to reduce fatalities and severe injuries from road accidents while improving the quality of care for casualties. Accurate and timely prediction of casualty severity can significantly enhance emergency response, allocate resources more effectively, and provide data-driven insights for preventative measures.

The current challenges in the health sector revolve around insufficient real-time data on casualty severity, which limits emergency response effectiveness, resource allocation, and subsequent medical interventions. Inadequate prediction models can lead to delays in providing appropriate treatment, with a high cost in terms of lives and hospital resources. There is an urgent need for advanced predictive models that integrate variables related to road accidents to predict casualty severity effectively.

The objective of this initiative is to build a robust predictive model for casualty severity, with the following goals:

- Predict Casualty Severity: Accurately predict the severity of casualties in road accidents based on various factors.

- Enhance Emergency Response: Provide emergency teams with valuable insights for optimal resource allocation and prioritization.
- Improve Health Outcomes: Tailor medical interventions to the severity of the injury, improving survival rates and recovery times.
- Data-Driven Policy Recommendations: Generate data-driven insights for policymakers to create better road safety policies.

2.0 - Data Preparation

2.1- Loading Data - Vehicle Dataset, Casualty Dataset and Data Guide

In [4]:
```
# Loading data from a CSV file into a DataFrame
df1 = pd.read_csv("vehicle_2023.csv")

# Displaying the information about the DataFrame such as the number of entr
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 189815 entries, 0 to 189814
Data columns (total 34 columns):
 #   Column                           Non-Null Count   Dtype  
---  ------                           --------------   -----  
 0   accident_index                   189815 non-null  object 
 1   accident_year                    189815 non-null  int64  
 2   accident_reference               189815 non-null  object 
 3   vehicle_reference                189815 non-null  int64  
 4   vehicle_type                     189815 non-null  int64  
 5   towing_and_articulation          189815 non-null  int64  
 6   vehicle_manoeuvre                189815 non-null  int64  
 7   vehicle_direction_from           189815 non-null  int64  
 8   vehicle_direction_to             189815 non-null  int64  
 9   vehicle_location_restricted_lane 189815 non-null  int64  
 10  junction_location                189815 non-null  int64  
 11  skidding_and_overturning         189815 non-null  int64  
 12  hit_object_in_carriageway        189815 non-null  int64  
 13  vehicle_leaving_carriageway      189815 non-null  int64  
 14  hit_object_off_carriageway       189815 non-null  int64  
 15  first_point_of_impact            189815 non-null  int64  
 16  vehicle_left_hand_drive          189815 non-null  int64  
 17  journey_purpose_of_driver        189815 non-null  int64  
 18  sex_of_driver                    189815 non-null  int64  
 19  age_of_driver                    189815 non-null  int64  
 20  age_band_of_driver               189815 non-null  int64  
 21  engine_capacity_cc               189815 non-null  int64  
 22  propulsion_code                  189815 non-null  int64  
 23  age_of_vehicle                   189815 non-null  int64  
 24  generic_make_model               189815 non-null  object 
 25  driver_imd_decile                189815 non-null  int64  
 26  driver_home_area_type            189815 non-null  int64  
 27  lsoa_of_driver                   189815 non-null  object 
 28  escooter_flag                    189815 non-null  int64  
 29  dir_from_e                       19527 non-null   float64
 30  dir_from_n                       19527 non-null   float64
 31  dir_to_e                         19363 non-null   float64
 32  dir_to_n                         19363 non-null   float64
 33  driver_distance_banding          189815 non-null  int64  
dtypes: float64(4), int64(26), object(4)
memory usage: 49.2+ MB
```

There are 189815 and 34 columns in the Vehicle Info dataset. We will drop and keep relevant variables based on judgement and interpretation from the business objective

In [5]:
```
# Dropping specific columns from the DataFrame by their index positions
df1 = df1.drop(df1.columns[[0, 1, 2, 7, 8, 19, 27, 28, 29, 30, 31, 32, 33]]

# Limiting the DataFrame to the first 6000 rows
df1 = df1[:6000]

# Displaying the modified DataFrame
df1
```

Out[5]:

	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_lo
0	1	11	0	4	
1	1	11	0	18	
2	2	9	0	9	
3	3	9	0	8	
4	1	9	0	18	
...	...	...	...	...	
5995	2	11	0	18	
5996	1	1	9	99	
5997	2	9	0	99	
5998	1	9	0	99	
5999	2	19	9	99	

6000 rows × 21 columns

In [6]:
```
#The updated df with the necessary columns
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 21 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   vehicle_reference               6000 non-null   int64 
 1   vehicle_type                    6000 non-null   int64 
 2   towing_and_articulation         6000 non-null   int64 
 3   vehicle_manoeuvre               6000 non-null   int64 
 4   vehicle_location_restricted_lane 6000 non-null   int64 
 5   junction_location               6000 non-null   int64 
 6   skidding_and_overturning        6000 non-null   int64 
 7   hit_object_in_carriageway       6000 non-null   int64 
 8   vehicle_leaving_carriageway     6000 non-null   int64 
 9   hit_object_off_carriageway      6000 non-null   int64 
 10  first_point_of_impact           6000 non-null   int64 
 11  vehicle_left_hand_drive         6000 non-null   int64 
 12  journey_purpose_of_driver       6000 non-null   int64 
 13  sex_of_driver                   6000 non-null   int64 
 14  age_band_of_driver              6000 non-null   int64 
 15  engine_capacity_cc              6000 non-null   int64 
 16  propulsion_code                 6000 non-null   int64 
 17  age_of_vehicle                  6000 non-null   int64 
 18  generic_make_model              6000 non-null   object
 19  driver_imd_decile               6000 non-null   int64 
 20  driver_home_area_type           6000 non-null   int64 
dtypes: int64(20), object(1)
memory usage: 984.5+ KB
```

In [7]:
```
# Loading data from Casualty into a DataFrame called df2
df2 = pd.read_csv("casualty_2023.csv")
#information about the columns
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132977 entries, 0 to 132976
Data columns (total 21 columns):
 #   Column                             Non-Null Count   Dtype 
---  ------                             --------------   ----- 
 0   accident_index                     132977 non-null  object
 1   accident_year                      132977 non-null  int64 
 2   accident_reference                 132977 non-null  object
 3   vehicle_reference                  132977 non-null  int64 
 4   casualty_reference                 132977 non-null  int64 
 5   casualty_class                     132977 non-null  int64 
 6   sex_of_casualty                    132977 non-null  int64 
 7   age_of_casualty                    132977 non-null  int64 
 8   age_band_of_casualty               132977 non-null  int64 
 9   casualty_severity                  132977 non-null  int64 
 10  pedestrian_location                132977 non-null  int64 
 11  pedestrian_movement                132977 non-null  int64 
 12  car_passenger                      132977 non-null  int64 
 13  bus_or_coach_passenger             132977 non-null  int64 
 14  pedestrian_road_maintenance_worker 132977 non-null  int64 
 15  casualty_type                      132977 non-null  int64 
 16  casualty_home_area_type            132977 non-null  int64 
 17  casualty_imd_decile                132977 non-null  int64 
 18  lsoa_of_casualty                   132977 non-null  object
 19  enhanced_casualty_severity         132977 non-null  int64 
 20  casualty_distance_banding          132977 non-null  int64 
dtypes: int64(18), object(3)
memory usage: 21.3+ MB
```

In [8]:
```
# Dropping specific columns from the DataFrame df2 by their index positions
df2 = df2.drop(df2.columns[[0, 1, 2, 4, 7, 13, 14, 15, 17, 18, 19, 20]], ax

# Limiting the DataFrame df2 to the first 6000 rows
df2 = df2[:6000]

# Displaying the modified DataFrame df2
df2
```

Out[8]:

	vehicle_reference	casualty_class	sex_of_casualty	age_band_of_casualty	casualty_sev
0	1	3	2	4	
1	2	1	1	5	
2	3	2	2	7	
3	1	1	1	8	
4	2	1	1	6	
...	...	...	...	...	
5995	1	3	1	5	
5996	1	1	1	7	
5997	1	1	2	6	
5998	1	1	1	8	
5999	1	1	1	8	

6000 rows × 9 columns

In [9]:
```
#The updated df with the necessary columns
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   vehicle_reference       6000 non-null   int64
 1   casualty_class          6000 non-null   int64
 2   sex_of_casualty         6000 non-null   int64
 3   age_band_of_casualty    6000 non-null   int64
 4   casualty_severity       6000 non-null   int64
 5   pedestrian_location     6000 non-null   int64
 6   pedestrian_movement     6000 non-null   int64
 7   car_passenger           6000 non-null   int64
 8   casualty_home_area_type 6000 non-null   int64
dtypes: int64(9)
memory usage: 422.0 KB
```

Merging the datasets

In [10]:
```
# Merging two DataFrames, df1 and df2, based on the 'vehicle_reference' col
df3 = pd.merge(df1, df2, on='vehicle_reference')

#Inspect the first few columns
df3.head()
```

Out[10]:

	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_locati
0	1	11	0	4	
1	1	11	0	4	
2	1	11	0	4	
3	1	11	0	4	
4	1	11	0	4	

5 rows × 29 columns

In [11]:
```
#Show the columns chosen for the analysis
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18516174 entries, 0 to 18516173
Data columns (total 29 columns):
 #   Column                           Dtype
---  ------                           -----
 0   vehicle_reference                int64
 1   vehicle_type                     int64
 2   towing_and_articulation          int64
 3   vehicle_manoeuvre                int64
 4   vehicle_location_restricted_lane int64
 5   junction_location                int64
 6   skidding_and_overturning         int64
 7   hit_object_in_carriageway        int64
 8   vehicle_leaving_carriageway      int64
 9   hit_object_off_carriageway       int64
 10  first_point_of_impact            int64
 11  vehicle_left_hand_drive          int64
 12  journey_purpose_of_driver        int64
 13  sex_of_driver                    int64
 14  age_band_of_driver               int64
 15  engine_capacity_cc               int64
 16  propulsion_code                  int64
 17  age_of_vehicle                   int64
 18  generic_make_model               object
 19  driver_imd_decile                int64
 20  driver_home_area_type            int64
 21  casualty_class                   int64
 22  sex_of_casualty                  int64
 23  age_band_of_casualty             int64
 24  casualty_severity                int64
 25  pedestrian_location              int64
 26  pedestrian_movement              int64
 27  car_passenger                    int64
 28  casualty_home_area_type          int64
dtypes: int64(28), object(1)
memory usage: 4.1+ GB
```

In [12]:
```
# Loading data from an Excel file named 'Data_Guide.xlsx' into a DataFrame
df4 = pd.read_excel('Data_Guide.xlsx')

# Displaying the first five rows of df4 to preview its content
df4.head()

# Note: This DataFrame, df4, contains descriptions for each data point in a
```

Out[12]:

	table	field name	code/format	label
0	accident	police_force	1	Metropolitan Police
1	accident	police_force	3	Cumbria
2	accident	police_force	4	Lancashire
3	accident	police_force	5	Merseyside
4	accident	police_force	6	Greater Manchester

In [13]:
```
#get information on the data guide
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1775 entries, 0 to 1774
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   table        1775 non-null   object
 1   field name   1775 non-null   object
 2   code/format  1722 non-null   object
 3   label        1723 non-null   object
dtypes: object(4)
memory usage: 55.6+ KB
```

In [14]:
```
# Filtering out rows from df4 where the 'table' column has the value 'accid
df4 = df4[df4['table'] != 'accident']

# Dropping rows in df4 where the 'code/format' column contains missing valu
df4 = df4.dropna(subset=['code/format'])

# Displaying the first five rows of the modified DataFrame df4 to preview t
df4.head()
```

Out[14]:

	table	field name	code/format	label
1384	vehicle	vehicle_type	1	Pedal cycle
1385	vehicle	vehicle_type	2	Motorcycle 50cc and under
1386	vehicle	vehicle_type	3	Motorcycle 125cc and under
1387	vehicle	vehicle_type	4	Motorcycle over 125cc and up to 500cc
1388	vehicle	vehicle_type	5	Motorcycle over 500cc

In [15]:
```
#Check the new output
df4.head(60)
```

Out[15]:

	table	field name	code/format	label
1384	vehicle	vehicle_type	1	Pedal cycle
1385	vehicle	vehicle_type	2	Motorcycle 50cc and under
1386	vehicle	vehicle_type	3	Motorcycle 125cc and under
1387	vehicle	vehicle_type	4	Motorcycle over 125cc and up to 500cc
1388	vehicle	vehicle_type	5	Motorcycle over 500cc
1389	vehicle	vehicle_type	8	Taxi/Private hire car
1390	vehicle	vehicle_type	9	Car
1391	vehicle	vehicle_type	10	Minibus (8 - 16 passenger seats)
1392	vehicle	vehicle_type	11	Bus or coach (17 or more pass seats)
1393	vehicle	vehicle_type	16	Ridden horse
1394	vehicle	vehicle_type	17	Agricultural vehicle
1395	vehicle	vehicle_type	18	Tram
1396	vehicle	vehicle_type	19	Van / Goods 3.5 tonnes mgw or under
1397	vehicle	vehicle_type	20	Goods over 3.5t. and under 7.5t
1398	vehicle	vehicle_type	21	Goods 7.5 tonnes mgw and over
1399	vehicle	vehicle_type	22	Mobility scooter
1400	vehicle	vehicle_type	23	Electric motorcycle
1401	vehicle	vehicle_type	90	Other vehicle
1402	vehicle	vehicle_type	97	Motorcycle - unknown cc
1403	vehicle	vehicle_type	98	Goods vehicle - unknown weight
1404	vehicle	vehicle_type	99	Unknown vehicle type (self rep only)
1405	vehicle	vehicle_type	103	Motorcycle - Scooter (1979-1998)
1406	vehicle	vehicle_type	104	Motorcycle (1979-1998)
1407	vehicle	vehicle_type	105	Motorcycle - Combination (1979-1998)
1408	vehicle	vehicle_type	106	Motorcycle over 125cc (1999-2004)
1409	vehicle	vehicle_type	108	Taxi (excluding private hire cars) (1979-2004)
1410	vehicle	vehicle_type	109	Car (including private hire cars) (1979-2004)
1411	vehicle	vehicle_type	110	Minibus/Motor caravan (1979-1998)
1412	vehicle	vehicle_type	113	Goods over 3.5 tonnes (1979-1998)
1413	vehicle	vehicle_type	-1	Data missing or out of range
1414	vehicle	towing_and_articulation	0	No tow/articulation
1415	vehicle	towing_and_articulation	1	Articulated vehicle
1416	vehicle	towing_and_articulation	2	Double or multiple trailer
1417	vehicle	towing_and_articulation	3	Caravan
1418	vehicle	towing_and_articulation	4	Single trailer
1419	vehicle	towing_and_articulation	5	Other tow
1420	vehicle	towing_and_articulation	9	unknown (self reported)
1421	vehicle	towing_and_articulation	-1	Data missing or out of range

	table	field name	code/format	label
1422	vehicle	vehicle_manoeuvre	1	Reversing
1423	vehicle	vehicle_manoeuvre	2	Parked
1424	vehicle	vehicle_manoeuvre	3	Waiting to go - held up
1425	vehicle	vehicle_manoeuvre	4	Slowing or stopping
1426	vehicle	vehicle_manoeuvre	5	Moving off
1427	vehicle	vehicle_manoeuvre	6	U-turn
1428	vehicle	vehicle_manoeuvre	7	Turning left
1429	vehicle	vehicle_manoeuvre	8	Waiting to turn left
1430	vehicle	vehicle_manoeuvre	9	Turning right
1431	vehicle	vehicle_manoeuvre	10	Waiting to turn right
1432	vehicle	vehicle_manoeuvre	11	Changing lane to left
1433	vehicle	vehicle_manoeuvre	12	Changing lane to right
1434	vehicle	vehicle_manoeuvre	13	Overtaking moving vehicle - offside
1435	vehicle	vehicle_manoeuvre	14	Overtaking static vehicle - offside
1436	vehicle	vehicle_manoeuvre	15	Overtaking - nearside
1437	vehicle	vehicle_manoeuvre	16	Going ahead left-hand bend
1438	vehicle	vehicle_manoeuvre	17	Going ahead right-hand bend
1439	vehicle	vehicle_manoeuvre	18	Going ahead other
1440	vehicle	vehicle_manoeuvre	99	unknown (self reported)
1441	vehicle	vehicle_manoeuvre	-1	Data missing or out of range
1442	vehicle	vehicle_direction_from	0	Parked
1443	vehicle	vehicle_direction_from	1	North

Mapping the data set according to the appropriate labels

In [16]:
```
#Use a for loop to map the index with the correct meaning of each value
def replace_codes_with_labels(df3, df4):
    # Looping through each column in df3
    for column in df3.columns:
        # Check if the current column is listed in df4's 'field name' colum
        if column in df4['field name'].values:
            # Creating a dictionary to map codes to labels for each 'field 
            mapping_dict = df4[df4['field name'] == column].set_index('code
            # If the mapping dictionary is not empty, replace df3's column 
            if mapping_dict:
                df3[column] = df3[column].map(mapping_dict).fillna(df3[colu
    # Return the updated DataFrame df3 after all columns have been processe
    return df3
```

In [17]:
```
#Apply the code
df3_transformed = replace_codes_with_labels(df3, df4)
```

In [18]:
```
#display the mapped data frame
df3_transformed.head(100)
```

Out[18]:

	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_loca
0	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
1	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
2	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
3	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
4	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
...	...	...	...	...	
95	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
96	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
97	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
98	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'
99	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main c'

100 rows × 29 columns

In [19]:
```
#Show a brief overview of the categorical columns in the mapped data frame
df3_transformed.describe(include = 'object')
```

Out[19]:

	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_location_restricted_
count	18516174	18516174	18516174	1851
unique	18	7	19	
top	Car	No tow/articulation	unknown (self reported)	On main c'way - not in restr
freq	9862985	16202126	7846361	995

4 rows × 27 columns

In [20]:
```
#Change the name of the transformed dataframe to understand the flow labell
df5=df3_transformed
```

In [21]:
```
#Investigate the last few records of the data set
df5.tail(5)
```

Out[21]:

	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicl
18516169	6	Car	No tow/articulation	Parked	On
18516170	6	Car	No tow/articulation	Parked	On
18516171	6	Car	No tow/articulation	Going ahead other	On
18516172	6	Car	No tow/articulation	Going ahead other	On
18516173	7	Car	No tow/articulation	Parked	On

5 rows × 29 columns

In [22]:
```
#We want to keep only 10,000 rows
df6 = df5.iloc[1799::1800]
#This is selecting every 1800th row starting from index 1799 to sample the
```

In [23]:
```
#get info on the new data frame
df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10286 entries, 1799 to 18514799
Data columns (total 29 columns):
 #   Column                            Non-Null Count  Dtype 
---  ------                            --------------  ----- 
 0   vehicle_reference                 10286 non-null  int64 
 1   vehicle_type                      10286 non-null  object
 2   towing_and_articulation           10286 non-null  object
 3   vehicle_manoeuvre                 10286 non-null  object
 4   vehicle_location_restricted_lane  10286 non-null  object
 5   junction_location                 10286 non-null  object
 6   skidding_and_overturning          10286 non-null  object
 7   hit_object_in_carriageway         10286 non-null  object
 8   vehicle_leaving_carriageway       10286 non-null  object
 9   hit_object_off_carriageway        10286 non-null  object
 10  first_point_of_impact             10286 non-null  object
 11  vehicle_left_hand_drive           10286 non-null  object
 12  journey_purpose_of_driver         10286 non-null  object
 13  sex_of_driver                     10286 non-null  object
 14  age_band_of_driver                10286 non-null  object
 15  engine_capacity_cc                10286 non-null  object
 16  propulsion_code                   10286 non-null  object
 17  age_of_vehicle                    10286 non-null  int64 
 18  generic_make_model                10286 non-null  object
 19  driver_imd_decile                 10286 non-null  object
 20  driver_home_area_type             10286 non-null  object
 21  casualty_class                    10286 non-null  object
 22  sex_of_casualty                   10286 non-null  object
 23  age_band_of_casualty              10286 non-null  object
 24  casualty_severity                 10286 non-null  object
 25  pedestrian_location               10286 non-null  object
 26  pedestrian_movement               10286 non-null  object
 27  car_passenger                     10286 non-null  object
 28  casualty_home_area_type           10286 non-null  object
dtypes: int64(2), object(27)
memory usage: 2.4+ MB
```

There 10286 entries, we will use this for the Exploratory Data Analysis

In [24]:
```
#Inspect the data
df6.head()
```

Out[24]:

	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_lo
1799	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main
3599	1	Bus or coach (17 or more pass seats)	No tow/articulation	Slowing or stopping	On main
5399	1	Bus or coach (17 or more pass seats)	No tow/articulation	Going ahead other	On main
7199	1	Bus or coach (17 or more pass seats)	No tow/articulation	Going ahead other	On main
8999	1	Bus or coach (17 or more pass seats)	No tow/articulation	Going ahead other	On main

5 rows × 29 columns

In [25]:
```
#Check the general descriptive stats on the categorical variables
df6.describe(include = 'object')
```

Out[25]:

	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_location_restricted_
count	10286	10286	10286	1(
unique	17	6	19	
top	Car	No tow/articulation	unknown (self reported)	On main c'way - not in restr
freq	5460	8998	4352	

4 rows × 27 columns

In [26]:
```
# Count the number of missing values in each column of DataFrame
missing_values_count = df6.isna().sum()

# Print the total number of missing values across all columns
print('Number of Missing Values : ' + str(missing_values_count.sum()))
```

```
Number of Missing Values : 0
```

2.2 - Variable Notes

Variables are categorised into multiple types : Nominal, Ordinal, Interval, Ratio.

To simplify, the types are narrowed down into 2 main types:

- Numeric : Variables containing numeric values.
- Categorical : Variables containing text data / each unique value indicates a category.

Input Data	Definition	Category
vehicle_type	The type of vehicle involved in the accident.e.g.,cars	Categorical
towing_and_articulation	Indicates whether the vehicle was towing a trailer or articulated.	Categorical
vehicle_manoeuvre	Describes the vehicle's movement before the accident	Categorical
vehicle_location_restricted_la ne	Specifies if the vehicle was in a restricted lane e.g., a bus lane	Categorical
junction_location	Indicates the vehicle's position concerning a junction	Categorical
skidding_and_overturning	Describes if the vehicle skidded, overturned	Categorical
hit_object_in_carriageway	Identifies objects that the vehicle hit within the carriageway	Categorical
vehicle_leaving_carriageway	Indicates whether the vehicle left the carriageway	Categorical
hit_object_off_carriageway	Identifies objects hit off the carriageway, such as trees	Categorical
First point of impact	The part of the vehicle that first made contact during the accident (e.g., front, rear, side).	Categorical
Vegicle left hand drive	vehicle is designed for left- hand drive (Yes or No).	Categorical
Journey purpose of driver	The reason for the journey at the time of the accident	Categorical
Sex of driver	The gender of the driver	Categorical
Age band of driver	Age group category of the driver	Categorical
Engine capacity cc	The size of the vehicle's engine in cubic centimeters (cc).	Numerical
Propulsion code	The type of fuel or propulsion system	Categorical
Age of vehicle	The number of years since the vehicle was first registered.	Numerical
Generic make model	A general description of the vehicle	Categorical
Driver imd decile	Representing the socio- economic status	Categorical
Driver Home Area Type	It categorizes the drivers' home locations	Categorical
Casualty Class	This describes whether the casualty type	Categorical
Sex of Casualty	This states gender	Categorical
Age Band of Casualty	age bands e.g., 21-25	Categorical
Casualty Severity	This states the intensity of the casuality	Categorical
Pedestrian Location	This describes the location the passenger	Categorical
Pedestrian Movement	This describes the direction the passenger	Categorical
Car Passenger	if there was a passenger and position in the car	Categorical
Casualty Home Area	It categorizes the drivers' home locations	Categorical

Descriptive Stats on Training Datasets

In [27]:
```
# Splitting DataFrame df6 into training and testing sets
# Set the seed for NumPy's random number generator
np.random.seed(1)

# Using 'vehicle_type' for stratification to maintain the proportion of dif
# Setting 'test_size' to 0.1 to allocate 10% of the data to the test set
# Using 'random_state' for reproducibility of the split
trainset, testset = train_test_split(df6, test_size=0.1, stratify=df6['vehi
```

In [28]:
```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9257 entries, 1151999 to 5570999
Data columns (total 29 columns):
 #   Column                             Non-Null Count  Dtype 
---  ------                             --------------  ----- 
 0   vehicle_reference                  9257 non-null   int64 
 1   vehicle_type                       9257 non-null   object
 2   towing_and_articulation            9257 non-null   object
 3   vehicle_manoeuvre                  9257 non-null   object
 4   vehicle_location_restricted_lane   9257 non-null   object
 5   junction_location                  9257 non-null   object
 6   skidding_and_overturning           9257 non-null   object
 7   hit_object_in_carriageway          9257 non-null   object
 8   vehicle_leaving_carriageway        9257 non-null   object
 9   hit_object_off_carriageway         9257 non-null   object
 10  first_point_of_impact              9257 non-null   object
 11  vehicle_left_hand_drive            9257 non-null   object
 12  journey_purpose_of_driver          9257 non-null   object
 13  sex_of_driver                      9257 non-null   object
 14  age_band_of_driver                 9257 non-null   object
 15  engine_capacity_cc                 9257 non-null   object
 16  propulsion_code                    9257 non-null   object
 17  age_of_vehicle                     9257 non-null   int64 
 18  generic_make_model                 9257 non-null   object
 19  driver_imd_decile                  9257 non-null   object
 20  driver_home_area_type              9257 non-null   object
 21  casualty_class                     9257 non-null   object
 22  sex_of_casualty                    9257 non-null   object
 23  age_band_of_casualty               9257 non-null   object
 24  casualty_severity                  9257 non-null   object
 25  pedestrian_location                9257 non-null   object
 26  pedestrian_movement                9257 non-null   object
 27  car_passenger                      9257 non-null   object
 28  casualty_home_area_type            9257 non-null   object
dtypes: int64(2), object(27)
memory usage: 2.1+ MB
```

In [29]:
```
# Remove the 'vehicle_reference' column from the trainset
trainset = trainset.loc[:, [col for col in trainset.columns if col != 'vehi

# Remove the 'vehicle_reference' column from the testset
testset = testset.loc[:, [col for col in testset.columns if col != 'vehicle
```

3.1 Target Variable

3.1.1 Casualty Severity

In [30]:
```
# Displaying the statistical summary for 'casualty_severity'
pd.DataFrame(trainset.loc[:, 'casualty_severity'].describe())
```

Out[30]:

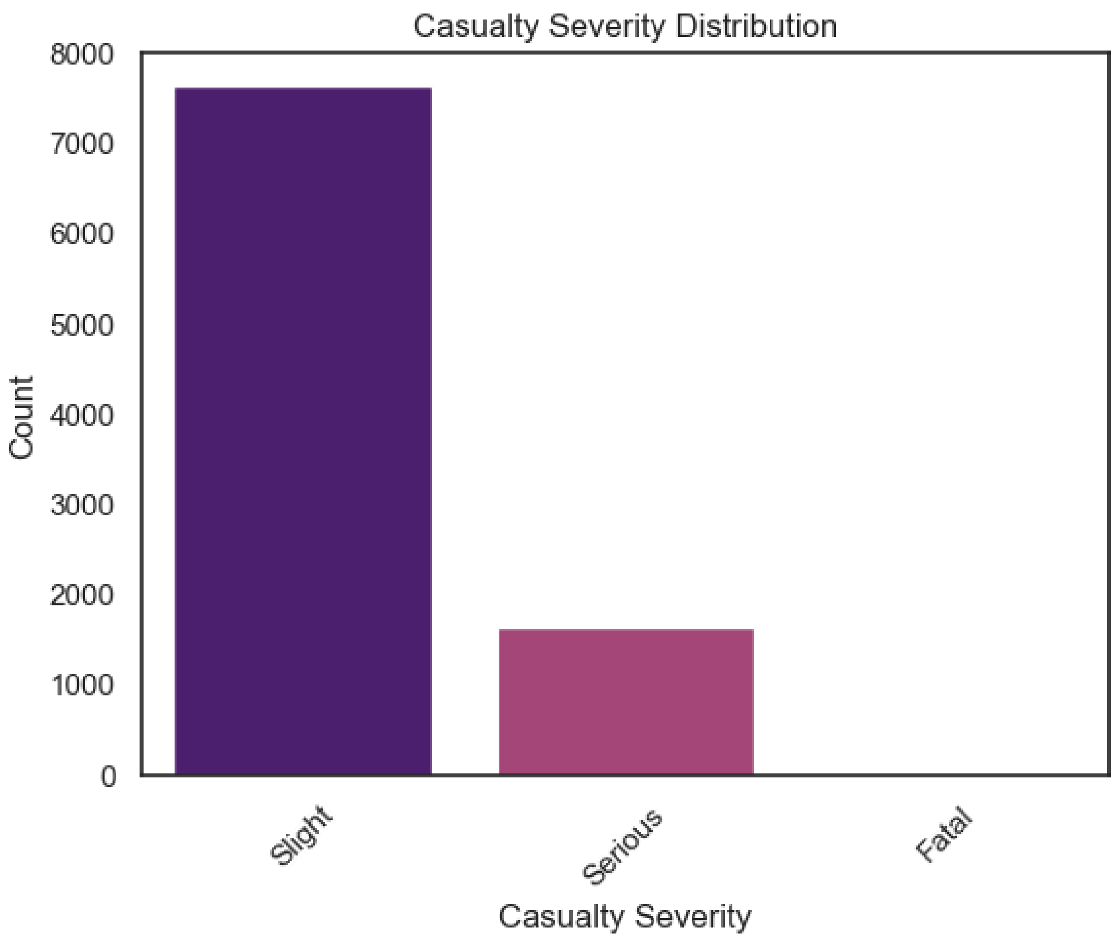
	casualty_severity
count	9257
unique	3
top	Slight
freq	7630

In [31]:
```
# Getting the value counts of the 'casualty_severity' variable
values = pd.DataFrame(trainset['casualty_severity'].value_counts())
values.columns = ['Casualty Severity Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['casualty_severity'].value_counts
percentages.columns = ['% Casualty Severity']

# Converting the percentage values to a string format with a '%' sign
percentages['% Casualty Severity'] = percentages['% Casualty Severity'].map

# Joining the values and percentages dataframes
summary_table = values.join(percentages)
summary_table
```

Out[31]:

	Casualty Severity Count	% Casualty Severity
Slight	7630	82.42%
Serious	1625	17.55%
Fatal	2	0.02%

In [32]:
```
# Define plot object for 'casualty_severity'
sns.set_style('white')
count = sns.countplot(data = trainset, x = 'casualty_severity',  palette='m

# Setting graph title and labels
count.set_title('Casualty Severity Distribution')
count.set(xlabel='Casualty Severity', ylabel='Count')

# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit

plt.ylim(0,8000)

# Showing the plot
plt.show()
```



3.2 Dependent Variables

3.2.1 Numerical Variables

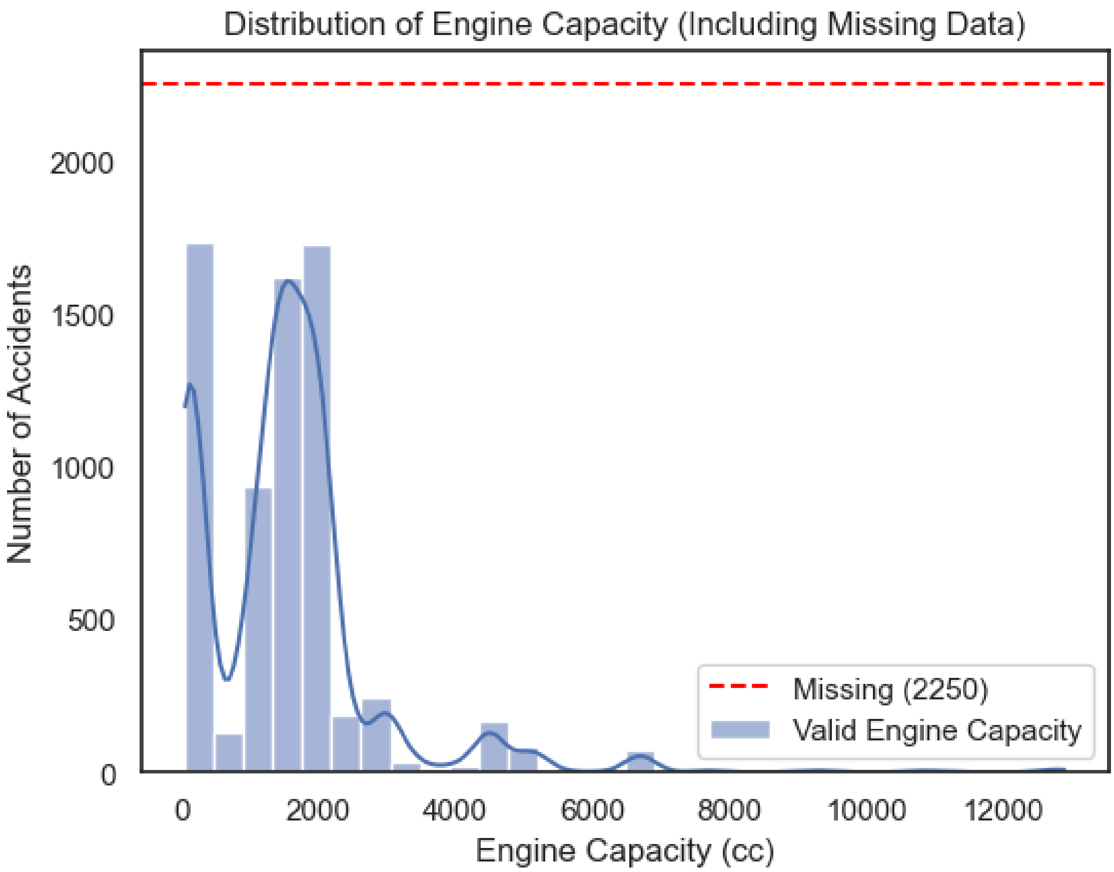
3.2.1.1 Engine Capacity cc

In [33]:
```
# Generate summary statistics for the 'engine_capacity_cc' variable
# Convert the output into a DataFrame for better readability
pd.DataFrame(trainset.loc[:,'engine_capacity_cc'].describe())
```

Out[33]:

	engine_capacity_cc
count	9257
unique	264
top	Data missing or out of range
freq	2250

In [34]:
```
sns.set_style("white")  # Removes background gridlines
# Separate numeric values and "Data missing or out of range"
trainset['engine_capacity_cc'] = trainset['engine_capacity_cc'].astype(str)
missing_count = trainset['engine_capacity_cc'].value_counts().get("Data mis

# Plot numeric values only
numeric_data = trainset[trainset['engine_capacity_cc'] != "Data missing or 
sns.histplot(numeric_data, bins=30, kde=True, label="Valid Engine Capacity"

# Add missing data count as a separate bar
plt.axhline(y=missing_count, color='red', linestyle='dashed', label=f'Missi

# Labels and title
plt.xlabel("Engine Capacity (cc)")
plt.ylabel("Number of Accidents")
plt.title("Distribution of Engine Capacity (Including Missing Data)")
plt.legend()
plt.show()
```



- The distribution shows that most vehicles have engine capacities below 3000cc, with very few above 6000cc, possibly high-performance or commercial vehicles. A significant portion (2,250 cases) is missing or out of range, shown by the red dashed line. This missing data may skew analysis, requiring imputation or exclusion for accuracy.

In [35]:
```
#Check the descriptive stats on the numeric dataset
numeric_data.describe()
```

Out[35]:
```
count     7007.000000
mean      1530.375482
std       1346.919051
min         49.000000
25%        600.000000
50%       1498.000000
75%       1984.000000
max      12902.000000
Name: engine_capacity_cc, dtype: float64
```

3.2.1.2 Age of Vehicle

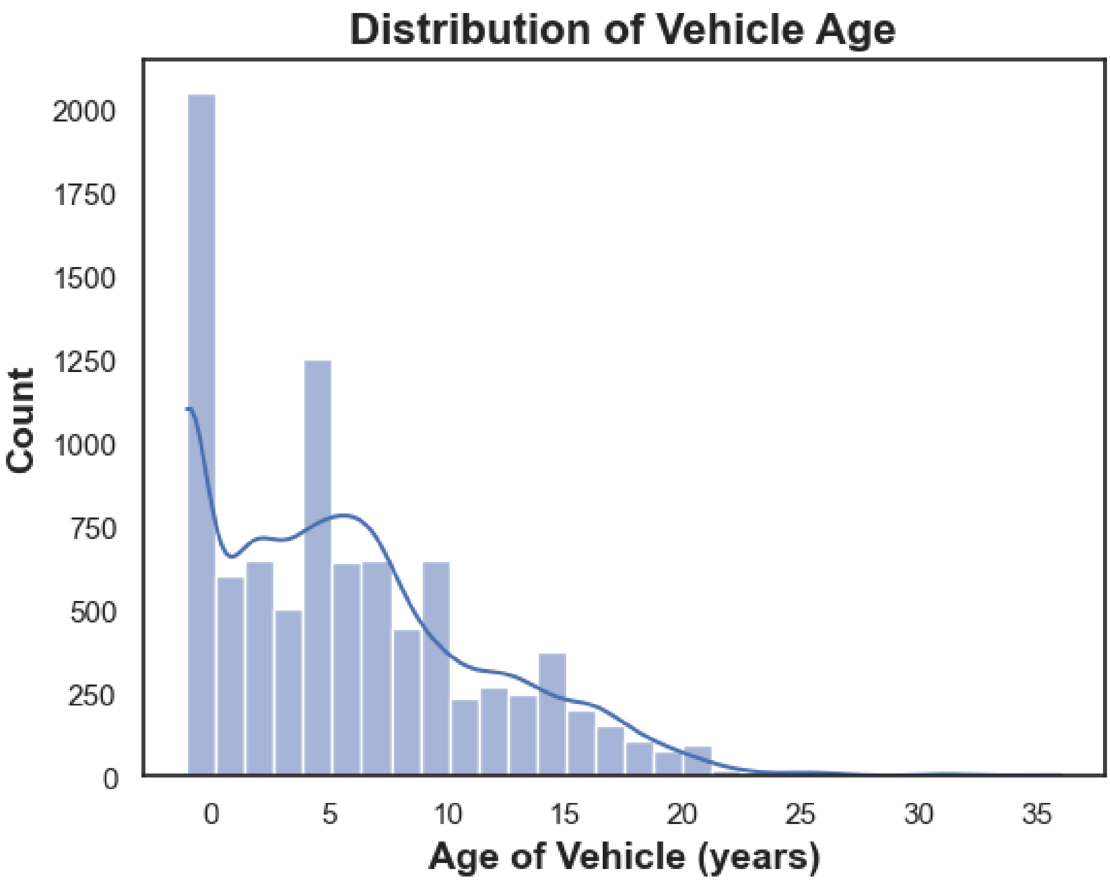
In [36]:
```
pd.DataFrame(trainset.loc[:,'age_of_vehicle'].describe())
```

Out[36]:

	age_of_vehicle
count	9257.000000
mean	5.655504
std	5.795719
min	-1.000000
25%	1.000000
50%	5.000000
75%	9.000000
max	36.000000

The average vehicle age is 5.66 years, with most between 1 and 9 years. The minimum value of -1 suggests data entry errors that need cleaning. The oldest vehicle is 36 years, however the mean suggests a relatively young vehicle fleet and the high standard deviation (5.8) indicates a wide range of vehicle ages.

In [37]:
```
# Set Seaborn style to a clean white background
sns.set_style("white")

# Create a histogram for the 'age_of_vehicle' column
# - Use 30 bins for better granularity
# - Includes a Kernel Density Estimate (KDE) to show the distribution curve
# - Sets the color to "skyblue" for better visibility
sns.histplot(trainset['age_of_vehicle'], bins=30, kde=True, )

# Set x-axis label with a descriptive title
plt.xlabel("Age of Vehicle (years)", fontsize=14, fontweight='bold')

# Set y-axis label to indicate count
plt.ylabel("Count", fontsize=14, fontweight='bold')

# Set the title of the plot
plt.title("Distribution of Vehicle Age", fontsize=16, fontweight='bold')

# Display the plot
plt.show()
```



- The vehicle age distribution is right-skewed, with most vehicles between 0-10 years old, peaking around newer vehicles (0-2 years). There are fewer older vehicles, but some exceed 30 years, possibly indicating classic or poorly maintained vehicles. The high frequency of very new cars suggests recent purchases or fleet vehicles.

3.2.2 Categorical variables

3.2.2.1 Vehicle Manoeuvre

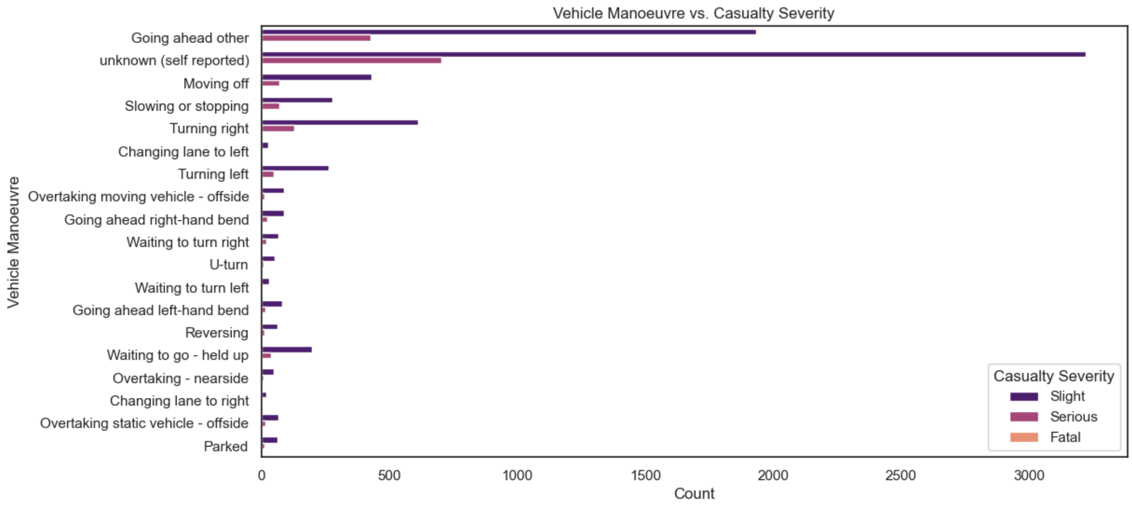
In [38]:
```
# Getting the value counts of the 'vehicle_manoeuvre' variable
# This counts the occurrences of each unique value in the 'vehicle_manoeuvr
values = pd.DataFrame(trainset['vehicle_manoeuvre'].value_counts())
values.columns = ['Vehicle Manoeuvre Count']  # Renaming the column for cla

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['vehicle_manoeuvre'].value_counts
percentages.columns = ['% Vehicle Manoeuvre']  # Renaming the column for cl

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Vehicle Manoeuvre'] = percentages['% Vehicle Manoeuvre'].map

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[38]:

	Vehicle Manoeuvre Count	% Vehicle Manoeuvre
unknown (self reported)	3924	42.39%
Going ahead other	2360	25.49%
Turning right	739	7.98%
Moving off	500	5.4%
Slowing or stopping	347	3.75%
Turning left	311	3.36%
Waiting to go - held up	236	2.55%
Going ahead right-hand bend	111	1.2%
Overtaking moving vehicle - offside	102	1.1%
Going ahead left-hand bend	99	1.07%
Waiting to turn right	85	0.92%
Overtaking static vehicle - offside	82	0.89%
Parked	77	0.83%
Reversing	75	0.81%
U-turn	62	0.67%
Overtaking - nearside	60	0.65%
Waiting to turn left	34	0.37%
Changing lane to left	32	0.35%
Changing lane to right	21	0.23%

- The most common manoeuvre is "unknown (self reported)" (41.9%), indicating a large portion of data lacks specific manoeuvre details.
- Rare manoeuvres like "Changing lane to right" (0.23%) highlight potential class imbalance in the dataset.

In [39]:
```
# Creating a count plot to visualize the relationship between 'vehicle_manc
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(12, 6))  # Setting the figure size for better visualiza
sns.countplot(y=trainset['vehicle_manoeuvre'], hue=trainset['casualty_sever
plt.title('Vehicle Manoeuvre vs. Casualty Severity')  # Adding a title to t
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Vehicle Manoeuvre')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- "Unknown (self reported)" and "Going ahead other" have the highest incident counts, with most casualties being Slight.
- Fatal incidents are relatively rare but appear more frequently in manoeuvres like "Turning right" and "Moving off", indicating potential risk factors.

3.2.2.2 Junction Location

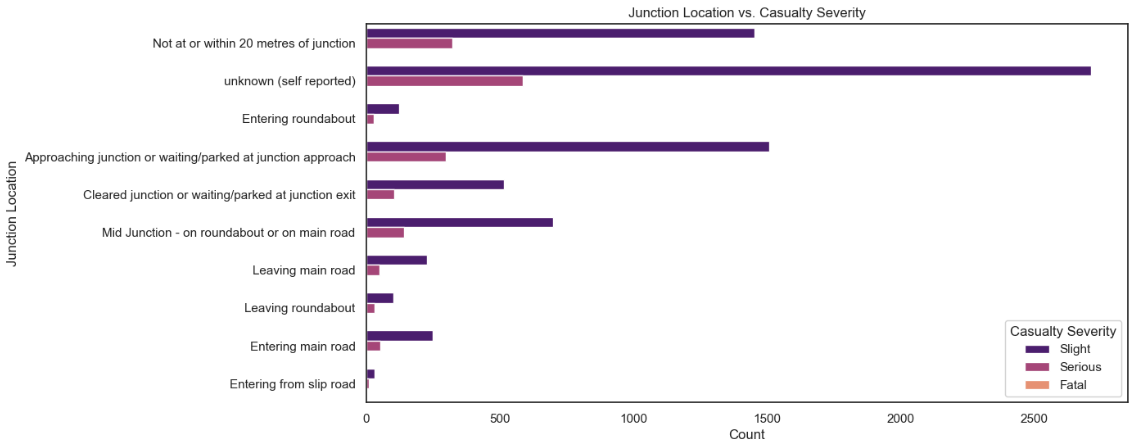
In [40]:
```
# Getting the value counts of the 'junction_location' variable
# This counts the occurrences of each unique value in the 'junction_locatic
values = pd.DataFrame(trainset['junction_location'].value_counts())
values.columns = ['Junction Location Count']  # Renaming the column for clc

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['junction_location'].value_counts
percentages.columns = ['% Junction Location']  # Renaming the column for cl

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Junction Location'] = percentages['% Junction Location'].map

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fc
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[40]:

	Junction Location Count	% Junction Location
unknown (self reported)	3300	35.65%
Approaching junction or waiting/parked at junction approach	1808	19.53%
Not at or within 20 metres of junction	1779	19.22%
Mid Junction - on roundabout or on main road	844	9.12%
Cleared junction or waiting/parked at junction exit	622	6.72%
Entering main road	303	3.27%
Leaving main road	277	2.99%
Entering roundabout	151	1.63%
Leaving roundabout	133	1.44%
Entering from slip road	40	0.43%

- "Unknown (self reported)" is the most common category (35.1%), indicating a significant portion of data lacks specific junction location details.
- Incidents are also frequent "Approaching junction" (19.79%) and "Not at or within 20 metres of junction" (19.28%), suggesting these areas may require further safety analysis.

In [41]:
```
# Creating a count plot to visualize the relationship between 'junction_loc
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(12, 6))  # Setting the figure size for better visualiza
sns.countplot(y=trainset['junction_location'], hue=trainset['casualty_sever
plt.title('Junction Location vs. Casualty Severity')  # Adding a title to t
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Junction Location')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- "Unknown (self reported)" has the highest incident counts, with most casualties being Slight, indicating a lack of detailed location data for many incidents.

3.2.2.3 Skidding and Overturning

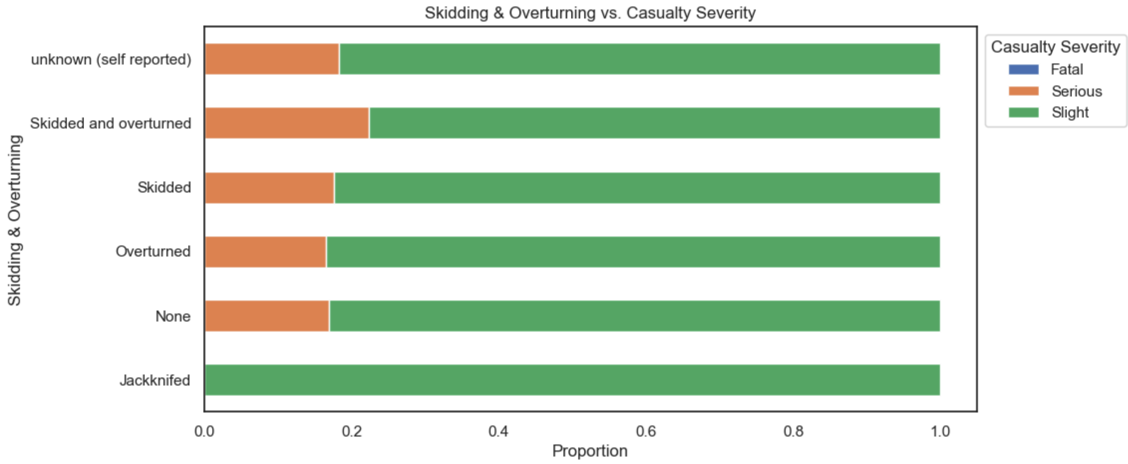
In [42]:
```
# Getting the value counts of the 'skidding_and_overturning' variable
# This counts the occurrences of each unique value in the 'skidding_and_ove
values = pd.DataFrame(trainset['skidding_and_overturning'].value_counts())
values.columns = ['Skidding and Overturning Count']  # Renaming the column

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['skidding_and_overturning'].value
percentages.columns = ['% Skidding and Overturning']  # Renaming the columr

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Skidding and Overturning'] = percentages['% Skidding and Ove

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[42]:

	Skidding and Overturning Count	% Skidding and Overturning
None	4861	52.51%
unknown (self reported)	3944	42.61%
Skidded	238	2.57%
Overturned	127	1.37%
Skidded and overturned	85	0.92%
Jackknifed	2	0.02%

- "None" is the most common category (52.95%), indicating that most incidents did not involve skidding or overturning.
- A significant portion of data is labeled as "unknown (self reported)" (42.11%), highlighting potential gaps in reporting or data collection.

In [43]:
```
# Creating a cross-tabulation (contingency table) to analyze the relationsh
# 'skidding_and_overturning' and 'casualty_severity'
# The normalize='index' parameter computes proportions row-wise (i.e., for
cross_tab = pd.crosstab(trainset['skidding_and_overturning'], trainset['cas

# Plotting the cross-tabulation as a stacked horizontal bar chart
cross_tab.plot(kind='barh', stacked=True, figsize=(10, 5))
plt.title('Skidding & Overturning vs. Casualty Severity')  # Adding a title
plt.xlabel('Proportion')  # Labeling the x-axis (proportions)
plt.ylabel('Skidding & Overturning')  # Labeling the y-axis (skidding/overt
plt.legend(title="Casualty Severity", bbox_to_anchor=(1, 1))  # Adding a le
plt.show()  # Displaying the plot
```



- "Skidded and overturned" has the highest proportion of Serious and Fatal incidents, making it the most severe scenario.
- "None" (no skidding or overturning) has the highest proportion of Slight incidents, indicating less severe outcomes in such cases.

3.2.2.4 Vehicle Leaving Carriageway

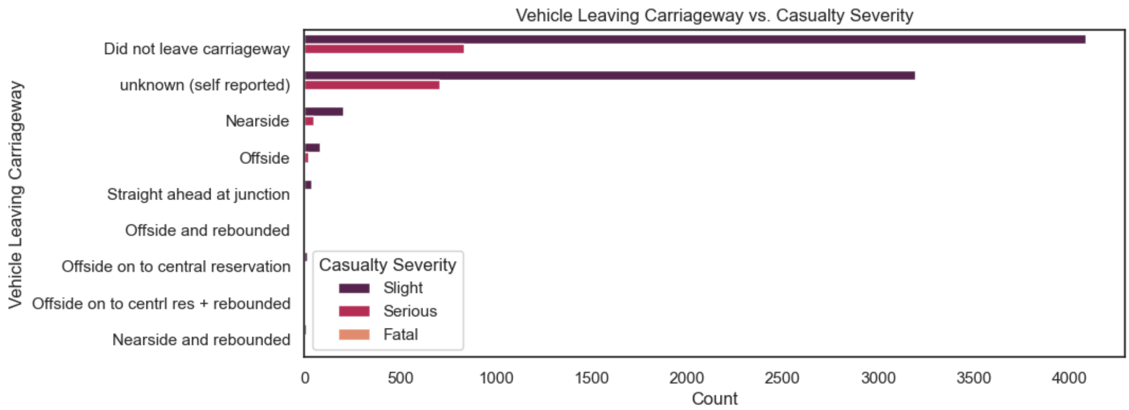
In [44]:
```
# Getting the value counts of the 'vehicle_leaving_carriageway' variable
# This counts the occurrences of each unique value in the 'vehicle_leaving_
values = pd.DataFrame(trainset['vehicle_leaving_carriageway'].value_counts(
values.columns = ['Vehicle Leaving Carriageway Count']  # Renaming the colu

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['vehicle_leaving_carriageway'].va
percentages.columns = ['% Vehicle Leaving Carriageway']  # Renaming the col

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Vehicle Leaving Carriageway'] = percentages['% Vehicle Leavi

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[44]:

	Vehicle Leaving Carriageway Count	% Vehicle Leaving Carriageway
Did not leave carriageway	4924	53.19%
unknown (self reported)	3901	42.14%
Nearside	251	2.71%
Offside	99	1.07%
Straight ahead at junction	39	0.42%
Offside on to central reservation	21	0.23%
Nearside and rebounded	12	0.13%
Offside and rebounded	8	0.09%
Offside on to centrl res + rebounded	2	0.02%

- "Did not leave carriageway" is the most common category (53.63%), indicating that in most incidents, vehicles remained on the carriageway.

In [45]:
```
# Creating a count plot to visualize the relationship between 'vehicle_leav
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(10, 4))  # Setting the figure size for better visualiz
sns.countplot(y=trainset['vehicle_leaving_carriageway'], hue=trainset['casu
plt.title('Vehicle Leaving Carriageway vs. Casualty Severity')  # Adding a
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Vehicle Leaving Carriageway')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- "Did not leave carriageway" has the highest incident counts, with most casualties being Slight, indicating less severe outcomes when vehicles remain on the carriageway.

3.2.2.5 Vehicle Type

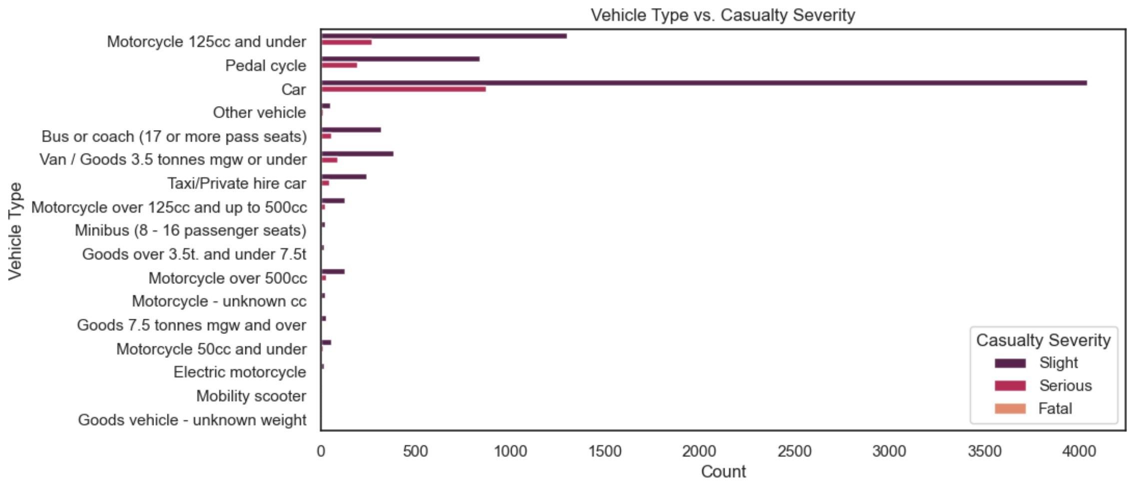
In [46]:
```
# Getting the value counts of the 'vehicle_type' variable
# This counts the occurrences of each unique value in the 'vehicle_type' co
values = pd.DataFrame(trainset['vehicle_type'].value_counts())
values.columns = ['Vehicle Type Count']  # Renaming the column for clarity

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['vehicle_type'].value_counts(norm
percentages.columns = ['% Vehicle Type']  # Renaming the column for clarity

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Vehicle Type'] = percentages['% Vehicle Type'].map(lambda x:

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[46]:

	Vehicle Type Count	% Vehicle Type
Car	4914	53.08%
Motorcycle 125cc and under	1569	16.95%
Pedal cycle	1036	11.19%
Van / Goods 3.5 tonnes mgw or under	473	5.11%
Bus or coach (17 or more pass seats)	375	4.05%
Taxi/Private hire car	292	3.15%
Motorcycle over 500cc	158	1.71%
Motorcycle over 125cc and up to 500cc	153	1.65%
Other vehicle	67	0.72%
Motorcycle 50cc and under	67	0.72%
Goods 7.5 tonnes mgw and over	36	0.39%
Motorcycle - unknown cc	31	0.33%
Minibus (8 - 16 passenger seats)	28	0.3%
Electric motorcycle	26	0.28%
Goods over 3.5t. and under 7.5t	17	0.18%
Goods vehicle - unknown weight	8	0.09%
Mobility scooter	7	0.08%

- Cars are the most common vehicle type involved in incidents (53.04%), followed by Motorcycles 125cc and under (16.98%).

In [47]:
```
# Creating a count plot to visualize the relationship between 'vehicle_type
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(10, 5))  # Setting the figure size for better visualiz
sns.countplot(y=trainset['vehicle_type'], hue=trainset['casualty_severity']
plt.title('Vehicle Type vs. Casualty Severity')  # Adding a title to the pl
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Vehicle Type')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- Cars have the highest incident counts, with most casualties being Slight, indicating less severe outcomes for car-related incidents.

3.2.2.6 Towing and Articulation

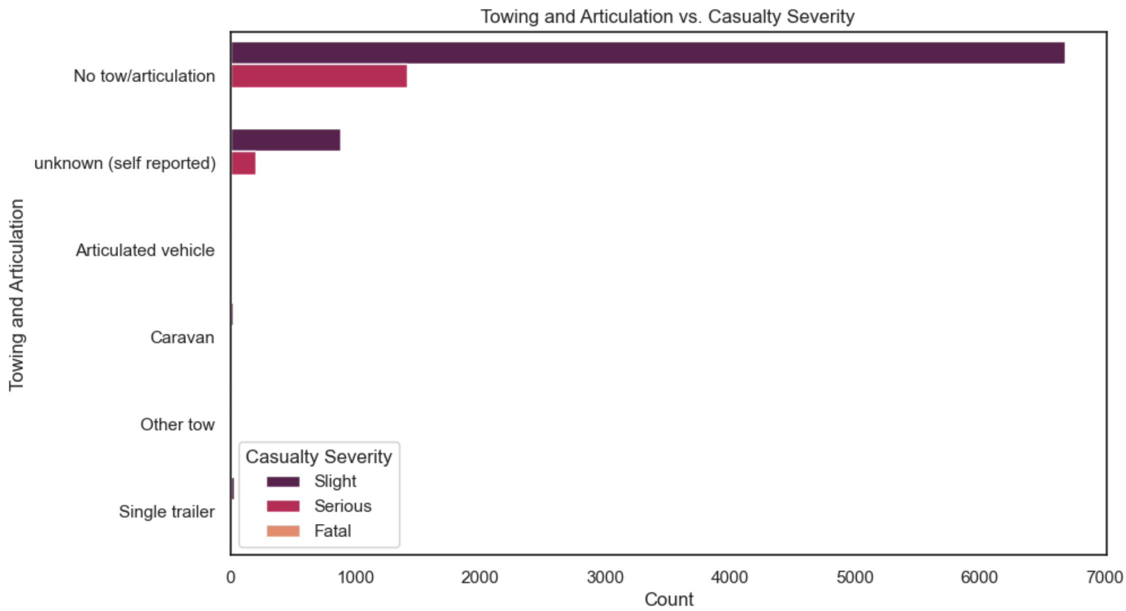
In [48]:
```
# Getting the value counts of the 'towing_and_articulation' variable
# This counts the occurrences of each unique value in the 'towing_and_artic
values = pd.DataFrame(trainset['towing_and_articulation'].value_counts())
values.columns = ['Towing and Articulation Count']  # Renaming the column f

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['towing_and_articulation'].value_
percentages.columns = ['% Towing and Articulation']  # Renaming the column

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Towing and Articulation'] = percentages['% Towing and Articu

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[48]:

	Towing and Articulation Count	% Towing and Articulation
No tow/articulation	8094	87.44%
unknown (self reported)	1079	11.66%
Single trailer	33	0.36%
Caravan	21	0.23%
Other tow	19	0.21%
Articulated vehicle	11	0.12%

- "No tow/articulation" is the most common category (87.92%), indicating that most vehicles involved in incidents were not towing or articulated.

In [49]:
```
# Creating a count plot to visualize the relationship between 'towing_and_a
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(10, 6))  # Setting the figure size for better visualiza
sns.countplot(y=trainset['towing_and_articulation'], hue=trainset['casualty
plt.title('Towing and Articulation vs. Casualty Severity')  # Adding a titl
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Towing and Articulation')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- "No tow/articulation" has the highest incident counts, with most casualties being Slight, indicating less severe outcomes for vehicles not towing or articulated.

3.2.2.7 Vehicle Location Restricted

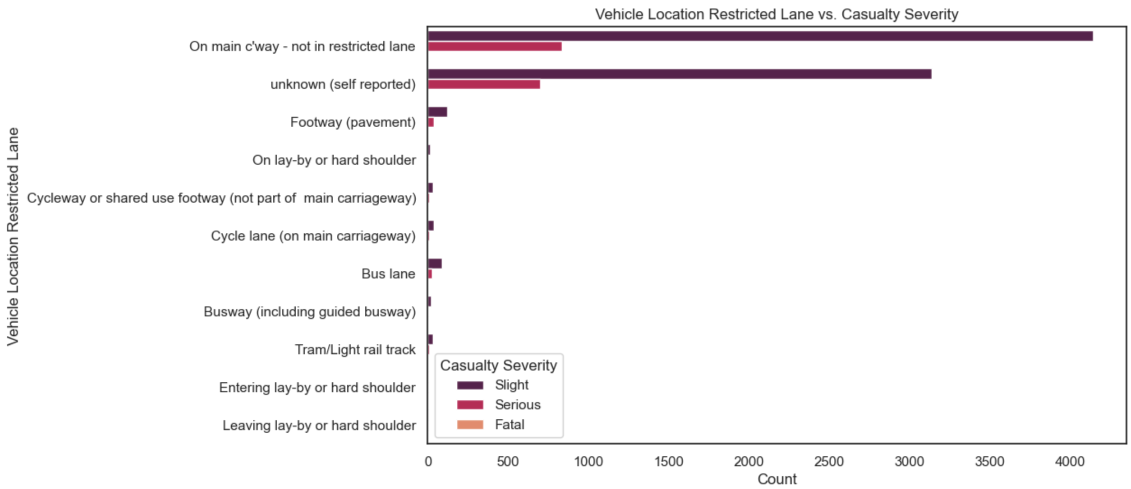
In [50]:
```
# Getting the value counts of the 'vehicle_location_restricted_lane' variab
# This counts the occurrences of each unique value in the 'vehicle_locatior
values = pd.DataFrame(trainset['vehicle_location_restricted_lane'].value_cc
values.columns = ['Vehicle Location Restricted Lane Count']  # Renaming the

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['vehicle_location_restricted_lane
percentages.columns = ['% Vehicle Location Restricted Lane']  # Renaming th

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Vehicle Location Restricted Lane'] = percentages['% Vehicle 

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[50]:

	Vehicle Location Restricted Lane Count	% Vehicle Location Restricted Lane
On main c'way - not in restricted lane	4978	53.78%
unknown (self reported)	3834	41.42%
Footway (pavement)	158	1.71%
Bus lane	112	1.21%
Cycle lane (on main carriageway)	43	0.46%
Cycleway or shared use footway (not part of main carriageway)	41	0.44%
Tram/Light rail track	39	0.42%
Busway (including guided busway)	23	0.25%
On lay-by or hard shoulder	14	0.15%
Entering lay-by or hard shoulder	8	0.09%
Leaving lay-by or hard shoulder	7	0.08%

- "On main carriageway - not in restricted lane" is the most common category (54.22%), indicating that most incidents occur on the main carriageway outside restricted lanes.

In [51]:
```
# Creating a count plot to visualize the relationship between 'vehicle_loca
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(10, 6))  # Setting the figure size for better visualiza
sns.countplot(y=trainset['vehicle_location_restricted_lane'], hue=trainset[
plt.title('Vehicle Location Restricted Lane vs. Casualty Severity')  # Addi
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Vehicle Location Restricted Lane')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- "On main carriageway - not in restricted lane" has the highest incident counts, with most casualties being Slight, indicating less severe outcomes for incidents on the main carriageway.

3.2.2.8 Hit Object in Carriageway

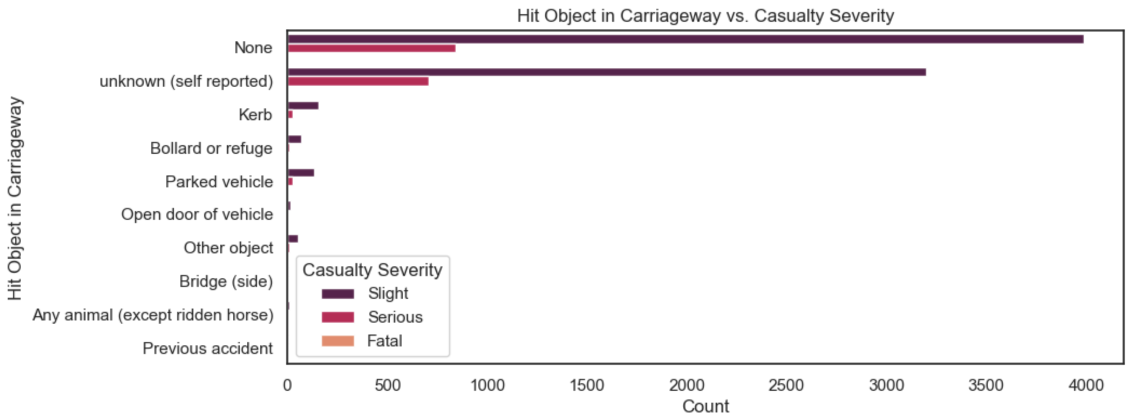
In [52]:
```
# Getting the value counts of the 'hit_object_in_carriageway' variable
# This counts the occurrences of each unique value in the 'hit_object_in_ca
values = pd.DataFrame(trainset['hit_object_in_carriageway'].value_counts()
values.columns = ['Hit Object in Carriageway Count']  # Renaming the columr

# The normalize attribute in the value_counts method computes the percentag
# This calculates the relative frequency (percentage) of each unique value
percentages = pd.DataFrame(round(trainset['hit_object_in_carriageway'].valu
percentages.columns = ['% Hit Object in Carriageway']  # Renaming the colum

# Converting the percentage values to a string format with a '%' sign for b
percentages['% Hit Object in Carriageway'] = percentages['% Hit Object in C

# Joining the values and percentages dataframes to create a summary table
# This combines the count and percentage information into a single table fo
summary_table = values.join(percentages)

# Displaying the summary table
summary_table
```

Out[52]:

	Hit Object in Carriageway Count	% Hit Object in Carriageway
None	4835	52.23%
unknown (self reported)	3905	42.18%
Kerb	182	1.97%
Parked vehicle	158	1.71%
Bollard or refuge	80	0.86%
Other object	62	0.67%
Open door of vehicle	18	0.19%
Any animal (except ridden horse)	10	0.11%
Previous accident	4	0.04%
Bridge (side)	3	0.03%

- "None" is the most common category (52.59%), indicating that in most incidents, no object was hit in the carriageway.

In [53]:
```
# Creating a count plot to visualize the relationship between 'hit_object_i
# The plot uses a horizontal orientation (y-axis) for better readability of
plt.figure(figsize=(10, 4))  # Setting the figure size for better visualiza
sns.countplot(y=trainset['hit_object_in_carriageway'], hue=trainset['casual
plt.title('Hit Object in Carriageway vs. Casualty Severity')  # Adding a ti
plt.xlabel('Count')  # Labeling the x-axis
plt.ylabel('Hit Object in Carriageway')  # Labeling the y-axis
plt.legend(title="Casualty Severity")  # Adding a legend to distinguish sev
plt.show()  # Displaying the plot
```



- "Unknown (self reported)" has the highest incident counts, with most casualties being Slight, indicating a lack of detailed information for many incidents.

3.2.2.9 First point of impact

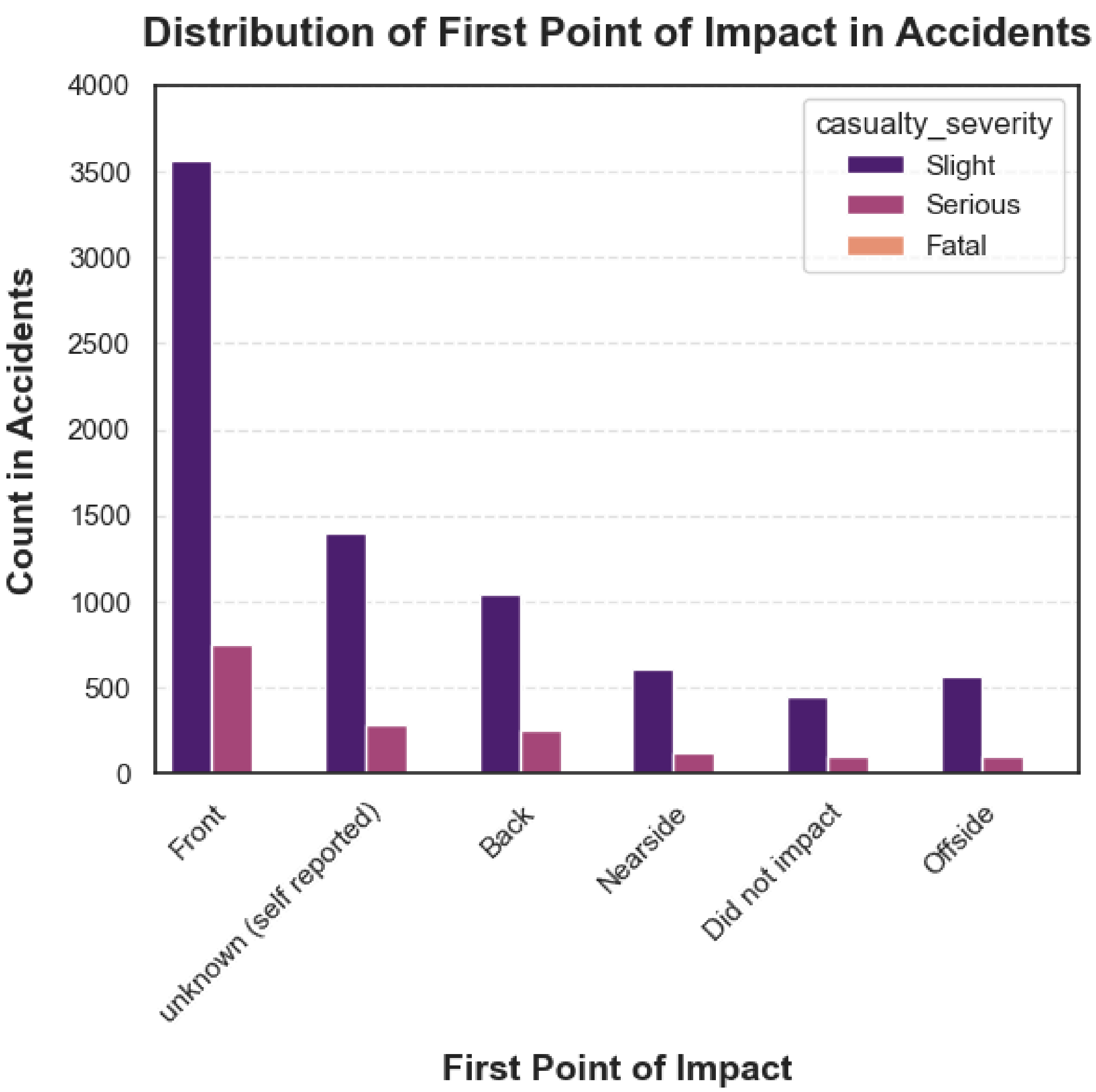
In [54]:
```
pd.DataFrame(trainset.loc[:,'first_point_of_impact'].describe())
```

Out[54]:

	first_point_of_impact
count	9257
unique	6
top	Front
freq	4318

The variable is categorised into six types, with "Front" being the most common. This suggests that frontal collisions are the most frequent, likely due to head-on crashes, rear-end collisions, or sudden braking scenarios. Understanding impact types can help improve vehicle safety measures and accident prevention strategies.

In [55]:
```
# Set Seaborn style
sns.set_style("white")

# Define plot object with improved style
count_plot = sns.countplot(x='first_point_of_impact', data=trainset, palett

# Set title and labels with better formatting
count_plot.set_title('Distribution of First Point of Impact in Accidents',
plt.xlabel('First Point of Impact', fontsize=14, fontweight='bold', labelpa
plt.ylabel('Count in Accidents', fontsize=14, fontweight='bold', labelpad=1


# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Add horizontal gridlines for better readability
plt.grid(axis='y', linestyle='--', alpha=0.5)

# Show the plot
plt.ylim(0, 4000)
plt.show()
```



The front of the vehicle is the most common first point of impact in accidents, suggesting frequent head-on or rear-end collisions. Self-reported unknown impacts are the second highest, indicating potential data uncertainty. Rear, nearside, offside, and non-impact cases occur less frequently.

3.2.2.10 Vehicle left hand drive

In [56]:
```
pd.DataFrame(trainset.loc[:,'vehicle_left_hand_drive'].describe())
```

Out[56]:

	vehicle_left_hand_drive
count	9257
unique	3
top	No
freq	7027

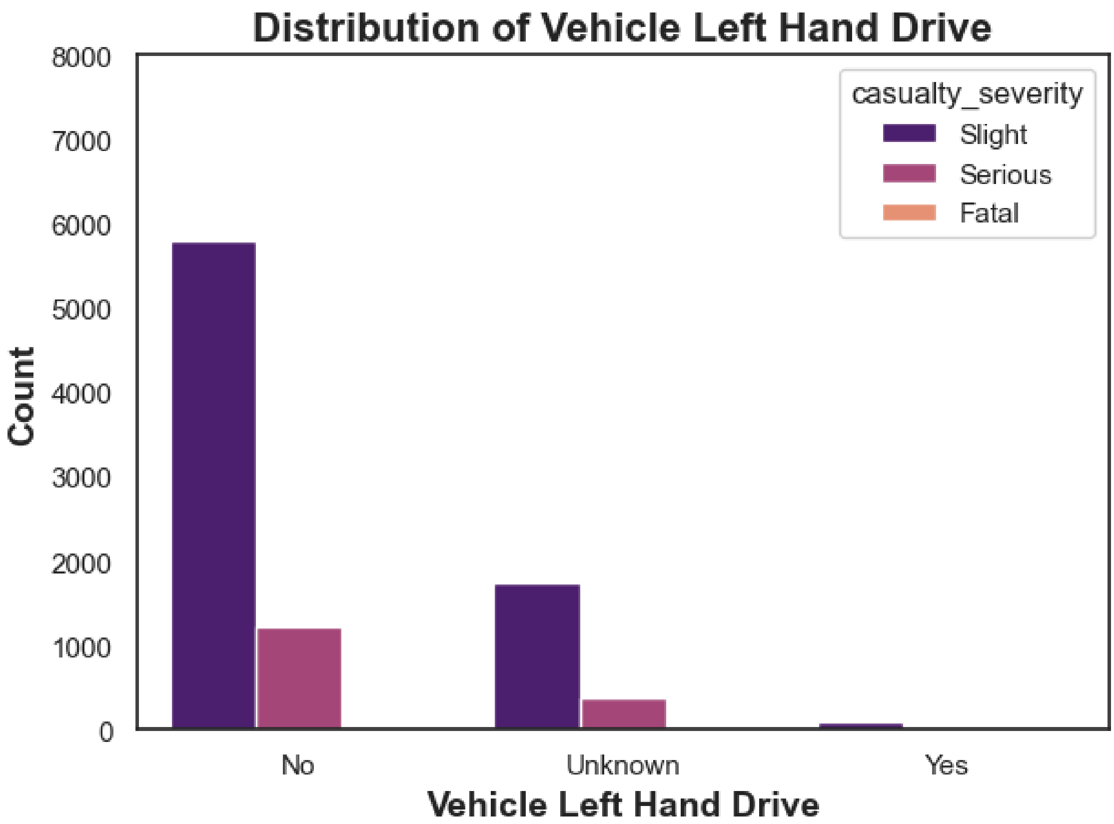
Most vehicles are not left-hand drive, indicating that right-hand drive vehicles dominate in this dataset. The presence of left-hand drive vehicles suggests international or imported cars.

In [57]:
```
# Getting the value counts of the 'vehicle_left_hand_drive' variable
values = pd.DataFrame(trainset['vehicle_left_hand_drive'].value_counts())
values.columns = ['vehicle_left_hand_drive Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['vehicle_left_hand_drive'].value_
percentages.columns = ['%vehicle_left_hand_drive']

# Converting the percentage values to a string format with a '%' sign
percentages['%vehicle_left_hand_drive'] = percentages['%vehicle_left_hand_d

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[57]:

	vehicle_left_hand_drive Count	%vehicle_left_hand_drive
No	7027	75.91%
Unknown	2118	22.88%
Yes	112	1.21%

In [58]:
```
# Bar Chart distribution
sns.set(style="white")

count_plot = sns.countplot(x='vehicle_left_hand_drive', data=trainset, pale

# Setting the title and labels
count_plot.set_title('Distribution of Vehicle Left Hand Drive', fontsize=16
plt.xlabel('Vehicle Left Hand Drive', fontsize=14, fontweight='bold')
plt.ylabel('Count', fontsize=14, fontweight='bold')


# Show the plot
plt.ylim(0, 8000)
plt.tight_layout()
plt.show()
```



Most vehicles in the dataset are right-hand drive, making up the majority. Only 112 left-hand drive vehicles exist, likely imports or special-use cars.

3.2.2.11 Journey purpose of driver

In [59]:
```
pd.DataFrame(trainset.loc[:,'journey_purpose_of_driver'].describe())
```

Out[59]:

	journey_purpose_of_driver
count	9257
unique	5
top	Not known
freq	5962

The journey purpose of the driver is mostly unknown, indicating data gaps in accident reporting. Among the five categories, the remaining provide meaningful insights into whether travel purpose affects accident risk.

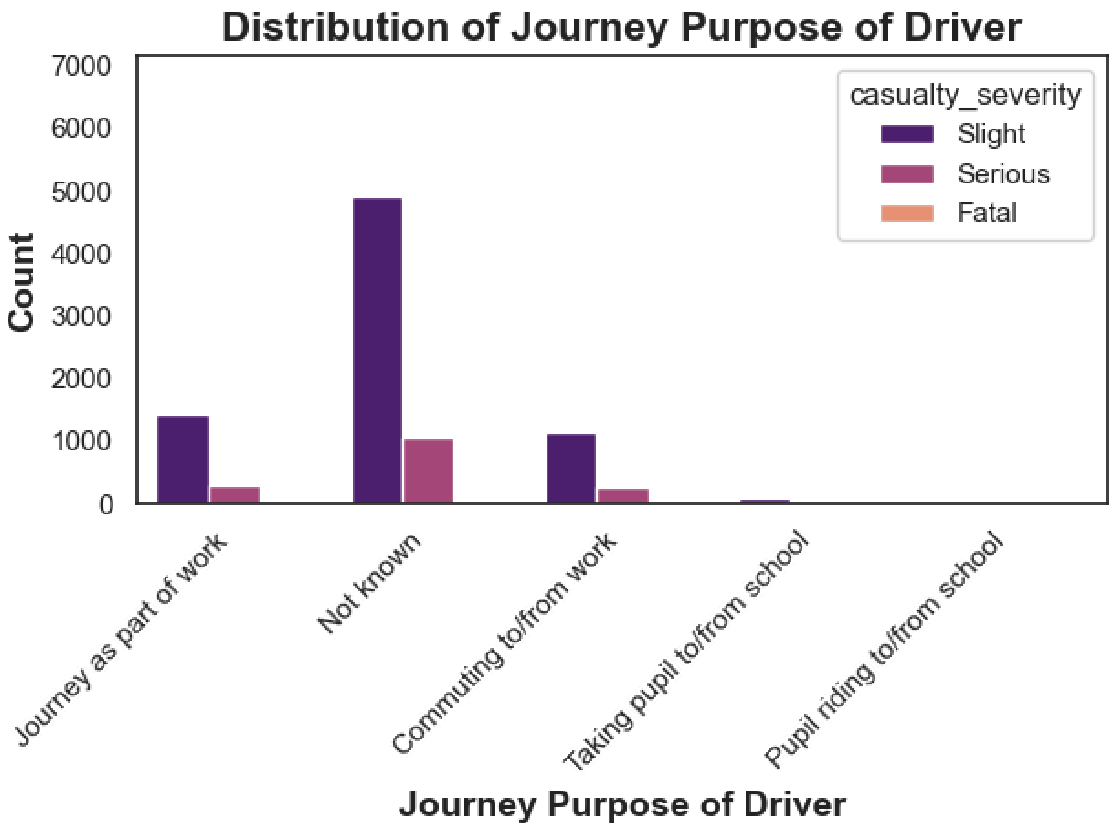
In [60]:
```
# Getting the value counts of the 'journey_purpose_of_driver' variable
values = pd.DataFrame(trainset['journey_purpose_of_driver'].value_counts()
values.columns = ['journey_purpose_of_driver Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['journey_purpose_of_driver'].valu
percentages.columns = ['%journey_purpose_of_driver']

# Converting the percentage values to a string format with a '%' sign
percentages['%journey_purpose_of_driver'] = percentages['%journey_purpose_o

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[60]:

	journey_purpose_of_driver Count	%journey_purpose_of_driver
Not known	5962	64.41%
Journey as part of work	1737	18.76%
Commuting to/from work	1409	15.22%
Taking pupil to/from school	130	1.4%
Pupil riding to/from school	19	0.21%

In [61]:

```
# Countplot for 'journey_purpose_of_driver'
sns.set(style="white")
count_plot = sns.countplot(x='journey_purpose_of_driver', data=trainset, pa

# Set title and labels
count_plot.set_title('Distribution of Journey Purpose of Driver', fontsize=
plt.xlabel('Journey Purpose of Driver', fontsize=14, fontweight='bold')
plt.ylabel('Count', fontsize=14, fontweight='bold')

# Adjust the y-axis to make sure smaller categories are visible
plt.ylim(0, trainset['journey_purpose_of_driver'].value_counts().max() * 1.

# Rotate x-axis labels because they are long
plt.xticks(rotation=45, ha='right')


# Show the plot
plt.tight_layout()
plt.show()
```



The majority of journey purposes are unknown. Among known cases, work-related journeys (part of work or commuting) dominate, suggesting higher accident risk for working drivers. School-related trips are rare, possibly indicating safer travel conditions or underreporting.

3.2.2.12 Sex of driver

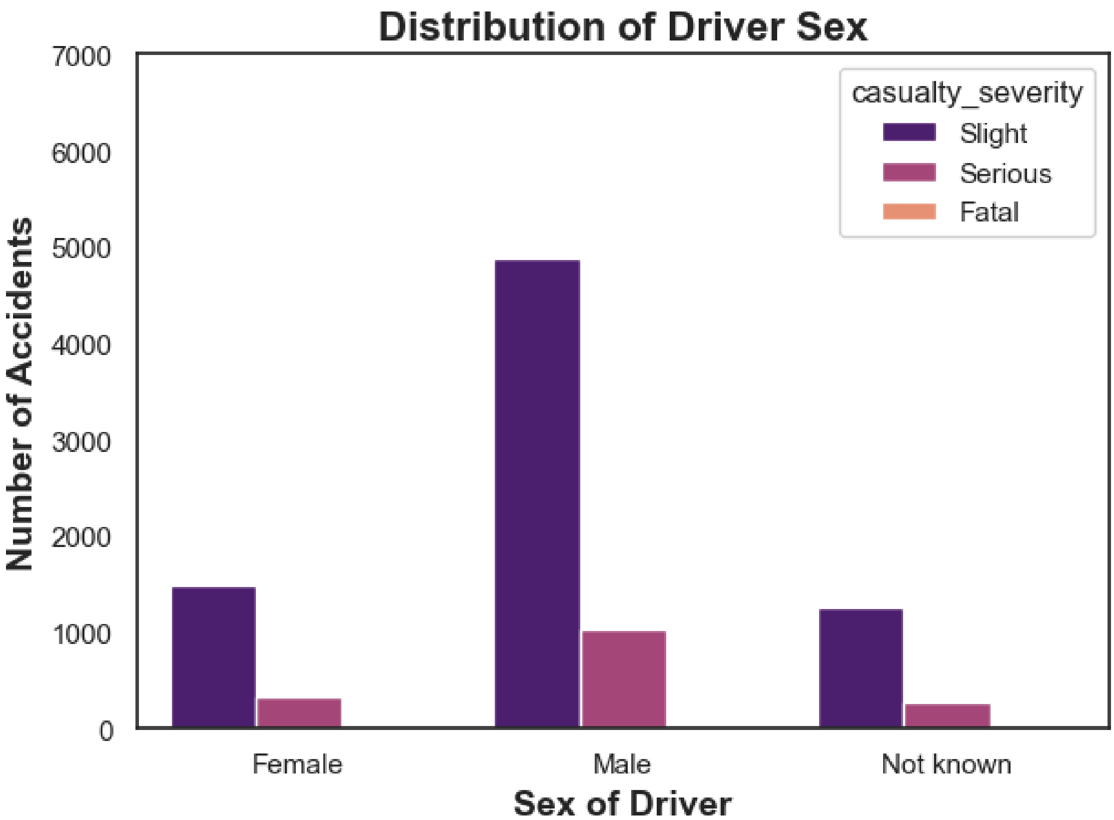
In [62]:
```
pd.DataFrame(trainset.loc[:,'sex_of_driver'].describe())
```

Out[62]:

	sex_of_driver
count	9257
unique	3
top	Male
freq	5913

Most drivers in the dataset are male, suggesting a higher involvement of men in reported accidents.

In [63]:
```
sns.set(style="white")

# Countplot
count_plot = sns.countplot(x='sex_of_driver', data=trainset, palette="magma

# Set the title and labels with larger font sizes for clarity
count_plot.set_title('Distribution of Driver Sex', fontsize=16, fontweight=
plt.xlabel('Sex of Driver', fontsize=14, fontweight='bold')
plt.ylabel('Number of Accidents', fontsize=14, fontweight='bold')

# Annotate each bar with the count value and percentage
total = len(trainset)  # Total number of rows in the dataset

for p in count_plot.patches:
    # Get the height of each bar (the count)
    height = p.get_height()

    # Calculate the percentage
    percentage = (height / total) * 100


# Show the plot
plt.ylim(0, 7000)
plt.tight_layout()
plt.show()
```



Most drivers in accidents are male (63.9%), significantly higher than female drivers (19.6%). This could indicate higher male driving exposure or riskier driving behavior.

3.2.2.13 Age band of driver

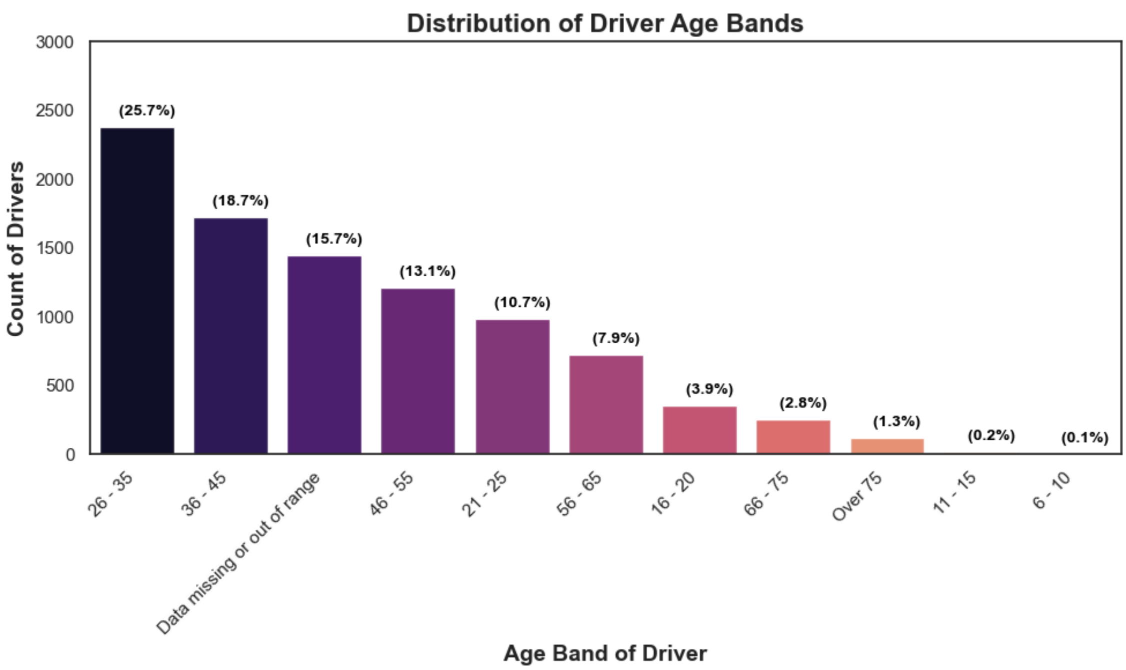
In [64]:
```
pd.DataFrame(trainset.loc[:,'age_band_of_driver'].describe())
```

Out[64]:

	age_band_of_driver
count	9257
unique	11
top	26 - 35
freq	2383

The 26-35 age group is the most common among drivers in accidents. With 11 unique age bands, accidents involve a diverse age range. Understanding whether younger or older drivers have higher accident severity could improve targeted road safety campaigns and driver training programs.

In [65]:
```
sns.set(style="white")

plt.figure(figsize=(10, 6))  # Adjust figure size to make the plot clearer

# Plot the countplot for 'age_band_of_driver'
count_plot = sns.countplot(x='age_band_of_driver', data=trainset, palette='

# Set the title and labels with larger font sizes for better readability
count_plot.set_title('Distribution of Driver Age Bands', fontsize=16, fontw
plt.xlabel('Age Band of Driver', fontsize=14, fontweight='bold')
plt.ylabel('Count of Drivers', fontsize=14, fontweight='bold')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')  # Rotate labels 45 degrees to avoid ov

# Calculate total number of drivers for percentage calculation
total = len(trainset)

# Annotate the bars with the count and percentage
for p in count_plot.patches:
    # Get the height of each bar (the count of each age band)
    height = p.get_height()

    # Calculate the percentage
    percentage = (height / total) * 100

    # Annotate with both count and percentage on top of the bars
    count_plot.annotate(f' ({percentage:.1f}%)',
                        (p.get_x() + p.get_width() / 2., height),  # Positi
                        ha='center', va='bottom',  # Horizontal center, ver
                        fontsize=10, fontweight='bold', color='black',
                        xytext=(5, 5), textcoords='offset points')  # Sligh

# Show the plot with tight layout to ensure labels and title fit properly
plt.ylim(0, 3000)
plt.tight_layout()
plt.show()
```



The 26-35 age group accounts for 25.7% of accidents, making them the most involved drivers. Accident frequency decreases with age, suggesting younger drivers may take more risks or have higher exposure.

3.2.2.14 Propulsion code

In [66]:
```
pd.DataFrame(trainset.loc[:,'propulsion_code'].describe())
```

Out[66]:

	propulsion_code
count	9257
unique	7
top	Petrol
freq	4484

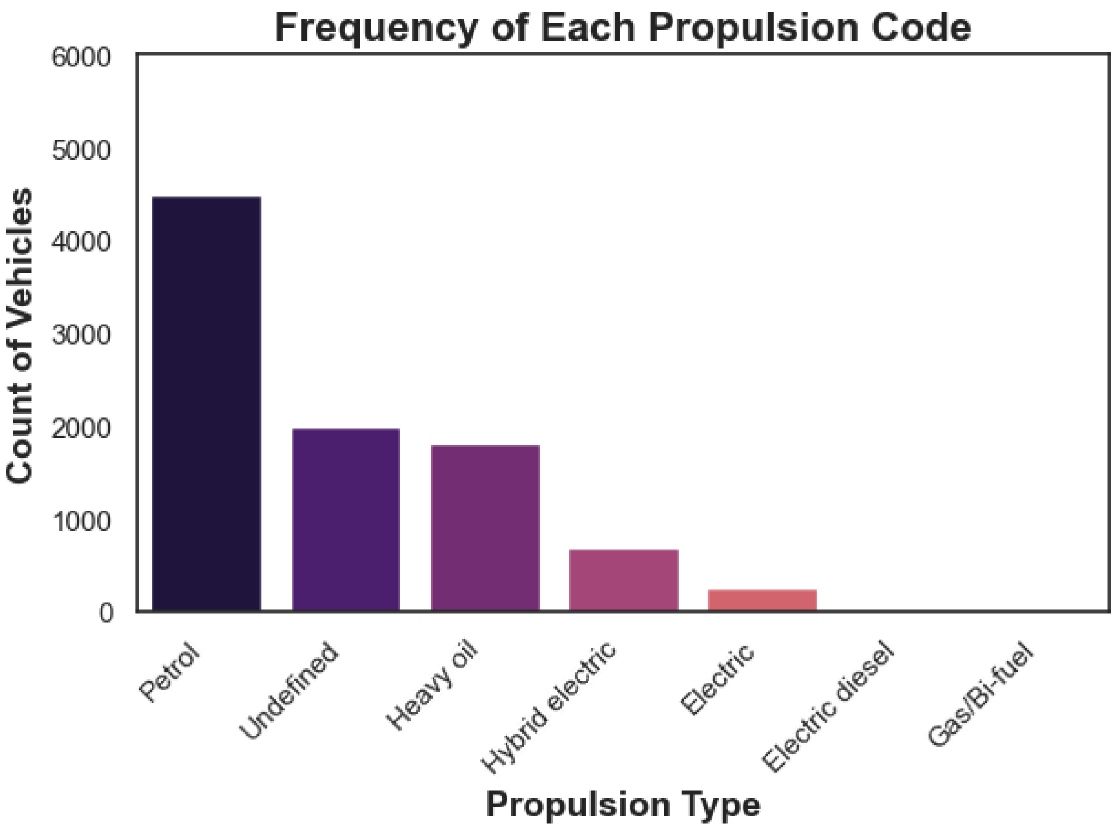
The majority of vehicles use petrol, followed by other propulsion types. With seven unique fuel types, the dataset represents a diverse vehicle mix.

In [67]:
```
# Getting the value counts of the 'propulsion_code' variable
values = pd.DataFrame(trainset['propulsion_code'].value_counts())
values.columns = ['propulsion_code Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['propulsion_code'].value_counts(r
percentages.columns = ['%propulsion_code']

# Converting the percentage values to a string format with a '%' sign
percentages['%propulsion_code'] = percentages['%propulsion_code'].map(lambc

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[67]:

	propulsion_code Count	%propulsion_code
Petrol	4484	48.44%
Undefined	1993	21.53%
Heavy oil	1816	19.62%
Hybrid electric	685	7.4%
Electric	254	2.74%
Electric diesel	21	0.23%
Gas/Bi-fuel	4	0.04%

In [68]:
```
# Bar plot with annotations for frequency
# Set the figure size
sns.set(style="white")
sns.countplot(x='propulsion_code', data=trainset, palette='magma')

# Title and axis labels
plt.title('Frequency of Each Propulsion Code', fontsize=16, fontweight='bol
plt.xlabel('Propulsion Type', fontsize=14, fontweight='bold')
plt.ylabel('Count of Vehicles', fontsize=14, fontweight='bold')

# Rotate x-axis labels for readability
plt.xticks(rotation=45, ha='right')


# Show the plot
plt.ylim(0, 6000)
plt.tight_layout()
plt.show()
```



Petrol-powered vehicles dominate, followed by heavy oil. Hybrid electric and electric vehicles are less common, suggesting limited adoption.

3.2.2.15 Generic make model

In [69]:
```
pd.DataFrame(trainset.loc[:,'generic_make_model'].describe())
```

Out[69]:

	generic_make_model
count	9257
unique	423
top	-1
freq	2185

The dataset contains 423 unique vehicle models, but "-1" appears most frequently, likely indicating missing or unclassified data.

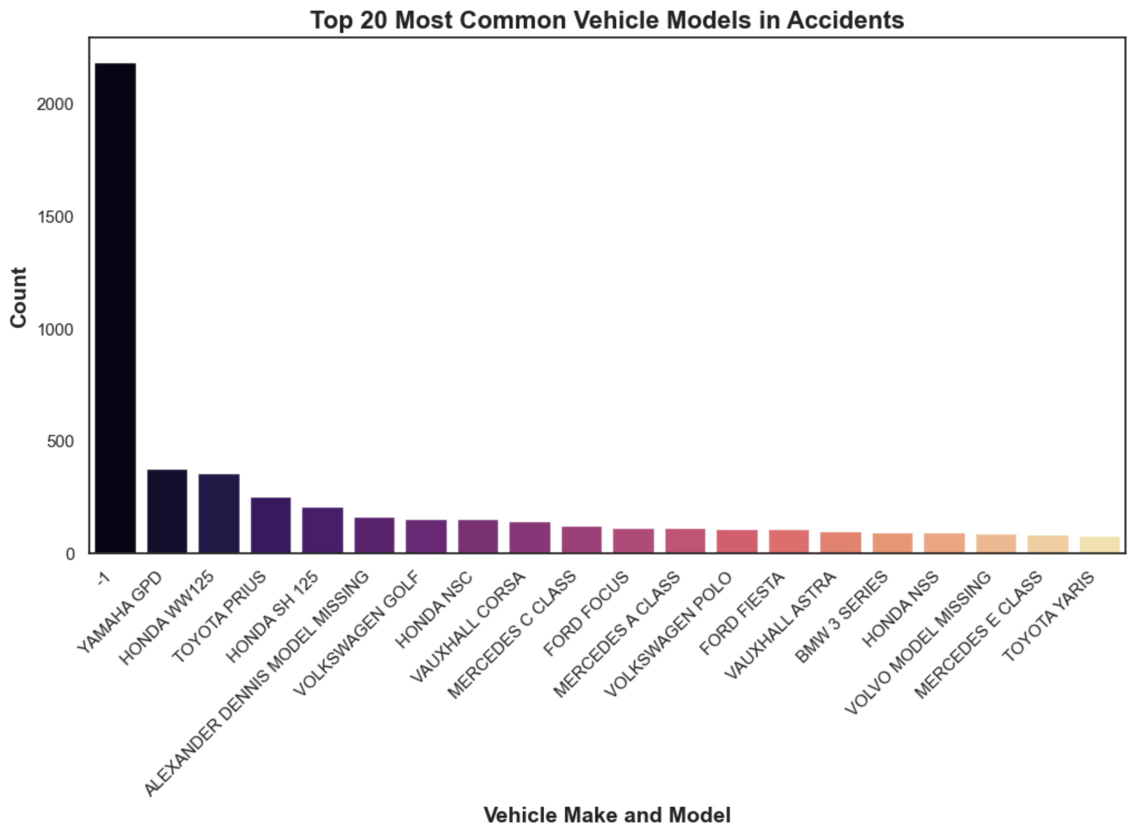
In [70]:
```
# Getting the value counts of the 'generic_make_model' variable
values = pd.DataFrame(trainset['generic_make_model'].value_counts())
values.columns = ['generic_make_model Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['generic_make_model'].value_count
percentages.columns = ['%generic_make_model']

# Converting the percentage values to a string format with a '%' sign
percentages['%generic_make_model'] = percentages['%generic_make_model'].map

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[70]:

	generic_make_model Count	%generic_make_model
-1	2185	23.6%
YAMAHA GPD	380	4.11%
HONDA WW125	361	3.9%
TOYOTA PRIUS	257	2.78%
HONDA SH 125	210	2.27%
...	...	...
LEXUS GS 450	1	0.01%
SEAT ALTEA	1	0.01%
PIAGGIO FLY	1	0.01%
JEEP GRAND CHEROKEE	1	0.01%
BMW S 1000	1	0.01%

423 rows × 2 columns

In [71]:
```
# Set the Seaborn style to a clean white background
sns.set(style="white")

# Create a figure with a specified size
plt.figure(figsize=(12, 6))

# Get the top 20 most common vehicle models in accidents
top_models = trainset['generic_make_model'].value_counts().nlargest(20)

# Create a bar plot with the "magma" color palette
sns.barplot(x=top_models.index, y=top_models.values, palette="magma")

# Set x-axis and y-axis labels
plt.xlabel("Vehicle Make and Model", fontsize=14, fontweight='bold')
plt.ylabel("Count", fontsize=14, fontweight='bold')

# Set the plot title with better readability
plt.title("Top 20 Most Common Vehicle Models in Accidents", fontsize=16, fo


# Rotate x-axis labels at 45 degrees for better readability
plt.xticks(rotation=45, ha='right')

# Display the plot
plt.show()
```



The most common entry is "-1", indicating a large amount of missing vehicle model data. Among known models, motorcycles (Yamaha, Honda WW125, Honda SH 125) and popular cars (Toyota Prius, Volkswagen Golf, Ford Focus) dominate. This suggests motorcycles may be overrepresented in accidents.

3.2.2.16 Driver IMD decile

In [72]:
```
pd.DataFrame(trainset.loc[:,'driver_imd_decile'].describe())
```

Out[72]:

	driver_imd_decile
count	9257
unique	11
top	Data missing or out of range
freq	1677

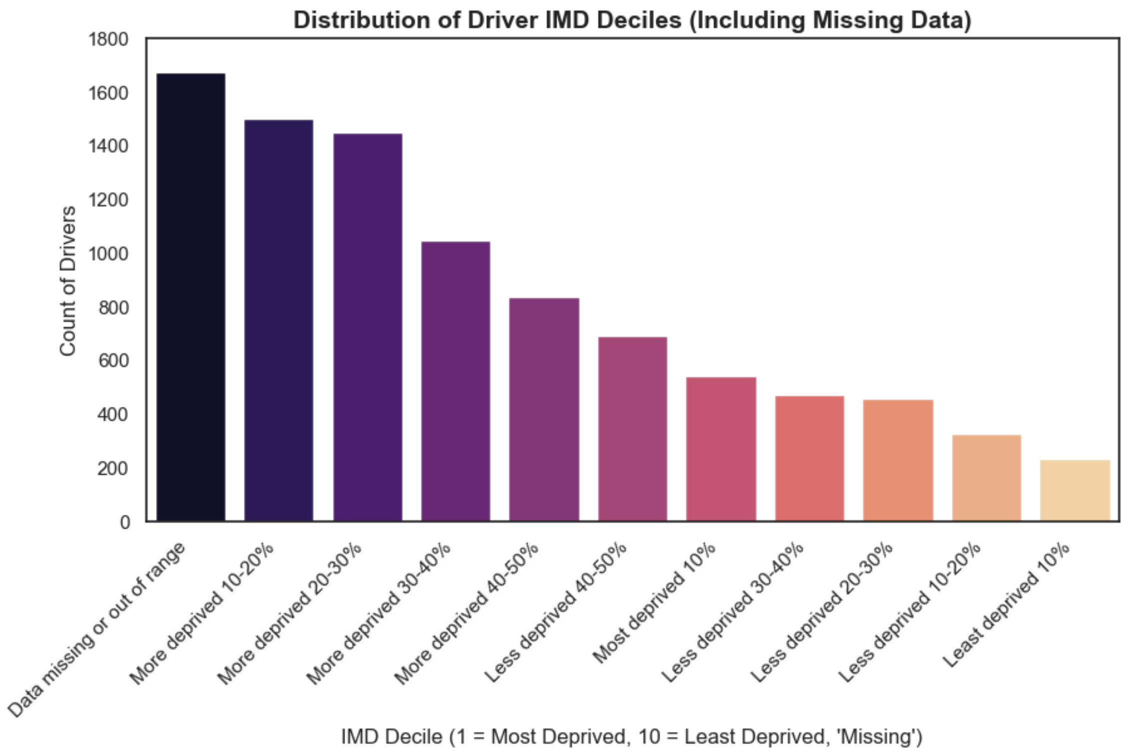
In [73]:
```
# Getting the value counts of the 'driver_imd_decile' variable
values = pd.DataFrame(trainset['driver_imd_decile'].value_counts())
values.columns = ['driver_imd_decile Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['driver_imd_decile'].value_counts
percentages.columns = ['%driver_imd_decile']

# Converting the percentage values to a string format with a '%' sign
percentages['%driver_imd_decile'] = percentages['%driver_imd_decile'].map(l

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[73]:

	driver_imd_decile Count	%driver_imd_decile
Data missing or out of range	1677	18.12%
More deprived 10-20%	1500	16.2%
More deprived 20-30%	1453	15.7%
More deprived 30-40%	1050	11.34%
More deprived 40-50%	839	9.06%
Less deprived 40-50%	694	7.5%
Most deprived 10%	545	5.89%
Less deprived 30-40%	473	5.11%
Less deprived 20-30%	460	4.97%
Less deprived 10-20%	329	3.55%
Least deprived 10%	237	2.56%

A significant portion of driver IMD decile data is missing. The 11 unique values indicate a wide socio-economic distribution among drivers. Investigating whether deprivation levels correlate with accident severity could provide insights into road safety disparities and targeted interventions for high-risk areas.

In [74]:
```
sns.set(style="white") # Seaborn style for a clean background.

plt.figure(figsize=(10, 5)) # Adjust the size

# Create a count plot with ordered categories
count_plot = sns.countplot(
    data=trainset,
    x='driver_imd_decile',
    order=trainset['driver_imd_decile'].value_counts().index,
    palette="magma"
)


# Set labels and title
plt.xlabel("IMD Decile (1 = Most Deprived, 10 = Least Deprived, 'Missing')"
plt.ylabel("Count of Drivers", fontsize=12)
plt.title("Distribution of Driver IMD Deciles (Including Missing Data)", fc

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.ylim(0,1800)
plt.show()
```



A large portion of driver IMD decile data is missing, affecting analysis. Most drivers come from more deprived areas (10-50%), while fewer are from less deprived areas (10-50%). This suggests higher accident involvement among drivers from deprived regions, warranting further investigation into socio-economic factors and road safety disparities.

3.2.2.17 Driver Home Area Type

In [75]:

```
pd.DataFrame(trainset.loc[:,'driver_home_area_type'].describe())
```

Out[75]:

	driver_home_area_type
count	9257
unique	4
top	Urban area
freq	7427

In [76]:

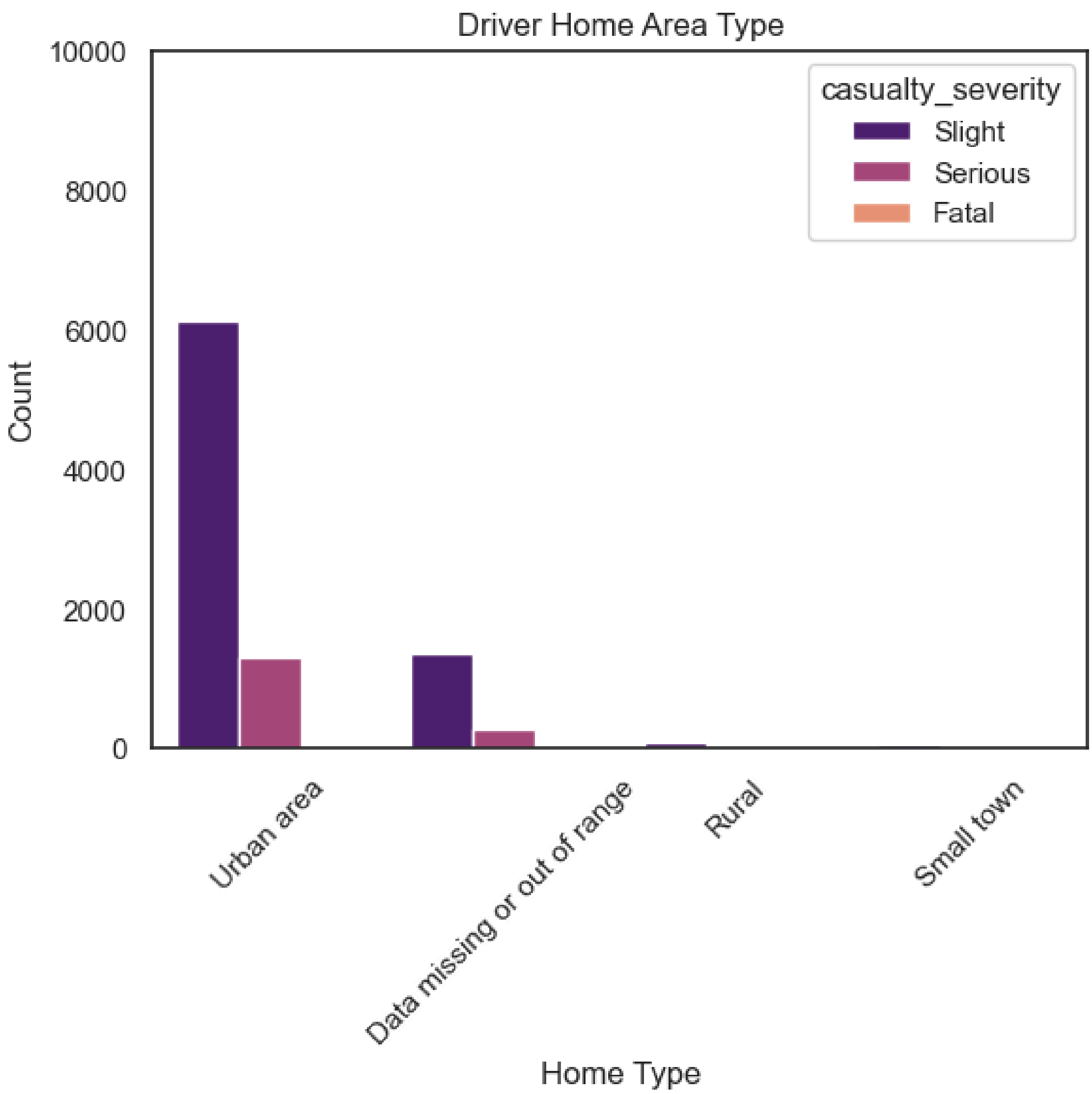
```
# Getting the value counts of the 'driver_home_area_type' variable
values = pd.DataFrame(trainset['driver_home_area_type'].value_counts())
values.columns = ['driver_home_area_type Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['driver_home_area_type'].value_co
percentages.columns = ['%driver_home_area_type']

# Converting the percentage values to a string format with a '%' sign
percentages['%driver_home_area_type'] = percentages['%driver_home_area_type

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[76]:

	driver_home_area_type Count	%driver_home_area_type
Urban area	7427	80.23%
Data missing or out of range	1657	17.9%
Rural	96	1.04%
Small town	77	0.83%

Majority of the drivers live in urban areas

In [77]:
```
# Define plot object with magma palette
count = sns.countplot(data=trainset, x='driver_home_area_type', hue='casual

# Setting graph title and labels
count.set_title('Driver Home Area Type')
count.set(xlabel='Home Type', ylabel='Count')


# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit

# Show the plot
plt.ylim(0,10000)
plt.show()
```



3.2.2.18 Sex_of_casualty

In [78]:
```
pd.DataFrame(trainset.loc[:,'sex_of_casualty'].describe())
```

Out[78]:

	sex_of_casualty
count	9257
unique	3
top	Male
freq	5873

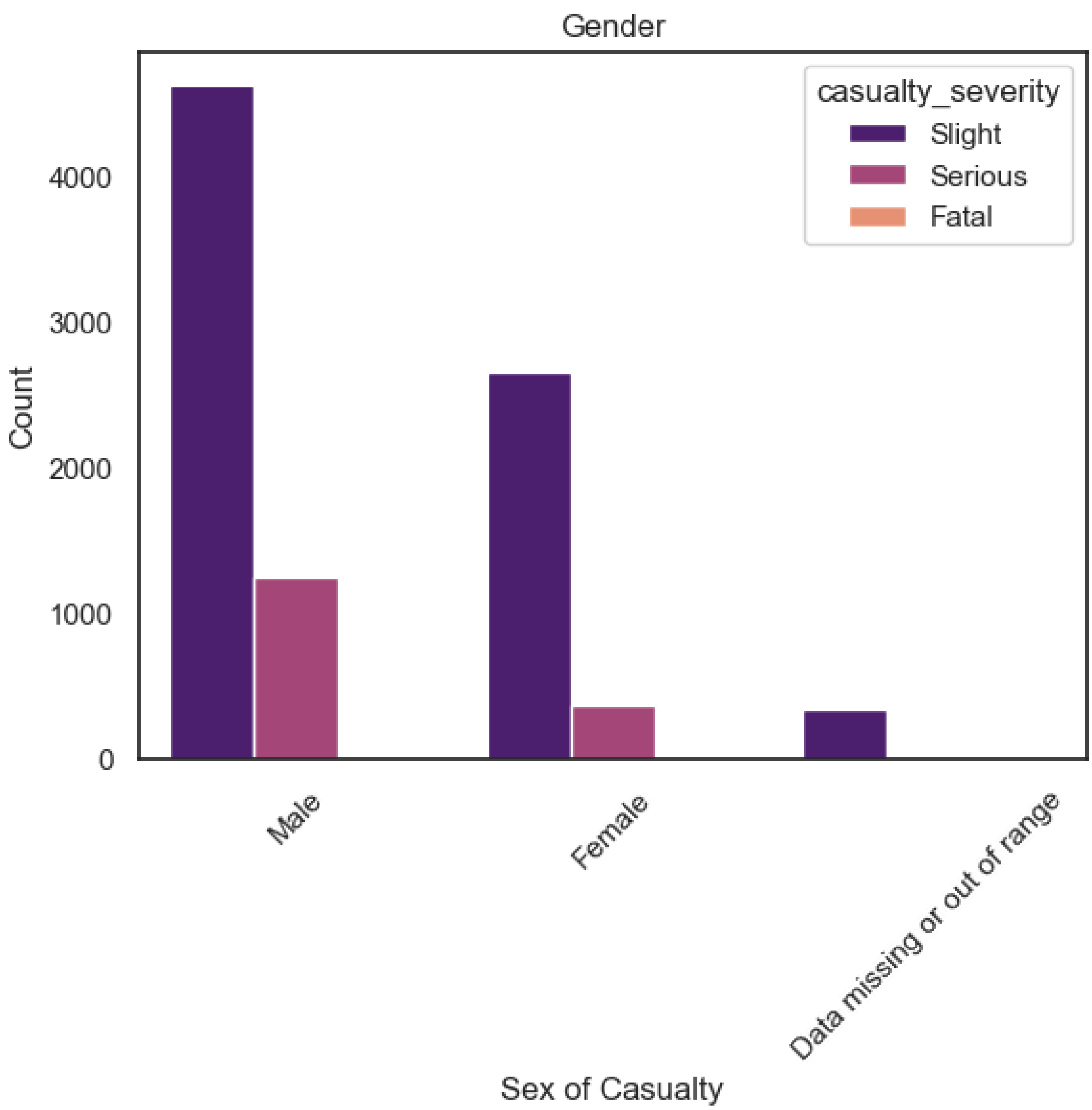
In [79]:
```
# Getting the value counts of the 'towing_and_articulation' variable
values = pd.DataFrame(trainset['sex_of_casualty'].value_counts())
values.columns = ['sex_of_casualty Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['sex_of_casualty'].value_counts(r
percentages.columns = ['%sex_of_casualty']

# Converting the percentage values to a string format with a '%' sign
percentages['%sex_of_casualty'] = percentages['%sex_of_casualty'].map(lambc

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[79]:

	sex_of_casualty Count	%sex_of_casualty
Male	5873	63.44%
Female	3033	32.76%
Data missing or out of range	351	3.79%

Majority of the casualties are male

In [80]:
```
# Define plot object with magma palette
count = sns.countplot(data=trainset, x='sex_of_casualty',hue='casualty_seve
# Setting graph title and labels
count.set_title('Gender')
count.set(xlabel='Sex of Casualty', ylabel='Count')


# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit

# Showing the plot **after** adding labels
plt.show()
```



3.2.2.19 Age_band_of_casualty

In [81]:
```
pd.DataFrame(trainset.loc[:,'age_band_of_casualty'].describe())
```

Out[81]:

	age_band_of_casualty
count	9257
unique	12
top	26 - 35
freq	2083

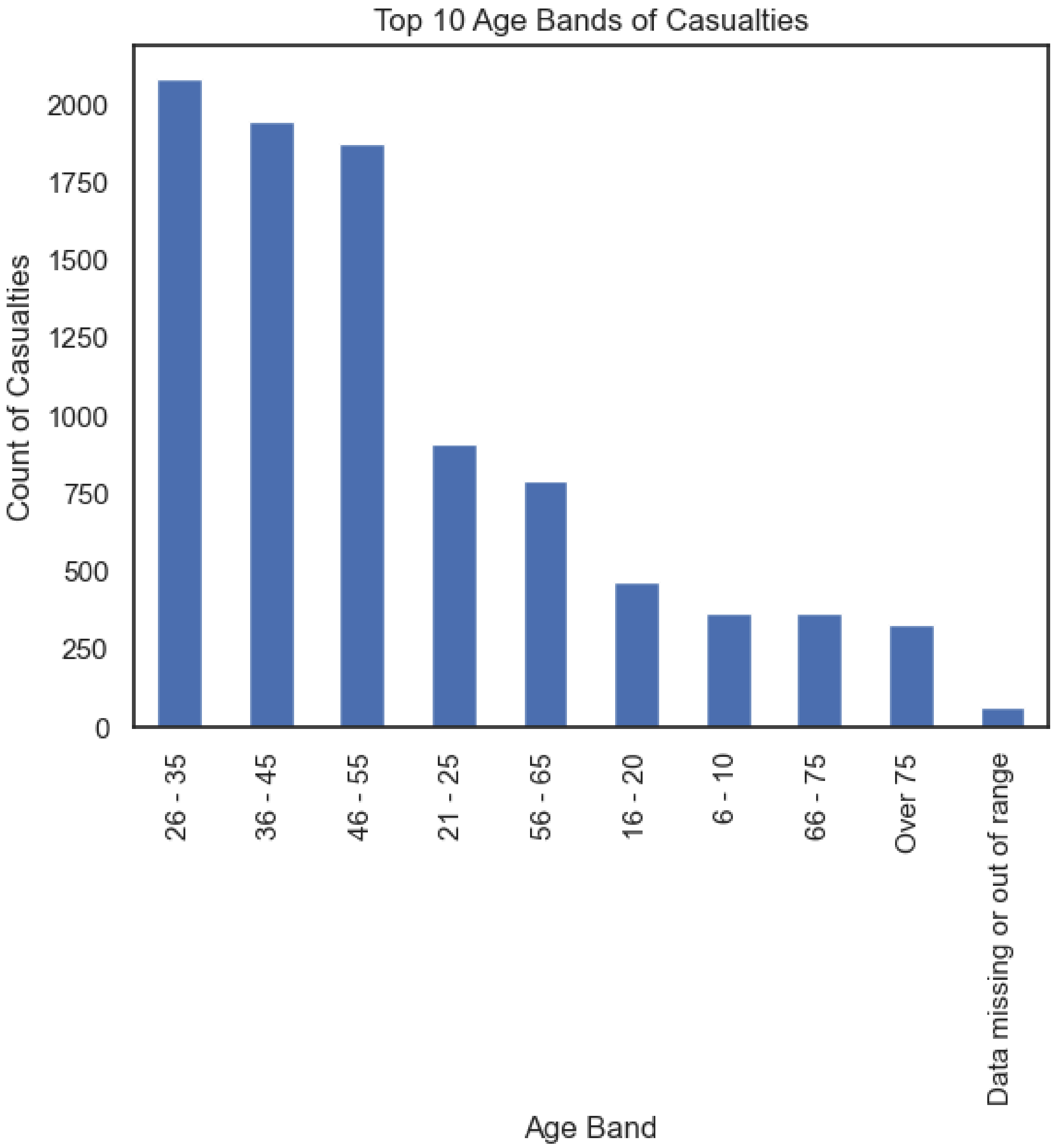
In [82]:
```
# Getting the value counts of the 'age_band_of_casualty' variable
values = pd.DataFrame(trainset['age_band_of_casualty'].value_counts())
values.columns = ['age_band_of_casualty Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['age_band_of_casualty'].value_cou
percentages.columns = ['%age_band_of_casualty']

# Converting the percentage values to a string format with a '%' sign
percentages['%age_band_of_casualty'] = percentages['%age_band_of_casualty']

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[82]:

	age_band_of_casualty Count	%age_band_of_casualty
26 - 35	2083	22.5%
36 - 45	1949	21.05%
46 - 55	1874	20.24%
21 - 25	914	9.87%
56 - 65	792	8.56%
16 - 20	466	5.03%
6 - 10	365	3.94%
66 - 75	364	3.93%
Over 75	334	3.61%
Data missing or out of range	66	0.71%
11 - 15	44	0.48%
0 - 5	6	0.06%

The 25-26 age band are most of the casualties

In [83]:
```
# Create a DataFrame of the top 10 most frequent age bands of casualties
top_age_bands = pd.DataFrame(trainset['age_band_of_casualty'].value_counts(

# Plotting the data
# 'kind='bar'' specifies that the plot should be a bar chart
plot = top_age_bands.plot(kind='bar' , legend=False)  # Turn off the legend

# Adding title and labels to the plot
plot.set_title('Top 10 Age Bands of Casualties')  # Set the title of the pl
plot.set_xlabel('Age Band')  # Set the x-axis label
plot.set_ylabel('Count of Casualties')  # Set the y-axis label

# Display the plot
plt.show()
```



3.2.2.20 Pedestrian_location

In [84]:
```
# Displaying the statistical summary for 'pedestrian_location'
pd.DataFrame(trainset.loc[:, 'pedestrian_location'].describe())
```

Out[84]:

	pedestrian_location
count	9257
unique	8
top	Not a Pedestrian
freq	7205

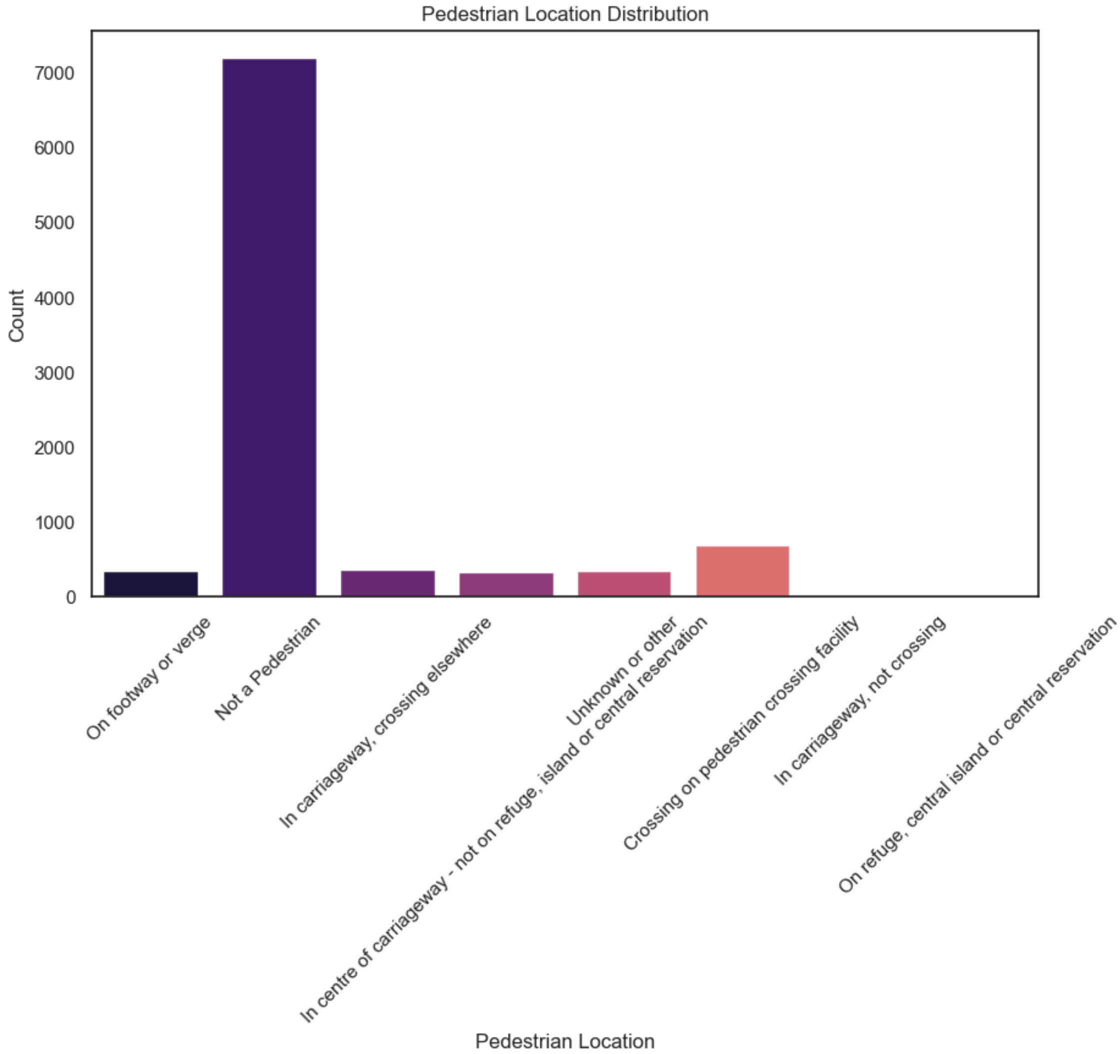
In [85]:
```
# Getting the value counts of the 'pedestrian_location' variable
values = pd.DataFrame(trainset['pedestrian_location'].value_counts())
values.columns = ['Pedestrian Location Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['pedestrian_location'].value_cour
percentages.columns = ['% Pedestrian Location']

# Converting the percentage values to a string format with a '%' sign
percentages['% Pedestrian Location'] = percentages['% Pedestrian Location']

# Joining the values and percentages dataframes
summary_table = values.join(percentages)
summary_table
```

Out[85]:

	Pedestrian Location Count	% Pedestrian Location
Not a Pedestrian	7205	77.83%
Crossing on pedestrian crossing facility	684	7.39%
In carriageway, crossing elsewhere	352	3.8%
Unknown or other	343	3.71%
On footway or verge	341	3.68%
In centre of carriageway - not on refuge, island or central reservation	326	3.52%
In carriageway, not crossing	5	0.05%
On refuge, central island or central reservation	1	0.01%

'Not a pedestrian' occurred the most as casualties

In [86]:
```
#Pick the fig size
plt.figure(figsize=(10, 6))

# Define plot object for 'pedestrian_location'
count = sns.countplot(data = trainset, x = 'pedestrian_location', palette =

# Setting graph title and labels
count.set_title('Pedestrian Location Distribution')
count.set(xlabel='Pedestrian Location', ylabel='Count')


# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit

# Showing the plot
plt.show()
```



3.2.2.21 Pedestrian_movement

In [87]:
```
# Displaying the statistical summary for 'pedestrian_movement'
pd.DataFrame(trainset.loc[:, 'pedestrian_movement'].describe())
```

Out[87]:

	pedestrian_movement
count	9257
unique	5
top	Not a Pedestrian
freq	7205

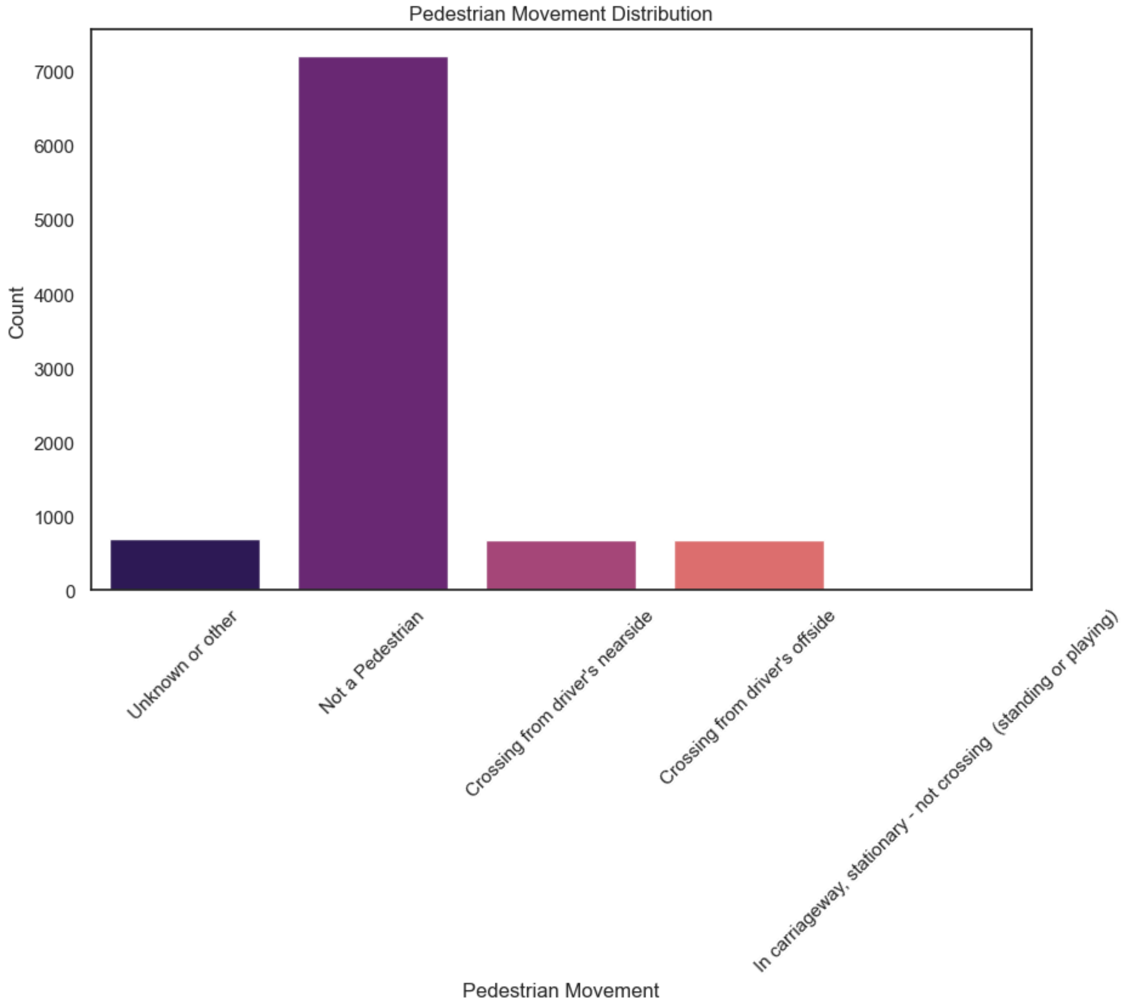
In [88]:
```
# Getting the value counts of the 'pedestrian_movement' variable
values = pd.DataFrame(trainset['pedestrian_movement'].value_counts())
values.columns = ['Pedestrian Movement Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['pedestrian_movement'].value_cour
percentages.columns = ['% Pedestrian Movement']

# Converting the percentage values to a string format with a '%' sign
percentages['% Pedestrian Movement'] = percentages['% Pedestrian Movement']

# Joining the values and percentages dataframes
summary_table = values.join(percentages)
summary_table
```

Out[88]:

	Pedestrian Movement Count	% Pedestrian Movement
Not a Pedestrian	7205	77.83%
Unknown or other	690	7.45%
Crossing from driver's offside	683	7.38%
Crossing from driver's nearside	674	7.28%
In carriageway, stationary - not crossing (standing or playing)	5	0.05%

Majority of the casualties were drivers and the second cause was not known

In [89]:
```
plt.figure(figsize=(10, 6))
# Define plot object for 'pedestrian_movement'
count = sns.countplot(data = trainset, x = 'pedestrian_movement', palette =

# Setting graph title and labels
count.set_title('Pedestrian Movement Distribution')
count.set(xlabel='Pedestrian Movement', ylabel='Count')


# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit

# Showing the plot
plt.show()
```



3.2.2.22 Car Passenger

In [90]:
```
# Displaying the statistical summary for 'car_passenger'
pd.DataFrame(trainset.loc[:, 'car_passenger'].describe())
```

Out[90]:

	car_passenger
count	9257
unique	4
top	Not car passenger
freq	8753

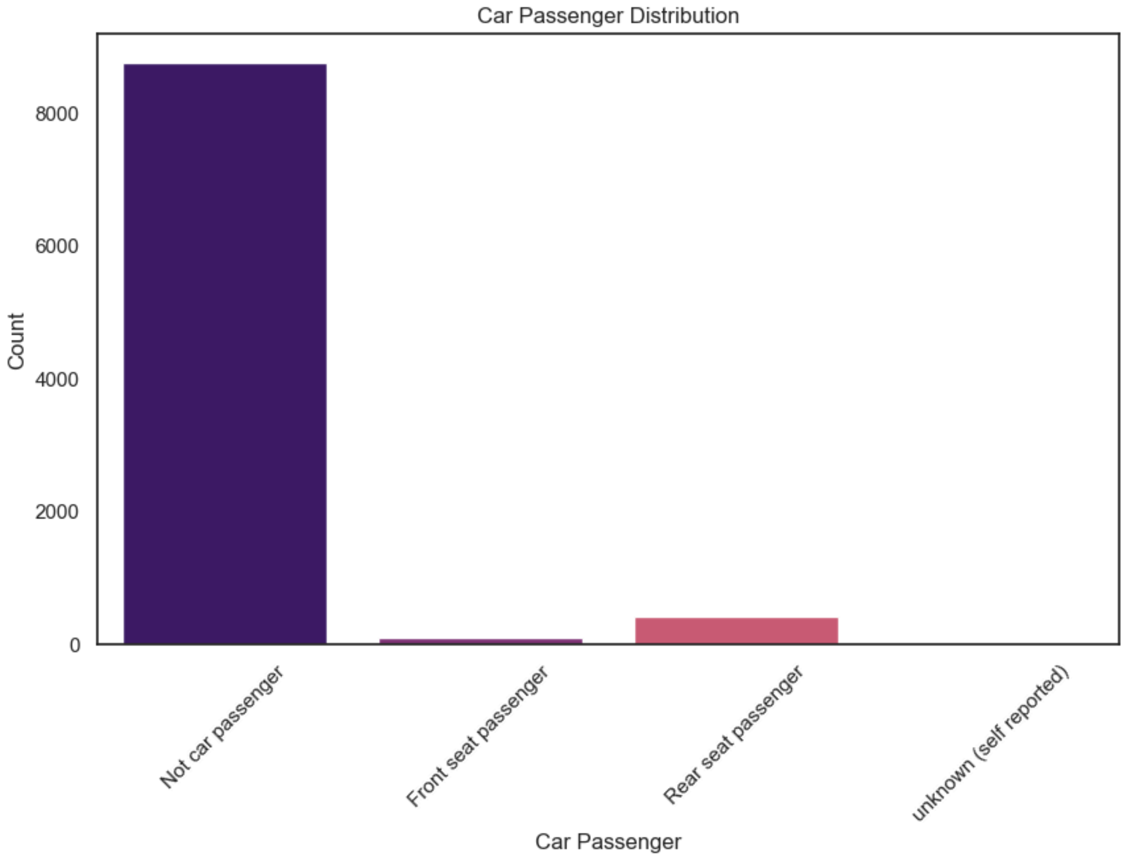
In [91]:
```
# Getting the value counts of the 'car_passenger' variable
values = pd.DataFrame(trainset['car_passenger'].value_counts())
values.columns = ['Car Passenger Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['car_passenger'].value_counts(nor
percentages.columns = ['% Car Passenger']

# Converting the percentage values to a string format with a '%' sign
percentages['% Car Passenger'] = percentages['% Car Passenger'].map(lambda

# Joining the values and percentages dataframes
summary_table = values.join(percentages)
summary_table
```

Out[91]:

	Car Passenger Count	% Car Passenger
Not car passenger	8753	94.56%
Rear seat passenger	404	4.36%
Front seat passenger	98	1.06%
unknown (self reported)	2	0.02%

Most casualties occur when the driver is alone

In [92]:
```
# Define plot object for 'car_passenger'
plt.figure(figsize=(10, 6))
count = sns.countplot(data = trainset ,x = 'car_passenger', palette = 'magm

# Setting graph title and labels
count.set_title('Car Passenger Distribution')
count.set(xlabel='Car Passenger', ylabel='Count')


# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit


# Showing the plot
plt.show()
```



3.2.2.23 Casualty Home Area Type

In [93]:
```
# Displaying the statistical summary for 'casualty_home_area_type'
pd.DataFrame(trainset.loc[:, 'casualty_home_area_type'].describe())
```

Out[93]:

	casualty_home_area_type
count	9257
unique	4
top	Urban area
freq	8763

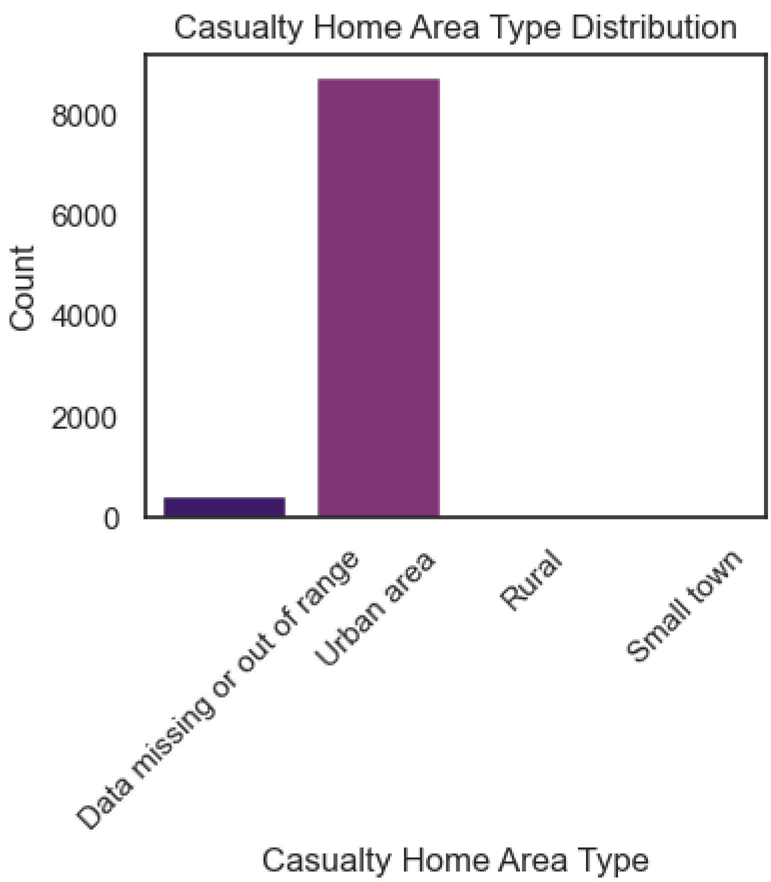
In [94]:
```
# Getting the value counts of the 'casualty_home_area_type' variable
values = pd.DataFrame(trainset['casualty_home_area_type'].value_counts())
values.columns = ['Casualty Home Area Type Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['casualty_home_area_type'].value_
percentages.columns = ['% Casualty Home Area Type']

# Converting the percentage values to a string format with a '%' sign
percentages['% Casualty Home Area Type'] = percentages['% Casualty Home Are
# Joining the values and percentages dataframes
summary_table = values.join(percentages)
summary_table
```

Out[94]:

	Casualty Home Area Type Count	% Casualty Home Area Type
Urban area	8763	94.66%
Data missing or out of range	465	5.02%
Rural	15	0.16%
Small town	14	0.15%

Majority of casualties from accidents occur in urban areas

In [95]:
```
# Define plot object for 'casualty_home_area_type'
plt.figure(figsize=(4, 3))
count = sns.countplot(data = trainset, x ='casualty_home_area_type', palett
# Setting graph title and labels
count.set_title('Casualty Home Area Type Distribution')
count.set(xlabel='Casualty Home Area Type', ylabel='Count')


# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit

# Showing the plot
plt.show()
```



3.2.2.24 Casualty Home Area Type

In [96]:
```
pd.DataFrame(trainset.loc[:,'casualty_class'].describe())
```

Out[96]:

	casualty_class
count	9257
unique	3
top	Driver or rider
freq	6333

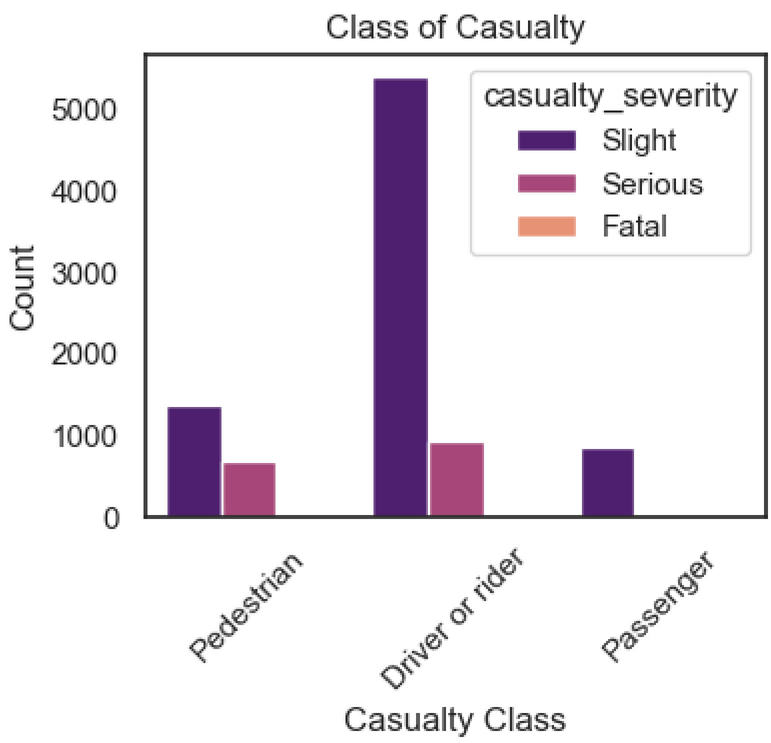
In [97]:
```
# Getting the value counts of the 'casualty_class' variable
values = pd.DataFrame(trainset['casualty_class'].value_counts())
values.columns = ['casualty_class Count']

# The normalize attribute in the value_counts method computes the percentag
percentages = pd.DataFrame(round(trainset['casualty_class'].value_counts(nc
percentages.columns = ['%casualty_class']

# Converting the percentage values to a string format with a '%' sign
percentages['%casualty_class'] = percentages['%casualty_class'].map(lambda

# Joining the values and percentages dataframes
values.join(percentages)
```

Out[97]:

	casualty_class Count	%casualty_class
Driver or rider	6333	68.41%
Pedestrian	2052	22.17%
Passenger	872	9.42%

The main casualties were either driver or riders

In [98]:
```
# Define plot object
plt.figure(figsize=(4, 3))
count = sns.countplot(data = trainset, x = 'casualty_class',hue = trainset[
# Setting graph title and labels
count.set_title('Class of Casualty')
count.set(xlabel='Casualty Class', ylabel='Count',)
# Slant the x-axis labels by setting a rotation
plt.xticks(rotation=45)  # Rotate labels to 45 degrees for better visibilit
# Showing the plot
plt.show()
```



4. Data Exploration and Preprocessing

4.1 Feature Selection

We will use a correlation to the casualty severity and rank them. This will be used for the individual tasks

In [99]:

```
# Define target variable and dataframe
target_variable = 'casualty_severity'
df = trainset  # Assuming 'trainset' is the dataframe

# Identify categorical columns (excluding target variable)
other_columns = df.select_dtypes(include=['object', 'category']).columns.tc
other_columns = [col for col in other_columns if col != target_variable]

# Compute Chi-Square test results
results = []

for column in other_columns:
    contingency_table = pd.crosstab(df[target_variable], df[column])
    chi2, p_value, dof, expected = chi2_contingency(contingency_table)
    results.append({
        'Column': column,
        'Chi-square Statistic': chi2,
        'P-value': f"{p_value:.2f}"
    })

results_df = pd.DataFrame(results)
sorted_results_df = results_df.sort_values('P-value').reset_index(drop=True

# Compute Phi-K correlation matrix
phik_matrix = df.phik_matrix()

# Extract Phi-K values for the target variable
phik_target = phik_matrix[target_variable].drop(target_variable).reset_inde
phik_target.columns = ['Column', 'Phi-K Correlation']

# Merge Chi-Square results with Phi-K results
final_results = sorted_results_df.merge(phik_target, on='Column')
final_results
```

```
interval columns not set, guessing: ['age_of_vehicle']
```

Out[99]:

	Column	Chi-square Statistic	P-value	Phi-K Correlation
0	casualty_home_area_type	50.661409	0.00	0.052196
1	pedestrian_movement	1223.185911	0.00	0.321204
2	vehicle_location_restricted_lane	122.054637	0.00	0.127945
3	pedestrian_location	2385.717763	0.00	0.490737
4	age_band_of_casualty	3027.388713	0.00	0.680963
5	sex_of_casualty	187.922041	0.00	0.297488
6	casualty_class	535.575403	0.00	0.442660
7	car_passenger	94.151361	0.00	0.073284
8	junction_location	25.294465	0.12	0.033661
9	first_point_of_impact	14.882586	0.14	0.039558
10	vehicle_left_hand_drive	4.758345	0.31	0.021841
11	journey_purpose_of_driver	6.473962	0.59	0.000000
12	towing_and_articulation	6.904867	0.73	0.000000
13	driver_home_area_type	3.007982	0.81	0.000000
14	sex_of_driver	1.593469	0.81	0.000000
15	skidding_and_overturning	6.117353	0.81	0.000000
16	age_band_of_driver	13.176921	0.87	0.000000
17	hit_object_in_carriageway	11.093587	0.89	0.000000
18	vehicle_leaving_carriageway	9.231189	0.90	0.000000
19	driver_imd_decile	11.298669	0.94	0.000000
20	hit_object_off_carriageway	12.781867	0.94	0.000000
21	propulsion_code	3.985619	0.98	0.000000
22	generic_make_model	477.048961	1.00	0.000000
23	vehicle_manoeuvre	17.724584	1.00	0.000000
24	engine_capacity_cc	251.458684	1.00	0.000000
25	vehicle_type	14.304888	1.00	0.000000

The higher the Phi-K score the more probable the variable is to a vaible predictor

4.2 Handling missing values

We checked each variable that had missing values denoted by 'Data Missing or out of range' or '-1'

Target Variable

4.2.1 casualty_severity

In [100]:
```
#Inspect the column
pd.DataFrame(trainset.loc[:,'casualty_severity'].unique())

#this column has no null values
```

Out[100]:

	0
0	Slight
1	Serious
2	Fatal

Dependent Variables - Categorical

4.2.2 Vehicle Type

In [101]:
```
pd.DataFrame(trainset['vehicle_type'].unique())
#this variable has no null values
```

Out[101]:

	0
0	Motorcycle 125cc and under
1	Pedal cycle
2	Car
3	Other vehicle
4	Bus or coach (17 or more pass seats)
5	Van / Goods 3.5 tonnes mgw or under
6	Taxi/Private hire car
7	Motorcycle over 125cc and up to 500cc
8	Minibus (8 - 16 passenger seats)
9	Goods over 3.5t. and under 7.5t
10	Motorcycle over 500cc
11	Motorcycle - unknown cc
12	Goods 7.5 tonnes mgw and over
13	Motorcycle 50cc and under
14	Electric motorcycle
15	Mobility scooter
16	Goods vehicle - unknown weight

4.2.3 Towing And Articulation

In [102]:
```
pd.DataFrame(trainset['towing_and_articulation'].unique())
#this variable has no null values
```

Out[102]:

	0
0	No tow/articulation
1	unknown (self reported)
2	Articulated vehicle
3	Caravan
4	Other tow
5	Single trailer

4.2.4 Vehicle Manouvre

In [103]:
```
pd.DataFrame(trainset['vehicle_manoeuvre'].unique())
#this variable has no null values
```

Out[103]:

	0
0	Going ahead other
1	unknown (self reported)
2	Moving off
3	Slowing or stopping
4	Turning right
5	Changing lane to left
6	Turning left
7	Overtaking moving vehicle - offside
8	Going ahead right-hand bend
9	Waiting to turn right
10	U-turn
11	Waiting to turn left
12	Going ahead left-hand bend
13	Reversing
14	Waiting to go - held up
15	Overtaking - nearside
16	Changing lane to right
17	Overtaking static vehicle - offside
18	Parked

4.2.5 Vehicle Location

In [104]:
```
pd.DataFrame(trainset['vehicle_location_restricted_lane'].unique())
#this variable has no null values
```

Out[104]:

	0
0	On main c'way - not in restricted lane
1	unknown (self reported)
2	Footway (pavement)
3	On lay-by or hard shoulder
4	Cycleway or shared use footway (not part of m...
5	Cycle lane (on main carriageway)
6	Bus lane
7	Busway (including guided busway)
8	Tram/Light rail track
9	Entering lay-by or hard shoulder
10	Leaving lay-by or hard shoulder

4.2.6 Junction Location

In [105]:
```
pd.DataFrame(trainset['junction_location'].unique())
#this variable has no null values
```

Out[105]:

	0
0	Not at or within 20 metres of junction
1	unknown (self reported)
2	Entering roundabout
3	Approaching junction or waiting/parked at junc...
4	Cleared junction or waiting/parked at junction...
5	Mid Junction - on roundabout or on main road
6	Leaving main road
7	Leaving roundabout
8	Entering main road
9	Entering from slip road

4.2.7 Skidding and Overturning

In [106]:
```
pd.DataFrame(trainset['skidding_and_overturning'].unique())
#this variable has no null values
```

Out[106]:

	0
0	Overturned
1	unknown (self reported)
2	None
3	Skidded
4	Skidded and overturned
5	Jackknifed

4.2.8 Hit Object in Carriageway

In [107]:
```
pd.DataFrame(trainset['hit_object_in_carriageway'].unique())
#this variable has no null values
```

Out[107]:

	0
0	None
1	unknown (self reported)
2	Kerb
3	Bollard or refuge
4	Parked vehicle
5	Open door of vehicle
6	Other object
7	Bridge (side)
8	Any animal (except ridden horse)
9	Previous accident

4.2.9 Hit Object Off Carriageway

In [108]:
```
pd.DataFrame(trainset['hit_object_off_carriageway'].unique())
#this variable has no null values
```

Out[108]:

	0
0	None
1	unknown (self reported)
2	Lamp post
3	Road sign or traffic signal
4	Other permanent object
5	Tree
6	Bus stop or bus shelter
7	Wall or fence
8	Central crash barrier
9	Near/Offside crash barrier
10	Entered ditch

4.2.10 Age band of Driver

In [109]:
```
pd.DataFrame(trainset['age_band_of_driver'].unique())
#this variable has  null values
```

Out[109]:

	0
0	36 - 45
1	Data missing or out of range
2	21 - 25
3	46 - 55
4	26 - 35
5	66 - 75
6	16 - 20
7	Over 75
8	56 - 65
9	11 - 15
10	6 - 10

In [110]:

```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['age_band_of_driver'] = encoder.fit_transform(trainset[['a

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['age_band_of_driver'] = trainset_encoded['age_band_of_driv

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['age_bar

# Decode the imputed data back to original categories
trainset['age_band_of_driver'] = encoder.inverse_transform(trainset_encodec

# Check the output
pd.DataFrame(trainset.loc[:,'age_band_of_driver'].value_counts())
```

Out[110]:

	age_band_of_driver
26 - 35	3835
36 - 45	1727
46 - 55	1214
21 - 25	987
56 - 65	729
16 - 20	359
66 - 75	255
Over 75	120
11 - 15	23
6 - 10	8

In [111]:
```
# Encode the categorical data in the testset as well
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['age_band_of_driver'] = encoder.fit_transform(testset[['age

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['age_band_of_driver'] = testset_encoded['age_band_of_driver

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['age_band_

# Decode the imputed data back to original categories
testset['age_band_of_driver'] = encoder.inverse_transform(testset_encoded_i

# Check the output
pd.DataFrame(testset.loc[:,'age_band_of_driver'].value_counts())
```

Out[111]:

	age_band_of_driver
26 - 35	415
36 - 45	180
46 - 55	150
21 - 25	102
56 - 65	96
16 - 20	44
66 - 75	24
Over 75	15
11 - 15	2
6 - 10	1

4.2.11 driver_imd_decile

In [112]:
```
pd.DataFrame(trainset.loc[:,'driver_imd_decile'].describe())
```

Out[112]:

	driver_imd_decile
count	9257
unique	11
top	Data missing or out of range
freq	1677

In [113]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['driver_imd_decile'] = encoder.fit_transform(trainset[['dr

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['driver_imd_decile'] = trainset_encoded['driver_imd_decile

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['driver_

# Decode the imputed data back to original categories
trainset['driver_imd_decile'] = encoder.inverse_transform(trainset_encoded_

# Check the output
pd.DataFrame(trainset.loc[:,'driver_imd_decile'].value_counts())
```

Out[113]:

	driver_imd_decile
More deprived 10-20%	3177
More deprived 20-30%	1453
More deprived 30-40%	1050
More deprived 40-50%	839
Less deprived 40-50%	694
Most deprived 10%	545
Less deprived 30-40%	473
Less deprived 20-30%	460
Less deprived 10-20%	329
Least deprived 10%	237

In [114]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['driver_imd_decile'] = encoder.fit_transform(testset[['driv

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['driver_imd_decile'] = testset_encoded['driver_imd_decile']

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['driver_im

# Decode the imputed data back to original categories
testset['driver_imd_decile'] = encoder.inverse_transform(testset_encoded_im

# Check the output
pd.DataFrame(testset.loc[:,'driver_imd_decile'].value_counts())
```

Out[114]:

	driver_imd_decile
More deprived 10-20%	344
More deprived 20-30%	171
More deprived 30-40%	122
More deprived 40-50%	92
Less deprived 40-50%	74
Most deprived 10%	65
Less deprived 20-30%	51
Less deprived 30-40%	48
Less deprived 10-20%	37
Least deprived 10%	25

4.2.12 generic_make_model

In [115]:
```
pd.DataFrame(trainset['generic_make_model'].unique())
```

Out[115]:

	0
0	YAMAHA GPD
1	-1
2	AUDI Q2
3	DACIA DUSTER
4	MERCEDES E CLASS
...	...
418	DACIA SANDERO
419	AUDI A8
420	MITSUBISHI MODEL MISSING
421	MERCEDES CLK CLASS
422	BMW S 1000

423 rows × 1 columns

In [116]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['generic_make_model'] = encoder.fit_transform(trainset[['g

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['-1']])[0, 0]

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['generic_make_model'] = trainset_encoded['generic_make_moc

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['generic

# Decode the imputed data back to original categories
trainset['generic_make_model'] = encoder.inverse_transform(trainset_encodec

# Check the output
pd.DataFrame(trainset.loc[:,'generic_make_model'].value_counts())
```

Out[116]:

	generic_make_model
MERCEDES EQA CLASS	2189
YAMAHA GPD	380
HONDA WW125	361
TOYOTA PRIUS	257
HONDA SH 125	210
...	...
LEXUS GS 450	1
SEAT ALTEA	1
PIAGGIO FLY	1
JEEP GRAND CHEROKEE	1
BMW S 1000	1

422 rows × 1 columns

In [117]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['generic_make_model'] = encoder.fit_transform(testset[['ger

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['-1']])[0, 0]

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['generic_make_model'] = testset_encoded['generic_make_model

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['generic_m

# Decode the imputed data back to original categories
testset['generic_make_model'] = encoder.inverse_transform(testset_encoded_i

# Check the output
pd.DataFrame(testset.loc[:,'generic_make_model'].value_counts())
```

Out[117]:

	generic_make_model
MERCEDES C CLASS	270
YAMAHA GPD	41
HONDA WW125	37
TOYOTA PRIUS	36
HONDA SH 125	25
...	...
LONDON TAXIS INT. TX4	1
MERCEDES SLK CLASS	1
SUBARU IMPREZA	1
DACIA DUSTER	1
BMW 6 SERIES	1

210 rows × 1 columns

4.2.13 driver_home_area_type

In [118]:
```
#check if the dataframe has been updated
pd.DataFrame(trainset.loc[:,'driver_home_area_type'].unique())
```

Out[118]:

	0
0	Urban area
1	Data missing or out of range
2	Rural
3	Small town

In [119]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['driver_home_area_type'] = encoder.fit_transform(trainset[

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['driver_home_area_type'] = trainset_encoded['driver_home_a

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['driver_

# Decode the imputed data back to original categories
trainset['driver_home_area_type'] = encoder.inverse_transform(trainset_enco

# If you want to see the modified dataframe

# Check the output
pd.DataFrame(trainset.loc[:,'driver_home_area_type'].value_counts())
```

Out[119]:

	driver_home_area_type
Urban area	7427
Small town	1734
Rural	96

In [120]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['driver_home_area_type'] = encoder.fit_transform(testset[['

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['driver_home_area_type'] = testset_encoded['driver_home_are

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['driver_ho

# Decode the imputed data back to original categories
testset['driver_home_area_type'] = encoder.inverse_transform(testset_encode

# Check the output
pd.DataFrame(testset.loc[:,'driver_home_area_type'].value_counts())
```

Out[120]:

	driver_home_area_type
Urban area	827
Small town	189
Rural	13

4.2.14 casualty_class

In [121]:
```
#Check if there any null values
pd.DataFrame(trainset.loc[:,'casualty_class'].unique())
```

Out[121]:

	0
0	Pedestrian
1	Driver or rider
2	Passenger

4.2.15 sex_of_casualty

In [122]:
```
#Check if there any null values
pd.DataFrame(trainset.loc[:,'sex_of_casualty'].unique())
```

Out[122]:

	0
0	Male
1	Female
2	Data missing or out of range

In [123]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['sex_of_casualty'] = encoder.fit_transform(trainset[['sex_

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['sex_of_casualty'] = trainset_encoded['sex_of_casualty'].r

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['sex_of_

# Decode the imputed data back to original categories
trainset['sex_of_casualty'] = encoder.inverse_transform(trainset_encoded_im

# If you want to see the modified dataframe

# Check the output
pd.DataFrame(trainset.loc[:,'sex_of_casualty'].value_counts())
```

Out[123]:

	sex_of_casualty
Male	5873
Female	3384

In [124]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['sex_of_casualty'] = encoder.fit_transform(testset[['sex_of

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['sex_of_casualty'] = testset_encoded['sex_of_casualty'].rep

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['sex_of_ca

# Decode the imputed data back to original categories
testset['sex_of_casualty'] = encoder.inverse_transform(testset_encoded_impu

# Check the output
pd.DataFrame(testset.loc[:,'sex_of_casualty'].value_counts())
```

Out[124]:

	sex_of_casualty
Male	647
Female	382

4.2.16 age_band_of_casualty

In [125]:
```
#Inspect the column
pd.DataFrame(trainset.loc[:,'age_band_of_casualty'].value_counts())
```

Out[125]:

	age_band_of_casualty
26 - 35	2083
36 - 45	1949
46 - 55	1874
21 - 25	914
56 - 65	792
16 - 20	466
6 - 10	365
66 - 75	364
Over 75	334
Data missing or out of range	66
11 - 15	44
0 - 5	6

In [126]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['age_band_of_casualty'] = encoder.fit_transform(trainset[[

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['age_band_of_casualty'] = trainset_encoded['age_band_of_ca

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['age_bar

# Decode the imputed data back to original categories
trainset['age_band_of_casualty'] = encoder.inverse_transform(trainset_encoc

# If you want to see the modified dataframe

# Check the output
pd.DataFrame(trainset.loc[:,'age_band_of_casualty'].value_counts())
```

Out[126]:

	age_band_of_casualty
26 - 35	2083
36 - 45	2015
46 - 55	1874
21 - 25	914
56 - 65	792
16 - 20	466
6 - 10	365
66 - 75	364
Over 75	334
11 - 15	44
0 - 5	6

In [127]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['age_band_of_casualty'] = encoder.fit_transform(testset[['a

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['age_band_of_casualty'] = testset_encoded['age_band_of_casu

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['age_band_

# Decode the imputed data back to original categories
testset['age_band_of_casualty'] = encoder.inverse_transform(testset_encodec

# Check the output
pd.DataFrame(testset.loc[:,'age_band_of_casualty'].value_counts())
```

Out[127]:

	age_band_of_casualty
26 - 35	242
36 - 45	226
46 - 55	220
21 - 25	92
56 - 65	77
Over 75	47
16 - 20	44
66 - 75	40
6 - 10	35
11 - 15	4
0 - 5	2

In [128]:
```
#Inspect the column
pd.DataFrame(trainset.loc[:,'casualty_severity'].unique())

#this column has no null values
```

Out[128]:

	0
0	Slight
1	Serious
2	Fatal

4.2.17 pedestrian_location

In [129]:
```
pd.DataFrame(trainset.loc[:,'pedestrian_location'].unique())
```

Out[129]:

	0
0	On footway or verge
1	Not a Pedestrian
2	In carriageway, crossing elsewhere
3	In centre of carriageway - not on refuge, isla...
4	Unknown or other
5	Crossing on pedestrian crossing facility
6	In carriageway, not crossing
7	On refuge, central island or central reservation

4.2.18 Pedestrian_movement

In [130]:
```
pd.DataFrame(trainset.loc[:,'pedestrian_movement'].unique())
```

Out[130]:

	0
0	Unknown or other
1	Not a Pedestrian
2	Crossing from driver's nearside
3	Crossing from driver's offside
4	In carriageway, stationary - not crossing (st...

4.2.19 car_passenger

In [131]:
```
pd.DataFrame(trainset.loc[:,'car_passenger'].unique())
```

Out[131]:

	0
0	Not car passenger
1	Front seat passenger
2	Rear seat passenger
3	unknown (self reported)

4.2.20 casualty_home_area_type

In [132]:
```
pd.DataFrame(trainset.loc[:,'casualty_home_area_type'].unique())
```

Out[132]:

	0
0	Data missing or out of range
1	Urban area
2	Rural
3	Small town

In [133]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['casualty_home_area_type'] = encoder.fit_transform(trainse

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['casualty_home_area_type'] = trainset_encoded['casualty_hc

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['casualt

# Decode the imputed data back to original categories
trainset['casualty_home_area_type'] = encoder.inverse_transform(trainset_er

# If you want to see the modified dataframe

# Check the output
pd.DataFrame(trainset.loc[:,'casualty_home_area_type'].value_counts())
```

Out[133]:

	casualty_home_area_type
Urban area	8763
Small town	479
Rural	15

In [134]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['casualty_home_area_type'] = encoder.fit_transform(testset[

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['casualty_home_area_type'] = testset_encoded['casualty_home

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['casualty_

# Decode the imputed data back to original categories
testset['casualty_home_area_type'] = encoder.inverse_transform(testset_enco

# Check the output
pd.DataFrame(testset.loc[:,'casualty_home_area_type'].value_counts())
```

Out[134]:

	casualty_home_area_type
Urban area	971
Rural	58

4.2.21 vehicle_leaving_carriageway

In [135]:
```
pd.DataFrame(trainset['vehicle_leaving_carriageway'].unique())
```

Out[135]:

	0
0	Did not leave carriageway
1	unknown (self reported)
2	Nearside
3	Offside
4	Straight ahead at junction
5	Offside and rebounded
6	Offside on to central reservation
7	Offside on to centrl res + rebounded
8	Nearside and rebounded

Dependent Variables - Numerical

In [136]:
```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9257 entries, 1151999 to 5570999
Data columns (total 28 columns):
 #   Column                            Non-Null Count  Dtype 
---  ------                            --------------  ----- 
 0   vehicle_type                      9257 non-null   object
 1   towing_and_articulation           9257 non-null   object
 2   vehicle_manoeuvre                 9257 non-null   object
 3   vehicle_location_restricted_lane  9257 non-null   object
 4   junction_location                 9257 non-null   object
 5   skidding_and_overturning          9257 non-null   object
 6   hit_object_in_carriageway         9257 non-null   object
 7   vehicle_leaving_carriageway       9257 non-null   object
 8   hit_object_off_carriageway        9257 non-null   object
 9   first_point_of_impact             9257 non-null   object
 10  vehicle_left_hand_drive           9257 non-null   object
 11  journey_purpose_of_driver         9257 non-null   object
 12  sex_of_driver                     9257 non-null   object
 13  age_band_of_driver                9257 non-null   object
 14  engine_capacity_cc                9257 non-null   object
 15  propulsion_code                   9257 non-null   object
 16  age_of_vehicle                    9257 non-null   int64 
 17  generic_make_model                9257 non-null   object
 18  driver_imd_decile                 9257 non-null   object
 19  driver_home_area_type             9257 non-null   object
 20  casualty_class                    9257 non-null   object
 21  sex_of_casualty                   9257 non-null   object
 22  age_band_of_casualty              9257 non-null   object
 23  casualty_severity                 9257 non-null   object
 24  pedestrian_location               9257 non-null   object
 25  pedestrian_movement               9257 non-null   object
 26  car_passenger                     9257 non-null   object
 27  casualty_home_area_type           9257 non-null   object
dtypes: int64(1), object(27)
memory usage: 2.0+ MB
```

4.2.22 vehicle_leaving_carriageway

We note that engine capacity was treated as categorical variable as the engine size tends to follow a certain ranking. We decided to handle these first as categorical but for analysis later we will use it as numeric

In [137]:
```
trainset['engine_capacity_cc'] = trainset['engine_capacity_cc'].astype(str)
testset['engine_capacity_cc'] = testset['engine_capacity_cc'].astype(str)
```

In [138]:
```
# Encode the categorical data
#encoder = OrdinalEncoder()
trainset_encoded = trainset.copy()
trainset_encoded['engine_capacity_cc'] = encoder.fit_transform(trainset[['e

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
trainset_encoded['engine_capacity_cc'] = trainset_encoded['engine_capacity_

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
trainset_encoded_imputed = imputer.fit_transform(trainset_encoded[['engine_

# Decode the imputed data back to original categories
trainset['engine_capacity_cc'] = encoder.inverse_transform(trainset_encodec

# If you want to see the modified dataframe

# Check the output
pd.DataFrame(trainset.loc[:,'engine_capacity_cc'].value_counts())
```

Out[138]:

	engine_capacity_cc
1790	2260
125	1247
1598	312
1995	265
124	210
...	...
1985	1
899	1
2364	1
2985	1
3246	1

263 rows × 1 columns

In [139]:
```
# Encode the categorical data
encoder = OrdinalEncoder()
testset_encoded = testset.copy()
testset_encoded['engine_capacity_cc'] = encoder.fit_transform(testset[['eng

# Identify the encoded value for 'Data missing or out of range'
encoded_missing_value = encoder.transform([['Data missing or out of range']

# Replace encoded 'Data missing or out of range' values with np.nan
testset_encoded['engine_capacity_cc'] = testset_encoded['engine_capacity_cc

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)
testset_encoded_imputed = imputer.fit_transform(testset_encoded[['engine_ca

# Decode the imputed data back to original categories
testset['engine_capacity_cc'] = encoder.inverse_transform(testset_encoded_i

# If you want to see the modified dataframe

# Check the output
pd.DataFrame(testset.loc[:,'engine_capacity_cc'].value_counts())
```

Out[139]:

	engine_capacity_cc
1600	251
125	134
1598	40
1798	34
1995	27
...	...
12777	1
1299	1
2487	1
49	1
1246	1

136 rows × 1 columns

4.2.23 age_of_vehicle

In [140]:
```
# Replace -1 with NaN for missing values
trainset['age_of_vehicle'] = trainset['age_of_vehicle'].replace(-1, np.nan)

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)

# Apply the imputer directly on 'age_of_vehicle' column (which is numeric)
trainset_imputed = trainset.copy()
trainset_imputed['age_of_vehicle'] = imputer.fit_transform(trainset_imputed
```

In [141]:
```
# Replace -1 with NaN for missing values
testset['age_of_vehicle'] = testset['age_of_vehicle'].replace(-1, np.nan)

# Set up the MICE imputer
imputer = IterativeImputer(max_iter=10, random_state=0)

# Apply the imputer directly on 'age_of_vehicle' column (which is numeric)
testset_imputed = testset.copy()
testset_imputed['age_of_vehicle'] = imputer.fit_transform(testset_imputed[[
```

Handling Outliers

In [142]:
```
# The check is on numerical variables
```

In [143]:
```
trainset['engine_capacity_cc'] = trainset['engine_capacity_cc'].astype(int)
testset['engine_capacity_cc'] = testset['engine_capacity_cc'].astype(int)
```

In [144]:
```
pd.DataFrame(trainset.loc[:,'engine_capacity_cc'].describe())
```

Out[144]:

	engine_capacity_cc
count	9257.000000
mean	1593.479637
std	1177.110824
min	49.000000
25%	1198.000000
50%	1790.000000
75%	1798.000000
max	12902.000000

In [145]:
```
trainset.head()
```

Out[145]:

	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_location_restricted
1151999	Motorcycle 125cc and under	No tow/articulation	Going ahead other	On main c'way - not in res
9503999	Pedal cycle	unknown (self reported)	unknown (self reported)	unknown (self rep
5639399	Car	No tow/articulation	unknown (self reported)	unknown (self rep
7336799	Car	No tow/articulation	unknown (self reported)	unknown (self rep
4147199	Car	No tow/articulation	unknown (self reported)	unknown (self rep

5 rows × 28 columns

In [146]:
```
trainset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9257 entries, 1151999 to 5570999
Data columns (total 28 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   vehicle_type                      9257 non-null   object
 1   towing_and_articulation           9257 non-null   object
 2   vehicle_manoeuvre                 9257 non-null   object
 3   vehicle_location_restricted_lane  9257 non-null   object
 4   junction_location                 9257 non-null   object
 5   skidding_and_overturning          9257 non-null   object
 6   hit_object_in_carriageway         9257 non-null   object
 7   vehicle_leaving_carriageway       9257 non-null   object
 8   hit_object_off_carriageway        9257 non-null   object
 9   first_point_of_impact             9257 non-null   object
 10  vehicle_left_hand_drive           9257 non-null   object
 11  journey_purpose_of_driver         9257 non-null   object
 12  sex_of_driver                     9257 non-null   object
 13  age_band_of_driver                9257 non-null   object
 14  engine_capacity_cc                9257 non-null   int32
 15  propulsion_code                   9257 non-null   object
 16  age_of_vehicle                    7262 non-null   float64
 17  generic_make_model                9257 non-null   object
 18  driver_imd_decile                 9257 non-null   object
 19  driver_home_area_type             9257 non-null   object
 20  casualty_class                    9257 non-null   object
 21  sex_of_casualty                   9257 non-null   object
 22  age_band_of_casualty              9257 non-null   object
 23  casualty_severity                 9257 non-null   object
 24  pedestrian_location               9257 non-null   object
 25  pedestrian_movement               9257 non-null   object
 26  car_passenger                     9257 non-null   object
 27  casualty_home_area_type           9257 non-null   object
dtypes: float64(1), int32(1), object(26)
memory usage: 2.0+ MB
```

In [147]:
```
pd.DataFrame(trainset.loc[:,'engine_capacity_cc'].describe())
```

Out[147]:

	engine_capacity_cc
count	9257.000000
mean	1593.479637
std	1177.110824
min	49.000000
25%	1198.000000
50%	1790.000000
75%	1798.000000
max	12902.000000

In [148]:
```
pd.DataFrame(testset.loc[:,'engine_capacity_cc'].describe())
```

Out[148]:

	engine_capacity_cc
count	1029.000000
mean	1539.448008
std	1056.794803
min	49.000000
25%	1199.000000
50%	1600.000000
75%	1798.000000
max	12777.000000

In [149]:
```
class OutlierTransformer(TransformerMixin, BaseEstimator):
    """
    This class transforms outliers in specified columns into NaN.
    The definition of an outlier is a value smaller than quantile1 - (1.5 *
    or larger than quantile3 + (1.5 * IQR).
    """

    def __init__(self, columns=None):
        self.columns = columns  # Columns to check for outliers (if None, w
        self.quantiles = None  # Initialize to store quantile values
        self.fitted = False

    def fit(self, X, y=None):
        """
        Compute the upper and lower bounds for outlier detection based on t
        """
        # Select only specified columns (or all numeric columns if none are
        if self.columns is not None:
            X = X[self.columns]

        # Select only numerical columns
        numeric_X = X.select_dtypes(include=['number'])

        # Compute the quantiles and IQR
        lower_quantile = numeric_X.quantile(0.25)
        upper_quantile = numeric_X.quantile(0.75)
        IQR = upper_quantile - lower_quantile

        # Calculate the upper and lower bounds for outlier detection
        lower_bound = lower_quantile - 1.5 * IQR
        upper_bound = upper_quantile + 1.5 * IQR

        # Store quantiles and bounds in a dictionary for easy access
        self.quantiles = {
            'lower_bound': lower_bound,
            'upper_bound': upper_bound
        }

        self.fitted = True
        return self

    def transform(self, X):
        """
        Replace outliers with NaN for selected columns.
        """
        if not self.fitted:
            raise ValueError("The transformer must be fitted before transfo

        # Select only specified columns (or all numeric columns if none are
        if self.columns is not None:
            X = X[self.columns]

        # Make a copy of the data to avoid modifying the original dataset
        X_copy = X.copy()

        # Count existing NaN values before transformation
        old_num_na = X_copy.isna().sum().sum()

        # Replace outliers with NaN
        for col in X_copy.columns:
            X_copy.loc[X_copy[col] < self.quantiles['lower_bound'][col], co
```

```
            X_copy.loc[X_copy[col] > self.quantiles['upper_bound'][col], cc

        # Print the number of outliers identified
        new_num_na = X_copy.isna().sum().sum()
        print(f"{new_num_na - old_num_na} outliers were identified and repl

        # Return the transformed DataFrame
        return X_copy
```

In [150]:

```
# Example usage:
# Assuming 'trainset' is your DataFrame and contains columns 'engine_capaci

# Initialize the transformer for specific columns
transformer = OutlierTransformer()

# Fit the transformer
transformer.fit(trainset)
```

Out[150]:

```
▾ OutlierTransformer
OutlierTransformer()
```

In [151]:

```
transformer = OutlierTransformer(columns=['engine_capacity_cc', 'age_of_veh

# Fit the transformer
transformer.fit(trainset)

# Transform the data to replace outliers with NaN
transformed_data = transformer.transform(trainset)
```

```
2440 outliers were identified and replaced with NaN.
```

In [152]:

```
transformed_data.head()
```

Out[152]:

	engine_capacity_cc	age_of_vehicle
1151999	NaN	2.0
9503999	1790.0	NaN
5639399	1498.0	2.0
7336799	999.0	4.0
4147199	1950.0	4.0

In [153]:

```
# Replace the original columns in the trainset with the transformed data
trainset[['engine_capacity_cc', 'age_of_vehicle']] = transformed_data[['eng
```

In [154]:
```
pd.DataFrame(trainset.loc[:,'engine_capacity_cc'].describe())
```

Out[154]:

	engine_capacity_cc
count	6887.000000
mean	1664.737186
std	340.027759
min	313.000000
25%	1490.000000
50%	1790.000000
75%	1797.000000
max	2597.000000

In [155]:
```
pd.DataFrame(trainset.loc[:,'age_of_vehicle'].describe())
```

Out[155]:

	age_of_vehicle
count	7192.000000
mean	7.304783
std	4.907147
min	0.000000
25%	3.000000
50%	6.000000
75%	10.000000
max	21.000000

In [156]:
```
#Do the same for the testset
transformer = OutlierTransformer(columns=['engine_capacity_cc', 'age_of_veh

# Fit the transformer
transformer.fit(testset)

# Transform the data to replace outliers with NaN
transformed_data = transformer.transform(testset)
```

```
279 outliers were identified and replaced with NaN.
```

In [157]:
```
# Separate numeric and categorical data
numeric_columns = trainset.select_dtypes(include=['number']).columns

# Apply mean imputation for numeric columns
num_imputer = SimpleImputer(strategy="mean")
trainset[numeric_columns] = num_imputer.fit_transform(trainset[numeric_colu
```

In [158]:
```
# Separate numeric and categorical data
numeric_columns = testset.select_dtypes(include=['number']).columns

# Apply mean imputation for numeric columns
num_imputer = SimpleImputer(strategy="mean")
testset[numeric_columns] = num_imputer.fit_transform(testset[numeric_columr
```

In [159]:
```
#Check the null values
missing_values_count = trainset.isna().sum()
missing_values_count
```

Out[159]:
```
vehicle_type                         0
towing_and_articulation              0
vehicle_manoeuvre                    0
vehicle_location_restricted_lane     0
junction_location                    0
skidding_and_overturning             0
hit_object_in_carriageway            0
vehicle_leaving_carriageway          0
hit_object_off_carriageway           0
first_point_of_impact                0
vehicle_left_hand_drive              0
journey_purpose_of_driver            0
sex_of_driver                        0
age_band_of_driver                   0
engine_capacity_cc                   0
propulsion_code                      0
age_of_vehicle                       0
generic_make_model                   0
driver_imd_decile                    0
driver_home_area_type                0
casualty_class                       0
sex_of_casualty                      0
age_band_of_casualty                 0
casualty_severity                    0
pedestrian_location                  0
pedestrian_movement                  0
car_passenger                        0
casualty_home_area_type              0
dtype: int64
```

In [160]:
```
missing_values_count = testset.isna().sum()
missing_values_count
```

Out[160]:
```
vehicle_type                       0
towing_and_articulation            0
vehicle_manoeuvre                  0
vehicle_location_restricted_lane   0
junction_location                  0
skidding_and_overturning           0
hit_object_in_carriageway          0
vehicle_leaving_carriageway        0
hit_object_off_carriageway         0
first_point_of_impact              0
vehicle_left_hand_drive            0
journey_purpose_of_driver          0
sex_of_driver                      0
age_band_of_driver                 0
engine_capacity_cc                 0
propulsion_code                    0
age_of_vehicle                     0
generic_make_model                 0
driver_imd_decile                  0
driver_home_area_type              0
casualty_class                     0
sex_of_casualty                    0
age_band_of_casualty               0
casualty_severity                  0
pedestrian_location                0
pedestrian_movement                0
car_passenger                      0
casualty_home_area_type            0
dtype: int64
```

6. Conclusion

In this Group assignment we successfully created a dataframe of casualty data. In the Individual assignment we will be generating a predictive model which predicts average Netflix rating based on independent variables which come from the casualty. By predicting ratings for casualty severity we should be able to determine help Predict Casualty Severity,Enhance Emergency Response,Improve Health Outcomes, Data-Driven Policy Recommendations

In [161]:
```
ytrain = pd.DataFrame(trainset.loc[:, 'casualty_severity'])  # This selects
Xtrain = trainset.drop(columns=['casualty_severity'])  # This excludes the
# This selects the 'casualty_severity' column as y
ytest = pd.DataFrame(testset.loc[:, 'casualty_severity'])
  # This excludes the 'casualty_severity' column for X
Xtest = testset.drop(columns=['casualty_severity'])
```

In [162]:
```
# Save the trainset as an Excel file
trainset.to_excel("trainset_inspection.xlsx", index=False)
testset.to_excel("testset_inspection.xlsx", index=False)
print("DataFrames saved successfully as excel")
```

```
DataFrames saved successfully as excel
```

In [166]:
```
%%javascript
var nb = IPython.notebook;
var kernel = IPython.notebook.kernel;
var command = "NOTEBOOK_FULL_PATH = '" + nb.notebook_path + "'";
kernel.execute(command);
```

In [167]:
```
import io
from nbformat import read, NO_CONVERT

with io.open(NOTEBOOK_FULL_PATH.split("/")[-1], 'r', encoding='utf-8') as f
    nb = read(f, NO_CONVERT)

word_count = 0
for cell in nb.cells:
    if cell.cell_type == "markdown":
        word_count += len(cell['source'].replace('#', '').lstrip().split('
print(f"Word count: {word_count}")
```

```
Word count: 1941
```

In [168]:
```
# Finish Timer
notebook_duration = round((time.time() - startnb)/60, 5)
print(f'The completion of the notebook took {notebook_duration} minutes.')
```

```
The completion of the notebook took 2.43911 minutes.
```

In [ ]: