ASTON UNIVERSITY

Group Number - 5

Candidate Numbers:

1. 838651
2. 478724
3. 240315

Big Data Coursework - UK Government (Department for Transport) Road Safety Dataset

INTRODUCTION

This notebook serves as a Data Preprocessing and Exploratory Data Analysis report on the United Kingdom Government's Department for Transport Road Safety dataset. It is part of a group assignment for the Big Data for Decision Making module of the MSc in Business Analytics course at Aston University, Birmingham. We will compile a dataset using the UK road safety data files available from the UK Department for Transport. The page includes a description of the data and links to download the actual data files.

The final dataset created from this group assignment will be used as input for the individual assignment, which involves modeling, evaluation, and deployment of machine learning predictive methods to address a business problem for stakeholders.

Both the group and individual assignments follow the Machine Learning Workflow and CRISP-DM methodologies for a comprehensive end-to-end project. The structure of the report is outlined below.

Table of Contents

1. Business Objective

Road accident statistics are vital for national governments in developing and monitoring road safety policies. They also support a wide range of research aimed at reducing risks to road users. This research can be government-funded, sponsored by independent entities, or conducted by organizations and individuals independently. Researchers and consultancies require access to this data to analyze the risks and consequences of accidents and to predict the potential impact of policy changes, regulations, and safety measures.

Expected Results

Improved Safety Outcomes: Stakeholders can anticipate a reduction in the severity of road accidents by proactively identifying and addressing high-risk factors.

Efficiency Gains: Predictive models help stakeholders allocate resources more effectively, resulting in cost savings in emergency response, infrastructure maintenance, and insurance claims management.

Data-Driven Decision Making: Utilizing predictive analytics enables stakeholders to make informed, data-driven decisions, leading to better policy development, risk management practices, and overall enhancements in road safety.

Business-Specific Problem:

Forecasting the Severity of Road Traffic Accidents to Enhance Resource Allocation and Emergency Response for Various Stakeholders.

The aim is to develop a predictive model that can accurately evaluate the severity of road traffic accidents by considering various factors such as the time of the accident, weather conditions, location, road type, vehicle type, and driver demographics. This model is intended to assist emergency services and public safety organizations in optimizing resource allocation and enhancing emergency response times by predicting whether an accident will result in slight, serious, or fatal injuries. Additionally, this prediction can be beneficial for insurance companies in risk assessment and premium pricing, as well as for policymakers in implementing targeted road safety measures.

Our target variable is the severity of road accidents, which means we are addressing a classification problem in machine learning. In this scenario, the goal is to predict the categorical class labels of new instances based on historical data. Specifically, we aim to classify the severity of a road accident into categories such as slight, serious, or fatal.

2. Data Understanding, Importing Libraries and Preparing Environment

Road safety casualty statistics, commonly referred to as STATS19, are collected independently by each police force. This name originates from the original form used for data collection.

With continuous improvements in data pipelines and validation processes, modern accident data tends to be more accurate and consistent. Significant efforts and funding have been dedicated to assisting officers in data collection.

Historically, police officers collected data using handwritten paper forms, which lacked the scrutiny applied to today's digital records. Now, many officers have access to mobile phone applications and advanced data systems that validate entries in real-time. For instance, officers can use a mobile phone's GPS to obtain precise coordinates, rather than relying on written descriptions based on nearby landmarks.

UK Road Safety Data

It is Published annually (since 1979) by UK Department for Transport. Each year's accidents data consists of 3 linked CSV files (via accident index):

• Collision (Accidents)

• Vehicles

• Casualties

We are utilizing data from three datasets for the year 2022: Collision, Vehicles, and Casualties. These datasets are linked using the Accident Index, a unique numerical identifier specific to each accident.

By merging the collision, vehicle, and casualty datasets, we can gain a more comprehensive understanding of each accident. This integration enhances the accuracy and utility of our predictive models by leveraging a broader range of features. It allows us to identify patterns and correlations that might not be apparent when analyzing a single dataset.

The Accident Index, which combines the accident year and accident reference number, serves as a unique ID. This ID enables us to join the Collision, Vehicle, and Casualty files, creating a comprehensive dataset with all relevant variables that can influence accident severity.

Importing Libraries and Preparing Environment

In [1]:
```
#Base Libraries
import re
import time
import datetime
import warnings
import numpy as np
import pandas as pd
#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
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
# Disable auto-scrolling for outputs
```

In [2]:
```
Start_time = time.time()
```

3. Data Preparation

3.1 Loading Datasets - 3 datasets of the year 2020, that are Collision, Vehicles and Casualties

We are using data from the 3 data files of the year 2022, that are mentioned as Collision, Vehicles and Casualties, by combining them together using Accident Index, which is the variable which links them together with a unique accident specific numerical identifier.

Combining data from the collision, vehicle, and casualty datasets would provide a more comprehensive view of each accident, potentially improving the accuracy and usefulness of your predictive models.

In [3]:
```
# Step 3.1: Loading the Data Files

# Load the Data Files
Casualty_data = pd.read_csv('Casualty-2022.csv')
Vehicle_data = pd.read_csv('Vehicle-2022.csv')
Collision_data = pd.read_csv('Collision-2022.csv')
```

```
C:\Users\bhara\AppData\Local\Temp\ipykernel_14984\2524049792.py:4: DtypeWarning: Columns (0,2) have mixed types. Specify dt
ype option on import or set low_memory=False.
  Casualty_data = pd.read_csv('Casualty-2022.csv')
C:\Users\bhara\AppData\Local\Temp\ipykernel_14984\2524049792.py:5: DtypeWarning: Columns (0,2) have mixed types. Specify dt
ype option on import or set low_memory=False.
  Vehicle_data = pd.read_csv('Vehicle-2022.csv')
C:\Users\bhara\AppData\Local\Temp\ipykernel_14984\2524049792.py:6: DtypeWarning: Columns (0,2) have mixed types. Specify dt
ype option on import or set low_memory=False.
  Collision_data = pd.read_csv('Collision-2022.csv')
```

3.2 Datasets View - 3 datasets of the year 2020, that are Collision, Vehicles and Casualties

We are providing the basic view of the three data files namely Collision, Vehicle and Casualty below, along with their Shape and information about the names and types of variables they hold for reference.

In [4]:
```
Collision_data.head()
```

Out[4]:

	accident_index	accident_year	accident_reference	location_easting_osgr	location_northing_osgr	longitude	latitude	police_force	accident_severity
0	2022010352073	2022	10352073	525199.0	177928.0	-0.198224	51.486454	1	3
1	2022010352573	2022	10352573	546214.0	179866.0	0.105042	51.498830	1	3
2	2022010352575	2022	10352575	551119.0	174789.0	0.173482	51.451924	1	3
3	2022010352578	2022	10352578	528889.0	192230.0	-0.139873	51.614153	1	3
4	2022010352580	2022	10352580	539773.0	190404.0	0.016495	51.595151	1	3

5 rows × 36 columns

In [5]:
```
Collision_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106004 entries, 0 to 106003
Data columns (total 36 columns):
 #   Column                                       Non-Null Count   Dtype
---  ------                                       --------------   -----
 0   accident_index                               106004 non-null  object
 1   accident_year                                106004 non-null  int64
 2   accident_reference                           106004 non-null  object
 3   location_easting_osgr                        105982 non-null  float64
 4   location_northing_osgr                       105982 non-null  float64
 5   longitude                                    105982 non-null  float64
 6   latitude                                     105982 non-null  float64
 7   police_force                                 106004 non-null  int64
 8   accident_severity                            106004 non-null  int64
 9   number_of_vehicles                           106004 non-null  int64
 10  number_of_casualties                         106004 non-null  int64
 11  date                                         106004 non-null  object
 12  day_of_week                                  106004 non-null  int64
 13  time                                         106004 non-null  object
 14  local_authority_district                     106004 non-null  int64
 15  local_authority_ons_district                 106004 non-null  object
 16  local_authority_highway                      106004 non-null  object
 17  first_road_class                             106004 non-null  int64
 18  first_road_number                            106004 non-null  int64
 19  road_type                                    106004 non-null  int64
 20  speed_limit                                  106004 non-null  int64
 21  junction_detail                              106004 non-null  int64
 22  junction_control                             106004 non-null  int64
 23  second_road_class                            106004 non-null  int64
 24  second_road_number                           106004 non-null  int64
 25  pedestrian_crossing_human_control            106004 non-null  int64
 26  pedestrian_crossing_physical_facilities      106004 non-null  int64
 27  light_conditions                             106004 non-null  int64
 28  weather_conditions                           106004 non-null  int64
 29  road_surface_conditions                      106004 non-null  int64
 30  special_conditions_at_site                   106004 non-null  int64
 31  carriageway_hazards                          106004 non-null  int64
 32  urban_or_rural_area                          106004 non-null  int64
 33  did_police_officer_attend_scene_of_accident  106004 non-null  int64
 34  trunk_road_flag                              106004 non-null  int64
 35  lsoa_of_accident_location                    106004 non-null  object
dtypes: float64(4), int64(25), object(7)
memory usage: 29.1+ MB
```

In [6]:
```
Collision_data.shape
```

Out[6]:
```
(106004, 36)
```

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

Out[7]:

	accident_index	accident_year	accident_reference	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre	vehicle_direction_from	v
0	2022010352073	2022	10352073	1	19	0	9	3	
1	2022010352073	2022	10352073	2	3	0	18	1	
2	2022010352573	2022	10352573	1	9	0	18	7	
3	2022010352573	2022	10352573	2	9	0	13	7	
4	2022010352575	2022	10352575	1	9	0	18	4	

5 rows × 28 columns

In [8]:
```
Vehicle_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193545 entries, 0 to 193544
Data columns (total 28 columns):
 #   Column                          Non-Null Count   Dtype 
---  ------                          --------------   ----- 
 0   accident_index                  193545 non-null  object
 1   accident_year                   193545 non-null  int64 
 2   accident_reference              193545 non-null  object
 3   vehicle_reference               193545 non-null  int64 
 4   vehicle_type                    193545 non-null  int64 
 5   towing_and_articulation         193545 non-null  int64 
 6   vehicle_manoeuvre               193545 non-null  int64 
 7   vehicle_direction_from          193545 non-null  int64 
 8   vehicle_direction_to            193545 non-null  int64 
 9   vehicle_location_restricted_lane  193545 non-null  int64 
 10  junction_location               193545 non-null  int64 
 11  skidding_and_overturning        193545 non-null  int64 
 12  hit_object_in_carriageway       193545 non-null  int64 
 13  vehicle_leaving_carriageway     193545 non-null  int64 
 14  hit_object_off_carriageway      193545 non-null  int64 
 15  first_point_of_impact           193545 non-null  int64 
 16  vehicle_left_hand_drive         193545 non-null  int64 
 17  journey_purpose_of_driver       193545 non-null  int64 
 18  sex_of_driver                   193545 non-null  int64 
 19  age_of_driver                   193545 non-null  int64 
 20  age_band_of_driver              193545 non-null  int64 
 21  engine_capacity_cc              193545 non-null  int64 
 22  propulsion_code                 193545 non-null  int64 
 23  age_of_vehicle                  193545 non-null  int64 
 24  generic_make_model              193545 non-null  object
 25  driver_imd_decile               193545 non-null  int64 
 26  driver_home_area_type           193545 non-null  int64 
 27  lsoa_of_driver                  193545 non-null  object
dtypes: int64(24), object(4)
memory usage: 41.3+ MB
```

In [9]:
```
Vehicle_data.shape
```

Out[9]:
```
(193545, 28)
```

In [10]:
```
Casualty_data.head()
```

Out[10]:

	accident_index	accident_year	accident_reference	vehicle_reference	casualty_reference	casualty_class	sex_of_casualty	age_of_casualty	age_band_of_
0	2022010352073	2022	10352073	2	1	1	1	17	
1	2022010352573	2022	10352573	1	1	1	2	42	
2	2022010352575	2022	10352575	1	1	1	1	20	
3	2022010352578	2022	10352578	1	1	1	1	46	
4	2022010352578	2022	10352578	1	2	2	1	45	

In [11]:
```
Casualty_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135480 entries, 0 to 135479
Data columns (total 19 columns):
 #   Column                              Non-Null Count   Dtype 
---  ------                              --------------   ----- 
 0   accident_index                      135480 non-null  object
 1   accident_year                       135480 non-null  int64 
 2   accident_reference                  135480 non-null  object
 3   vehicle_reference                   135480 non-null  int64 
 4   casualty_reference                  135480 non-null  int64 
 5   casualty_class                      135480 non-null  int64 
 6   sex_of_casualty                     135480 non-null  int64 
 7   age_of_casualty                     135480 non-null  int64 
 8   age_band_of_casualty                135480 non-null  int64 
 9   casualty_severity                   135480 non-null  int64 
 10  pedestrian_location                 135480 non-null  int64 
 11  pedestrian_movement                 135480 non-null  int64 
 12  car_passenger                       135480 non-null  int64 
 13  bus_or_coach_passenger              135480 non-null  int64 
 14  pedestrian_road_maintenance_worker  135480 non-null  int64 
 15  casualty_type                       135480 non-null  int64 
 16  casualty_home_area_type             135480 non-null  int64 
 17  casualty_imd_decile                 135480 non-null  int64 
 18  lsoa_of_casualty                    135480 non-null  object
dtypes: int64(16), object(3)
memory usage: 19.6+ MB
```

In [12]:
```
Casualty_data.shape
```

Out[12]:
```
(135480, 19)
```

3.3 Datasets Variable Selection - From Collision, Vehicles and Casualties

This selection is based on specific logic. Since many of the variables are not related to predicting accident severity, they will be excluded from our analysis.

The names of the selected variables from all three data files can be seen in the next block of code. The included variables will be analyzed for their importance using statistical methods. Additional variables may be dropped later based on their predictive power, which will be assessed using statistical methods. More variables can also be added through feature engineering as needed.

In [13]:
```
# Step 6: Filter Relevant Variables
# List of relevant variables for each dataset
Collision = [
    'accident_index', 'longitude', 'latitude', 'accident_severity',
    'number_of_vehicles', 'number_of_casualties', 'date', 'day_of_week',
    'time', 'road_type', 'speed_limit', 'light_conditions',
    'weather_conditions', 'road_surface_conditions', 'urban_or_rural_area',
    'did_police_officer_attend_scene_of_accident'
]

Vehicle = [
    'accident_index', 'vehicle_type', 'vehicle_manoeuvre',
    'sex_of_driver', 'age_of_driver', 'age_of_vehicle'
]

Casualty = [
    'accident_index', 'casualty_class', 'sex_of_casualty',
    'age_of_casualty', 'casualty_severity', 'casualty_type'
]

# Filter the datasets to keep only the relevant variables
Collision_data = Collision_data[Collision]
Vehicle_data = Vehicle_data[Vehicle]
Casualty_data = Casualty_data[Casualty]
```

3.4 Merging Datasets, Droping missing values and Removing Duplicates - Collision, Vehicles and Casualties.

Datasets are merged here using the unique accident_index variable present in all datasets corresponding to each case of Accident. The view, shape and information about the merged files are also presented below.

In [14]:
```
# Step 7: Merge the Datasets
# Merge collision and vehicle data on 'accident_index'
new = pd.merge(Collision_data, Vehicle_data, on='accident_index', how='inner')

# Merge the result with casualty data on 'accident_index'
df = pd.merge(new, Casualty_data, on='accident_index', how='inner')
```

In [15]:
```
df.shape
```

Out[15]:
```
(168884, 26)
```

In [16]:
```
# Check for missing values in the dataset
missing_values = df.isnull().sum()

# Print the number of missing values for each column
print("Missing values in each column:")
print(missing_values)
```

```
Missing values in each column:
accident_index                                0
longitude                                    34
latitude                                     34
accident_severity                             0
number_of_vehicles                            0
number_of_casualties                          0
date                                          0
day_of_week                                   0
time                                          0
road_type                                     0
speed_limit                                   0
light_conditions                              0
weather_conditions                            0
road_surface_conditions                       0
urban_or_rural_area                           0
did_police_officer_attend_scene_of_accident   0
vehicle_type                                  0
vehicle_manoeuvre                             0
sex_of_driver                                 0
age_of_driver                                 0
age_of_vehicle                                0
casualty_class                                0
sex_of_casualty                               0
age_of_casualty                               0
casualty_severity                             0
casualty_type                                 0
dtype: int64
```

In [17]:
```
# Step 8: Handle Missing Data
# Drop rows with missing values (example)
df = df.dropna()
```

In [18]:
```
# Check for missing values in the dataset
missing_values = df.isnull().sum()

# Print the number of missing values for latitude and longitude as they had some missing values
print("Missing values in each column:")
print(missing_values[['longitude', 'longitude']])
```

```
Missing values in each column:
longitude    0
longitude    0
dtype: int64
```

Many of our variable rows currently have missing values represented by -1. We plan to eliminate these records to reduce the computational burden of imputing values based on certain criteria, and to ensure our analysis is not significantly impacted.

In [19]:
```python
# check -1 values in the final_data dataframe

# List of columns that use -1 as missing data
columns_with_minus_one = [
    'accident_index', 'longitude', 'latitude', 'accident_severity',
    'number_of_vehicles', 'number_of_casualties', 'date', 'day_of_week',
    'time', 'road_type', 'speed_limit', 'light_conditions',
    'weather_conditions', 'road_surface_conditions', 'urban_or_rural_area',
    'did_police_officer_attend_scene_of_accident', 'vehicle_type', 'vehicle_manoeuvre',
    'sex_of_driver', 'age_of_driver', 'age_of_vehicle', 'casualty_class', 'sex_of_casualty',
    'age_of_casualty', 'casualty_severity', 'casualty_type'
]

# Function to count -1 values in specified columns
def count_minus_one_values(df, columns):
    minus_one_counts = {}
    for column in columns:
        minus_one_counts[column] = (df[column] == -1).sum()
    return minus_one_counts

# Count -1 values before removal
minus_one_counts_before = count_minus_one_values(df, columns_with_minus_one)
print("Counts of -1 values before removal:")
for column, count in minus_one_counts_before.items():
    print(f"{column}: {count}")

# Filter out rows where any of these columns contain -1
for column in columns_with_minus_one:
    df = df[df[column] != -1]
```

```
Counts of -1 values before removal:
accident_index: 0
longitude: 0
latitude: 0
accident_severity: 0
number_of_vehicles: 0
number_of_casualties: 0
date: 0
day_of_week: 0
time: 0
road_type: 0
speed_limit: 0
light_conditions: 0
weather_conditions: 0
road_surface_conditions: 533
urban_or_rural_area: 0
did_police_officer_attend_scene_of_accident: 0
vehicle_type: 171
vehicle_manoeuvre: 1529
sex_of_driver: 0
age_of_driver: 24592
age_of_vehicle: 32726
casualty_class: 0
sex_of_casualty: 1797
age_of_casualty: 4254
casualty_severity: 0
casualty_type: 50
```

In [20]:
```
# Count -1 values after removal
minus_one_counts_after = count_minus_one_values(df, columns_with_minus_one)
print("\nCounts of -1 values after removal:")
for column, count in minus_one_counts_after.items():
    print(f"{column}: {count}")

# Verify the changes
print("\nRemaining data after removing rows with -1 in columns:")
print(df.shape)
```

```
Counts of -1 values after removal:
accident_index: 0
longitude: 0
latitude: 0
accident_severity: 0
number_of_vehicles: 0
number_of_casualties: 0
date: 0
day_of_week: 0
time: 0
road_type: 0
speed_limit: 0
light_conditions: 0
weather_conditions: 0
road_surface_conditions: 0
urban_or_rural_area: 0
did_police_officer_attend_scene_of_accident: 0
vehicle_type: 0
vehicle_manoeuvre: 0
sex_of_driver: 0
age_of_driver: 0
age_of_vehicle: 0
casualty_class: 0
sex_of_casualty: 0
age_of_casualty: 0
casualty_severity: 0
casualty_type: 0

Remaining data after removing rows with -1 in columns:
(118104, 26)
```

Now we remove duplicate Values based on Accident Index, as it can skew our analysis significantly.

In [21]:
```
# Check for duplicate rows based on 'accident_index'
duplicate_count = df.duplicated(subset='accident_index').sum()
print(f'Number of duplicate rows based on accident_index: {duplicate_count}')

# Get the number of unique accident_index
unique_accident_indices = df['accident_index'].nunique()
print(f'Number of unique accident_index: {unique_accident_indices}')

# Check the shape of the dataset
print(f'Shape of the dataset: {df.shape}')

# Remove duplicate rows based on 'accident_index', keeping the first occurrence
df = df.drop_duplicates(subset='accident_index', keep='first')

# Verify the changes
duplicate_count_after = df.duplicated(subset='accident_index').sum()
print(f'Number of duplicate rows after removal: {duplicate_count_after}')
print(f'Shape of data after removing duplicate rows: {df.shape}')
```

```
Number of duplicate rows based on accident_index: 60573
Number of unique accident_index: 57531
Shape of the dataset: (118104, 26)
Number of duplicate rows after removal: 0
Shape of data after removing duplicate rows: (57531, 26)
```

The Final data file has thus 57531 records and a total of 26 variables to hold. This is the data prepared for Analysis and would be further cleaned and pre-processed as needed.

4. Feature Dataset Construction

By modifying and creating the features, we can capture more nuanced information that might be important for predicting accident severity. This process not only enhances the predictive power of your models but also makes them more interpretable, providing valuable insights for your stakeholders.

First we need to convert categorical variables into categories which are currently labeled as Integer due to the number encoding of categories. And also the date and time variables labeled as objects, needs to be converted to datetime format.

In [22]:
```
# List of categorical variables
categorical_cols = [
    'accident_severity', 'day_of_week', 'road_type', 'light_conditions',
    'weather_conditions', 'road_surface_conditions', 'urban_or_rural_area',
    'did_police_officer_attend_scene_of_accident',
    'vehicle_type', 'vehicle_manoeuvre', 'sex_of_driver', 'casualty_class',
    'sex_of_casualty', 'casualty_severity', 'casualty_type'
]

# Convert categorical variables to category dtype
for col in categorical_cols:
    df[col] = df[col].astype('category')

# Print the dataframe to verify changes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 57531 entries, 0 to 168883
Data columns (total 26 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   accident_index                               57531 non-null  object
 1   longitude                                    57531 non-null  float64
 2   latitude                                     57531 non-null  float64
 3   accident_severity                            57531 non-null  category
 4   number_of_vehicles                           57531 non-null  int64
 5   number_of_casualties                         57531 non-null  int64
 6   date                                         57531 non-null  object
 7   day_of_week                                  57531 non-null  category
 8   time                                         57531 non-null  object
 9   road_type                                    57531 non-null  category
 10  speed_limit                                  57531 non-null  int64
 11  light_conditions                             57531 non-null  category
 12  weather_conditions                           57531 non-null  category
 13  road_surface_conditions                      57531 non-null  category
 14  urban_or_rural_area                          57531 non-null  category
 15  did_police_officer_attend_scene_of_accident  57531 non-null  category
 16  vehicle_type                                 57531 non-null  category
 17  vehicle_manoeuvre                            57531 non-null  category
 18  sex_of_driver                                57531 non-null  category
 19  age_of_driver                                57531 non-null  int64
 20  age_of_vehicle                               57531 non-null  int64
 21  casualty_class                               57531 non-null  category
 22  sex_of_casualty                              57531 non-null  category
 23  age_of_casualty                              57531 non-null  int64
 24  casualty_severity                            57531 non-null  category
 25  casualty_type                                57531 non-null  category
dtypes: category(15), float64(2), int64(6), object(3)
memory usage: 6.1+ MB
```

We have successfully converted all categorical variables to the appropriate categorical datatype. The numerical variables are already correctly classified, with 2 as floats and 6 as integers. Additionally, we have 3 object-type variables.

However, since the object-type variables include Date and Time, which represent temporal data, they should not remain as object types. Therefore, we will combine them and convert their datatype to datetime format. This conversion will enable us to extract more meaningful features and facilitate easier temporal calculations.

In [23]:
```
# Combine 'date' and 'time' into a single datetime column
df['datetime'] = pd.to_datetime(df['date'].astype(str) + ' ' + df['time'].astype(str), format='%d/%m/%Y %H:%M')

# View the first few rows of the DataFrame with data and time columns
df[['date', 'time', 'datetime']].info()
print('\n')
# View the first few rows of the DataFrame with data and time columns
df[['date', 'time', 'datetime']].head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 57531 entries, 0 to 168883
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      57531 non-null  object
 1   time      57531 non-null  object
 2   datetime  57531 non-null  datetime64[ns]
dtypes: datetime64[ns](1), object(2)
memory usage: 1.8+ MB
```

Out[23]:

	date	time	datetime
0	05/01/2022	16:40	2022-01-05 16:40:00
4	01/01/2022	01:15	2022-01-01 01:15:00
6	01/01/2022	02:24	2022-01-01 02:24:00
10	01/01/2022	02:30	2022-01-01 02:30:00
22	01/01/2022	02:55	2022-01-01 02:55:00

In [24]:
```
# Drop the original 'date' and 'time' columns
df.drop(columns=['date', 'time'], inplace=True)

# Verify the data types
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 57531 entries, 0 to 168883
Data columns (total 25 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   accident_index                               57531 non-null  object
 1   longitude                                    57531 non-null  float64
 2   latitude                                     57531 non-null  float64
 3   accident_severity                            57531 non-null  category
 4   number_of_vehicles                           57531 non-null  int64
 5   number_of_casualties                         57531 non-null  int64
 6   day_of_week                                  57531 non-null  category
 7   road_type                                    57531 non-null  category
 8   speed_limit                                  57531 non-null  int64
 9   light_conditions                             57531 non-null  category
 10  weather_conditions                           57531 non-null  category
 11  road_surface_conditions                      57531 non-null  category
 12  urban_or_rural_area                          57531 non-null  category
 13  did_police_officer_attend_scene_of_accident  57531 non-null  category
 14  vehicle_type                                 57531 non-null  category
 15  vehicle_manoeuvre                            57531 non-null  category
 16  sex_of_driver                                57531 non-null  category
 17  age_of_driver                                57531 non-null  int64
 18  age_of_vehicle                               57531 non-null  int64
 19  casualty_class                               57531 non-null  category
 20  sex_of_casualty                              57531 non-null  category
 21  age_of_casualty                              57531 non-null  int64
 22  casualty_severity                            57531 non-null  category
 23  casualty_type                                57531 non-null  category
 24  datetime                                     57531 non-null  datetime64[ns]
dtypes: category(15), datetime64[ns](1), float64(2), int64(6), object(1)
memory usage: 5.7+ MB
None
```

In [25]:
```
df.shape
```

Out[25]: (57531, 25)

Before we start splitting the data, we need to map the category code of categorical varibles back to their respective labels using the Data Guide. This will ensure that the visualizations are more interpretable and meaningful for analysis.

In [26]:
```
# Define the mappings
accident_severity_mapping = { 1: 'Fatal', 2: 'Serious', 3: 'Slight'
}

day_of_week_mapping = {
    1: 'Sunday', 2: 'Monday', 3: 'Tuesday', 4: 'Wednesday', 5: 'Thursday', 6: 'Friday', 7: 'Saturday'
}

road_type_mapping = {
    1: 'Roundabout', 2: 'One way street', 3: 'Dual carriageway', 6: 'Single carriageway', 7: 'Slip road',
    9: 'Unknown', 12: 'One way street/Slip road', -1: 'Data missing or out of range'
}

light_conditions_mapping = {
    1: 'Daylight', 4: 'Darkness - lights lit', 5: 'Darkness - lights unlit', 6: 'Darkness - no lighting',
    7: 'Darkness - lighting unknown', -1: 'Data missing or out of range'
}

weather_conditions_mapping = {
    1: 'Fine no high winds', 2: 'Raining no high winds', 3: 'Snowing no high winds', 4: 'Fine + high winds',
    5: 'Raining + high winds', 6: 'Snowing + high winds', 7: 'Fog or mist', 8: 'Other', 9: 'Unknown',
    -1: 'Data missing or out of range'
}

road_surface_conditions_mapping = {
    1: 'Dry', 2: 'Wet or damp', 3: 'Snow', 4: 'Frost or ice', 5: 'Flood over 3cm. deep', 6: 'Oil or diesel',
    7: 'Mud', -1: 'Data missing or out of range', 9: 'unknown (self reported)'
}

urban_or_rural_area_mapping = {
    1: 'Urban', 2: 'Rural', 3: 'Unallocated', -1: 'Data missing or out of range'
}

did_police_officer_attend_scene_of_accident_mapping = {
    1: 'Yes', 2: 'No', 3: 'No - self reported',
    -1: 'Data missing or out of range'
}

vehicle_type_mapping = {
    1: 'Pedal cycle', 2: 'Motorcycle 50cc under', 3: 'Motorcycle 125cc under', 4: 'Motorcycle 125cc-500cc',
    5: 'Motorcycle 500cc+', 8: 'Taxi/Private hire car', 9: 'Car', 10: 'Minibus (8 - 16 seats)',
    11: 'Bus or coach (17+ seats)', 16: 'Ridden horse', 17: 'Agricultural vehicle', 18: 'Tram',
    19: 'Van / Goods 3.5 tonnes mgw', 20: 'Goods 3.5t.- 7.5t', 21: 'Goods 7.5 tonnes mgw+',
    22: 'Mobility scooter', 23: 'Electric motorcycle', 90: 'Other vehicle', 97: 'Motorcycle - unknown cc',
    98: 'Goods vehicle - unknown weight', 99: 'Unknown vehicle type (self rep only)', -1: 'Data missing or out of range'
}

vehicle_manoeuvre_mapping = {
    1: 'Reversing', 2: 'Parked', 3: 'Waiting to go - held up', 4: 'Slowing or stopping', 5: 'Moving off', 6: 'U-turn',
    7: 'Turning left', 8: 'Waiting to turn left', 9: 'Turning right', 10: 'Waiting to turn right', 11: 'Changing lane to lef
    12: 'Changing lane to right', 13: 'Overtaking moving vehicle - offside', 14: 'Overtaking static vehicle - offside',
    15: 'Overtaking - nearside', 16: 'Going ahead left-hand bend', 17: 'Going ahead right-hand bend', 18: 'Going ahead other
    99: 'unknown (self reported)', -1: 'Data missing or out of range'
}

sex_of_driver_mapping = {
    1: 'Male', 2: 'Female', 3: 'Not known', -1: 'Data missing or out of range'
}

casualty_class_mapping = {
    1: 'Driver or rider', 2: 'Passenger', 3: 'Pedestrian'
}

sex_of_casualty_mapping = {
    1: 'Male', 2: 'Female', 9: 'unknown (self reported)', -1: 'Data missing or out of range'
}

casualty_severity_mapping = {
    1: 'Fatal', 2: 'Serious', 3: 'Slight'
}

casualty_type_mapping = {
    0: 'Pedestrian', 1: 'Cyclist', 2: 'Motorcycle 50cc', 3: 'Motorcycle 125cc',
    4: 'Motorcycle 125cc to 500cc', 5: 'Motorcycle over 500cc',
    8: 'Taxi/Private hire car', 9: 'Car occupant', 10: 'Minibus (8 - 16 passenger)', 11: 'Bus or coach(17+)',
    16: 'Horse rider', 17: 'Agricultural vehicle', 18: 'Tram', 19: 'Van / Goods vehicle (3.5 tonnes mgw)',
    20: 'Goods vehicle (3.5t.- 7.5t.)', 21: 'Goods vehicle (7.5 tonnes mgw+)',
    22: 'Mobility scooter rider', 23: 'Electric motorcycle', 90: 'Other vehicle occupant', 97: 'Motorcycle - unknown cc',
    98: 'Goods vehicle (unknown weight)', 99: 'Unknown vehicle type (self rep only)', -1: 'Data missing or out of range'
}
```

In [27]:
```
# Apply the mappings to the final_data DataFrame
df['accident_severity'] = df['accident_severity'].map(accident_severity_mapping)
df['day_of_week'] = df['day_of_week'].map(day_of_week_mapping)
df['road_type'] = df['road_type'].map(road_type_mapping)
df['light_conditions'] = df['light_conditions'].map(light_conditions_mapping)
df['weather_conditions'] = df['weather_conditions'].map(weather_conditions_mapping)
df['road_surface_conditions'] = df['road_surface_conditions'].map(road_surface_conditions_mapping)
df['urban_or_rural_area'] = df['urban_or_rural_area'].map(urban_or_rural_area_mapping)
df['did_police_officer_attend_scene_of_accident'] = df['did_police_officer_attend_scene_of_accident'].map(did_police_officer
df['vehicle_type'] = df['vehicle_type'].map(vehicle_type_mapping)
df['vehicle_manoeuvre'] = df['vehicle_manoeuvre'].map(vehicle_manoeuvre_mapping)
df['sex_of_driver'] = df['sex_of_driver'].map(sex_of_driver_mapping)
df['casualty_class'] = df['casualty_class'].map(casualty_class_mapping)
df['sex_of_casualty'] = df['sex_of_casualty'].map(sex_of_casualty_mapping)
df['casualty_severity'] = df['casualty_severity'].map(casualty_severity_mapping)
df['casualty_type'] = df['casualty_type'].map(casualty_type_mapping)
```

In [28]:
```
df.head()
```

Out[28]:

	accident_index	longitude	latitude	accident_severity	number_of_vehicles	number_of_casualties	day_of_week	road_type	speed_limit	light_condit
0	2022010352073	-0.198224	51.486454	Slight	2	1	Wednesday	Single carriageway	30	Darkness - li
4	2022010352575	0.173482	51.451924	Slight	2	1	Saturday	Single carriageway	30	Darkness - li
6	2022010352578	-0.139873	51.614153	Slight	2	2	Saturday	Single carriageway	30	Darkness - li
10	2022010352580	0.016495	51.595151	Slight	4	3	Saturday	Dual carriageway	50	Darkness - li
22	2022010352588	0.061626	51.512146	Serious	1	5	Saturday	Single carriageway	30	Darkness - li

5 rows × 25 columns

In [29]:
```
# Replace 'Serious' and 'Fatal' with 'Serious' in the 'accident_severity' column
df['accident_severity'] = df['accident_severity'].replace({'Serious': 'Serious', 'Fatal': 'Serious'})

# Verify the changes
print(df['accident_severity'].value_counts())
```

```
accident_severity
Slight     44226
Serious    13305
Name: count, dtype: int64
```

In [30]:
```
# Replace 'Serious' and 'Fatal' with 'Serious' in the 'casualty_severity' column
df['casualty_severity'] = df['casualty_severity'].replace({'Serious': 'Serious', 'Fatal': 'Serious'})

# Verify the changes
print(df['casualty_severity'].value_counts())
```

```
casualty_severity
Slight     45618
Serious    11913
Name: count, dtype: int64
```

5. Data Splitting(Train-Test split)

No we are doing a train-test split of our data to prepare it for further analysis with exploratory data analysis. We will use StratifiedShuffleSplit method of ScikitLearn Library of Python. It would split data in equal proportions of all three categories of out Target variable accident_severity, Slight, Severe and Fatal.

Only Training data shall be used for further analysis from this point. This early Train-Test split would allow us to remove bias and data snooping where we may be tempted to fit the modal correctly to the test set before evaluation.

In [31]:
```
from sklearn.model_selection import StratifiedShuffleSplit
import pandas as pd

# Assuming 'df' is your DataFrame with the complete dataset
# Define the features (X) and the target (y)
X = df.drop(columns=['accident_severity'])
y = df['accident_severity']

# Initialize StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=42)

# Perform the split
for train_index, test_index in sss.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

# Combine the features and target back for train and test datasets
train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)

# Save the train and test data to CSV files
train_data.to_csv('train_data.csv', index=False)
test_data.to_csv('test_data.csv', index=False)

print("Train and test datasets have been created and saved successfully.")
```

```
Train and test datasets have been created and saved successfully.
```

Checking Proportionality of Target variables across its categories.

In [32]:
```
# Check distribution in original data
print("Original data distribution:")
print(df['accident_severity'].value_counts(normalize=True))

# Check distribution in training set
print("\nTraining set distribution:")
print(train_data['accident_severity'].value_counts(normalize=True))

# Check distribution in test set
print("\nTest set distribution:")
print(test_data['accident_severity'].value_counts(normalize=True))

print("Proportionality of accident_severity is maintained across Slight, Serious and Fatal categories.")
```

```
Original data distribution:
accident_severity
Slight     0.768733
Serious    0.231267
Name: proportion, dtype: float64

Training set distribution:
accident_severity
Slight     0.768742
Serious    0.231258
Name: proportion, dtype: float64

Test set distribution:
accident_severity
Slight     0.768714
Serious    0.231286
Name: proportion, dtype: float64
Proportionality of accident_severity is maintained across Slight, Serious and Fatal categories.
```

6. Descriptive Statistics and Exploratory Data Analysis

In this section, we are exploring the data to understand and highlight key areas which requires attention and preprocessing before we start building predictive models. This would help us to decide what transformations have to be done to make data more suitable for predictive analytics.

We do not want to do any 'data snooping' as this would cause bias in model development. Therefore, all analysis in this section is done on the train_data.

Also we would be creating and using functions to achieve our tasks by calling a block of code to reuse it again.

6.1 Univariate Analysis

In this section, we investigate the variables of the dataframes one at a time in univariate analysis first. The analysis is separated into numeric variables and categorical variable (Includes Target Variable accident_severity).

A selection of these variables will be used in the predictive model so the descriptive statistics are an important way to improve our understanding of them.

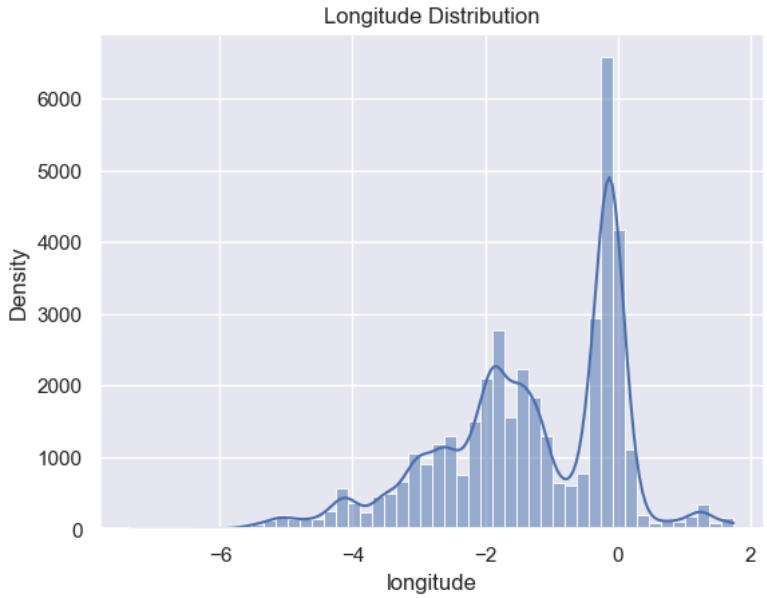
(a) Numerical variables

In [33]:
```
#For the variable - longitude
train_data['longitude'].describe()
```

Out[33]:
```
count    40271.000000
mean        -1.309141
std          1.331101
min         -7.353562
25%         -2.115102
50%         -1.235893
75%         -0.150048
max          1.751344
Name: longitude, dtype: float64
```

In [34]:
```
sns.histplot(train_data['longitude'], bins=50, kde=True)
# Add titles and labels
plt.title('Longitude Distribution')
plt.xlabel('longitude')
plt.ylabel('Density')
```

Out[34]:
```
Text(0, 0.5, 'Density')
```

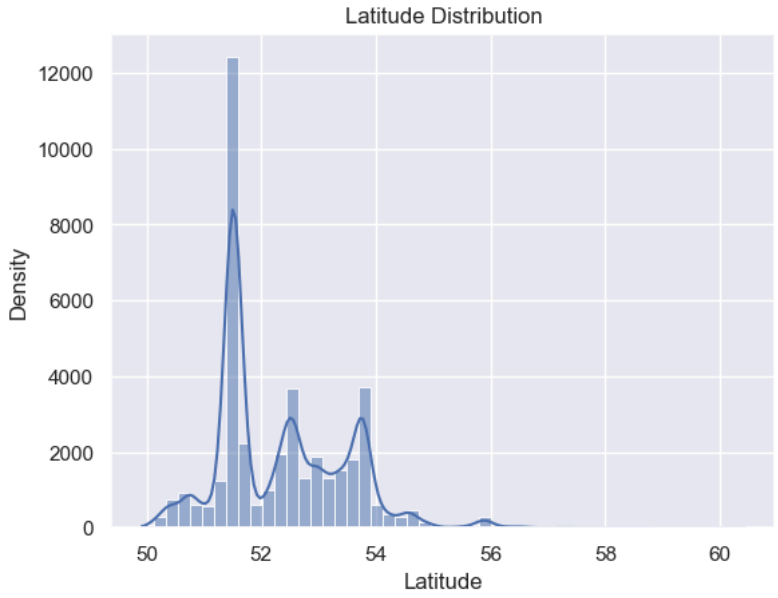


In [35]:
```
#For the variable - latitude
train_data['latitude'].describe()
```

Out[35]:
```
count    40271.000000
mean        52.328658
std          1.115957
min         49.914329
25%         51.501509
50%         52.126103
75%         53.188895
max         60.435892
Name: latitude, dtype: float64
```

In [36]:
```
sns.histplot(train_data['latitude'], bins=50, kde=True)
# Add titles and labels
plt.title('Latitude Distribution')
plt.xlabel('Latitude')
plt.ylabel('Density')
```

Out[36]:
```
Text(0, 0.5, 'Density')
```

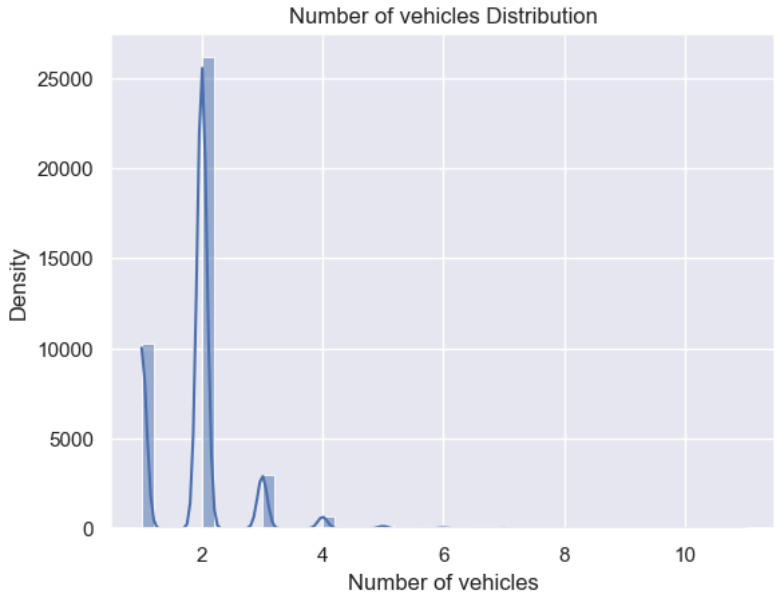


In [37]:
```
#For the variable - number_of_vehicles
train_data['number_of_vehicles'].describe()
```

Out[37]:
```
count    40271.000000
mean         1.873507
std          0.679613
min          1.000000
25%          1.000000
50%          2.000000
75%          2.000000
max         11.000000
Name: number_of_vehicles, dtype: float64
```

In [38]:
```
sns.histplot(train_data['number_of_vehicles'], bins=50, kde=True)
# Add titles and labels
plt.title('Number of vehicles Distribution')
plt.xlabel('Number of vehicles')
plt.ylabel('Density')
```

Out[38]:
```
Text(0, 0.5, 'Density')
```

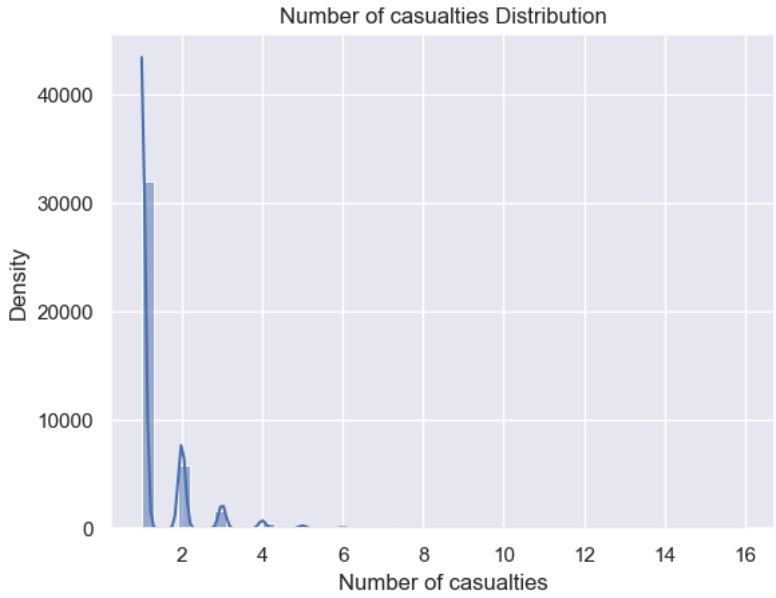


In [39]:
```
#For the variable - number_of_casualties
train_data['number_of_casualties'].describe()
```

Out[39]:
```
count    40271.000000
mean         1.305580
std          0.733557
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max         16.000000
Name: number_of_casualties, dtype: float64
```

In [40]:
```
sns.histplot(train_data['number_of_casualties'], bins=50, kde=True)
# Add titles and labels
plt.title('Number of casualties Distribution')
plt.xlabel('Number of casualties')
plt.ylabel('Density')
```

Out[40]:
```
Text(0, 0.5, 'Density')
```

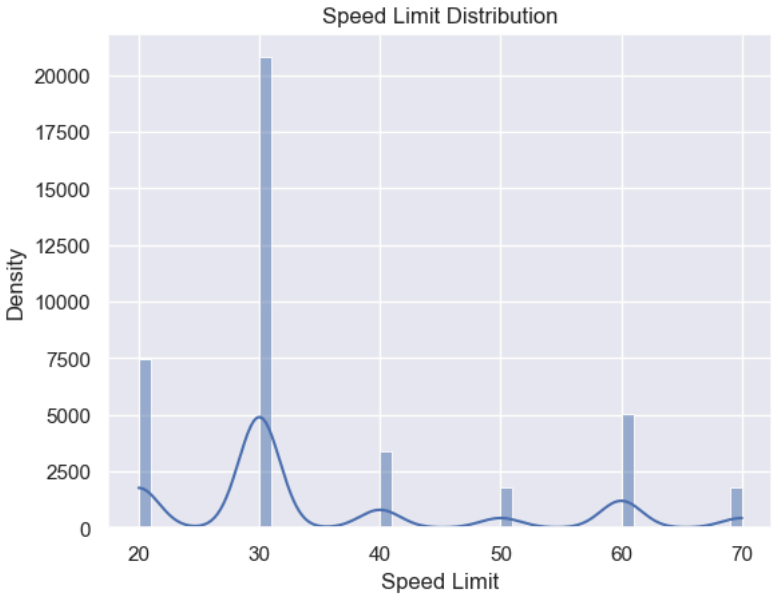


In [41]:
```
#For the variable - speed limit
train_data['speed_limit'].describe()
```

Out[41]:
```
count    40271.000000
mean        35.424996
std         14.136604
min         20.000000
25%         30.000000
50%         30.000000
75%         40.000000
max         70.000000
Name: speed_limit, dtype: float64
```

In [42]:
```
sns.histplot(train_data['speed_limit'], bins=50, kde=True)
# Add titles and labels
plt.title('Speed Limit Distribution')
plt.xlabel('Speed Limit')
plt.ylabel('Density')
```

Out[42]:
```
Text(0, 0.5, 'Density')
```



In [43]:
```
#For the variable - age_of_driver
train_data['age_of_driver'].describe()
```

Out[43]:
```
count    40271.000000
mean        41.089742
std         16.893333
min          5.000000
25%         27.500000
50%         38.000000
75%         52.000000
max         99.000000
Name: age_of_driver, dtype: float64
```

In [44]:
```
sns.histplot(train_data['age_of_driver'], bins=50, kde=True)
# Add titles and labels
plt.title('Driver Age Distribution')
plt.xlabel('Driver Age')
plt.ylabel('Density')
```

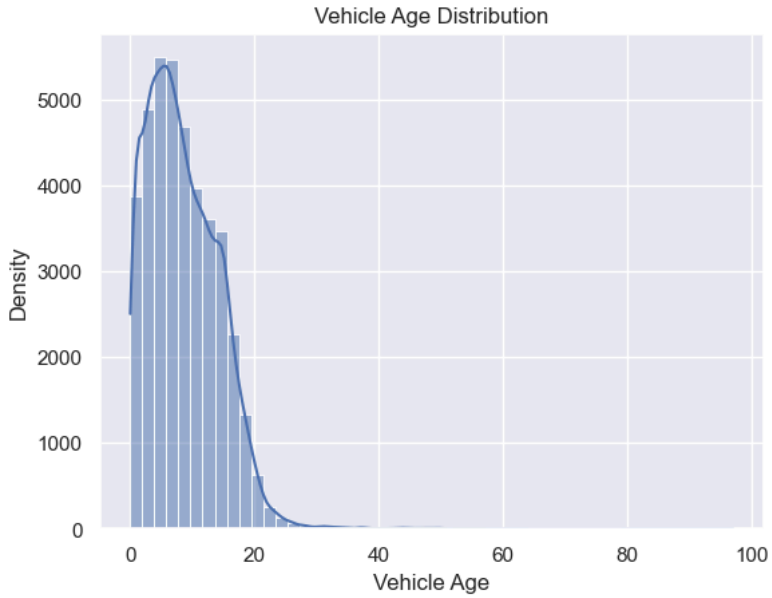
Out[44]:
```
Text(0, 0.5, 'Density')
```

In [45]:
```
#For the variable - age_of_vehicle
train_data['age_of_vehicle'].describe()
```

Out[45]:
```
count    40271.000000
mean         8.508381
std          5.765091
min          0.000000
25%          4.000000
50%          8.000000
75%         12.000000
max         97.000000
Name: age_of_vehicle, dtype: float64
```

In [46]:
```
sns.histplot(train_data['age_of_vehicle'], bins=50, kde=True)
# Add titles and labels
plt.title('Vehicle Age Distribution')
plt.xlabel('Vehicle Age')
plt.ylabel('Density')
```

Out[46]:
```
Text(0, 0.5, 'Density')
```

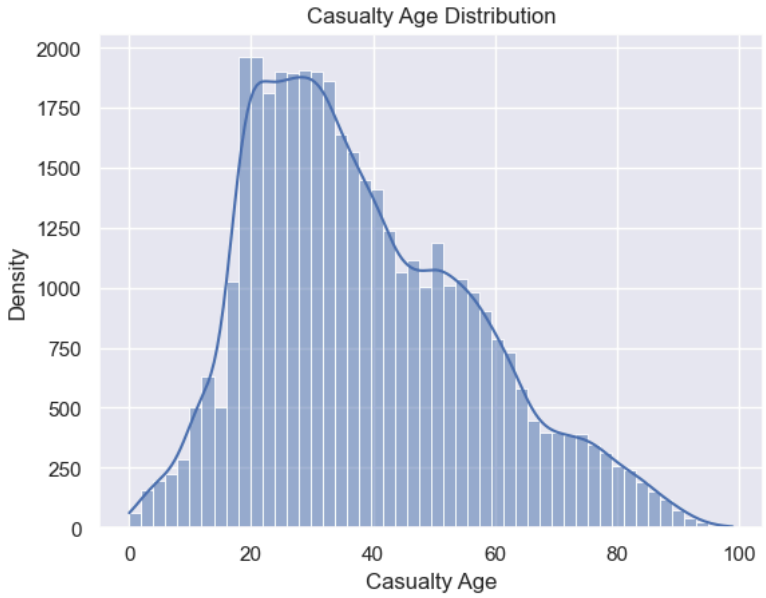


In [47]:
```
#For the variable - age_of_casualty
train_data['age_of_casualty'].describe()
```

Out[47]:
```
count    40271.000000
mean        38.539247
std         18.480451
min          0.000000
25%         24.000000
50%         35.000000
75%         51.000000
max         99.000000
Name: age_of_casualty, dtype: float64
```

In [48]:
```
sns.histplot(train_data['age_of_casualty'], bins=50, kde=True)
# Add titles and labels
plt.title('Casualty Age Distribution')
plt.xlabel('Casualty Age')
plt.ylabel('Density')
```

Out[48]:
```
Text(0, 0.5, 'Density')
```



(b) Categorical variables

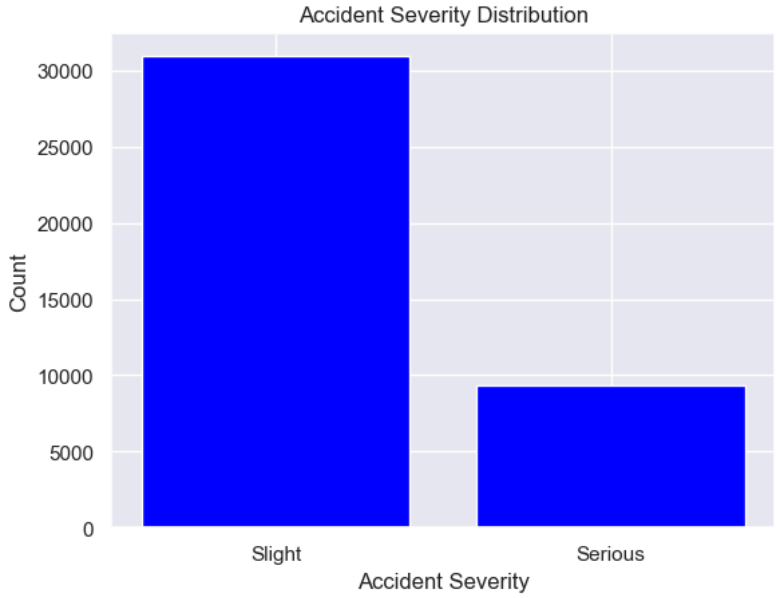
Now for Categorical Variables, we are starting with target variable (Dependent variable) that is accident_severity, followed by other independent categorical variables.

Bar charts are given for each of the categorical variables to illustrate their value counts, along with value count table and descriptive statistics.

In [49]:
```
#For the variable - accident_severity
train_data['accident_severity'].describe()
```

Out[49]:
```
count      40271
unique         2
top       Slight
freq       30958
Name: accident_severity, dtype: object
```

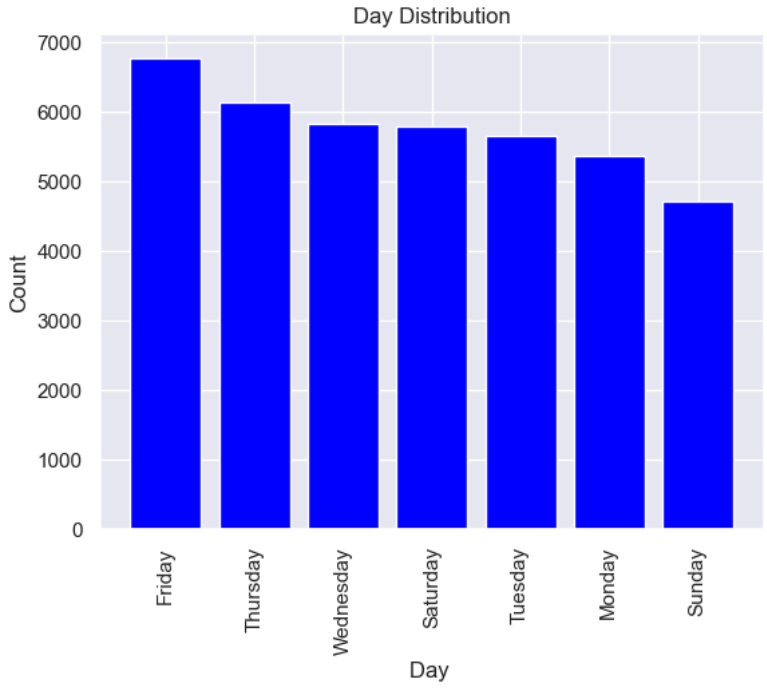
In [50]:
```
count = train_data['accident_severity'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Accident Severity Distribution')
plt.xlabel('Accident Severity')
plt.ylabel('Count')

# Show plot
plt.show()
```



In [51]:
```
#For the variable - day_of_week
train_data['day_of_week'].describe()
```

Out[51]:
```
count      40271
unique         7
top       Friday
freq        6778
Name: day_of_week, dtype: object
```

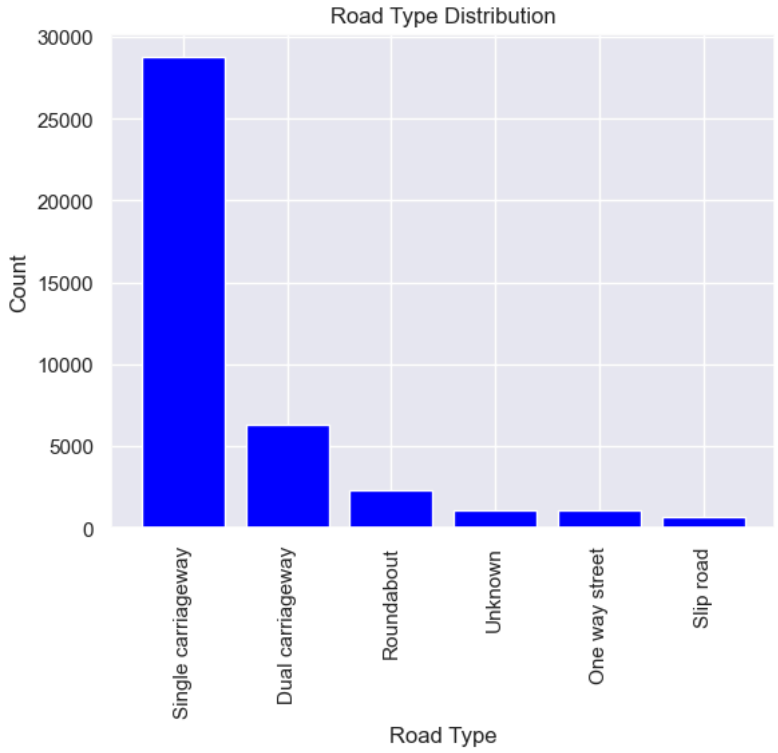
In [52]:
```
count = train_data['day_of_week'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Day Distribution')
plt.xlabel('Day ')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [53]:
```
#For the variable - road_type
train_data['road_type'].describe()
```

Out[53]:
```
count                  40271
unique                     6
top       Single carriageway
freq                   28764
Name: road_type, dtype: object
```

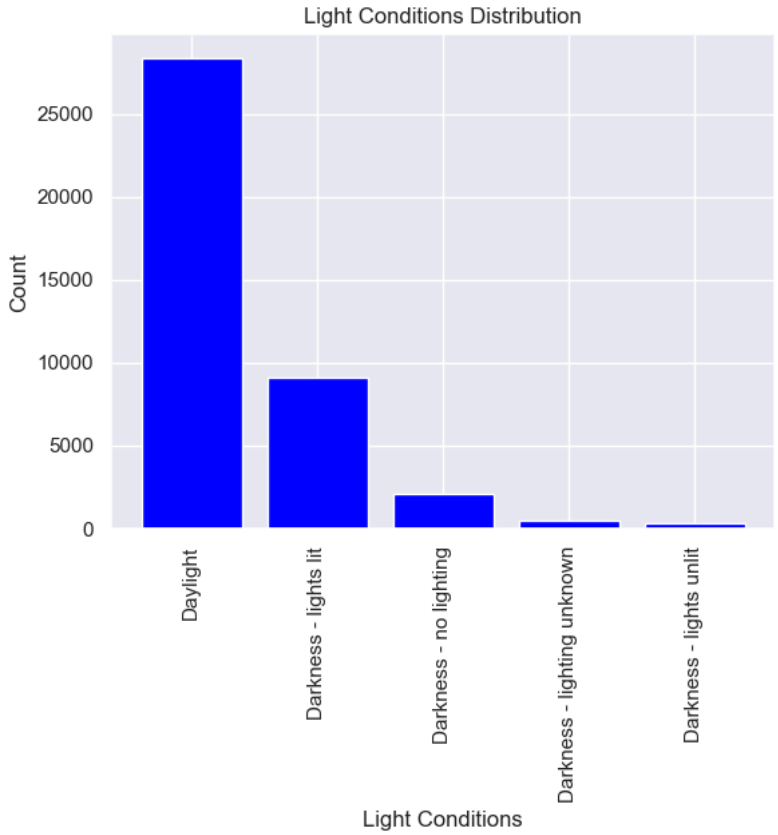
In [54]:
```
count = train_data['road_type'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Road Type Distribution')
plt.xlabel('Road Type')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [55]:
```
#For the variable - light_conditions
train_data['light_conditions'].describe()
```

Out[55]:
```
count        40271
unique           5
top       Daylight
freq         28365
Name: light_conditions, dtype: object
```

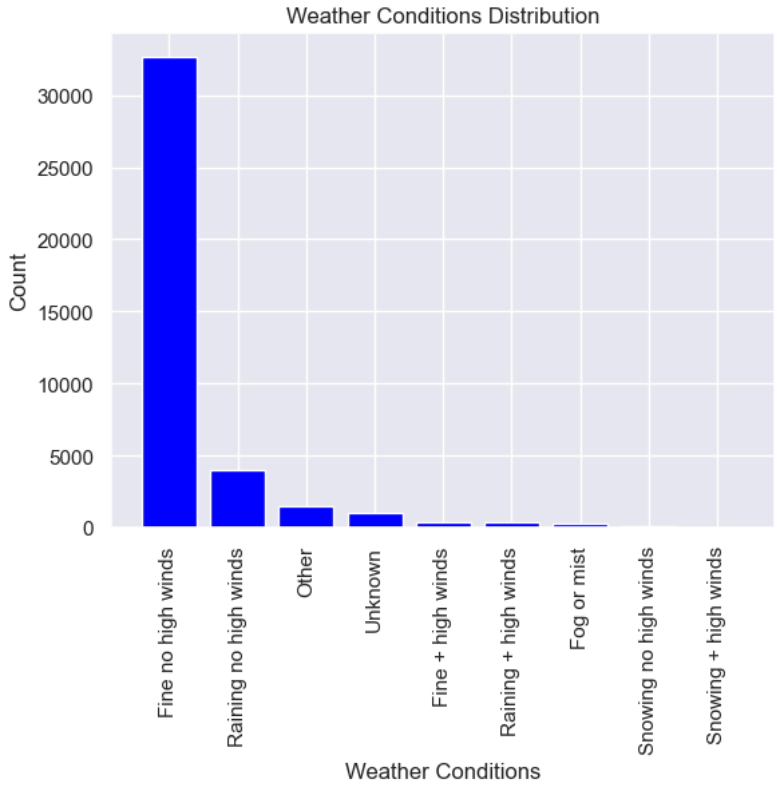
In [56]:
```
count = train_data['light_conditions'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Light Conditions Distribution')
plt.xlabel('Light Conditions')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [57]:
```
#For the variable - weather_conditions
train_data['weather_conditions'].describe()
```

Out[57]:
```
count                  40271
unique                     9
top       Fine no high winds
freq                   32656
Name: weather_conditions, dtype: object
```

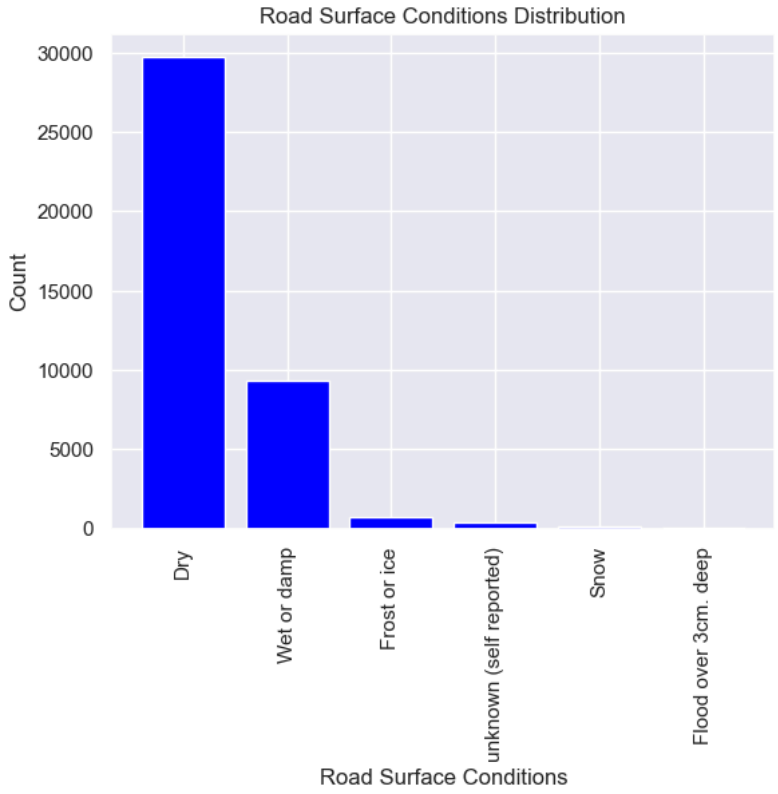
In [58]:
```
count = train_data['weather_conditions'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Weather Conditions Distribution')
plt.xlabel('Weather Conditions')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [59]:
```
#For the variable - road_surface_conditions
train_data['road_surface_conditions'].describe()
```

Out[59]:
```
count     40271
unique        6
top         Dry
freq      29704
Name: road_surface_conditions, dtype: object
```

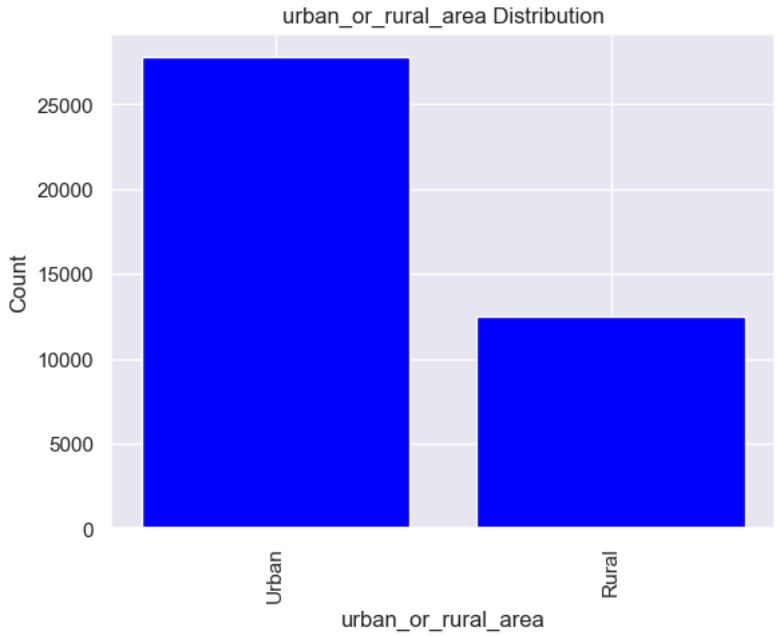
In [60]:
```
count = train_data['road_surface_conditions'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Road Surface Conditions Distribution')
plt.xlabel('Road Surface Conditions')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [61]:
```
#For the variable - urban_or_rural_area
train_data['urban_or_rural_area'].describe()
```

Out[61]:
```
count     40271
unique        2
top       Urban
freq      27777
Name: urban_or_rural_area, dtype: object
```

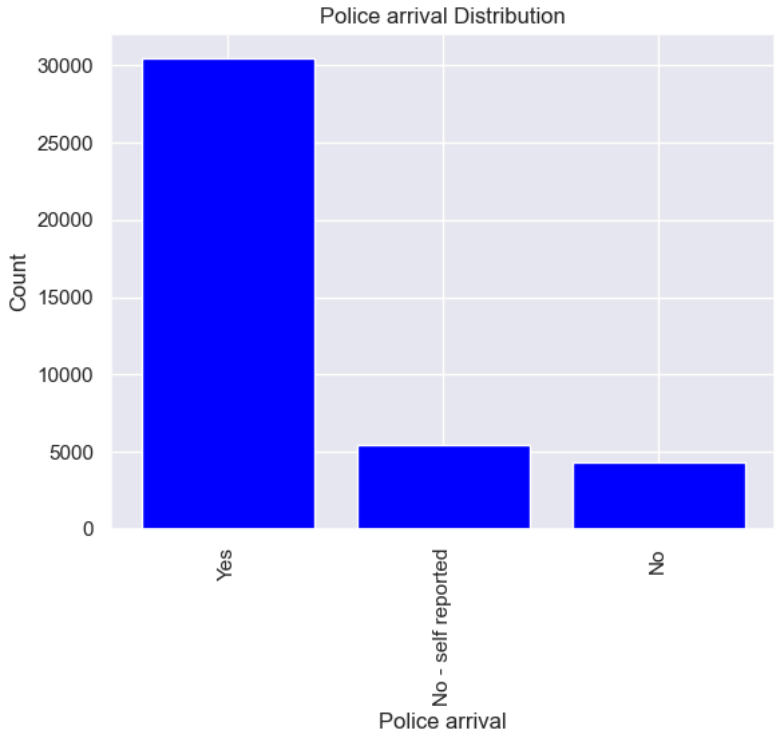
In [62]:
```
count = train_data['urban_or_rural_area'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('urban_or_rural_area Distribution')
plt.xlabel('urban_or_rural_area')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [63]:
```
#For the variable - did_police_officer_attend_scene_of_accident
train_data['did_police_officer_attend_scene_of_accident'].describe()
```

Out[63]:
```
count     40271
unique        3
top         Yes
freq      30471
Name: did_police_officer_attend_scene_of_accident, dtype: object
```

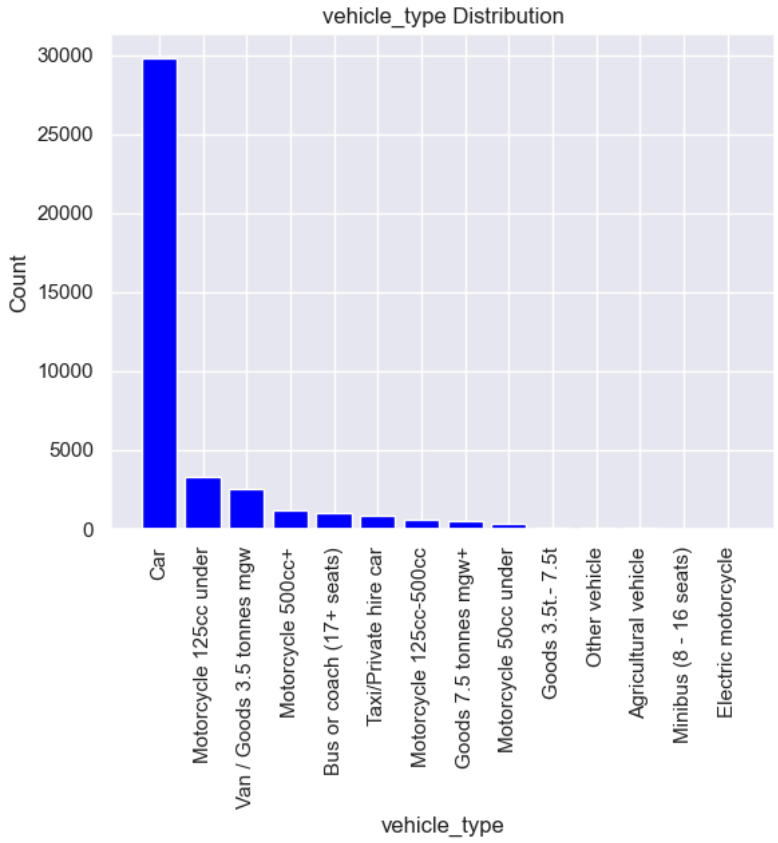
In [64]:
```
count = train_data['did_police_officer_attend_scene_of_accident'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('Police arrival Distribution')
plt.xlabel('Police arrival')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [65]:
```
#For the variable - vehicle_type
train_data['vehicle_type'].describe()
```

Out[65]:
```
count     40271
unique       14
top         Car
freq      29823
Name: vehicle_type, dtype: object
```

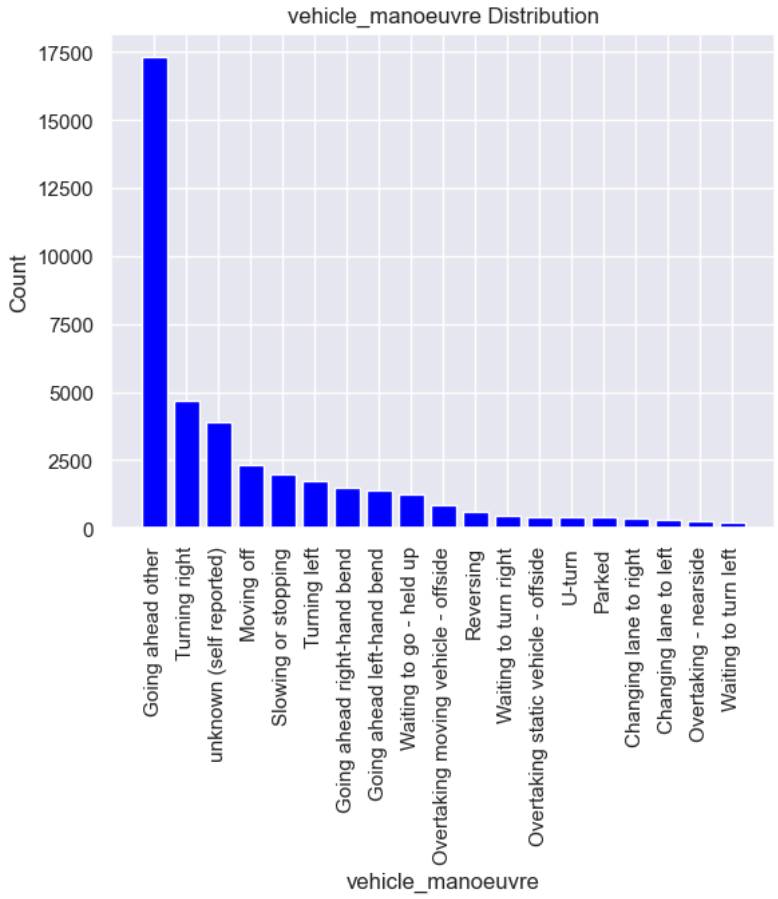
In [66]:
```
count = train_data['vehicle_type'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('vehicle_type Distribution')
plt.xlabel('vehicle_type')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [67]:
```
#For the variable - vehicle_manoeuvre
train_data['vehicle_manoeuvre'].describe()
```

Out[67]:
```
count                 40271
unique                   19
top       Going ahead other
freq                  17311
Name: vehicle_manoeuvre, dtype: object
```

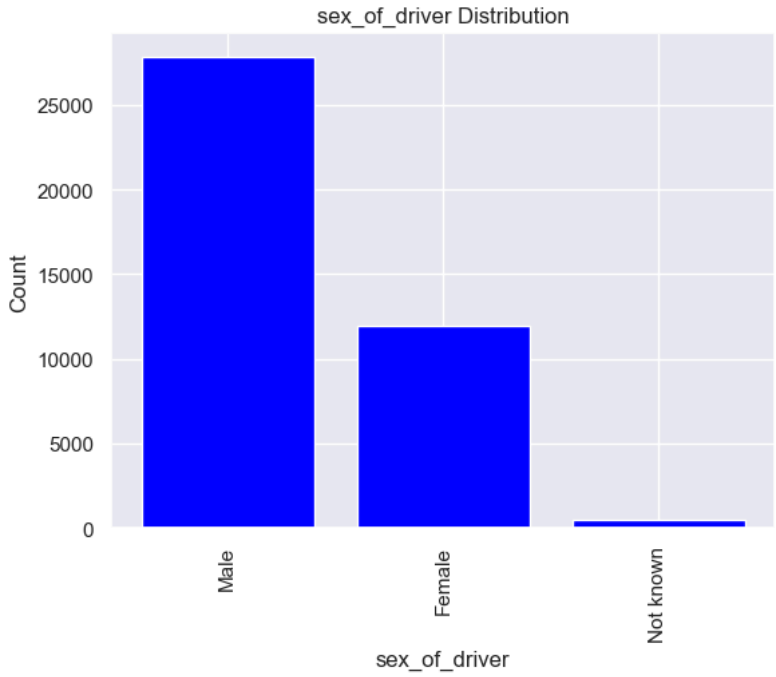
In [68]:
```
count = train_data['vehicle_manoeuvre'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('vehicle_manoeuvre Distribution')
plt.xlabel('vehicle_manoeuvre')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [69]:
```
#For the variable - sex_of_driver
train_data['sex_of_driver'].describe()
```

Out[69]:
```
count     40271
unique        3
top        Male
freq      27847
Name: sex_of_driver, dtype: object
```

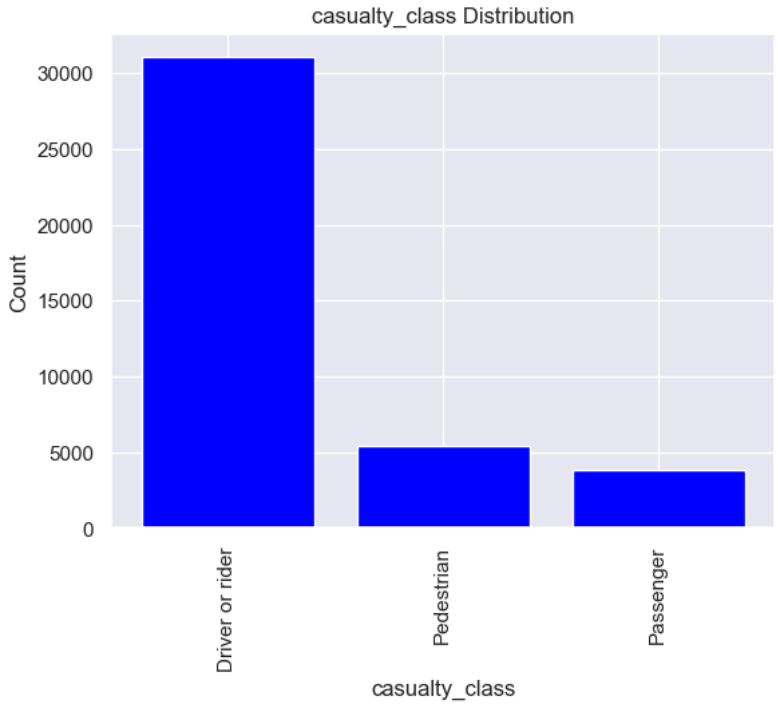
In [70]:
```
count = train_data['sex_of_driver'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('sex_of_driver Distribution')
plt.xlabel('sex_of_driver')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [71]:
```
#For the variable - casualty_class
train_data['casualty_class'].describe()
```

Out[71]:
```
count               40271
unique                  3
top       Driver or rider
freq                31038
Name: casualty_class, dtype: object
```

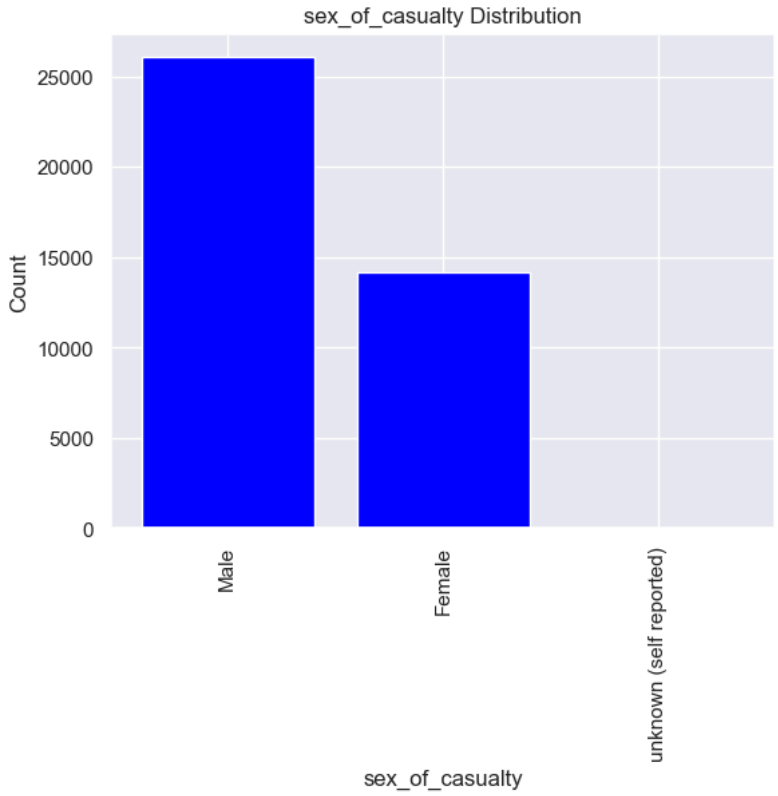
In [72]:
```
count = train_data['casualty_class'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('casualty_class Distribution')
plt.xlabel('casualty_class')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [73]:
```
#For the variable - sex_of_casualty
train_data['sex_of_casualty'].describe()
```

Out[73]:
```
count     40271
unique        3
top        Male
freq      26072
Name: sex_of_casualty, dtype: object
```

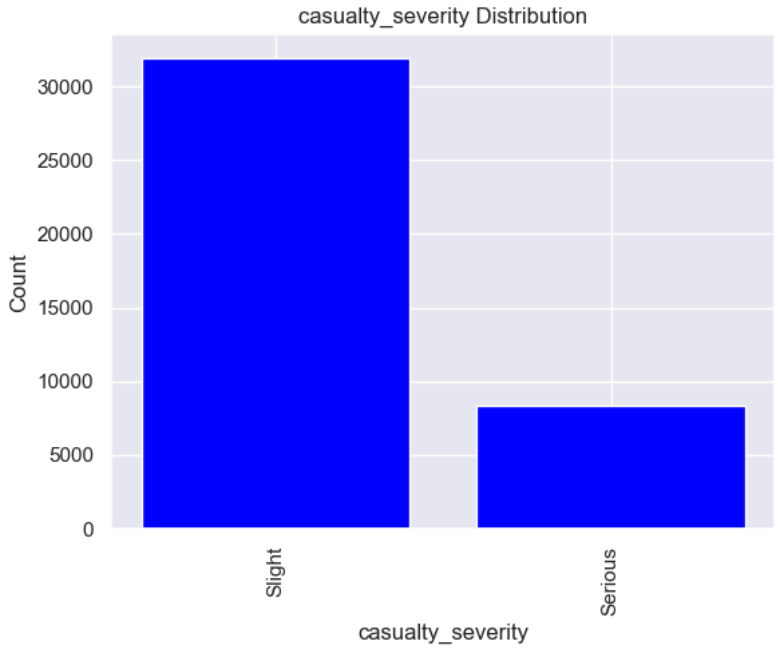
In [74]:
```
count = train_data['sex_of_casualty'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('sex_of_casualty Distribution')
plt.xlabel('sex_of_casualty')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [75]:
```
#For the variable - casualty_severity
train_data['casualty_severity'].describe()
```

Out[75]:
```
count      40271
unique         2
top       Slight
freq       31923
Name: casualty_severity, dtype: object
```

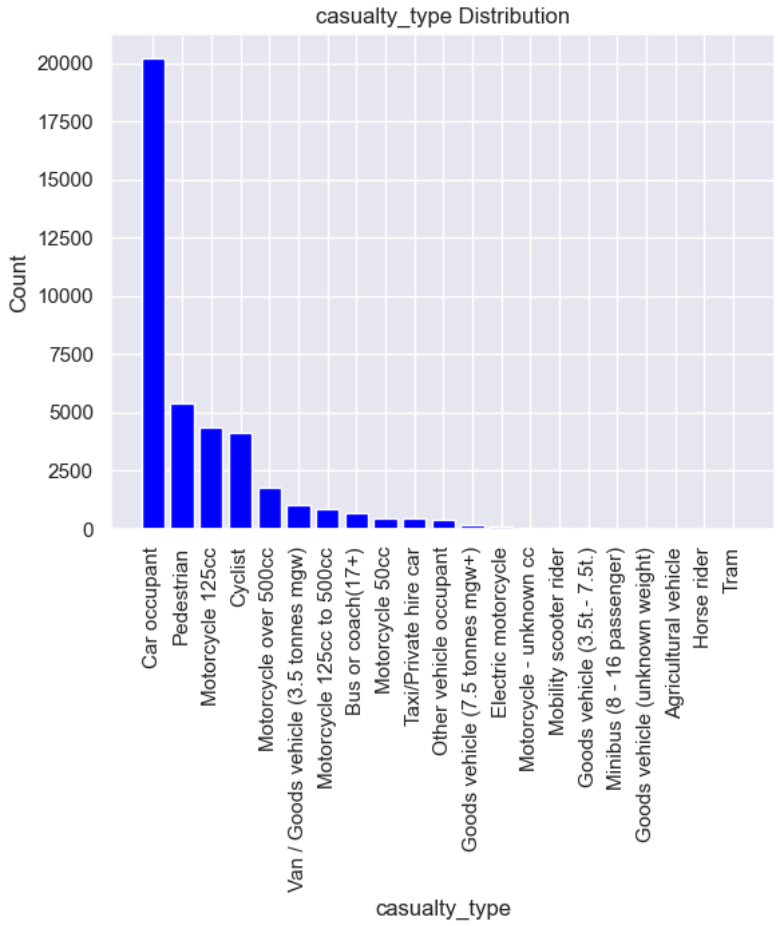
In [76]:
```
count = train_data['casualty_severity'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('casualty_severity Distribution')
plt.xlabel('casualty_severity')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



In [77]:
```
#For the variable - casualty_type
train_data['casualty_type'].describe()
```

Out[77]:
```
count            40271
unique              21
top       Car occupant
freq             20204
Name: casualty_type, dtype: object
```

In [78]:
```
count = train_data['casualty_type'].value_counts()
# Create bar graph
plt.bar(count.index, count.values, color='blue')

# Add titles and labels
plt.title('casualty_type Distribution')
plt.xlabel('casualty_type')
plt.ylabel('Count')
plt.tick_params(axis='x', rotation=90)

# Show plot
plt.show()
```



From this visualizations, we can get the general idea of the overall distribution of the data.

6.2 Bivariate Analysis

We now visualize their relationships with each other using appropriate graphs for categorical and numerical variables. We will mostly see how they are associated with our target variable to understand their usability in our Models.

After understanding the Numerical corelations, we'll visualize the impact of all variables on accident_severity along with underlying associations and relationships.

First we will start with correlation matrix for numerical variables

In [79]:
```
# Matrix plots for numerical variables against each other
#Computes correlation coefficients (e.g., Pearson or Spearman) to identify relationships between numeric variables.
corr_matrix = train_data.corr(method="pearson", numeric_only=True)
corr_matrix
```

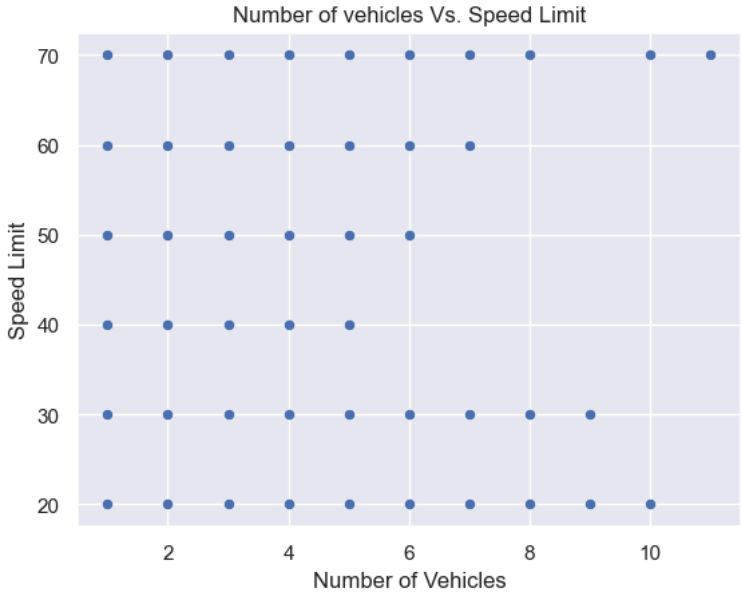
Out[79]:

	longitude	latitude	number_of_vehicles	number_of_casualties	speed_limit	age_of_driver	age_of_vehicle	age_of_casualty
longitude	1.000000	-0.134266	0.034904	-0.068319	-0.248305	-0.051046	-0.116690	-0.048461
latitude	-0.134266	1.000000	-0.011319	0.063810	0.191405	0.027196	0.040553	0.020456
number_of_vehicles	0.034904	-0.011319	1.000000	0.202761	0.055015	-0.012216	-0.032066	0.044814
number_of_casualties	-0.068319	0.063810	0.202761	1.000000	0.173779	-0.018803	0.033733	0.006790
speed_limit	-0.248305	0.191405	0.055015	0.173779	1.000000	0.015805	0.100874	0.074380
age_of_driver	-0.051046	0.027196	-0.012216	-0.018803	0.015805	1.000000	0.018175	0.507868
age_of_vehicle	-0.116690	0.040553	-0.032066	0.033733	0.100874	0.018175	1.000000	0.016443
age_of_casualty	-0.048461	0.020456	0.044814	0.006790	0.074380	0.507868	0.016443	1.000000

Here we will only see correlation between 2 pair of variables further. Those variable pairs are number_of_vehicles-speed_limit and number_of_casualties-age_of_driver.

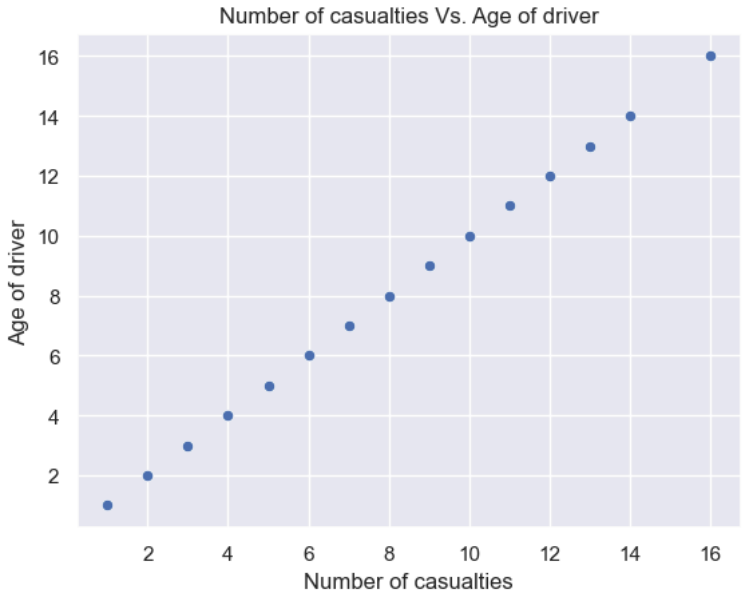
(i) Correlation between number_of_vehicles and speed_limit using scatterplot

In [80]:
```
# Bi-variate visualization of Number of vehicles Vs. Speed Limit
sns.scatterplot(x=train_data['number_of_vehicles'], y=train_data['speed_limit'])

# Add titles and labels
plt.title('Number of vehicles Vs. Speed Limit')
plt.xlabel('Number of Vehicles')
plt.ylabel('Speed Limit')

# Show plot
plt.show()
```



(ii) Correlation between number_of_casualties and age_of_driver using scatterplot

In [81]:
```
# Bi-variate visualization of car price vs Engine_Size
sns.scatterplot(x=train_data['number_of_casualties'], y=train_data['number_of_casualties'])

# Add titles and labels
plt.title('Number of casualties Vs. Age of driver')
plt.xlabel('Number of casualties')
plt.ylabel('Age of driver')

# Show plot
plt.show()
```

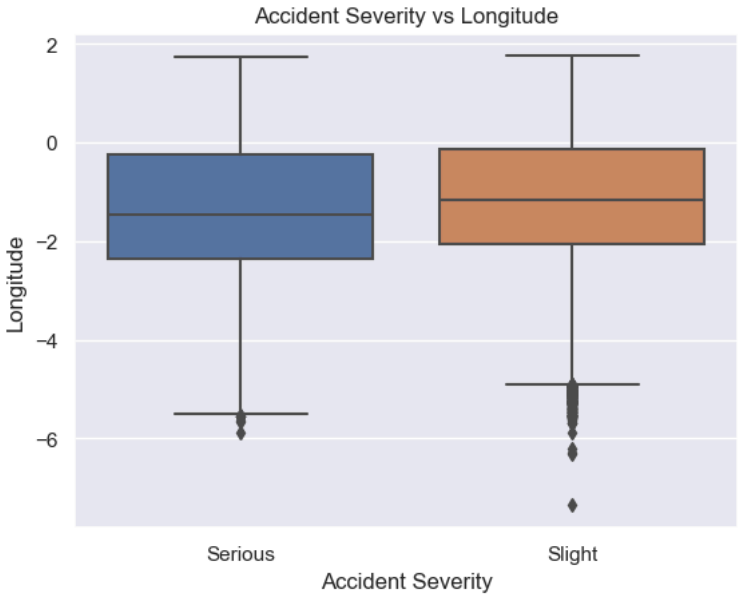


(iii) Bivariate analysis for numerical variables

In [82]:
```
# Group by the accident_severity and describe the longitude
grouped_stats = train_data.groupby('accident_severity')['longitude'].describe()
print(grouped_stats)
```

```
                     count      mean       std       min       25%       50%  \
accident_severity
Serious             9313.0 -1.491207  1.347870 -5.880985 -2.350975 -1.467822
Slight             30958.0 -1.254371  1.321136 -7.353562 -2.045892 -1.154747

                         75%       max
accident_severity
Serious            -0.232400  1.746887
Slight             -0.134932  1.751344
```

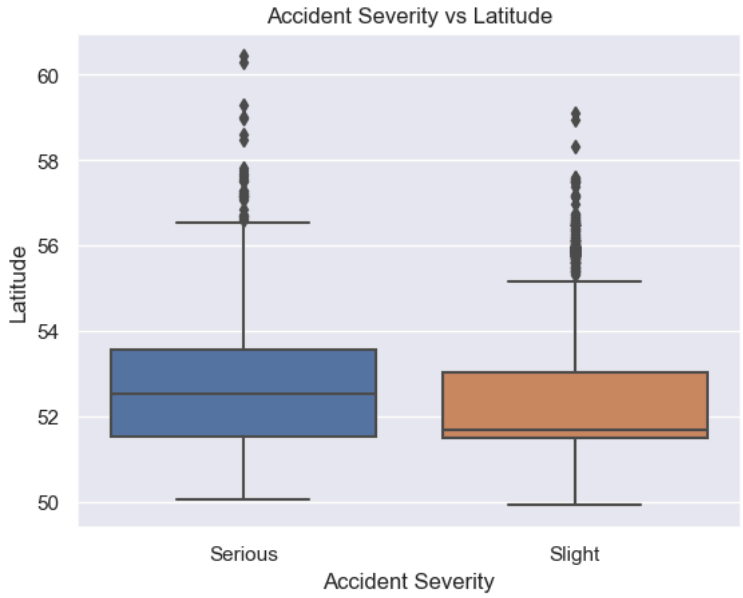
In [83]:
```
# Bi-variate visualization of Accident Severity vs Longitude
sns.boxplot(x=train_data['accident_severity'], y=df['longitude'])

# Add titles and labels
plt.title('Accident Severity vs Longitude')
plt.xlabel('Accident Severity')
plt.ylabel('Longitude')

# Show plot
plt.show()
```



In [84]:
```
# Group by the accident_severity and describe the latitude
grouped_stats = train_data.groupby('accident_severity')['latitude'].describe()
print(grouped_stats)
```

```
                     count       mean       std        min        25%  \
accident_severity
Serious             9313.0  52.574943  1.208662  50.035665  51.532388
Slight             30958.0  52.254569  1.075565  49.914329  51.494849

                         50%        75%        max
accident_severity
Serious            52.513676  53.556668  60.435892
Slight             51.691222  53.011148  59.101597
```

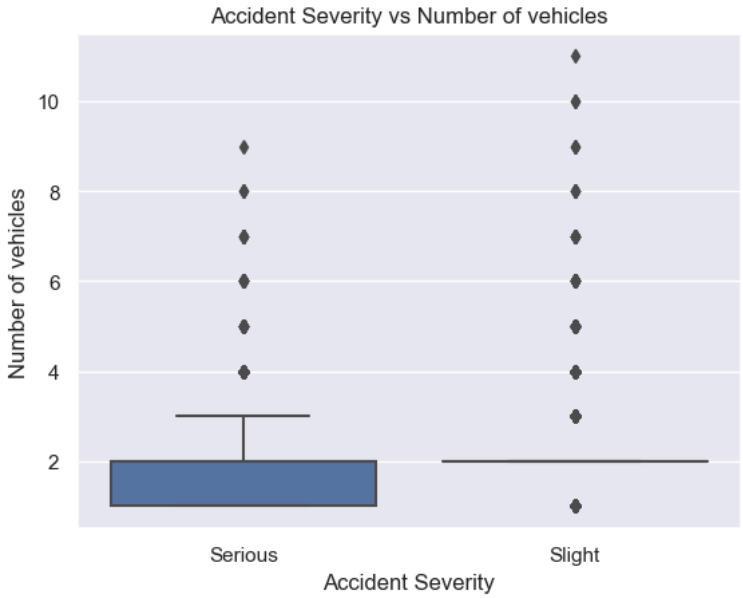
In [85]:
```
# Bi-variate visualization of Accident Severity vs Latitude
sns.boxplot(x=train_data['accident_severity'], y=df['latitude'])

# Add titles and labels
plt.title('Accident Severity vs Latitude')
plt.xlabel('Accident Severity')
plt.ylabel('Latitude')

# Show plot
plt.show()
```



In [86]:
```
# Group by the accident_severity and describe the number of vehicles
grouped_stats = train_data.groupby('accident_severity')['number_of_vehicles'].describe()
print(grouped_stats)
```

```
                     count      mean       std  min  25%  50%  75%   max
accident_severity
Serious             9313.0  1.789112  0.762342  1.0  1.0  2.0  2.0   9.0
Slight             30958.0  1.898895  0.650552  1.0  2.0  2.0  2.0  11.0
```

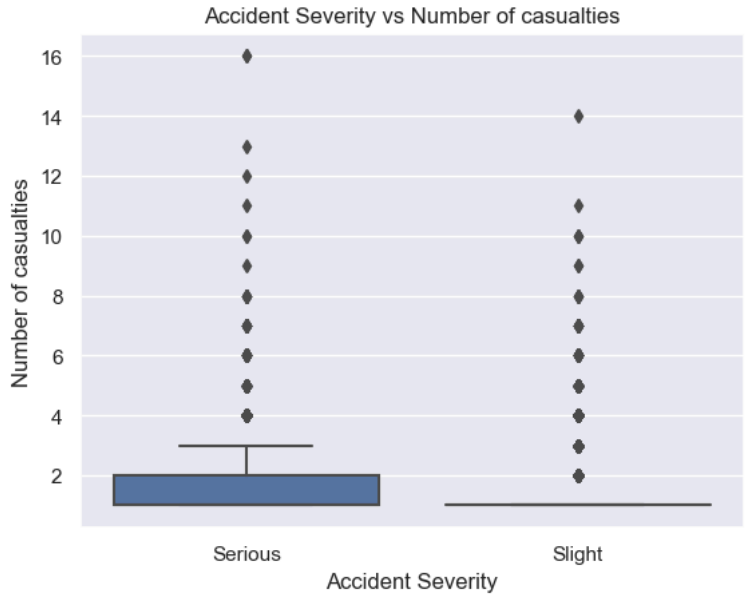
In [87]:
```
# Bi-variate visualization of Accident Severity vs number of vehicles
sns.boxplot(x=train_data['accident_severity'], y=df['number_of_vehicles'])

# Add titles and labels
plt.title('Accident Severity vs Number of vehicles')
plt.xlabel('Accident Severity')
plt.ylabel('Number of vehicles')

# Show plot
plt.show()
```



In [88]:
```
# Group by the accident_severity and describe the number of casualties
grouped_stats = train_data.groupby('accident_severity')['number_of_casualties'].describe()
print(grouped_stats)
```

```
                    count      mean       std  min  25%  50%  75%   max
accident_severity
Serious            9313.0  1.438742  0.934800  1.0  1.0  1.0  2.0  16.0
Slight            30958.0  1.265521  0.655889  1.0  1.0  1.0  1.0  14.0
```

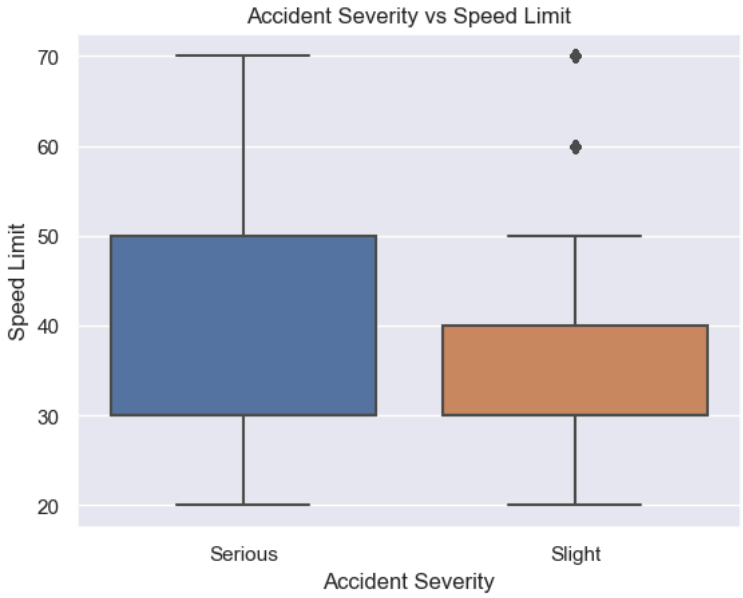
In [89]:
```
# Bi-variate visualization of Accident Severity vs number of casualties
sns.boxplot(x=train_data['accident_severity'], y=df['number_of_casualties'])

# Add titles and labels
plt.title('Accident Severity vs Number of casualties')
plt.xlabel('Accident Severity')
plt.ylabel('Number of casualties')

# Show plot
plt.show()
```



In [90]:
```
# Group by the accident_severity and describe the speed limit
grouped_stats = train_data.groupby('accident_severity')['speed_limit'].describe()
print(grouped_stats)
```

```
                    count       mean        std   min   25%   50%   75%   max
accident_severity
Serious            9313.0  38.108021  14.892453  20.0  30.0  30.0  50.0  70.0
Slight            30958.0  34.617869  13.799741  20.0  30.0  30.0  40.0  70.0
```

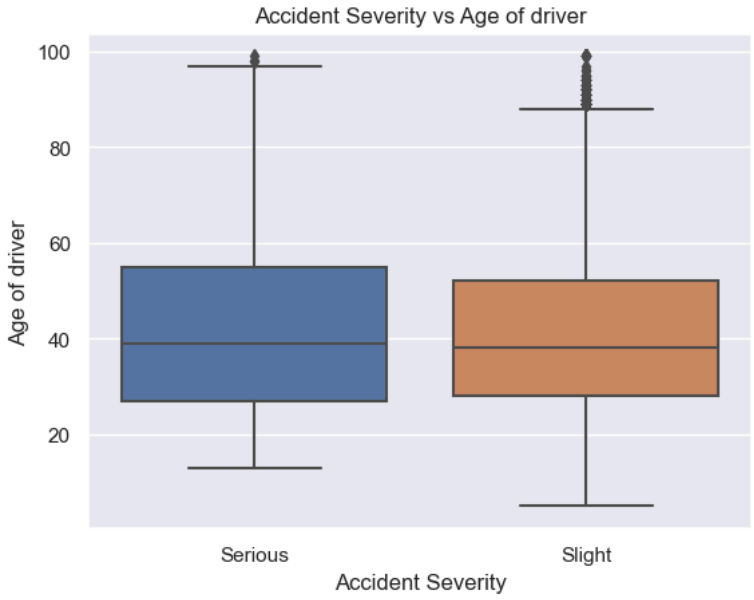
In [91]:
```
# Bi-variate visualization of Accident Severity vs speed limit
sns.boxplot(x=train_data['accident_severity'], y=df['speed_limit'])

# Add titles and labels
plt.title('Accident Severity vs Speed Limit')
plt.xlabel('Accident Severity')
plt.ylabel('Speed Limit')

# Show plot
plt.show()
```



In [92]:
```
# Group by the accident_severity and describe the age of driver
grouped_stats = train_data.groupby('accident_severity')['age_of_driver'].describe()
print(grouped_stats)
```

```
                     count       mean        std   min   25%   50%   75%   max
accident_severity
Serious             9313.0  42.247074  17.873201  13.0  27.0  39.0  55.0  99.0
Slight             30958.0  40.741585  16.571734   5.0  28.0  38.0  52.0  99.0
```

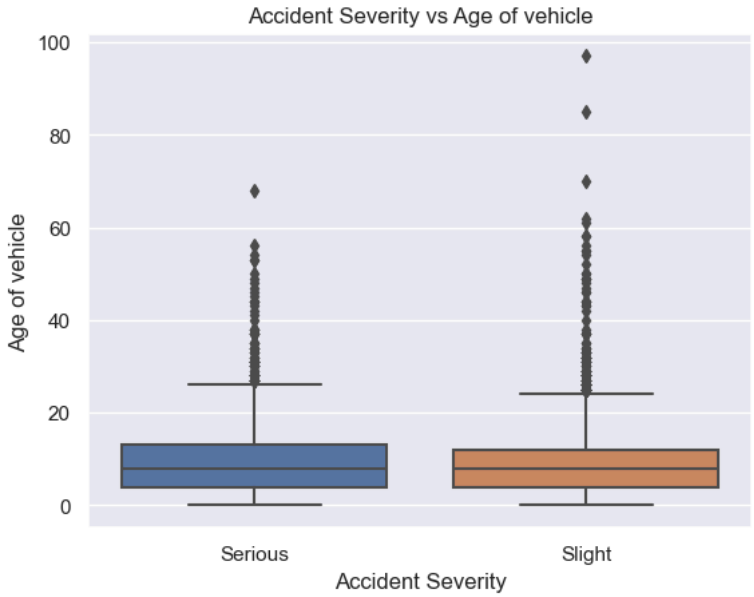
In [93]:
```
# Bi-variate visualization of Accident Severity vs age of driver
sns.boxplot(x=train_data['accident_severity'], y=df['age_of_driver'])

# Add titles and labels
plt.title('Accident Severity vs Age of driver')
plt.xlabel('Accident Severity')
plt.ylabel('Age of driver')

# Show plot
plt.show()
```



In [94]:
```
# Group by the accident_severity and describe the age of vehicle
grouped_stats = train_data.groupby('accident_severity')['age_of_vehicle'].describe()
print(grouped_stats)
```

```
                    count      mean       std  min  25%  50%   75%   max
accident_severity
Serious             9313.0  9.008590  6.174956  0.0  4.0  8.0  13.0  68.0
Slight             30958.0  8.357904  5.627376  0.0  4.0  8.0  12.0  97.0
```

In [95]:
```
# Bi-variate visualization of Accident Severity vs age of vehicle
sns.boxplot(x=train_data['accident_severity'], y=df['age_of_vehicle'])

# Add titles and labels
plt.title('Accident Severity vs Age of vehicle')
plt.xlabel('Accident Severity')
plt.ylabel('Age of vehicle')

# Show plot
plt.show()
```



In [96]:
```
# Group by the accident_severity and describe the age of casualty
grouped_stats = train_data.groupby('accident_severity')['age_of_casualty'].describe()
print(grouped_stats)
```

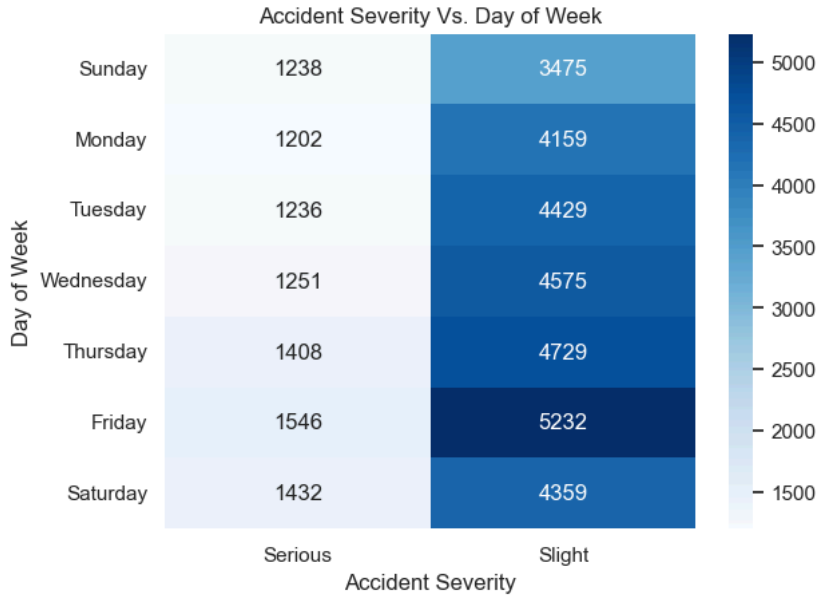
```
                    count       mean        std  min   25%   50%   75%   max
accident_severity
Serious             9313.0  40.668528  20.432501  0.0  24.0  37.0  56.0  98.0
Slight             30958.0  37.898701  17.802100  0.0  24.0  35.0  50.0  99.0
```

In [97]:
```
# Bi-variate visualization of Accident Severity vs age of casualty
sns.boxplot(x=train_data['accident_severity'], y=df['age_of_casualty'])

# Add titles and labels
plt.title('Accident Severity vs Age of casualty')
plt.xlabel('Accident Severity')
plt.ylabel('Age of casualty')

# Show plot
plt.show()
```

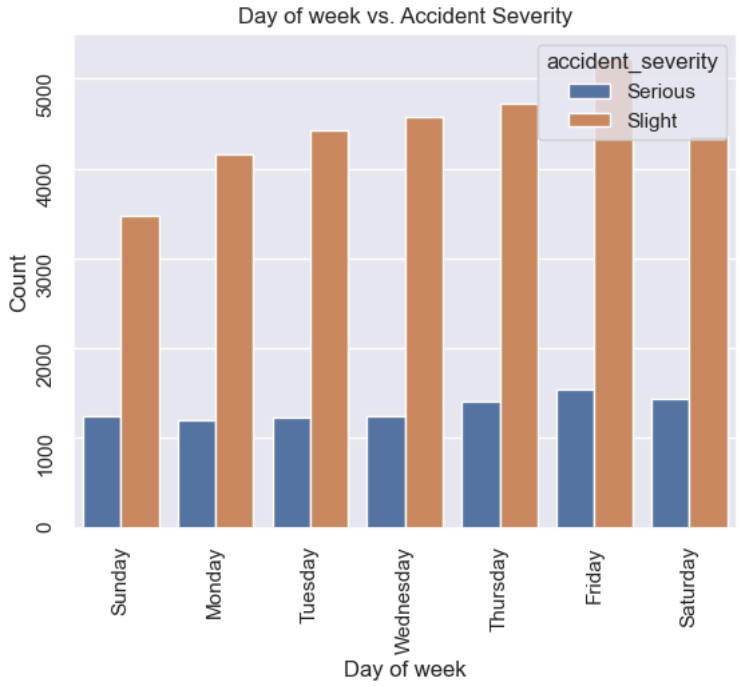
(iv) Bivariate analysis for categorical variables

In [98]:
```
# Create a contingency table - Day of week
contingency_table = pd.crosstab(train_data['day_of_week'], train_data['accident_severity'])
```

In [99]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Day of Week')
plt.xlabel('Accident Severity')
plt.ylabel('Day of Week')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

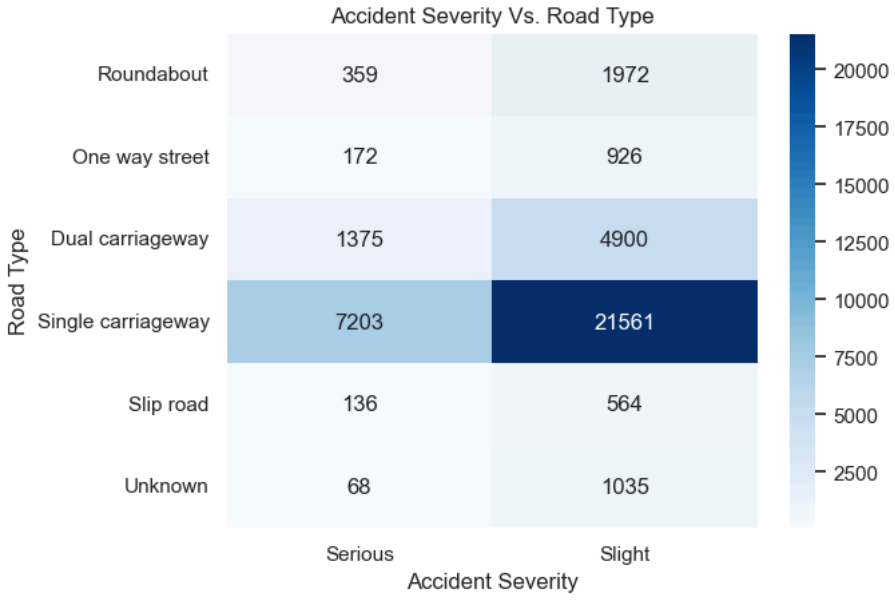


In [100]:
```
# Count plot
sns.countplot(data=train_data, x='day_of_week', hue='accident_severity')
plt.title('Day of week vs. Accident Severity')
plt.xlabel('Day of week')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

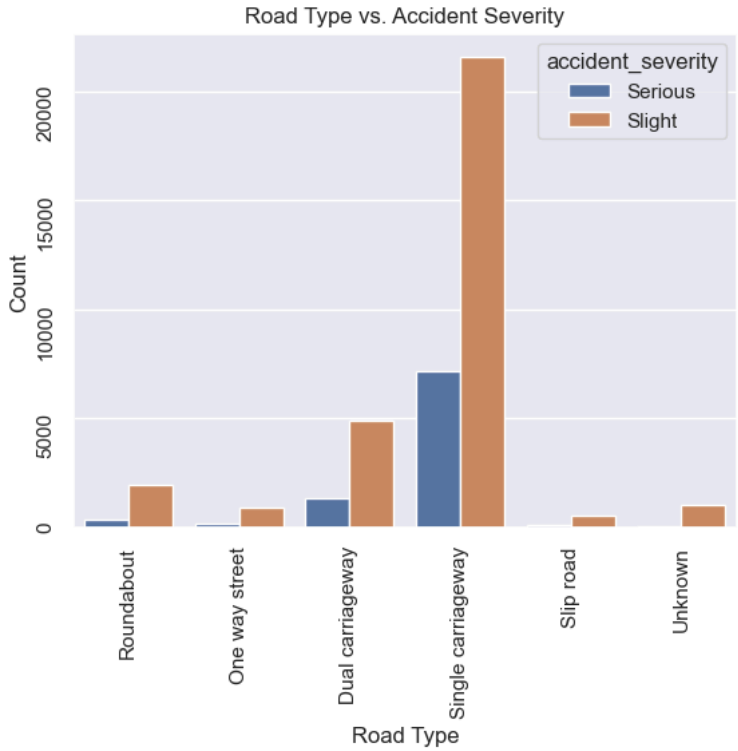


In [101]:
```
# Create a contingency table - Road Type
contingency_table = pd.crosstab(train_data['road_type'], train_data['accident_severity'])
```

In [102]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Road Type')
plt.xlabel('Accident Severity')
plt.ylabel('Road Type')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

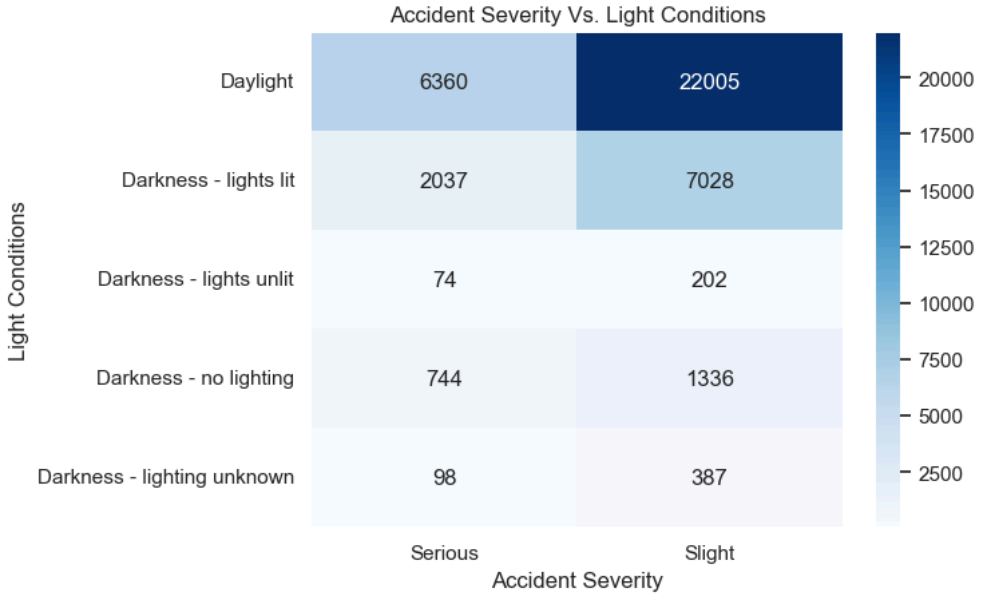


In [103]:
```
# Count plot
sns.countplot(data=train_data, x='road_type', hue='accident_severity')
plt.title('Road Type vs. Accident Severity')
plt.xlabel('Road Type')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

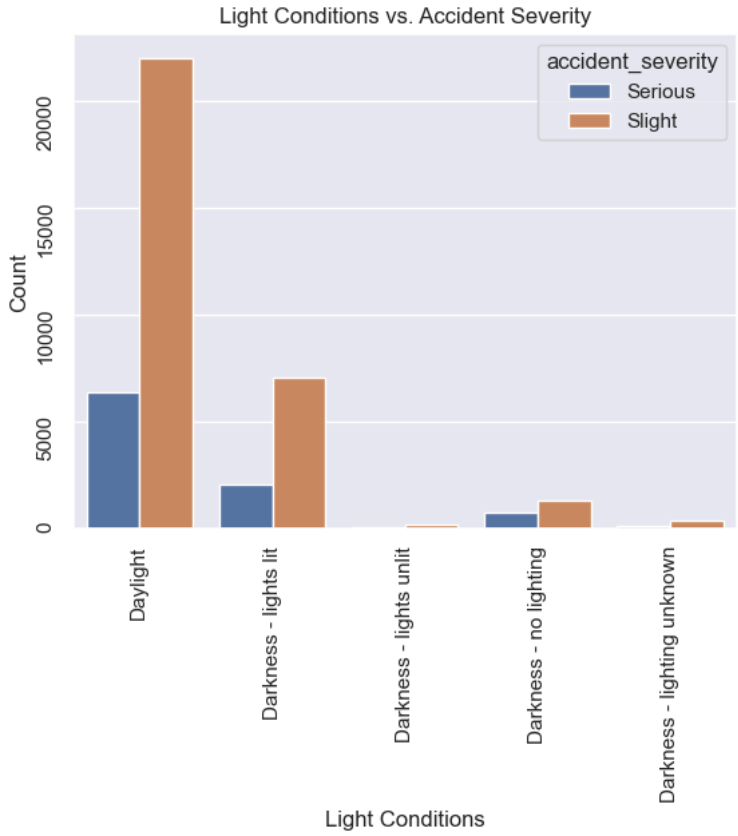


In [104]:
```
# Create a contingency table - Light Conditions
contingency_table = pd.crosstab(train_data['light_conditions'], train_data['accident_severity'])
```

In [105]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Light Conditions')
plt.xlabel('Accident Severity')
plt.ylabel('Light Conditions')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

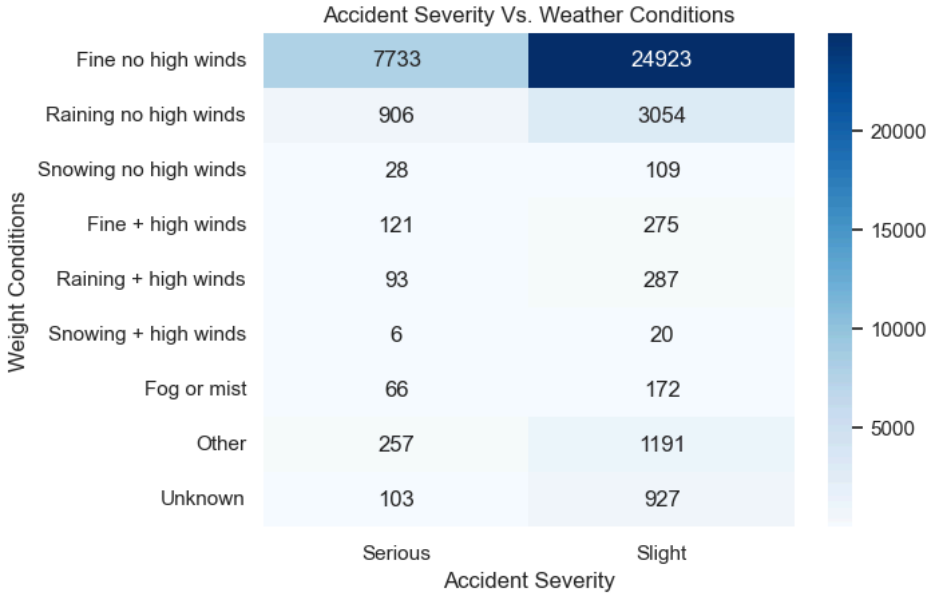


In [106]:
```
# Count plot
sns.countplot(data=train_data, x='light_conditions', hue='accident_severity')
plt.title('Light Conditions vs. Accident Severity')
plt.xlabel('Light Conditions')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

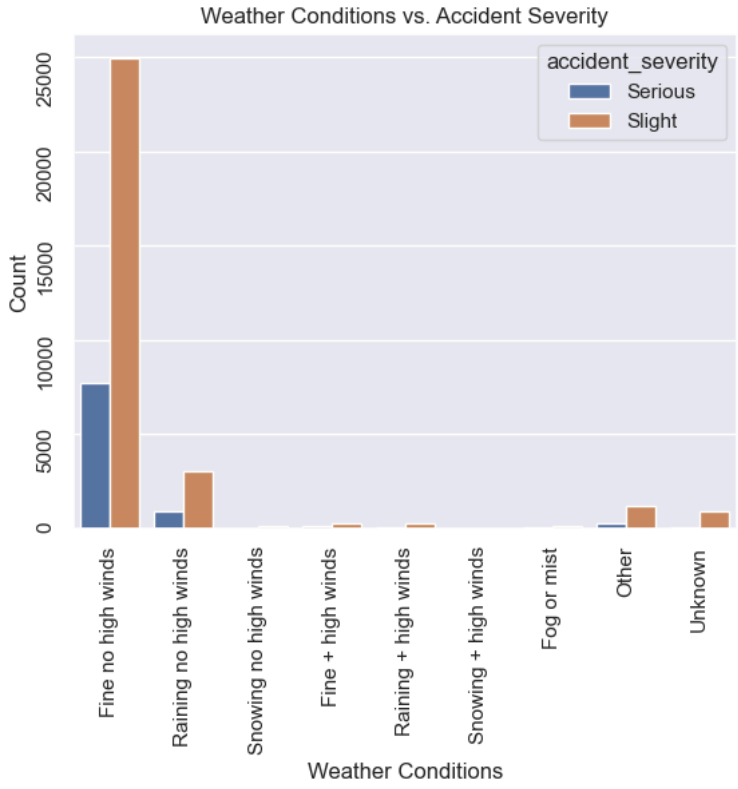


In [107]:
```
# Create a contingency table - Weather Conditions
contingency_table = pd.crosstab(train_data['weather_conditions'], train_data['accident_severity'])
```

In [108]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Weather Conditions')
plt.xlabel('Accident Severity')
plt.ylabel('Weight Conditions')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

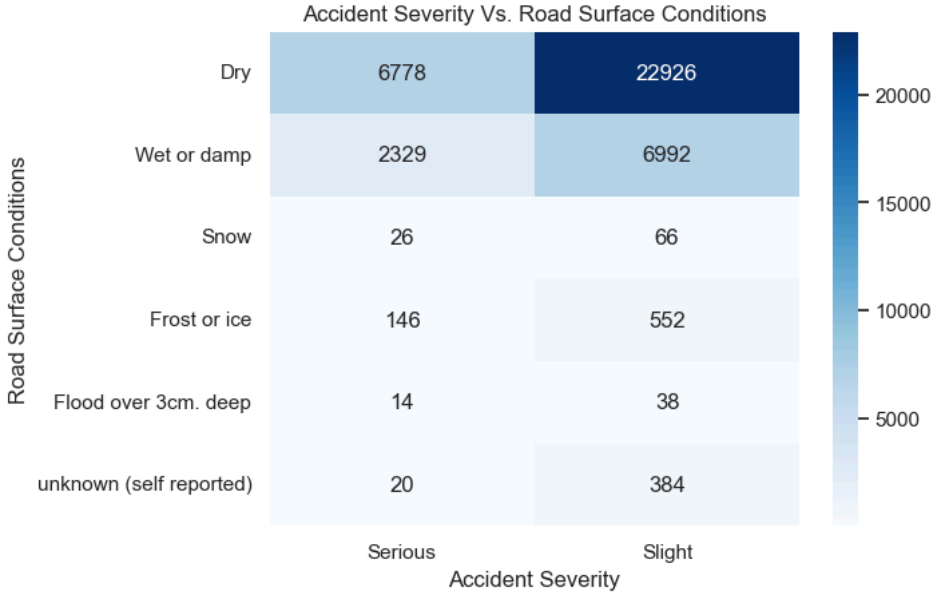


In [109]:
```
# Count plot
sns.countplot(data=train_data, x='weather_conditions', hue='accident_severity')
plt.title('Weather Conditions vs. Accident Severity')
plt.xlabel('Weather Conditions')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

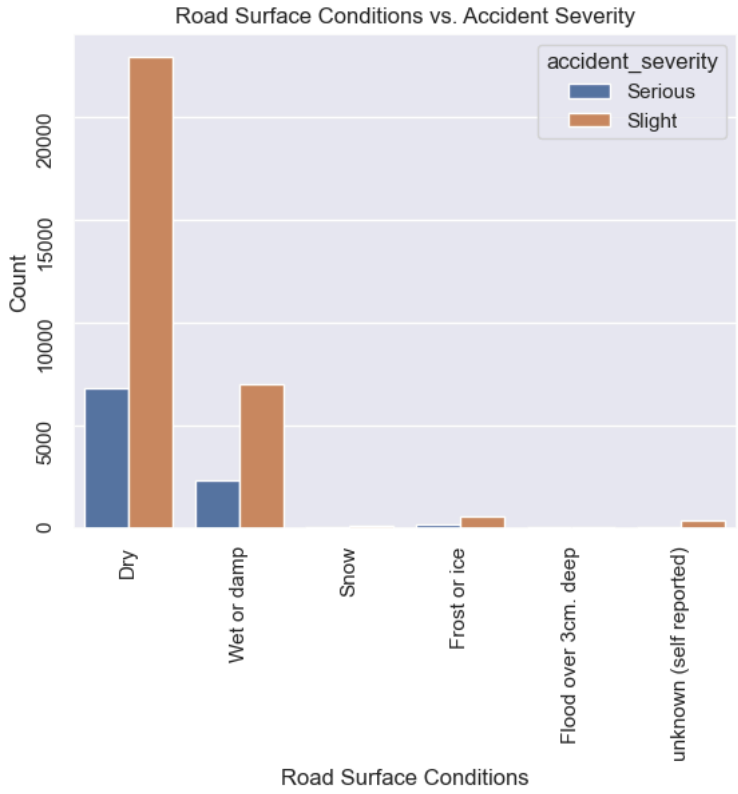


In [110]:
```
# Create a contingency table - road_surface_conditions
contingency_table = pd.crosstab(train_data['road_surface_conditions'], train_data['accident_severity'])
```

In [111]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Road Surface Conditions')
plt.xlabel('Accident Severity')
plt.ylabel('Road Surface Conditions')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

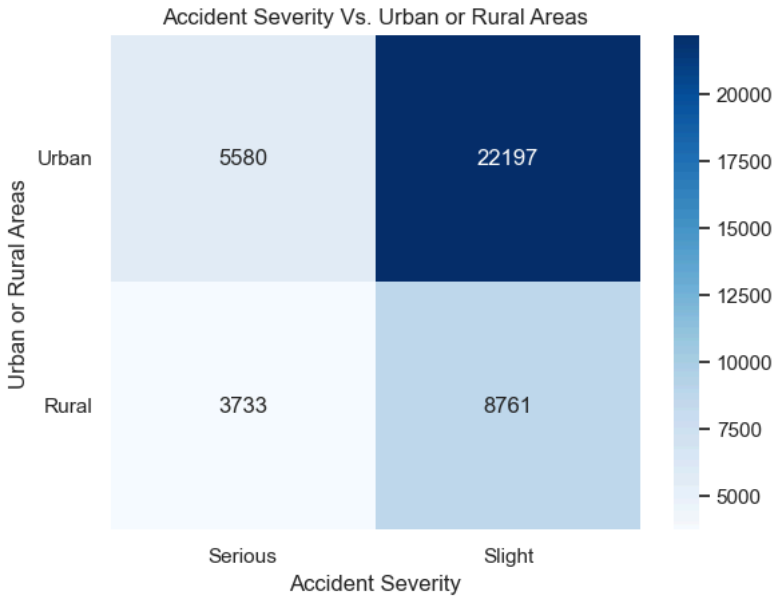


In [112]:
```
# Count plot
sns.countplot(data=train_data, x='road_surface_conditions', hue='accident_severity')
plt.title('Road Surface Conditions vs. Accident Severity')
plt.xlabel('Road Surface Conditions')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

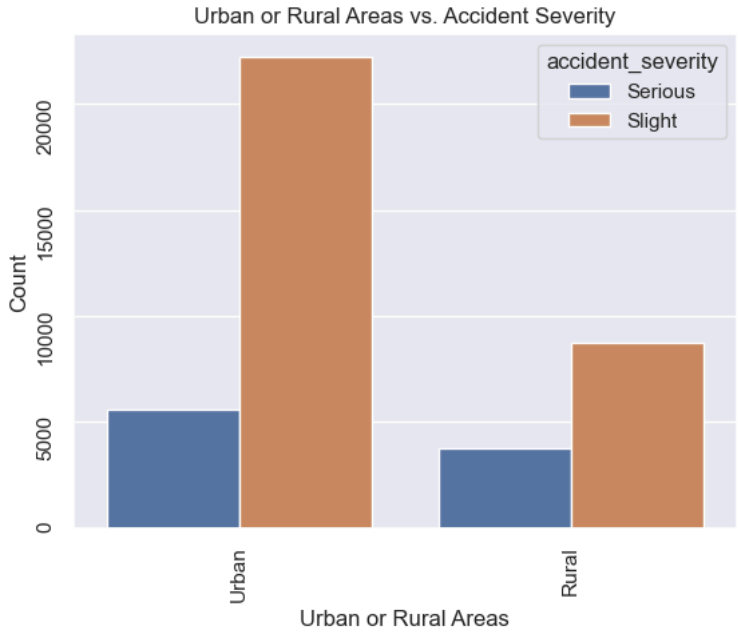


In [113]:
```
# Create a contingency table - urban_or_rural_area
contingency_table = pd.crosstab(train_data['urban_or_rural_area'], train_data['accident_severity'])
```

In [114]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Urban or Rural Areas')
plt.xlabel('Accident Severity')
plt.ylabel('Urban or Rural Areas')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

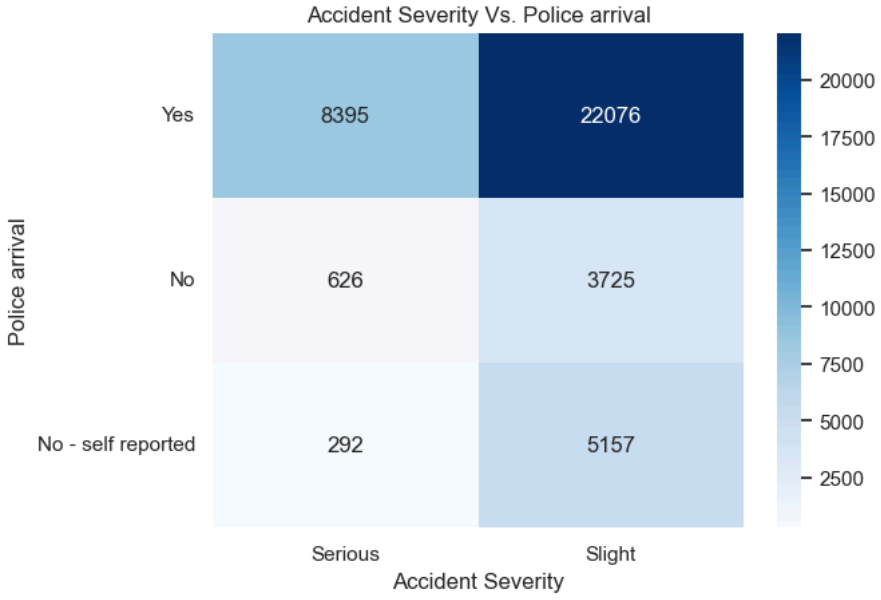


In [115]:
```
# Count plot
sns.countplot(data=train_data, x='urban_or_rural_area', hue='accident_severity')
plt.title('Urban or Rural Areas vs. Accident Severity')
plt.xlabel('Urban or Rural Areas')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

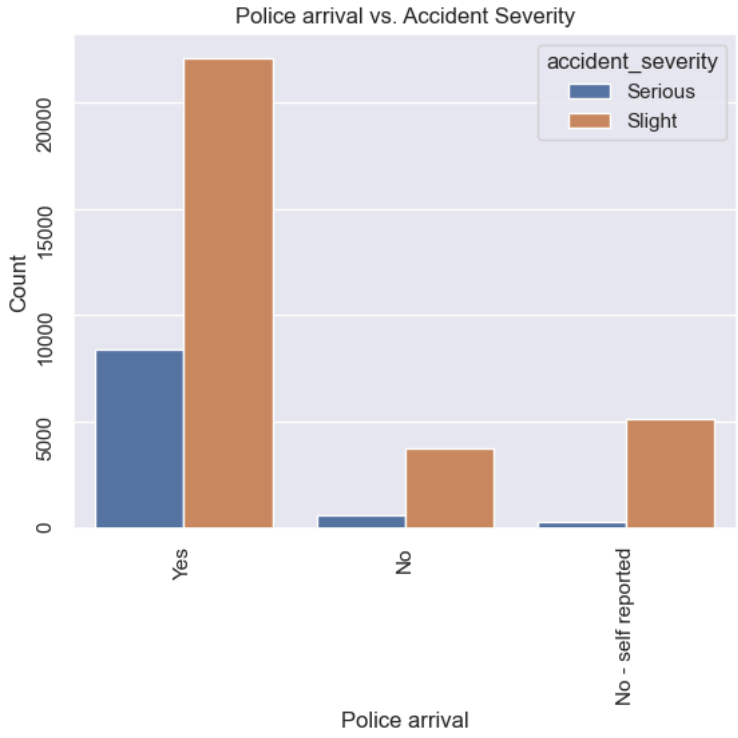


In [116]:
```
# Create a contingency table - did_police_officer_attend_scene_of_accident
contingency_table = pd.crosstab(train_data['did_police_officer_attend_scene_of_accident'], train_data['accident_severity'])
```

In [117]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Police arrival')
plt.xlabel('Accident Severity')
plt.ylabel('Police arrival')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

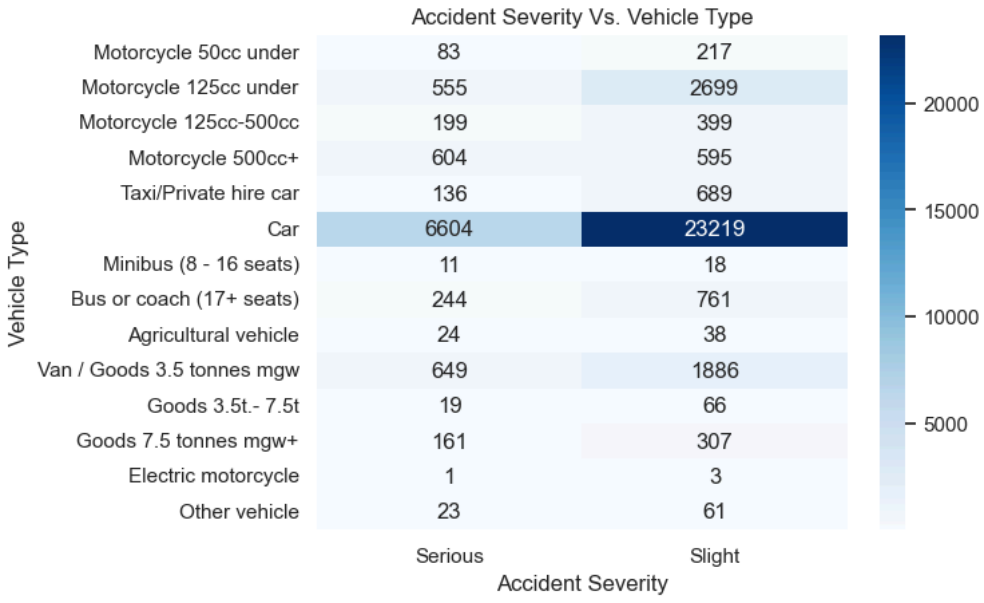


In [118]:
```
# Count plot
sns.countplot(data=train_data, x='did_police_officer_attend_scene_of_accident', hue='accident_severity')
plt.title('Police arrival vs. Accident Severity')
plt.xlabel('Police arrival')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

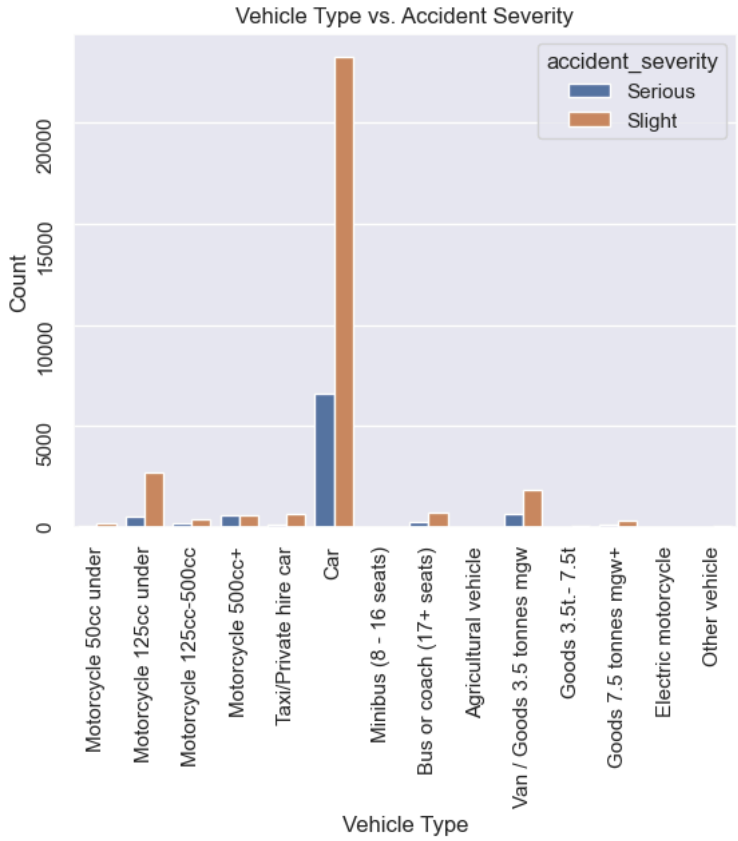


In [119]:
```
# Create a contingency table - vehicle_type
contingency_table = pd.crosstab(train_data['vehicle_type'], train_data['accident_severity'])
```

In [120]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Vehicle Type')
plt.xlabel('Accident Severity')
plt.ylabel('Vehicle Type')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

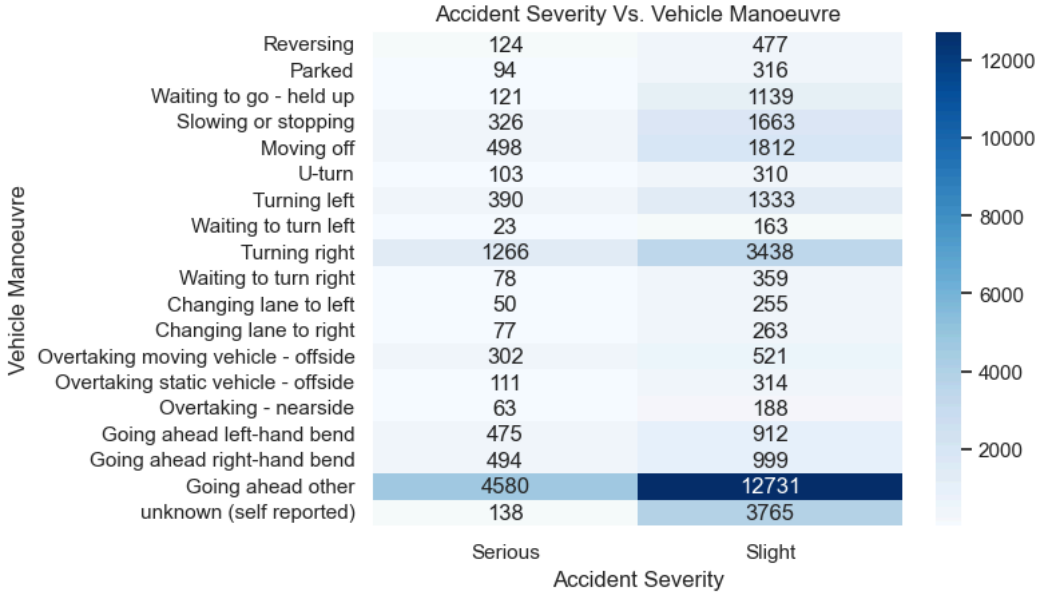


In [121]:
```
# Count plot
sns.countplot(data=train_data, x='vehicle_type', hue='accident_severity')
plt.title('Vehicle Type vs. Accident Severity')
plt.xlabel('Vehicle Type')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

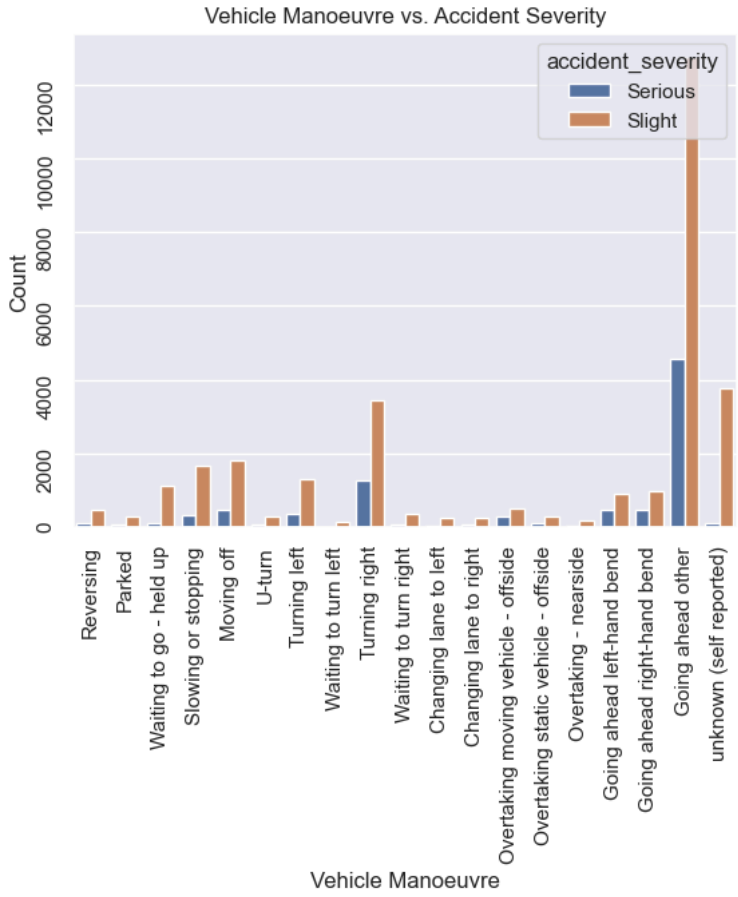


In [122]:
```
# Create a contingency table - vehicle_manoeuvre
contingency_table = pd.crosstab(train_data['vehicle_manoeuvre'], train_data['accident_severity'])
```

In [123]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Vehicle Manoeuvre')
plt.xlabel('Accident Severity')
plt.ylabel('Vehicle Manoeuvre')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

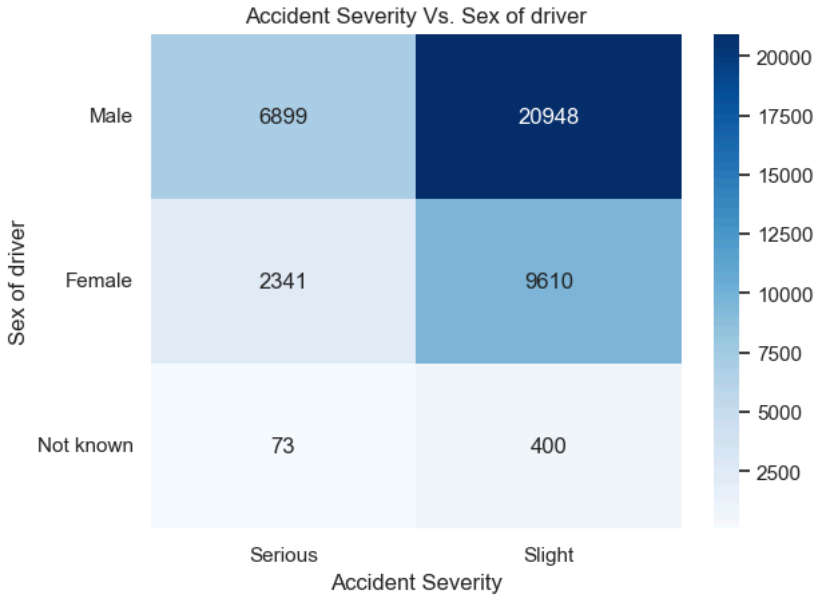


In [124]:
```
# Count plot
sns.countplot(data=train_data, x='vehicle_manoeuvre', hue='accident_severity')
plt.title('Vehicle Manoeuvre vs. Accident Severity')
plt.xlabel('Vehicle Manoeuvre')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=90)
# Adjust layout
plt.show()
```

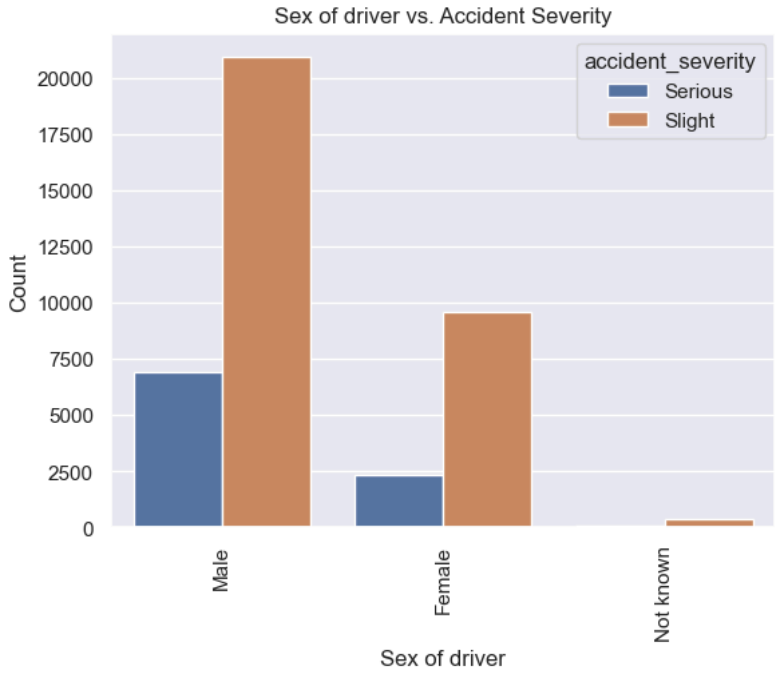


In [125]:
```
# Create a contingency table - sex_of_driver
contingency_table = pd.crosstab(train_data['sex_of_driver'], train_data['accident_severity'])
```

In [126]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Sex of driver')
plt.xlabel('Accident Severity')
plt.ylabel('Sex of driver')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

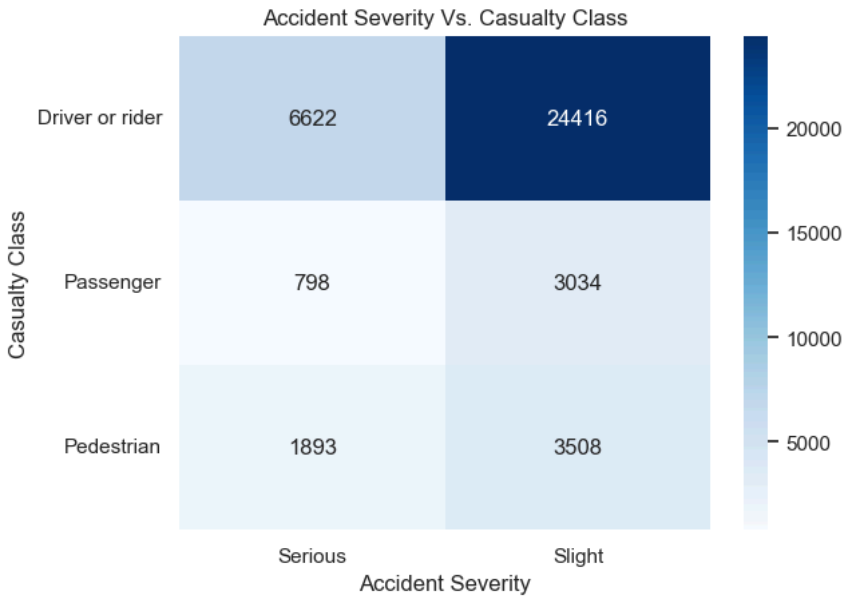


In [127]:
```
# Count plot
sns.countplot(data=train_data, x='sex_of_driver', hue='accident_severity')
plt.title('Sex of driver vs. Accident Severity')
plt.xlabel('Sex of driver')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=0)
# Adjust layout
plt.show()
```

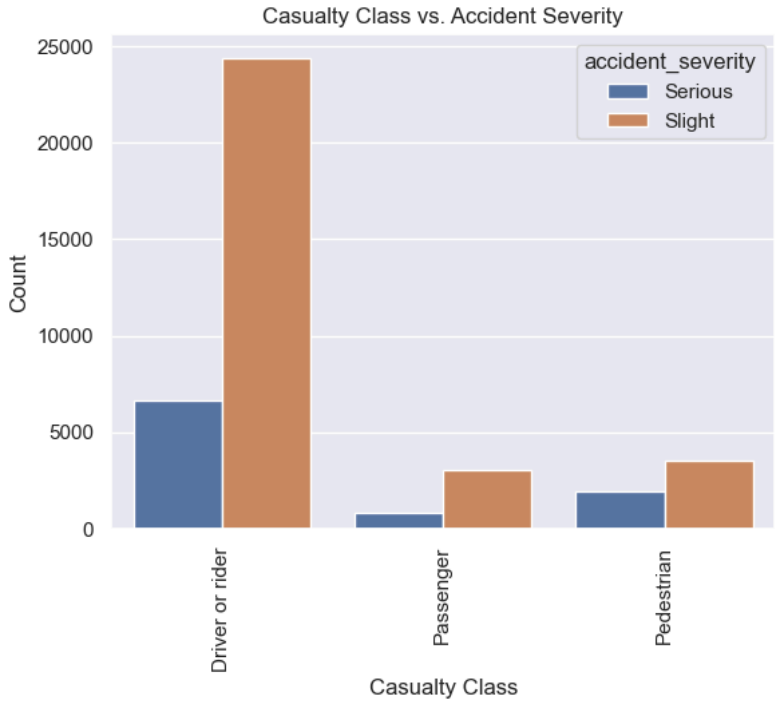


In [128]:
```
# Create a contingency table - casualty_class
contingency_table = pd.crosstab(train_data['casualty_class'], train_data['accident_severity'])
```

In [129]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Casualty Class')
plt.xlabel('Accident Severity')
plt.ylabel('Casualty Class')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

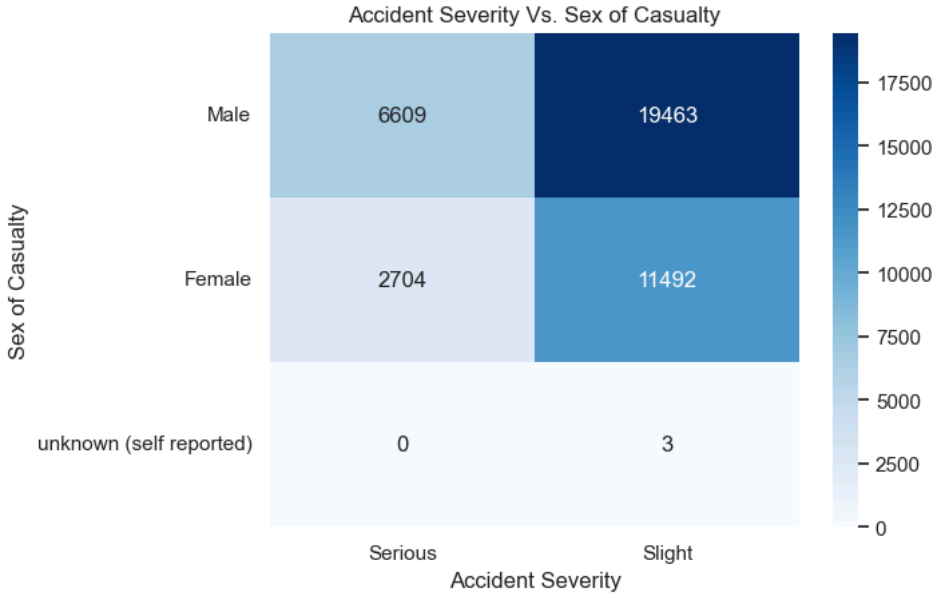


In [130]:
```
# Count plot
sns.countplot(data=train_data, x='casualty_class', hue='accident_severity')
plt.title('Casualty Class vs. Accident Severity')
plt.xlabel('Casualty Class')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=0)
# Adjust layout
plt.show()
```

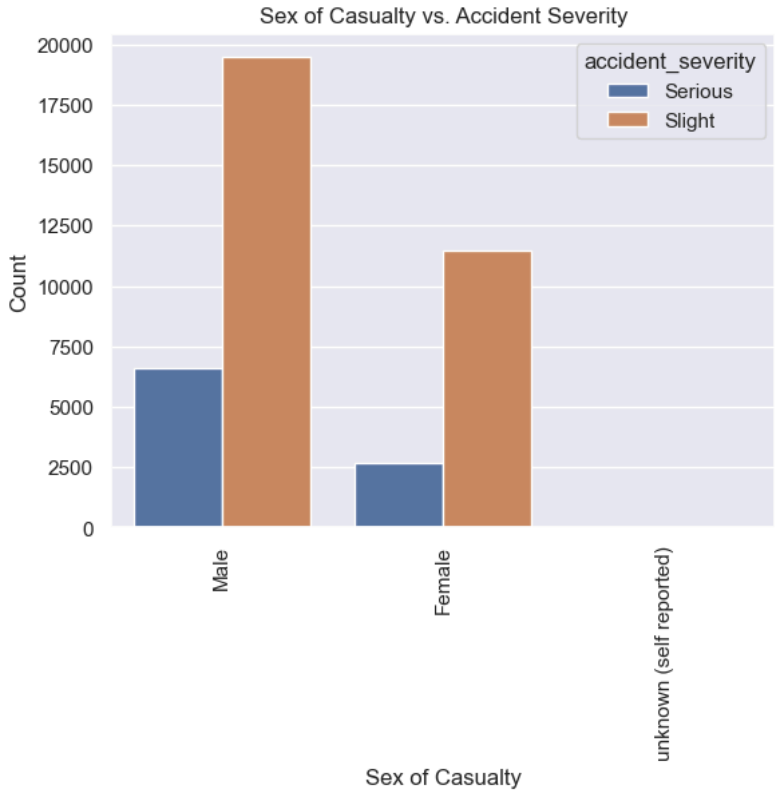


In [131]:
```
# Create a contingency table - sex_of_casualty
contingency_table = pd.crosstab(train_data['sex_of_casualty'], train_data['accident_severity'])
```

In [132]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Sex of Casualty')
plt.xlabel('Accident Severity')
plt.ylabel('Sex of Casualty')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

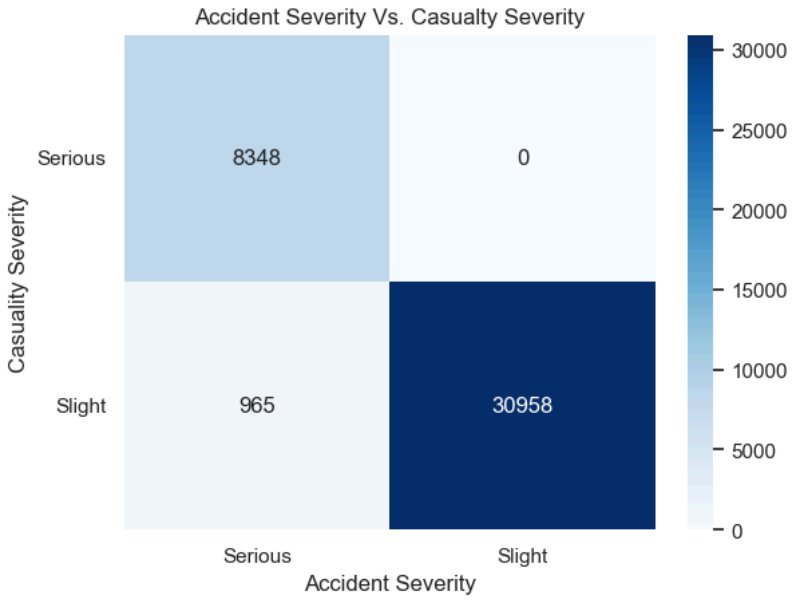


In [133]:
```
# Count plot
sns.countplot(data=train_data, x='sex_of_casualty', hue='accident_severity')
plt.title('Sex of Casualty vs. Accident Severity')
plt.xlabel('Sex of Casualty')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=0)
# Adjust layout
plt.show()
```

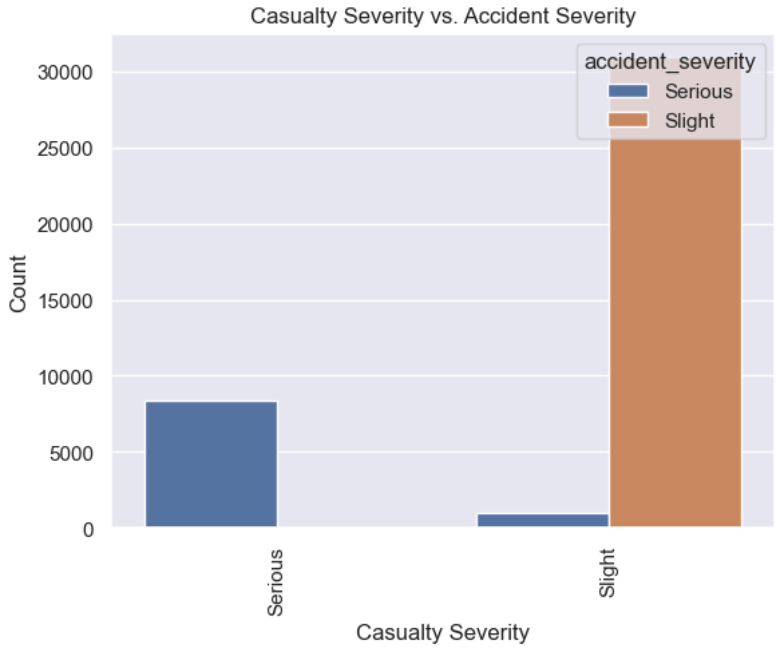


In [134]:
```
# Create a contingency table - casualty_severity
contingency_table = pd.crosstab(train_data['casualty_severity'], train_data['accident_severity'])
```

In [135]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Casualty Severity')
plt.xlabel('Accident Severity')
plt.ylabel('Casuality Severity')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```

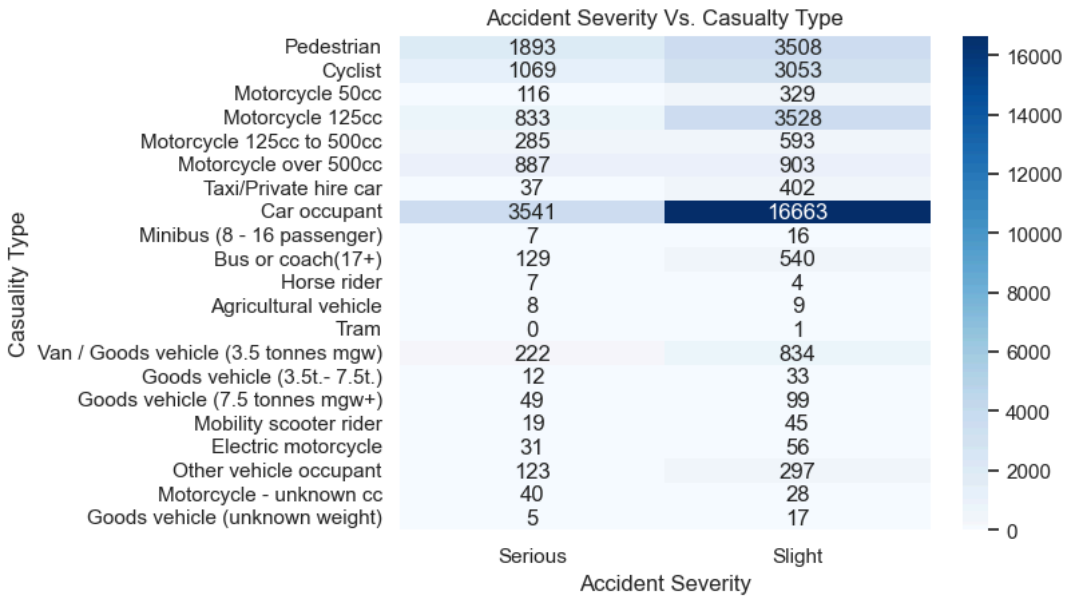


In [136]:
```
# Count plot
sns.countplot(data=train_data, x='casualty_severity', hue='accident_severity')
plt.title('Casualty Severity vs. Accident Severity')
plt.xlabel('Casualty Severity')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=0)
# Adjust layout
plt.show()
```

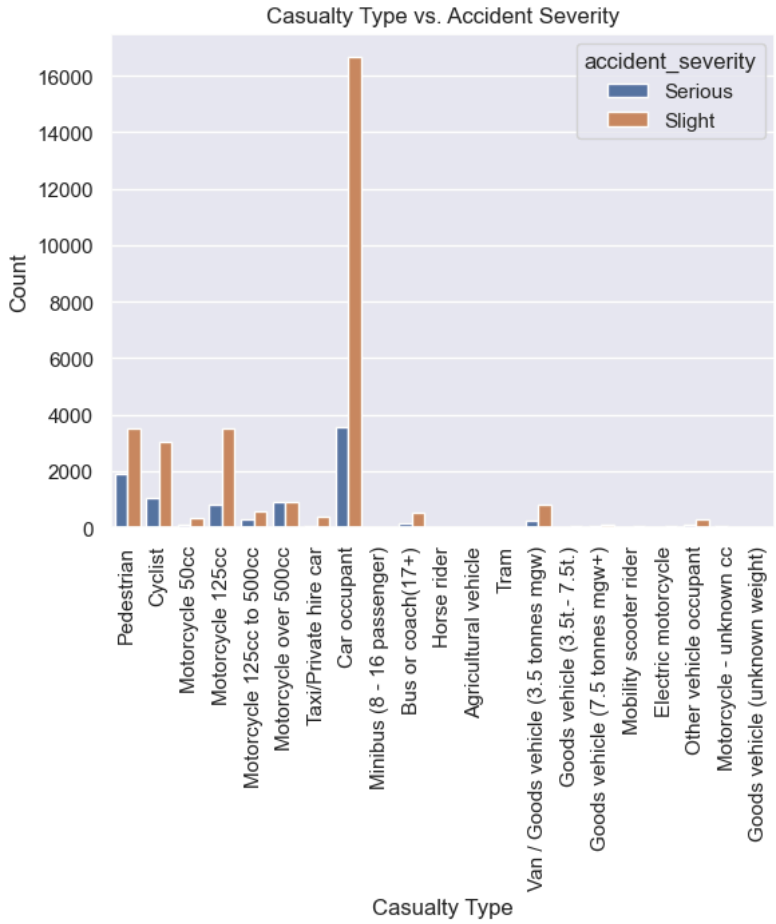


In [137]:
```
# Create a contingency table - casualty_type
contingency_table = pd.crosstab(train_data['casualty_type'], train_data['accident_severity'])
```

In [138]:
```
# Heatmap
sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues')
plt.title('Accident Severity Vs. Casualty Type')
plt.xlabel('Accident Severity')
plt.ylabel('Casuality Type')
plt.tick_params(axis='x', rotation=0)
plt.tick_params(axis='y', rotation=0)
```



In [139]:
```
# Count plot
sns.countplot(data=train_data, x='casualty_type', hue='accident_severity')
plt.title('Casualty Type vs. Accident Severity')
plt.xlabel('Casualty Type')
plt.ylabel("Count")
plt.legend(title='accident_severity', loc='upper right')
plt.tick_params(axis='x', rotation=90)
plt.tick_params(axis='y', rotation=0)
# Adjust layout
plt.show()
```



(v) Chi-square tests to understand Association between categorical variables and target variable accident severity

The Chi-square test of independence is used to determine if there is a significant association between two categorical variables. When applied to understand the association between categorical variables and a target variable (like accident severity), it helps answer whether the distribution of accident severity is different across the categories of another variable (e.g., road surface conditions, weather conditions, etc.).

In [140]:
```
from scipy.stats import chi2_contingency
```

In [141]:
```
# Create a contingency table, passing a single column for the index at a time
chi2_results = []
for col in ['day_of_week', 'road_type', 'light_conditions',
    'weather_conditions', 'road_surface_conditions', 'urban_or_rural_area',
    'did_police_officer_attend_scene_of_accident',
    'vehicle_type', 'vehicle_manoeuvre', 'sex_of_driver', 'casualty_class',
    'sex_of_casualty', 'casualty_severity', 'casualty_type']:
    contingency_table = pd.crosstab(train_data[col], train_data['accident_severity'])

    # Perform the Chi-square test
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    chi2_results.append((col, chi2, p, dof))
```

In [142]:
```
# Print results
for col, chi2, p, dof in chi2_results:
    print(f"Feature: {col}")
    print("Chi-square statistic:", chi2)
    print("P-value:", p)
    print("Degrees of freedom:", dof)
    print("\n")

    # Interpretation of the results
    alpha = 0.05  # significance level
    if p < alpha:
        print(f"There is a significant association between {col} and 'accident_severity' (reject null hypothesis).")
    else:
        print(f"There is no significant association between {col} and 'accident_severity' (fail to reject null hypothesis)."
```

```
Feature: day_of_week
Chi-square statistic: 50.932673183470136
P-value: 3.055538611382338e-09
Degrees of freedom: 6


There is a significant association between day_of_week and 'accident_severity' (reject null hypothesis).
Feature: road_type
Chi-square statistic: 361.0717970619349
P-value: 7.228829713318978e-76
Degrees of freedom: 5


There is a significant association between road_type and 'accident_severity' (reject null hypothesis).
Feature: light_conditions
Chi-square statistic: 201.55565120894622
P-value: 1.7394005780967014e-42
Degrees of freedom: 4


There is a significant association between light_conditions and 'accident_severity' (reject null hypothesis).
Feature: weather_conditions
Chi-square statistic: 145.23093990684936
P-value: 1.9341967202547868e-27
Degrees of freedom: 8


There is a significant association between weather_conditions and 'accident_severity' (reject null hypothesis).
Feature: road_surface_conditions
Chi-square statistic: 98.50455638759897
P-value: 1.0918648005639404e-19
Degrees of freedom: 5


There is a significant association between road_surface_conditions and 'accident_severity' (reject null hypothesis).
Feature: urban_or_rural_area
Chi-square statistic: 464.0316629070293
P-value: 6.375878217578723e-103
Degrees of freedom: 1


There is a significant association between urban_or_rural_area and 'accident_severity' (reject null hypothesis).
Feature: did_police_officer_attend_scene_of_accident
Chi-square statistic: 1490.0280925918341
P-value: 0.0
Degrees of freedom: 2


There is a significant association between did_police_officer_attend_scene_of_accident and 'accident_severity' (reject null
hypothesis).
Feature: vehicle_type
Chi-square statistic: 698.9139323241739
P-value: 5.876021288531459e-141
Degrees of freedom: 13


There is a significant association between vehicle_type and 'accident_severity' (reject null hypothesis).
Feature: vehicle_manoeuvre
Chi-square statistic: 1469.7653609392826
P-value: 1.491000308327125e-301
Degrees of freedom: 18


There is a significant association between vehicle_manoeuvre and 'accident_severity' (reject null hypothesis).
Feature: sex_of_driver
Chi-square statistic: 142.45293329405783
P-value: 1.1661074493532001e-31
Degrees of freedom: 2


There is a significant association between sex_of_driver and 'accident_severity' (reject null hypothesis).
Feature: casualty_class
Chi-square statistic: 499.29918868239076
P-value: 3.789298066281703e-109
Degrees of freedom: 2


There is a significant association between casualty_class and 'accident_severity' (reject null hypothesis).
Feature: sex_of_casualty
Chi-square statistic: 206.19769792707993
P-value: 1.6777946474272785e-45
Degrees of freedom: 2


There is a significant association between sex_of_casualty and 'accident_severity' (reject null hypothesis).
Feature: casualty_severity
Chi-square statistic: 35001.50937592414
P-value: 0.0
Degrees of freedom: 1
```

```
There is a significant association between casualty_severity and 'accident_severity' (reject null hypothesis).
Feature: casualty_type
Chi-square statistic: 1748.3445300120677
P-value: 0.0
Degrees of freedom: 20


There is a significant association between casualty_type and 'accident_severity' (reject null hypothesis).
```

7. Data Transformation and Preprocessing

Based on the Exploration of the data along with descriptive statistics we generate, here are the final preprocessing steps we need to do before preparing predictive models. The change will be applied to both training and test sets, based on understanding of Training data we have now from EDA.

The key idea is to use the insights from EDA to define a cleaning and transformation pipeline that can be applied to both the training and test sets. This avoids data leakage and ensures that both sets undergo the same preprocessing steps.

7.1 Data wrangling for categorical variables

Here we will first do Cardinality reduction which involves reducing the number of unique values in categorical variables. High cardinality can lead to problems in machine learning models, such as overfitting, high memory usage, and increased computational time.

We will do it for selected few categorical variables depending on the distribution and value count of number of occurrences of each category.

In [143]:
```
def reduce_cardinality(data):

    # Frequent categories retained; infrequent categories grouped into 'Other'
    # Reduce cardinality for 'road_type'
    data['road_type'] = data['road_type'].replace({
        'Single carriageway': 'Single carriageway',
        'Dual carriageway': 'Dual carriageway',
        'Roundabout': 'Roundabout',
        'Slip road': 'Other',
        'One way street': 'Other',
        'Unknown': 'Other'
    }).fillna('Other')

    # Convert 'light_conditions' to binary
    data['light_conditions'] = data['light_conditions'].replace({
        'Daylight': 'Light',
        'Darkness - lights lit': 'Dark',
        'Darkness - no lighting': 'Dark',
        'Darkness - lighting unknown': 'Dark',
        'Darkness - lights unlit': 'Dark'
    }).fillna('Dark')

    # Reduce cardinality for 'weather_conditions'
    data['weather_conditions'] = data['weather_conditions'].replace({
        'Fine no high winds': 'Fine Weather',
        'Raining no high winds': 'Disturbance Weather',
        'Other': 'Disturbance Weather',
        'Raining + high winds': 'Disturbance Weather',
        'Fine + high winds': 'Disturbance Weather',
        'Unknown': 'Disturbance Weather',
        'Fog or mist': 'Disturbance Weather',
        'Snowing no high winds': 'Disturbance Weather',
        'Snowing + high winds': 'Disturbance Weather'
    }).fillna('Disturbance Weather')

    # Convert 'urban_or_rural_area' to binary
    data['urban_or_rural_area'] = data['urban_or_rural_area'].replace({
        'Urban': 'Urban Area',
        'Rural': 'Rural Area'
    }).fillna('Urban Area')

    # Convert 'did_police_officer_attend_scene_of_accident' to binary
    data['did_police_officer_attend_scene_of_accident'] = data['did_police_officer_attend_scene_of_accident'].replace({
        'Yes': 'Attended',
        'No': 'Not Attended',
        'No - self reported': 'Not Attended'
    }).fillna('Not Attended')

    # Reduce cardinality for 'vehicle_type'
    data['vehicle_type'] = data['vehicle_type'].replace({
        'Car': 'Car/Passenger',
        'Van / Goods 3.5 tonnes mgw': 'Goods Carrier',
        'Motorcycle 125cc under': 'Motorcycle/Two-Wheeler',
        'Motorcycle 500cc+': 'Motorcycle/Two-Wheeler',
        'Taxi/Private hire car': 'Car/Passenger',
        'Goods 7.5 tonnes mgw+': 'Goods Carrier',
        'Bus or coach (17+ seats)': 'Bus/Coach',
        'Motorcycle 125cc-500cc': 'Motorcycle/Two-Wheeler',
        'Motorcycle 50cc under': 'Motorcycle/Two-Wheeler',
        'Goods vehicle - unknown weight': 'Goods Carrier',
        'Agricultural vehicle': 'Motorcycle/Two-Wheeler',
        'Goods 3.5t.- 7.5t': 'Goods Carrier',
        'Other vehicle': 'Motorcycle/Two-Wheeler',
        'Motorcycle - unknown cc': 'Motorcycle/Two-Wheeler',
        'Minibus (8 - 16 seats)': 'Bus/Coach',
        'Electric motorcycle': 'Motorcycle/Two-Wheeler',
        'Mobility scooter': 'Motorcycle/Two-Wheeler'
    }).fillna('Motorcycle/Two-Wheeler')

    # Reduce cardinality for 'vehicle_manoeuvre'
    data['vehicle_manoeuvre'] = data['vehicle_manoeuvre'].replace({
        'Going ahead other': 'Going ahead',
        'Going ahead right-hand bend': 'Going ahead',
        'Going ahead left-hand bend': 'Going ahead',
        'Turning right': 'Turning',
        'Turning left': 'Turning',
        'Slowing or stopping': 'Stopping or Slowing',
        'Overtaking moving vehicle - offside': 'Overtaking',
        'Overtaking static vehicle - offside': 'Overtaking',
        'Overtaking - nearside': 'Overtaking',
        'Waiting to go - held up': 'Waiting or Holding',
        'Waiting to turn right': 'Waiting or Holding',
        'Waiting to turn left': 'Waiting or Holding',
        'Changing lane to left': 'Changing lanes',
        'Changing lane to right': 'Changing lanes',
        'Reversing': 'Miscellaneous',
        'Parked': 'Miscellaneous',
        'U-turn': 'Miscellaneous',
        'unknown (self reported)': 'Miscellaneous'
    }).fillna('Miscellaneous')  # Fill any NaNs with 'Miscellaneous'
```

```
    # Replace 'Male' with 'Male' (or any label you prefer), and 'Female' or 'Not known' with 'Female/Unknown'
    data['sex_of_driver'] = data['sex_of_driver'].replace({
        'Male': 'Male',
        'Female': 'Female',
        'Not known': 'Female'
    }).fillna('Female')

    # Reduce cardinality for 'casualty_class'
    data['casualty_class'] = data['casualty_class'].replace({
       'Driver or rider': 'Driver or rider',
       'Pedestrian': 'Pedestrian/Passenger',
       'Passenger': 'Pedestrian/Passenger'
    })

    # Reduce cardinality for 'casualty_type'
    data['casualty_type'] = data['casualty_type'].replace({
        'Car occupant': 'Car occupant',
        'Pedestrian': 'Pedestrian/Cyclist',
        'Cyclist': 'Pedestrian/Cyclist',
        'Motorcycle 125cc': 'Motorcycle',
        'Motorcycle over 500cc': 'Motorcycle',
        'Van / Goods vehicle (3.5 tonnes mgw)': 'Goods vehicle/Bus/Other',
        'Motorcycle 125cc to 500cc': 'Motorcycle',
        'Motorcycle 50cc': 'Motorcycle',
        'Taxi/Private hire car': 'Car occupant',
        'Bus or coach(17+)': 'Goods vehicle/Bus/Other',
        'Goods vehicle (7.5 tonnes mgw+)': 'Goods vehicle/Bus/Other',
        'Other vehicle occupant': 'Goods vehicle/Bus/Other',
        'Motorcycle - unknown cc': 'Motorcycle',
        'Goods vehicle (unknown weight)': 'Goods vehicle/Bus/Other',
        'Goods vehicle (3.5t.- 7.5t.)': 'Goods vehicle/Bus/Other',
        'Mobility scooter rider': 'Goods vehicle/Bus/Other',
        'Agricultural vehicle': 'Goods vehicle/Bus/Other',
        'Electric motorcycle': 'Motorcycle',
        'Minibus (8 - 16 passenger)': 'Car occupant',
        'Horse rider': 'Pedestrian/Cyclist',
        'Tram': 'Goods vehicle/Bus/Other'
    }).fillna('Pedestrian/Cyclist')

    # Add more transformations for other variables as needed

    return data
```

In [144]:
```
# Apply reduce_cardinality function to both train_data and test_data
train_data = reduce_cardinality(train_data)
test_data = reduce_cardinality(test_data)
```

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

Out[145]:

	accident_index	longitude	latitude	number_of_vehicles	number_of_casualties	day_of_week	road_type	speed_limit	light_conditions	weather_
86666	2022170S12142	-1.405515	54.566513	2	2	Monday	Single carriageway	50	Dark	Disturbar
105262	2022221257763	-2.153893	52.267966	1	1	Saturday	Single carriageway	30	Dark	F
68375	2022131179676	-1.845597	53.859536	1	1	Friday	Single carriageway	30	Light	F
143891	2022520203486	-2.644064	50.945872	2	1	Monday	Single carriageway	30	Light	F
30751	2022010392849	-0.174342	51.499817	2	1	Monday	Single carriageway	20	Light	F

5 rows × 25 columns

Feature Engineering and Dropping insignificant variables further.

(A) Dropping variables Accident Index and Age of Casualty

Accident Index was used to merge the datasets. It does not have any predictive power to be used in our models to predict accident severity.

Age of Casualty variable is being removed after through consideration and comparing it with age of driver using correlation matrix and Scatter plot. This two were moderately positively related with correlation coefficient equal to 0.504. This is not a very strong correlation, but it is also not weak. It indicates that there is a noticeable trend between the two variables, which can also be seen in Scatter plot of two variables where there was a straight sharp line with equal x and y coordinates. This signifies that in those cases mostly, the driver and casualty was the same person involved.

Also Age of driver is related directly to the severity of accident compared to age of casualty as situation is not in his control most of the time. His age does'st make as asignificant factor, and hence we are removing them.

In [146]:
```
# List of columns to drop
columns_to_drop = ['accident_index', 'age_of_casualty']

# Drop the columns from train_data
train_data = train_data.drop(columns=columns_to_drop)

# Drop the columns from test_data
test_data = test_data.drop(columns=columns_to_drop)

# Display the first few rows to confirm
print(train_data.shape)
print(test_data.shape)
```

```
(40271, 23)
(17260, 23)
```

(B) Feature Engineering #Feature 1

Location of Accident- This feature can be engineered by using latitude and longitude together, which can be used to analyse the nearby landmarks near the coordinates of the accident location. This can help in optimizing the Emergency Response time to reduce the fatality of the accident and casualty damage. We will construct this feature during individual assignment if required.

(C) Feature Engineering #Feature 2

Converting age of vehicle and age of driver into categorical age bands- This features would help us in making more effective predictions on effect of age of vehicles and driver on the severity of the accidents.

In [147]:
```
# Define age bands for 'age_of_driver'
train_data['age_of_driver_band'] = pd.cut(train_data['age_of_driver'],
                                          bins=[0, 17, 24, 34, 44, 54, 64, np.inf],
                                          labels=['Underage/Teen', 'Young Adult', 'Adult',
                                                  'Middle-aged', 'Mature', 'Senior', 'Elderly'])
test_data['age_of_driver_band'] = pd.cut(test_data['age_of_driver'],
                                         bins=[0, 17, 24, 34, 44, 54, 64, np.inf],
                                         labels=['Underage/Teen', 'Young Adult', 'Adult',
                                                 'Middle-aged', 'Mature', 'Senior', 'Elderly'])

# Define age bands for 'age_of_vehicle'
train_data['age_of_vehicle_band'] = pd.cut(train_data['age_of_vehicle'],
                                           bins=[-1, 2, 5, 10, 20, np.inf],
                                           labels=['New', 'Relatively New', 'Moderately Old',
                                                   'Old', 'Very Old'])
test_data['age_of_vehicle_band'] = pd.cut(test_data['age_of_vehicle'],
                                          bins=[-1, 2, 5, 10, 20, np.inf],
                                          labels=['New', 'Relatively New', 'Moderately Old',
                                                  'Old', 'Very Old'])

# Drop the original age columns
train_data.drop(columns=['age_of_driver', 'age_of_vehicle'], inplace=True)
test_data.drop(columns=['age_of_driver', 'age_of_vehicle'], inplace=True)
```

In [148]:
```
# Convert age band columns to category type
train_data['age_of_driver_band'] = train_data['age_of_driver_band'].astype('category')
test_data['age_of_driver_band'] = test_data['age_of_driver_band'].astype('category')

train_data['age_of_vehicle_band'] = train_data['age_of_vehicle_band'].astype('category')
test_data['age_of_vehicle_band'] = test_data['age_of_vehicle_band'].astype('category')
```

In [149]:
```
train_data.head()
```

Out[149]:

	longitude	latitude	number_of_vehicles	number_of_casualties	day_of_week	road_type	speed_limit	light_conditions	weather_conditions	road
86666	-1.405515	54.566513	2	2	Monday	Single carriageway	50	Dark	Disturbance Weather	
105262	-2.153893	52.267966	1	1	Saturday	Single carriageway	30	Dark	Fine Weather	
68375	-1.845597	53.859536	1	1	Friday	Single carriageway	30	Light	Fine Weather	
143891	-2.644064	50.945872	2	1	Monday	Single carriageway	30	Light	Fine Weather	
30751	-0.174342	51.499817	2	1	Monday	Single carriageway	20	Light	Fine Weather	

5 rows × 23 columns

(D) Feature Engineering #Feature 3

Constructing Time based features from Datetime column which would help us to understand following time based

In [150]:
```
def create_time_based_features(train_data, test_data):


    # Extract month and hour
    for df in [train_data, test_data]:
        df['month'] = df['datetime'].dt.month
        df['hour'] = df['datetime'].dt.hour

        # Create a feature to indicate if the accident happened on a weekend
        df['is_weekend'] = df['datetime'].dt.dayofweek.apply(lambda x: 1 if x >= 5 else 0)

        # Drop the 'datetime' column as it is no longer needed
        df.drop(columns=['datetime'], inplace=True)

    return train_data, test_data

# Apply the function to your datasets
train_data, test_data = create_time_based_features(train_data, test_data)
```

In [151]:
```
# Optimize data types
train_data['is_weekend'] = train_data['is_weekend'].astype('int8')
test_data['is_weekend'] = test_data['is_weekend'].astype('int8')
```

In [152]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 40271 entries, 86666 to 12098
Data columns (total 25 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   longitude                                    40271 non-null  float64
 1   latitude                                     40271 non-null  float64
 2   number_of_vehicles                           40271 non-null  int64
 3   number_of_casualties                         40271 non-null  int64
 4   day_of_week                                  40271 non-null  category
 5   road_type                                    40271 non-null  category
 6   speed_limit                                  40271 non-null  int64
 7   light_conditions                             40271 non-null  category
 8   weather_conditions                           40271 non-null  category
 9   road_surface_conditions                      40271 non-null  category
 10  urban_or_rural_area                          40271 non-null  category
 11  did_police_officer_attend_scene_of_accident  40271 non-null  category
 12  vehicle_type                                 40271 non-null  category
 13  vehicle_manoeuvre                            40271 non-null  category
 14  sex_of_driver                                40271 non-null  category
 15  casualty_class                               40271 non-null  category
 16  sex_of_casualty                              40271 non-null  category
 17  casualty_severity                            40271 non-null  category
 18  casualty_type                                40271 non-null  category
 19  accident_severity                            40271 non-null  category
 20  age_of_driver_band                           40271 non-null  category
 21  age_of_vehicle_band                          40271 non-null  category
 22  month                                        40271 non-null  int32
 23  hour                                         40271 non-null  int32
 24  is_weekend                                   40271 non-null  int8
dtypes: category(17), float64(2), int32(2), int64(3), int8(1)
memory usage: 3.9 MB
```

7.3 Creation of Dummy Variables

Using One-Hot Encoder to create Dummy Variables to handle Categorical variables

In [153]:
```
from sklearn.preprocessing import OneHotEncoder

# Separate the target and predictors again
ytrain = train_data["accident_severity"].copy()
Xtrain = train_data.drop("accident_severity", axis=1)
ytest = test_data["accident_severity"].copy()
Xtest = test_data.drop("accident_severity", axis=1)

# check the size of the Xtrain and Xtest datasets
print(Xtrain.shape)
print(Xtest.shape)

# Identify categorical columns
categorical_cols = Xtrain.select_dtypes(include=['category', 'object']).columns.tolist()

# Apply one-hot encoding to categorical columns
encoder = OneHotEncoder(drop='first', sparse=False, handle_unknown='ignore')
Xtrain_encoded = pd.DataFrame(encoder.fit_transform(Xtrain[categorical_cols]), columns=encoder.get_feature_names_out(categor
Xtest_encoded = pd.DataFrame(encoder.transform(Xtest[categorical_cols]), columns=encoder.get_feature_names_out(categorical_c

# Drop original categorical columns and concatenate the encoded columns
Xtrain = Xtrain.drop(columns=categorical_cols).reset_index(drop=True)
Xtest = Xtest.drop(columns=categorical_cols).reset_index(drop=True)

Xtrain = pd.concat([Xtrain, Xtrain_encoded], axis=1)
Xtest = pd.concat([Xtest, Xtest_encoded], axis=1)
```

```
(40271, 24)
(17260, 24)

C:\Users\bhara\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `
sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its defau
lt value.
  warnings.warn(
```

In [154]:
```
# check if the dummies are produced correctly in the trainset
Xtrain_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40271 entries, 0 to 40270
Data columns (total 46 columns):
 #   Column                                                  Non-Null Count  Dtype
---  ------                                                  --------------  -----
 0   day_of_week_Monday                                      40271 non-null  float64
 1   day_of_week_Saturday                                    40271 non-null  float64
 2   day_of_week_Sunday                                      40271 non-null  float64
 3   day_of_week_Thursday                                    40271 non-null  float64
 4   day_of_week_Tuesday                                     40271 non-null  float64
 5   day_of_week_Wednesday                                   40271 non-null  float64
 6   road_type_Other                                         40271 non-null  float64
 7   road_type_Roundabout                                    40271 non-null  float64
 8   road_type_Single carriageway                            40271 non-null  float64
 9   light_conditions_Light                                  40271 non-null  float64
 10  weather_conditions_Fine Weather                         40271 non-null  float64
 11  road_surface_conditions_Flood over 3cm. deep            40271 non-null  float64
 12  road_surface_conditions_Frost or ice                    40271 non-null  float64
 13  road_surface_conditions_Snow                            40271 non-null  float64
 14  road_surface_conditions_Wet or damp                     40271 non-null  float64
 15  road_surface_conditions_unknown (self reported)         40271 non-null  float64
 16  urban_or_rural_area_Urban Area                          40271 non-null  float64
 17  did_police_officer_attend_scene_of_accident_Not Attended  40271 non-null  float64
 18  vehicle_type_Car/Passenger                              40271 non-null  float64
 19  vehicle_type_Goods Carrier                              40271 non-null  float64
 20  vehicle_type_Motorcycle/Two-Wheeler                     40271 non-null  float64
 21  vehicle_manoeuvre_Going ahead                           40271 non-null  float64
 22  vehicle_manoeuvre_Miscellaneous                         40271 non-null  float64
 23  vehicle_manoeuvre_Moving off                            40271 non-null  float64
 24  vehicle_manoeuvre_Overtaking                            40271 non-null  float64
 25  vehicle_manoeuvre_Stopping or Slowing                   40271 non-null  float64
 26  vehicle_manoeuvre_Turning                               40271 non-null  float64
 27  vehicle_manoeuvre_Waiting or Holding                    40271 non-null  float64
 28  sex_of_driver_Male                                      40271 non-null  float64
 29  casualty_class_Pedestrian/Passenger                     40271 non-null  float64
 30  sex_of_casualty_Male                                    40271 non-null  float64
 31  sex_of_casualty_unknown (self reported)                 40271 non-null  float64
 32  casualty_severity_Slight                                40271 non-null  float64
 33  casualty_type_Goods vehicle/Bus/Other                   40271 non-null  float64
 34  casualty_type_Motorcycle                                40271 non-null  float64
 35  casualty_type_Pedestrian/Cyclist                        40271 non-null  float64
 36  age_of_driver_band_Elderly                              40271 non-null  float64
 37  age_of_driver_band_Mature                               40271 non-null  float64
 38  age_of_driver_band_Middle-aged                          40271 non-null  float64
 39  age_of_driver_band_Senior                               40271 non-null  float64
 40  age_of_driver_band_Underage/Teen                        40271 non-null  float64
 41  age_of_driver_band_Young Adult                          40271 non-null  float64
 42  age_of_vehicle_band_New                                 40271 non-null  float64
 43  age_of_vehicle_band_Old                                 40271 non-null  float64
 44  age_of_vehicle_band_Relatively New                      40271 non-null  float64
 45  age_of_vehicle_band_Very Old                            40271 non-null  float64
dtypes: float64(46)
memory usage: 14.1 MB
```

In [155]:
```
# check if the dummies are produced correctly in the testset
Xtest_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17260 entries, 0 to 17259
Data columns (total 46 columns):
 #   Column                                                Non-Null Count  Dtype  
---  ------                                                --------------  -----  
 0   day_of_week_Monday                                    17260 non-null  float64
 1   day_of_week_Saturday                                  17260 non-null  float64
 2   day_of_week_Sunday                                    17260 non-null  float64
 3   day_of_week_Thursday                                  17260 non-null  float64
 4   day_of_week_Tuesday                                   17260 non-null  float64
 5   day_of_week_Wednesday                                 17260 non-null  float64
 6   road_type_Other                                       17260 non-null  float64
 7   road_type_Roundabout                                  17260 non-null  float64
 8   road_type_Single carriageway                          17260 non-null  float64
 9   light_conditions_Light                                17260 non-null  float64
 10  weather_conditions_Fine Weather                       17260 non-null  float64
 11  road_surface_conditions_Flood over 3cm. deep          17260 non-null  float64
 12  road_surface_conditions_Frost or ice                  17260 non-null  float64
 13  road_surface_conditions_Snow                          17260 non-null  float64
 14  road_surface_conditions_Wet or damp                   17260 non-null  float64
 15  road_surface_conditions_unknown (self reported)       17260 non-null  float64
 16  urban_or_rural_area_Urban Area                        17260 non-null  float64
 17  did_police_officer_attend_scene_of_accident_Not Attended  17260 non-null  float64
 18  vehicle_type_Car/Passenger                            17260 non-null  float64
 19  vehicle_type_Goods Carrier                            17260 non-null  float64
 20  vehicle_type_Motorcycle/Two-Wheeler                   17260 non-null  float64
 21  vehicle_manoeuvre_Going ahead                         17260 non-null  float64
 22  vehicle_manoeuvre_Miscellaneous                       17260 non-null  float64
 23  vehicle_manoeuvre_Moving off                          17260 non-null  float64
 24  vehicle_manoeuvre_Overtaking                          17260 non-null  float64
 25  vehicle_manoeuvre_Stopping or Slowing                 17260 non-null  float64
 26  vehicle_manoeuvre_Turning                             17260 non-null  float64
 27  vehicle_manoeuvre_Waiting or Holding                  17260 non-null  float64
 28  sex_of_driver_Male                                    17260 non-null  float64
 29  casualty_class_Pedestrian/Passenger                   17260 non-null  float64
 30  sex_of_casualty_Male                                  17260 non-null  float64
 31  sex_of_casualty_unknown (self reported)               17260 non-null  float64
 32  casualty_severity_Slight                              17260 non-null  float64
 33  casualty_type_Goods vehicle/Bus/Other                 17260 non-null  float64
 34  casualty_type_Motorcycle                              17260 non-null  float64
 35  casualty_type_Pedestrian/Cyclist                      17260 non-null  float64
 36  age_of_driver_band_Elderly                            17260 non-null  float64
 37  age_of_driver_band_Mature                             17260 non-null  float64
 38  age_of_driver_band_Middle-aged                        17260 non-null  float64
 39  age_of_driver_band_Senior                             17260 non-null  float64
 40  age_of_driver_band_Underage/Teen                      17260 non-null  float64
 41  age_of_driver_band_Young Adult                        17260 non-null  float64
 42  age_of_vehicle_band_New                               17260 non-null  float64
 43  age_of_vehicle_band_Old                               17260 non-null  float64
 44  age_of_vehicle_band_Relatively New                    17260 non-null  float64
 45  age_of_vehicle_band_Very Old                          17260 non-null  float64
dtypes: float64(46)
memory usage: 6.1 MB
```

In [156]:
```
#check final Trainig data
Xtrain.info()
Xtest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40271 entries, 0 to 40270
Data columns (total 54 columns):
 #   Column                                                Non-Null Count  Dtype  
---  ------                                                --------------  -----  
 0   longitude                                             40271 non-null  float64
 1   latitude                                              40271 non-null  float64
 2   number_of_vehicles                                    40271 non-null  int64  
 3   number_of_casualties                                  40271 non-null  int64  
 4   speed_limit                                           40271 non-null  int64  
 5   month                                                 40271 non-null  int32  
 6   hour                                                  40271 non-null  int32  
 7   is_weekend                                            40271 non-null  int8   
 8   day_of_week_Monday                                    40271 non-null  float64
 9   day_of_week_Saturday                                  40271 non-null  float64
 10  day_of_week_Sunday                                    40271 non-null  float64
 11  day_of_week_Thursday                                  40271 non-null  float64
 12  day_of_week_Tuesday                                   40271 non-null  float64
 13  day_of_week_Wednesday                                 40271 non-null  float64
 14  road_type_Other                                       40271 non-null  float64
```

7.4 Handling Outliers for Numerical variables

Outliers are a major point of concern, as they may skew our analysis in wrong direction, but effecting the spread of out data, and we may get mislead about the centrality of the data or about the focal point of our analysis. So handling outliers is very necessary, and is mostly a logic based process. Here we will use variety of methods to handle them for numerical variables we have.

(longitude, latitude, number_of_vehicles, number_of_casualties, speed_limit, age_of_driver, age_of_vehicle, age_of_casualty)

Since speed limit has no outliers and is having descerte values in the form of (20, 30, 40, 50, 60, 70), we are leaving it as it is. Rest of them are as follows.

Using Isolation Forest Method for detecting outliers and removing them.

In [157]:
```
from sklearn.ensemble import IsolationForest
from sklearn.impute import SimpleImputer

# Standardize the numerical features
numerical_features = ['longitude', 'latitude', 'number_of_vehicles', 'number_of_casualties', 'speed_limit']

'''# Separate continuous and discrete numerical features
continuous_numerical_features = ['longitude', 'latitude']
discrete_numerical_features = ['number_of_vehicles', 'number_of_casualties', 'speed_limit', 'month', 'hour', 'is_weekend']''

# Apply outlier detection only on the numerical features
Xtrain_numerical = Xtrain[numerical_features]
Xtest_numerical = Xtest[numerical_features]

Xtrain_numerical.info()
Xtest_numerical.info()

# Initialize Isolation Forest
clf = IsolationForest(n_estimators=100, random_state=0, contamination=0.20)

# Fit on training data
clf.fit(Xtrain_numerical)

# Predict outliers in the training data
yhat_train = clf.predict(Xtrain_numerical)

# Filter training data based on the prediction
Xtrain_filtered = Xtrain[yhat_train == 1].reset_index(drop=True)
ytrain_filtered = ytrain[yhat_train == 1].reset_index(drop=True)

# Impute missing values for all numerical features
imputer = SimpleImputer(strategy='mean')
# Fit the imputer on the TRAINING data
imputer.fit(Xtrain_numerical)

# Transform BOTH training and test data using the fitted imputer
Xtrain_imputed = pd.DataFrame(imputer.transform(Xtrain_numerical), columns=Xtrain_numerical.columns)
Xtest_imputed = pd.DataFrame(imputer.transform(Xtest_numerical), columns=Xtest_numerical.columns)

# Predict outliers in the test data
yhat_test = clf.predict(Xtest_imputed)

# Filter test data based on the prediction
Xtest_filtered = Xtest[yhat_test == 1].reset_index(drop=True)
ytest_filtered = ytest[yhat_test == 1].reset_index(drop=True)

# Check the shape of the filtered datasets
print("Filtered Xtrain shape:", Xtrain_filtered.shape)
print("Filtered Xtest shape:", Xtest_filtered.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40271 entries, 0 to 40270
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   longitude            40271 non-null  float64
 1   latitude             40271 non-null  float64
 2   number_of_vehicles   40271 non-null  int64  
 3   number_of_casualties  40271 non-null  int64  
 4   speed_limit          40271 non-null  int64  
dtypes: float64(2), int64(3)
memory usage: 1.5 MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17260 entries, 0 to 17259
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   longitude            17260 non-null  float64
 1   latitude             17260 non-null  float64
 2   number_of_vehicles   17260 non-null  int64  
 3   number_of_casualties  17260 non-null  int64  
 4   speed_limit          17260 non-null  int64  
dtypes: float64(2), int64(3)
memory usage: 674.3 KB
Filtered Xtrain shape: (32217, 54)
Filtered Xtest shape: (13727, 54)
```

In [158]:
```
Xtrain_filtered.info()
Xtest_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32217 entries, 0 to 32216
Data columns (total 54 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   longitude                                             32217 non-null  float64
 1   latitude                                              32217 non-null  float64
 2   number_of_vehicles                                    32217 non-null  int64
 3   number_of_casualties                                  32217 non-null  int64
 4   speed_limit                                           32217 non-null  int64
 5   month                                                 32217 non-null  int32
 6   hour                                                  32217 non-null  int32
 7   is_weekend                                            32217 non-null  int8
 8   day_of_week_Monday                                    32217 non-null  float64
 9   day_of_week_Saturday                                  32217 non-null  float64
 10  day_of_week_Sunday                                    32217 non-null  float64
 11  day_of_week_Thursday                                  32217 non-null  float64
 12  day_of_week_Tuesday                                   32217 non-null  float64
 13  day_of_week_Wednesday                                 32217 non-null  float64
```

In [159]:
```
Xtrain_filtered.head()
```

Out[159]:

	longitude	latitude	number_of_vehicles	number_of_casualties	speed_limit	month	hour	is_weekend	day_of_week_Monday	day_of_week_Saturday
0	-2.153893	52.267966	1	1	30	12	16	1	0.0	1.0
1	-1.845597	53.859536	1	1	30	5	15	0	0.0	0.0
2	-2.644064	50.945872	2	1	30	8	20	0	1.0	0.0
3	-0.174342	51.499817	2	1	20	8	19	0	1.0	0.0
4	-0.449676	51.481362	2	1	40	12	8	0	0.0	0.0

5 rows × 54 columns

In [160]:
```
Xtest_filtered.head()
```

Out[160]:

	longitude	latitude	number_of_vehicles	number_of_casualties	speed_limit	month	hour	is_weekend	day_of_week_Monday	day_of_week_Saturday
0	-0.073378	51.517436	3	1	20	4	9	0	0.0	0.0
1	1.367290	52.657601	2	1	40	10	15	0	0.0	0.0
2	0.014591	51.446256	2	1	20	9	6	0	0.0	0.0
3	-0.060023	51.561352	2	1	20	8	19	0	0.0	0.0
4	-1.952994	53.603817	2	2	60	8	14	1	0.0	1.0

5 rows × 54 columns

7.5 Scaling and Standardization

In [161]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Fit the scaler on the training data and transform both train and test data
Xtrain_filtered[numerical_features] = scaler.fit_transform(Xtrain_filtered[numerical_features])
Xtest_filtered[numerical_features] = scaler.transform(Xtest_filtered[numerical_features])

# Add the target variable back to the datasets
Xtrain_filtered['accident_severity'] = ytrain_filtered.reset_index(drop=True)
Xtest_filtered['accident_severity'] = ytest_filtered.reset_index(drop=True)

Xtrain_filtered.info()
Xtest_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32217 entries, 0 to 32216
Data columns (total 55 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   longitude                                             32217 non-null  float64
 1   latitude                                              32217 non-null  float64
 2   number_of_vehicles                                    32217 non-null  float64
 3   number_of_casualties                                  32217 non-null  float64
 4   speed_limit                                           32217 non-null  float64
 5   month                                                 32217 non-null  int32
 6   hour                                                  32217 non-null  int32
 7   is_weekend                                            32217 non-null  int8
 8   day_of_week_Monday                                    32217 non-null  float64
 9   day_of_week_Saturday                                  32217 non-null  float64
 10  day_of_week_Sunday                                    32217 non-null  float64
 11  day_of_week_Thursday                                  32217 non-null  float64
 12  day_of_week_Tuesday                                   32217 non-null  float64
 13  day_of_week_Wednesday                                 32217 non-null  float64
```

8. Conclusion and Data Exporting

In [162]:
```
# Save the processed data

Xtrain_filtered.to_csv('Xtrain.csv', index=False)
Xtest_filtered.to_csv('Xtest.csv', index=False)
ytrain_filtered.to_csv('ytrain.csv', index=False)
ytest_filtered.to_csv('ytest.csv', index=False)
```

In [163]:
```
Xtrain_filtered.shape
```

Out[163]: (32217, 55)

In [164]:
```
ytrain_filtered.shape
```

Out[164]: (32217,)

In [165]:
```
Xtest_filtered.shape
```

Out[165]: (13727, 55)

In [166]:
```
ytest_filtered.shape
```

Out[166]: (13727,)

In [167]:
```
from IPython.display import FileLink

# Save the DataFrame to a CSV file
df.to_csv('Xtrain_filtered.csv', index=False)

# Create a download link
FileLink('Xtrain_filtered.csv')
```

Out[167]: Xtrain_filtered.csv (Xtrain_filtered.csv)

In [168]:
```
from IPython.display import FileLink

# Save the DataFrame to a CSV file
df.to_csv('ytrain_filtered.csv', index=False)

# Create a download link
FileLink('ytrain_filtered.csv')
```

Out[168]: ytrain_filtered.csv (ytrain_filtered.csv)

In [169]:
```
from IPython.display import FileLink

# Save the DataFrame to a CSV file
df.to_csv('Xtest_filtered.csv', index=False)

# Create a download link
FileLink('Xtest_filtered.csv')
```

Out[169]: Xtest_filtered.csv (Xtest_filtered.csv)

In [170]:
```
from IPython.display import FileLink

# Save the DataFrame to a CSV file
df.to_csv('ytest_filtered.csv', index=False)

# Create a download link
FileLink('ytest_filtered.csv')
```

Out[170]: ytest_filtered.csv (ytest_filtered.csv)

In [ ]: