1 BUSINESS UNDERSTANDING

1.1 MAIN OBJECTIVE

Predict the primary contributory cause of a car accident, given information about the car, the people in the car, the road and weather conditions.

1.1.1 Reason

1.1.1.1 Improving Road Safety – Identifying key factors that contribute to accidents can help traffic authorities take preventive measures.

1.1.1.2 Assisting Insurance and Law Enforcement – Automated accident cause prediction can assist in claims processing and accident investigations.

1.1.1.3 Enhancing Autonomous Vehicles – Understanding accident causes can help improve AI models for self-driving cars to avoid risky situations.

1.1.1.4 Data-Driven Decision Making – Analyzing accident data can lead to better infrastructure planning, traffic regulations, and awareness campaigns.

1.1.1.5 Reducing Human Error – Many accidents are due to driver mistakes; predicting causes can help design better driver-assist systems.

1.2 KEY BUSINESS QUESTIONS

1.2.1 What are the most common contributory causes of car accidents?

1.2.2 How do road and weather conditions impact the likelihood of different accident causes?

1.2.3 Do certain car models or vehicle types have a higher risk of specific accident causes?

1.2.4 Is blood alcohol content a major factor in accident occurence?

1.2.5 How do the above factors correlate with accident causes?

1.2.6 What machine learning techniques are most effective for accident cause prediction?

1.3 SUCCESS CRITERIA.

1.3.1 Identify which factors (car conditions, driver behavior, road, or weather) contribute most to accidents.

1.3.2 Evaluate different classification models to determine the most effective for predicting accident causes.

1.3.3 Ability to deploy the model for real-time accident cause predictions.

1.3.4 Analyze how accident causes vary by different factors.

1.4 Technologies To Be Applied

1. **Python (Jupyter Notebook)** - for data understanding and data cleaning.
2. **GitHub** - to store my work as an accessible repository

3. **MS-Powerpoint** - -This will be used to present my work

2 DATA UNDERSTANDING

2.1 DATA SOURCE

The data was obtained from Chicago Data Portal :https://data.cityofchicago.org/ (https://data.cityofchicago.org/). Data used are:

1. Traffic crashes- Crashes.
2. Traffic Crashes - Vehicles.
3. Traffic Crashes - People.

2.2 OVERVIEW

2.2.1 CRASH DATA

Crash data shows information about each traffic crash on city streets within the City of Chicago limits and under the jurisdiction of Chicago Police Department (CPD). Data are shown as is from the electronic crash reporting system (E-Crash) at CPD, excluding any personally identifiable information. Records are added to the data portal when a crash report is finalized or when amendments are made to an existing report in E-Crash. D

As per Illinois statute, only crashes with a property damage value of $1,500 or more or involving bodily injury to any person(s) and that happen on a public roadway and that involve at least one moving vehicle, except bike dooring, are considered reportable crashes. However, CPD records every reported traffic crash event, regardless of the statute of limitations, and hence any formal Chicago crash dataset released by Illinois Department of Transportation may not include all the crashes listed here.

The column names are:

1. **CRASH_RECORD_ID** This number can be used to link to the same crash in the Vehicles and People datasets. This number also serves as a unique ID in this dataset.
2. **FIRST_CRASH_TYPE** Type of first collision in crash.
3. **TRAFFICWAY_TYPE** Trafficway type, as determined by reporting officer.
4. **LANE_CNT** Total number of through lanes in either direction, excluding turn lanes, as determined by reporting officer (0 = intersection).
5. **ALIGNMENT** Street alignment at crash location, as determined by reporting officer.
6. **ROADWAY_SURFACE_COND** Road surface condition, as determined by reporting officer.
7. **CRASH_TYPE** A general severity classification for the crash. Can be either Injury and/or Tow Due to Crash or No Injury / Drive Away.
8. **INTERSECTION_RELATED_I** A field observation by the police officer whether an intersection played a role in the crash. Does not represent whether or not the crash occurred within the intersection.
9. **NOT_RIGHT_OF_WAY_I** Whether the crash begun or first contact was made outside of the public right-of-way.
10. **HIT_AND_RUN_I** Crash did/did not involve a driver who caused the crash and fled the scene without exchanging information and/or rendering aid.
11. **DAMAGE** A field observation of estimated damage.
12. **DATE_POLICE_NOTIFIED** Calendar date on which police were notified of the crash.
13. **PRIM_CONTRIBUTORY_CAUSE** The factor which was most significant in causing the crash, as determined by officer judgment
14. **SEC_CONTRIBUTORY_CAUSE** The factor which was second most significant in causing the crash, as determined by officer judgment.
15. **STREET_NO** Street address number of crash location, as determined by reporting officer.
16. **STREET_DIRECTION** Street address direction (N,E,S,W) of crash location, as determined by reporting officer.
17. **STREET_NAME** Street address name of crash location, as determined by reporting officer.
18. **BEAT_OF_OCCURRENCE** Chicago Police Department Beat ID. Boundaries available at https://data.cityofchicago.org/d/aerh-rz74 (https://data.cityofchicago.org/d/aerh-rz74).
19. **PHOTOS_TAKEN_I** Whether the Chicago Police Department took photos at the location of the crash.
20. **CRASH_DATE_EST_I** Crash date estimated by desk officer or reporting party (only used in cases where crash is reported at police station days after the crash).
21. **STATEMENTS_TAKEN_I** Whether statements were taken from unit(s) involved in crash.
22. **DOORING_I** Whether crash involved a motor vehicle occupant opening a door into the travel path of a bicyclist, causing a crash.
23. **WORK_ZONE_I** Whether the crash occurred in an active work zone.
24. **WORK_ZONE_TYPE** The type of work zone, if any.
25. **WORKERS_PRESENT_I** Whether construction workers were present in an active work zone at crash location.
26. **NUM_UNITS** Number of units involved in the crash. A unit can be a motor vehicle, a pedestrian, a bicyclist, or another non-passenger roadway user. Each unit represents a mode of traffic with an independent trajectory.
27. **MOST_SEVERE_INJURY** Most severe injury sustained by any person involved in the crash.
28. **INJURIES_TOTAL** Total persons sustaining fatal, incapacitating, non-incapacitating, and possible injuries as determined by the reporting officer.

29. **INJURIES_FATAL** Total persons sustaining fatal injuries in the crash.
30. **INJURIES_INCAPACITATING** Total persons sustaining incapacitating/serious injuries in the crash as determined by the reporting officer. Any injury other than fatal injury, which prevents the injured person from walking, driving, or normally continuing the activities they were capable of performing before the injury occurred. Includes severe lacerations, broken limbs, skull or chest injuries, and abdominal injuries.
31. **CRASH_DATE** Date and time of crash as entered by the reporting officer.
32. **INJURIES_NON_INCAPACITATING** Total persons sustaining non-incapacitating injuries in the crash as determined by the reporting officer. Any injury, other than fatal or incapacitating injury, which is evident to observers at the scene of the crash. Includes lump on head, abrasions, bruises, and minor lacerations.
33. **INJURIES_REPORTED_NOT_EVIDENT** Total persons sustaining possible injuries in the crash as determined by the reporting officer. Includes momentary unconsciousness, claims of injuries not evident, limping, complaint of pain, nausea, and hysteria.
34. **INJURIES_NO_INDICATION** Total persons sustaining no injuries in the crash as determined by the reporting officer.
35. **INJURIES_UNKNOWN** Total persons for whom injuries sustained, if any, are unknown.
36. **CRASH_HOUR** The hour of the day component of CRASH_DATE.
37. **CRASH_DAY_OF_WEEK** The day of the week component of CRASH_DATE. Sunday=1
38. **CRASH_MONTH** The month component of CRASH_DATE.
39. **LATITUDE** The latitude of the crash location, as determined by reporting officer, as derived from the reported address of crash.
40. **LONGITUDE** The longitude of the crash location, as determined by reporting officer, as derived from the reported address of crash.
41. **LOCATION** The crash location, as determined by reporting officer, as derived from the reported address of crash, in a column type that allows for mapping and other geographic analysis in the data portal software.
42. **POSTED_SPEED_LIMIT** Posted speed limit, as determined by reporting officer.
43. **TRAFFIC_CONTROL_DEVICE** Traffic control device present at crash location, as determined by reporting officer.
44. **DEVICE_CONDITION** Condition of traffic control device, as determined by reporting officer.
45. **WEATHER_CONDITION** Weather condition at time of crash, as determined by reporting officer.
46. **LIGHTING_CONDITION** Light condition at time of crash, as determined by reporting officer.
47. **ROAD_DEFECT** Road defects, as determined by reporting officer.
48. **REPORT_TYPE** Administrative report type (at scene, at desk, amended).

2.2.2 PEOPLE DATA

This data contains information about people involved in a crash and if any injuries were sustained. This dataset should be used in combination with the traffic Crash and Vehicle dataset. Each record corresponds to an occupant in a vehicle listed in the Crash dataset. Some people involved in a crash may not have been an occupant in a motor vehicle, but may have been a pedestrian, bicyclist, or using another non-motor vehicle mode of transportation.

The column names are:

1. **PERSON_ID** A unique identifier for each person record. IDs starting with P indicate passengers. IDs starting with O indicate a person who was not a passenger in the vehicle (e.g., driver, pedestrian, cyclist, etc.).
2. **PERSON_TYPE** Type of roadway user involved in crash
3. **CRASH_RECORD_ID** This number can be used to link to the same crash in the Crashes and Vehicles datasets. This number also serves as a unique ID in the Crashes dataset.
4. **VEHICLE_ID** The corresponding CRASH_UNIT_ID from the Vehicles dataset.
5. **CRASH_DATE** Date and time of crash as entered by the reporting officer
6. **SEAT_NO** Code for seating position of motor vehicle occupant: 1= driver, 2= center front, 3 = front passenger, 4 = second row left, 5 = second row center, 6 = second row right, 7 = enclosed passengers, 8 = exposed passengers, 9= unknown position, 10 = third row left, 11 = third row center, 12 = third row right
7. **CITY** City of residence of person involved in crash
8. **STATE** State of residence of person involved in crash
9. **ZIPCODE** ZIP Code of residence of person involved in crash
10. **SEX** Gender of person involved in crash, as determined by reporting officer
11. **AGE** Age of person involved in crash
12. **DRIVERS_LICENSE_STATE** State issuing driver's license of person involved in crash
13. **DRIVERS_LICENSE_CLASS** Class of driver's license of person involved in crash
14. **SAFETY_EQUIPMENT** Safety equipment used by vehicle occupant in crash, if any
15. **AIRBAG_DEPLOYED** Whether vehicle occupant airbag deployed as result of crash
16. **EJECTION** Whether vehicle occupant was ejected or extricated from the vehicle as a result of crash
17. **INJURY_CLASSIFICATION** Severity of injury person sustained in the crash
18. **HOSPITAL** Hospital to which person injured in the crash was taken
19. **EMS_AGENCY** EMS agency who transported person injured in crash to the hospital
20. **EMS_RUN_NO** EMS agency run number
21. **DRIVER_ACTION** Driver action that contributed to the crash, as determined by reporting officer
22. **DRIVER_VISION** What, if any, objects obscured the driver’s vision at time of crash
23. **PHYSICAL_CONDITION** Driver’s apparent physical condition at time of crash, as observed by the reporting officer
24. **PEDPEDAL_ACTION** Action of pedestrian or cyclist at the time of crash
25. **PEDPEDAL_VISIBILITY** Visibility of pedestrian of cyclist safety equipment in use at time of crash
26. **PEDPEDAL_LOCATION** Location of pedestrian or cyclist at the time of crash
27. **BAC_RESULT** Status of blood alcohol concentration testing for driver or other person involved in crash

28. **BAC_RESULT_VALUE** Driver's blood alcohol concentration test result (fatal crashes may include pedestrian or cyclist results)
29. **CELL_PHONE_USE** Whether person was/was not using cellphone at the time of the crash, as determined by the reporting officer

2.2.3 VEHICLES DATA

This dataset contains information about vehicles (or units as they are identified in crash reports) involved in a traffic crash. This dataset should be used in conjunction with the tCrash and People dataset. "Vehicle" information includes motor vehicle and non-motor vehicle modes of transportation, such as bicycles and pedestrians. Each mode of transportation involved in a crash is a "unit" and get one entry here. Each vehicle, each pedestrian, each motorcyclist, and each bicyclist is considered an independent unit that can have a trajectory separate from the other units.

The columns are:

1. **CRASH_UNIT_ID** A unique identifier for each vehicle record.
2. **CRASH_RECORD_ID** This number can be used to link to the same crash in the Crashes and People datasets. This number also serves as a unique ID in the Crashes dataset.
3. **CRASH_DATE** Date and time of crash as entered by the reporting officer
4. **UNIT_NO** A unique ID for each unit within a specific crash report.
5. **UNIT_TYPE** The type of unit.
6. **NUM_PASSENGERS** Number of passengers in the vehicle. The driver is not included. More information on passengers is in the People dataset.
7. **VEHICLE_ID** Unique vehicle identifier.
8. **CMRC_VEH_I** Chinese Machine Reading Comprehension.
9. **MAKE** The make (brand) of the vehicle, if relevant.
10. **MODEL** The model of the vehicle, if relevant.
11. **LIC_PLATE_STATE** The state issuing the license plate of the vehicle, if relevant.
12. **VEHICLE_YEAR** The model year of the vehicle, if relevant.
13. **VEHICLE_DEFECT** Defects prior to the accident.
14. **VEHICLE_TYPE** The type of vehicle, if relevant.
15. **VEHICLE_USE** The normal use of the vehicle, if relevant.
16. **TRAVEL_DIRECTION** The direction in which the unit was traveling prior to the crash, as determined by the reporting officer.
17. **MANEUVER** The action the unit was taking prior to the crash, as determined by the reporting officer.
18. **TOWED_I** Indicator of whether the vehicle was towed.
19. **FIRE_I** Was there a fire due to the accident.
20. **OCCUPANT_CNT** The number of people in the unit, as determined by the reporting officer.
21. **EXCEED_SPEED_LIMIT_I** Indicator of whether the unit was speeding, as determined by the reporting officer.
22. **TOWED_BY** Entity that towed the unit, if relevant.
23. **TOWED_TO** Location to which the unit was towed, if relevant.
24. **AREA_00_I** whether the accident occurred in a specific area or zone (designated as "Area 00")
25. **AREA_01_I** whether the accident occurred in a specific area or zone (designated as "Area 01")
26. **AREA_02_I** whether the accident occurred in a specific area or zone (designated as "Area 02")
27. **AREA_03_I** whether the accident occurred in a specific area or zone (designated as "Area 03")
28. **AREA_04_I** whether the accident occurred in a specific area or zone (designated as "Area 04")
29. **AREA_05_I** whether the accident occurred in a specific area or zone (designated as "Area 05")
30. **AREA_06_I** whether the accident occurred in a specific area or zone (designated as "Area 06")
31. **AREA_07_I** whether the accident occurred in a specific area or zone (designated as "Area 07")
32. **AREA_08_I** whether the accident occurred in a specific area or zone (designated as "Area 08")
33. **AREA_09_I** whether the accident occurred in a specific area or zone (designated as "Area 09")
34. **AREA_10_I** whether the accident occurred in a specific area or zone (designated as "Area 10")
35. **AREA_11_I** whether the accident occurred in a specific area or zone (designated as "Area 11")
36. **AREA_12_I** whether the accident occurred in a specific area or zone (designated as "Area 12")
37. **AREA_99_I** whether the accident occurred in a specific area or zone (designated as "Area 99")
38. **FIRST_CONTACT_POINT** The location on the vehicle that first made contact during the crash.
39. **CMV_ID** Commercial Motor Vehicle Identifier, which is used to track commercial vehicles involved in accidents.
40. **USDOT_NO** Refers to the United States Department of Transportation (USDOT) Number, which is a unique identifier assigned to commercial vehicles and trucking companies by the Federal Motor Carrier Safety Administration (FMCSA).
41. **CCMC_NO** Refers to a Commercial Carrier or Motor Carrier Number, which may be used to track licensed commercial vehicles in a specific state or jurisdiction.
42. **ILCC_NO** Stands for Illinois Commerce Commission (ILCC) Number.
43. **COMMERCIAL_SRC** Stands for Commercial Source and could indicate the source of commercial vehicle information in the crash report.
44. **GVWR** Represents the maximum allowable weight of a vehicle, including its own weight plus passengers, cargo, and fuel.
45. **CARRIER_NAME** Refers to the name of the commercial carrier or trucking company associated with a vehicle involved in the accident.
46. **CARRIER_STATE** Refers to the state where the commercial carrier (trucking company) is registered.
47. **CARRIER_CITY** Refers to the city where the commercial carrier (trucking company) is registered.
48. **HAZMAT_PLACARDS_I** Hazardous Materials placards.
49. **HAZMAT_NAME** Hazardous Materials name.
50. **UN_NO** Refers to the United Nations Number (UN Number) assigned to hazardous materials involved in an accident.

51. **HAZMAT_PRESENT_I** Hazardous Materials present.
52. **HAZMAT_REPORT_I** Signifies whether a hazardous materials (HAZMAT) report was filed for the accident.
53. **HAZMAT_REPORT_NO** Refers to the unique report number assigned to hazardous material (HAZMAT) incidents.
54. **MCS_REPORT_I** Motor Carrier Safety (MCS) report.
55. **MCS_REPORT_NO** Refers to the Motor Carrier Safety (MCS) report number assigned to commercial vehicle accidents.
56. **HAZMAT_VIO_CAUSE_CRASH_I** Whether a Hazardous Materials (HAZMAT) violation contributed to causing the crash.
57. **MCS_VIO_CAUSE_CRASH_I** Whether a Motor Carrier Safety (MCS) violation contributed to causing the crash.
58. **IDOT_PERMIT_NO** Refers to the Illinois Department of Transportation (IDOT) permit number issued for oversized, overweight, or special-use vehicles involved in a crash.
59. **WIDE_LOAD_I** Specifies whether a vehicle involved in the crash was classified as a wide load.
60. **TRAILER1_WIDTH** Refers to the width (in feet or inches) of the first trailer attached to a truck or commercial vehicle involved in a crash.
61. **TRAILER2_WIDTH** Refers to the width (in feet or inches) of the second trailer attached to a truck or commercial vehicle involved in a crash.
62. **TRAILER1_LENGTH** Refers to the length (in feet or inches) of the first trailer attached to a truck or commercial vehicle involved in a crash.
63. **TRAILER2_LENGTH** Refers to the length (in feet or inches) of the second trailer attached to a truck or commercial vehicle involved in a crash.
64. **TOTAL_VEHICLE_LENGTH** Represents the total length of a vehicle (or vehicle combination, including trailers) involved in a crash, measured in feet or inches.
65. **AXLE_CNT** Represents the number of axles on a vehicle involved in a crash.
66. **VEHICLE_CONFIG** Refers to the configuration or type of vehicle involved in a crash, particularly for commercial vehicles and trucks.
67. **CARGO_BODY_TYPE** Describes the type of cargo body on a commercial vehicle involved in a crash.
68. **LOAD_TYPE** Refers to the type of load a commercial vehicle was carrying at the time of the crash.
69. **HAZMAT_OUT_OF_SERVICE_I** Specifies whether a vehicle carrying hazardous materials (HAZMAT) was placed out of service due to safety violations at the time of the crash.
70. **MCS_OUT_OF_SERVICE_I** Specifies whether a Motor Carrier Safety (MCS) violation caused a vehicle to be placed out of service at the time of the crash.
71. **HAZMAT_CLASS** Refers to the hazard classification of dangerous goods being transported at the time of a crash.

2.3 DATA EXPLORATION

Looking into the data to understand the structure before cleaning and analysis.

2.3.1 IMPORTING REQUIRED LIBRARIES.

Python has many libraries which are used in undertking different tasks mostly statistical and mathematical operations. In this step **I** will be importing libraries required for this project.

In [1]:
```
#Import the different required libraries for this project.
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
import csv
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, StratifiedKFold
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc, accuracy_score, precision_scor
from xgboost import XGBClassifier, callback
from lightgbm import LGBMClassifier
```

2.3.2 LOADING THE DATASETS

Load the datasets to the notebook and convert them to a dataframe. This step is crucial for performing different operations required on the data.

In [2]:
```
# Loading  the data sets.

# Open the Crashes CSV file and load it into a DataFrame
with open("Traffic_Crashes_-_Crashes_20250217.csv", mode="r", encoding="utf-8") as file:
    accidents_df = pd.read_csv(file)

# Open the Vehicles CSV file and load it into a DataFrame
with open("Traffic_Crashes_-_Vehicles_20250217.csv", mode="r", encoding="utf-8") as file:
    vehicles_df = pd.read_csv(file)

# Open the People CSV file and load it into a DataFrame
with open("Traffic_Crashes_-_People_20250217.csv", mode="r", encoding="utf-8") as file:
    people_df = pd.read_csv(file)
```

```
C:\Users\Administrator\anaconda3\envs\learn-env\lib\site-packages\IPython\core\interactiveshell.py:3145: DtypeWarn
ing: Columns (20,39,40,41,43,47,48,49,52,54,57,58,60,70) have mixed types.Specify dtype option on import or set lo
w_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
C:\Users\Administrator\anaconda3\envs\learn-env\lib\site-packages\IPython\core\interactiveshell.py:3145: DtypeWarn
ing: Columns (19,28) have mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

2.3.2.1 View the first ten rows of the three datasets.

To get a general view of the data.

In [3]:
```
# Ensure all columns are displayed.
pd.set_option("display.max_columns", None)

# Return the first ten rows of accidents dataset.
accidents_df.head(10)
```

Out[3]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_
0	00a530520c84927490b597a6220ff3f2a3347472ef3086...	NaN	01/15/2025 01:50:00 PM	30	NO C
1	204c3ca48ff3cdf0737e7f10b54ef5f8c49fbe30b5e53e...	NaN	12/04/2024 05:39:00 PM	30	U
2	0173d218723a1d608d77b7142ca9b9a4afd7463e824162...	Y	09/23/2023 10:39:00 AM	30	TRAFF
3	26205ab4bcce4f6c3751a4947c94e5ad28b0dc0738cb27...	NaN	12/22/2019 08:11:00 PM	30	TRAFF
4	262653b489eadfd76fb48900ea9a48717357c1fdc9dbb8...	NaN	07/01/2023 02:00:00 PM	30	TRAFF
5	0235f4b4460f50115c7f6cb0d2bedeea7ce92b9b0f5943...	NaN	01/15/2025 06:40:00 PM	25	TRAFF
6	019686d77a8168054af5b142442f775929fc1d9e100b36...	NaN	08/09/2019 09:10:00 PM	30	NO C
7	01c36d50980b25ab3086fe56ddebe44aad49e0d51355ff...	NaN	08/21/2023 07:48:00 PM	25	NO C
8	262a5cf2fc5f98193c94d52b5b094b1d3ec8d2199b57d5...	NaN	12/24/2019 08:20:00 PM	30	TRAFF
9	1b6c82c982bb0d01618cad1f4703403371cf19eb568dd2...	NaN	11/16/2024 08:50:00 PM	30	TRAFF

In [4]:
```
# Return first ten rows of vehicles dataset.
vehicles_df.head(10)
```

Out[4]:

	CRASH_UNIT_ID	CRASH_RECORD_ID	CRASH_DATE	UNIT_NO	UNIT_TYPE	NUM_PASSENGERS
0	10	2e31858c0e411f0bdcb337fb7c415aa93763cf2f23e02f...	08/04/2015 12:40:00 PM	1	DRIVER	NaN
1	100	e73b35bd7651b0c6693162bee0666db159b28901437009...	07/31/2015 05:50:00 PM	1	DRIVER	NaN
2	1000	f2b1adeb85a15112e4fb7db74bff440d6ca53ff7a21e10...	09/02/2015 11:45:00 AM	1	DRIVER	NaN
3	10000	15a3e24fce3ce7cd2b02d44013d1a93ff2fbdca80632ec...	10/31/2015 09:30:00 PM	2	DRIVER	NaN
4	100000	1d3c178880366c77deaf06b8c3198429112a1c8e8807ed...	11/16/2016 01:00:00 PM	2	PARKED	NaN
5	1000001	57945047eb951f4152fde013708fa029caf5d33fc2a2c2...	11/25/2020 05:46:00 PM	1	DRIVER	NaN
6	1000002	57945047eb951f4152fde013708fa029caf5d33fc2a2c2...	11/25/2020 05:46:00 PM	2	DRIVER	1.0
7	1000003	7cf1e0da7413f35c258ad67a3f0d08b10c9cf94a59bb4e...	11/25/2020 06:39:00 PM	1	DRIVER	NaN
8	1000004	7cf1e0da7413f35c258ad67a3f0d08b10c9cf94a59bb4e...	11/25/2020 06:39:00 PM	2	PARKED	NaN
9	1000005	7cf1e0da7413f35c258ad67a3f0d08b10c9cf94a59bb4e...	11/25/2020 06:39:00 PM	3	DRIVER	NaN

In [5]:
```
# Return the first ten rows of peple dataset
people_df.head(10)
```

Out[5]:

	PERSON_ID	PERSON_TYPE	CRASH_RECORD_ID	VEHICLE_ID	CRASH_DATE	SEAT_NO	CITY
0	O749947	DRIVER	81dc0de2ed92aa62baccab641fa377be7feb1cc47e6554...	834816.0	09/28/2019 03:30:00 AM	NaN	CHICAG
1	O871921	DRIVER	af84fb5c8d996fcd3aefd36593c3a02e6e7509eeb27568...	827212.0	04/13/2020 10:50:00 PM	NaN	CHICAG
2	O10018	DRIVER	71162af7bf22799b776547132ebf134b5b438dcf3dac6b...	9579.0	11/01/2015 05:00:00 AM	NaN	Na
3	O10038	DRIVER	c21c476e2ccc41af550b5d858d22aaac4ffc88745a1700...	9598.0	11/01/2015 08:00:00 AM	NaN	Na
4	O10039	DRIVER	eb390a4c8e114c69488f5fb8a097fe629f5a92fd528cf4...	9600.0	11/01/2015 10:15:00 AM	NaN	Na
5	O10041	DRIVER	dd1bce4bd6d0be4c247714dcabab44e6563c62b913229b...	9601.0	11/01/2015 11:00:00 AM	NaN	Na
6	O10062	DRIVER	4bd2ee6bb306902b99a9c2ae55cf4fcffec00879e39759...	9621.0	11/01/2015 12:30:00 PM	NaN	Na
7	O10066	DRIVER	9c03b6fcc6d72cf3ee2cb9ea754ea7e4617ab965142552...	9623.0	10/31/2015 04:00:00 PM	NaN	Na
8	O1007	DRIVER	2e7e0c1682100a200a46c25a532eda4febb00a2ae49ea9...	958.0	09/02/2015 02:35:00 PM	NaN	Na
9	O10088	DRIVER	8bf8069b3d839b732fd35d2c9d8caddeddbb4b8978e84d...	9645.0	11/01/2015 01:00:00 PM	NaN	Na

Check number of rows and columns

To get a view of the number of entries and features in the dataset and if the data is sufficient for analysis and modeling.

In [6]:
```
# Check number of rows and columns in the dataset.
print(f"Accidents Dataframe: {accidents_df.shape}")
print(f"Vehicles Dataframe: {vehicles_df.shape}")
print(f"People Dataframe: {people_df.shape}")
```

```
Accidents Dataframe: (919074, 48)
Vehicles Dataframe: (1874570, 71)
People Dataframe: (2017954, 29)
```

2.3.2.2 Check the columns in the dataset

In [7]:
```
# Print the columns of the accidents dataframe.
print(accidents_df.columns)

# Print the columns of the vehicles dataframe.
print(vehicles_df.columns)

# Print the columns of the people dataframe.
print(people_df.columns)
```

```
Index(['CRASH_RECORD_ID', 'CRASH_DATE_EST_I', 'CRASH_DATE',
       'POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE', 'LANE_CNT', 'ALIGNMENT', 'ROADWAY_SURFACE_COND',
       'ROAD_DEFECT', 'REPORT_TYPE', 'CRASH_TYPE', 'INTERSECTION_RELATED_I',
       'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'DAMAGE', 'DATE_POLICE_NOTIFIED',
       'PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO',
       'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRENCE',
       'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'DOORING_I', 'WORK_ZONE_I',
       'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I', 'NUM_UNITS',
       'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_FATAL',
       'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
       'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION',
       'INJURIES_UNKNOWN', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
       'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
Index(['CRASH_UNIT_ID', 'CRASH_RECORD_ID', 'CRASH_DATE', 'UNIT_NO',
       'UNIT_TYPE', 'NUM_PASSENGERS', 'VEHICLE_ID', 'CMRC_VEH_I', 'MAKE',
       'MODEL', 'LIC_PLATE_STATE', 'VEHICLE_YEAR', 'VEHICLE_DEFECT',
       'VEHICLE_TYPE', 'VEHICLE_USE', 'TRAVEL_DIRECTION', 'MANEUVER',
       'TOWED_I', 'FIRE_I', 'OCCUPANT_CNT', 'EXCEED_SPEED_LIMIT_I', 'TOWED_BY',
       'TOWED_TO', 'AREA_00_I', 'AREA_01_I', 'AREA_02_I', 'AREA_03_I',
       'AREA_04_I', 'AREA_05_I', 'AREA_06_I', 'AREA_07_I', 'AREA_08_I',
       'AREA_09_I', 'AREA_10_I', 'AREA_11_I', 'AREA_12_I', 'AREA_99_I',
       'FIRST_CONTACT_POINT', 'CMV_ID', 'USDOT_NO', 'CCMC_NO', 'ILCC_NO',
       'COMMERCIAL_SRC', 'GVWR', 'CARRIER_NAME', 'CARRIER_STATE',
       'CARRIER_CITY', 'HAZMAT_PLACARDS_I', 'HAZMAT_NAME', 'UN_NO',
       'HAZMAT_PRESENT_I', 'HAZMAT_REPORT_I', 'HAZMAT_REPORT_NO',
       'MCS_REPORT_I', 'MCS_REPORT_NO', 'HAZMAT_VIO_CAUSE_CRASH_I',
       'MCS_VIO_CAUSE_CRASH_I', 'IDOT_PERMIT_NO', 'WIDE_LOAD_I',
       'TRAILER1_WIDTH', 'TRAILER2_WIDTH', 'TRAILER1_LENGTH',
       'TRAILER2_LENGTH', 'TOTAL_VEHICLE_LENGTH', 'AXLE_CNT', 'VEHICLE_CONFIG',
       'CARGO_BODY_TYPE', 'LOAD_TYPE', 'HAZMAT_OUT_OF_SERVICE_I',
       'MCS_OUT_OF_SERVICE_I', 'HAZMAT_CLASS'],
      dtype='object')
Index(['PERSON_ID', 'PERSON_TYPE', 'CRASH_RECORD_ID', 'VEHICLE_ID',
       'CRASH_DATE', 'SEAT_NO', 'CITY', 'STATE', 'ZIPCODE', 'SEX', 'AGE',
       'DRIVERS_LICENSE_STATE', 'DRIVERS_LICENSE_CLASS', 'SAFETY_EQUIPMENT',
       'AIRBAG_DEPLOYED', 'EJECTION', 'INJURY_CLASSIFICATION', 'HOSPITAL',
       'EMS_AGENCY', 'EMS_RUN_NO', 'DRIVER_ACTION', 'DRIVER_VISION',
       'PHYSICAL_CONDITION', 'PEDPEDAL_ACTION', 'PEDPEDAL_VISIBILITY',
       'PEDPEDAL_LOCATION', 'BAC_RESULT', 'BAC_RESULT VALUE',
       'CELL_PHONE_USE'],
      dtype='object')
```

2.3.2.3 Check the details of the data in each column

In [8]:
```
#Check the details of the data in each column of the accidents dataframe.
accidents_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 919074 entries, 0 to 919073
Data columns (total 48 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   CRASH_RECORD_ID                 919074 non-null  object
 1   CRASH_DATE_EST_I                67696 non-null   object
 2   CRASH_DATE                      919074 non-null  object
 3   POSTED_SPEED_LIMIT              919074 non-null  int64
 4   TRAFFIC_CONTROL_DEVICE          919074 non-null  object
 5   DEVICE_CONDITION                919074 non-null  object
 6   WEATHER_CONDITION               919074 non-null  object
 7   LIGHTING_CONDITION              919074 non-null  object
 8   FIRST_CRASH_TYPE                919074 non-null  object
 9   TRAFFICWAY_TYPE                 919074 non-null  object
 10  LANE_CNT                        199023 non-null  float64
 11  ALIGNMENT                       919074 non-null  object
 12  ROADWAY_SURFACE_COND            919074 non-null  object
 13  ROAD_DEFECT                     919074 non-null  object
 14  REPORT_TYPE                     890132 non-null  object
 15  CRASH_TYPE                      919074 non-null  object
 16  INTERSECTION_RELATED_I          211256 non-null  object
 17  NOT_RIGHT_OF_WAY_I              41746 non-null   object
 18  HIT_AND_RUN_I                   288212 non-null  object
 19  DAMAGE                          919074 non-null  object
 20  DATE_POLICE_NOTIFIED            919074 non-null  object
 21  PRIM_CONTRIBUTORY_CAUSE         919074 non-null  object
 22  SEC_CONTRIBUTORY_CAUSE          919074 non-null  object
 23  STREET_NO                       919074 non-null  int64
 24  STREET_DIRECTION                919070 non-null  object
 25  STREET_NAME                     919073 non-null  object
 26  BEAT_OF_OCCURRENCE              919069 non-null  float64
 27  PHOTOS_TAKEN_I                  12654 non-null   object
 28  STATEMENTS_TAKEN_I              21306 non-null   object
 29  DOORING_I                       2878 non-null    object
 30  WORK_ZONE_I                     5083 non-null    object
 31  WORK_ZONE_TYPE                  3917 non-null    object
 32  WORKERS_PRESENT_I               1307 non-null    object
 33  NUM_UNITS                       919074 non-null  int64
 34  MOST_SEVERE_INJURY              917055 non-null  object
 35  INJURIES_TOTAL                  917069 non-null  float64
 36  INJURIES_FATAL                  917069 non-null  float64
 37  INJURIES_INCAPACITATING         917069 non-null  float64
 38  INJURIES_NON_INCAPACITATING     917069 non-null  float64
 39  INJURIES_REPORTED_NOT_EVIDENT   917069 non-null  float64
 40  INJURIES_NO_INDICATION          917069 non-null  float64
 41  INJURIES_UNKNOWN                917069 non-null  float64
 42  CRASH_HOUR                      919074 non-null  int64
 43  CRASH_DAY_OF_WEEK               919074 non-null  int64
 44  CRASH_MONTH                     919074 non-null  int64
 45  LATITUDE                        912344 non-null  float64
 46  LONGITUDE                       912344 non-null  float64
 47  LOCATION                        912344 non-null  object
dtypes: float64(11), int64(6), object(31)
memory usage: 336.6+ MB
```

In [9]:
```
#Check the details of the data in each column of the vehicles dataframe.
vehicles_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1874570 entries, 0 to 1874569
Data columns (total 71 columns):
 #   Column                     Dtype
---  ------                     -----
 0   CRASH_UNIT_ID              int64
 1   CRASH_RECORD_ID            object
 2   CRASH_DATE                 object
 3   UNIT_NO                    int64
 4   UNIT_TYPE                  object
 5   NUM_PASSENGERS             float64
 6   VEHICLE_ID                 float64
 7   CMRC_VEH_I                 object
 8   MAKE                       object
 9   MODEL                      object
 10  LIC_PLATE_STATE            object
 11  VEHICLE_YEAR               float64
 12  VEHICLE_DEFECT             object
 13  VEHICLE_TYPE               object
 14  VEHICLE_USE                object
 15  TRAVEL_DIRECTION           object
 16  MANEUVER                   object
 17  TOWED_I                    object
 18  FIRE_I                     object
 19  OCCUPANT_CNT               float64
 20  EXCEED_SPEED_LIMIT_I       object
 21  TOWED_BY                   object
 22  TOWED_TO                   object
 23  AREA_00_I                  object
 24  AREA_01_I                  object
 25  AREA_02_I                  object
 26  AREA_03_I                  object
 27  AREA_04_I                  object
 28  AREA_05_I                  object
 29  AREA_06_I                  object
 30  AREA_07_I                  object
 31  AREA_08_I                  object
 32  AREA_09_I                  object
 33  AREA_10_I                  object
 34  AREA_11_I                  object
 35  AREA_12_I                  object
 36  AREA_99_I                  object
 37  FIRST_CONTACT_POINT        object
 38  CMV_ID                     float64
 39  USDOT_NO                   object
 40  CCMC_NO                    object
 41  ILCC_NO                    object
 42  COMMERCIAL_SRC             object
 43  GVWR                       object
 44  CARRIER_NAME               object
 45  CARRIER_STATE              object
 46  CARRIER_CITY               object
 47  HAZMAT_PLACARDS_I          object
 48  HAZMAT_NAME                object
 49  UN_NO                      object
 50  HAZMAT_PRESENT_I           object
 51  HAZMAT_REPORT_I            object
 52  HAZMAT_REPORT_NO           object
 53  MCS_REPORT_I               object
 54  MCS_REPORT_NO              object
 55  HAZMAT_VIO_CAUSE_CRASH_I   object
 56  MCS_VIO_CAUSE_CRASH_I      object
 57  IDOT_PERMIT_NO             object
 58  WIDE_LOAD_I                object
 59  TRAILER1_WIDTH             object
 60  TRAILER2_WIDTH             object
 61  TRAILER1_LENGTH            float64
 62  TRAILER2_LENGTH            float64
 63  TOTAL_VEHICLE_LENGTH       float64
 64  AXLE_CNT                   float64
 65  VEHICLE_CONFIG             object
 66  CARGO_BODY_TYPE            object
 67  LOAD_TYPE                  object
 68  HAZMAT_OUT_OF_SERVICE_I    object
 69  MCS_OUT_OF_SERVICE_I       object
 70  HAZMAT_CLASS               object
```

```
dtypes: float64(9), int64(2), object(60)
memory usage: 1015.4+ MB
```

In [10]:
```
#Check the details of the data in each column of the people dataframe.
people_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2017954 entries, 0 to 2017953
Data columns (total 29 columns):
 #   Column                 Dtype
---  ------                 -----
 0   PERSON_ID              object
 1   PERSON_TYPE            object
 2   CRASH_RECORD_ID        object
 3   VEHICLE_ID             float64
 4   CRASH_DATE             object
 5   SEAT_NO                float64
 6   CITY                   object
 7   STATE                  object
 8   ZIPCODE                object
 9   SEX                    object
 10  AGE                    float64
 11  DRIVERS_LICENSE_STATE  object
 12  DRIVERS_LICENSE_CLASS  object
 13  SAFETY_EQUIPMENT       object
 14  AIRBAG_DEPLOYED        object
 15  EJECTION               object
 16  INJURY_CLASSIFICATION  object
 17  HOSPITAL               object
 18  EMS_AGENCY             object
 19  EMS_RUN_NO             object
 20  DRIVER_ACTION          object
 21  DRIVER_VISION          object
 22  PHYSICAL_CONDITION     object
 23  PEDPEDAL_ACTION        object
 24  PEDPEDAL_VISIBILITY    object
 25  PEDPEDAL_LOCATION      object
 26  BAC_RESULT             object
 27  BAC_RESULT VALUE       float64
 28  CELL_PHONE_USE         object
dtypes: float64(4), object(25)
memory usage: 446.5+ MB
```

2.3.2.4 Check statistical measures of the columns datasets

Returns the count, mean, standard deviation, minimum value, the quartiles and the maximum value.

Numerical columns

In [11]:
```
# Check the statistical measures of the numerical columns in the accidents dataset.
accidents_df.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
POSTED_SPEED_LIMIT	919074.0	28.422494	6.094825	0.000000	30.000000	30.00000	30.000000	9.900000e+01
LANE_CNT	199023.0	13.329369	2961.496986	0.000000	2.000000	2.00000	4.000000	1.191625e+06
STREET_NO	919074.0	3690.001741	2878.789151	0.000000	1256.000000	3201.00000	5562.000000	4.511000e+05
BEAT_OF_OCCURRENCE	919069.0	1245.968037	704.738584	111.000000	715.000000	1212.00000	1822.000000	6.100000e+03
NUM_UNITS	919074.0	2.035084	0.451578	1.000000	2.000000	2.00000	2.000000	1.800000e+01
INJURIES_TOTAL	917069.0	0.194600	0.572706	0.000000	0.000000	0.00000	0.000000	2.100000e+01
INJURIES_FATAL	917069.0	0.001182	0.037253	0.000000	0.000000	0.00000	0.000000	4.000000e+00
INJURIES_INCAPACITATING	917069.0	0.019572	0.163874	0.000000	0.000000	0.00000	0.000000	1.000000e+01
INJURIES_NON_INCAPACITATING	917069.0	0.108791	0.425041	0.000000	0.000000	0.00000	0.000000	2.100000e+01
INJURIES_REPORTED_NOT_EVIDENT	917069.0	0.065055	0.327762	0.000000	0.000000	0.00000	0.000000	1.500000e+01
INJURIES_NO_INDICATION	917069.0	1.999925	1.156199	0.000000	1.000000	2.00000	2.000000	6.100000e+01
INJURIES_UNKNOWN	917069.0	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000e+00
CRASH_HOUR	919074.0	13.194152	5.574052	0.000000	9.000000	14.00000	17.000000	2.300000e+01
CRASH_DAY_OF_WEEK	919074.0	4.121909	1.979492	1.000000	2.000000	4.00000	6.000000	7.000000e+00
CRASH_MONTH	919074.0	6.695867	3.432188	1.000000	4.000000	7.00000	10.000000	1.200000e+01
LATITUDE	912344.0	41.855123	0.339058	0.000000	41.783317	41.87497	41.924602	4.202278e+01
LONGITUDE	912344.0	-87.673606	0.689437	-87.936193	-87.721851	-87.67433	-87.633678	0.000000e+00

In [12]:
```
# Check the statistical measures of the numerical columns in the vehicles dataset.
vehicles_df.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
CRASH_UNIT_ID	1874570.0	1.002621e+06	580227.256063	2.0	499381.25	1004238.5	1505643.75	2006773.0
UNIT_NO	1874570.0	3.580253e+00	2759.402017	0.0	1.00	2.0	2.00	3778035.0
NUM_PASSENGERS	277422.0	1.469746e+00	1.060403	1.0	1.00	1.0	2.00	59.0
VEHICLE_ID	1830743.0	9.535227e+05	551137.177757	2.0	476558.50	951730.0	1430398.50	1912460.0
VEHICLE_YEAR	1540494.0	2.014310e+03	135.926212	1900.0	2007.00	2013.0	2017.00	9999.0
OCCUPANT_CNT	1830743.0	1.079227e+00	0.781517	0.0	1.00	1.0	1.00	99.0
CMV_ID	18857.0	1.051360e+04	6073.276035	1.0	5183.00	10558.0	15782.00	20954.0
TRAILER1_LENGTH	2449.0	4.847244e+01	20.537571	1.0	45.00	53.0	53.00	740.0
TRAILER2_LENGTH	70.0	4.427143e+01	28.008240	1.0	24.25	50.0	53.00	123.0
TOTAL_VEHICLE_LENGTH	2987.0	5.321828e+01	31.730950	1.0	35.00	53.0	66.00	999.0
AXLE_CNT	4523.0	9.449480e+00	386.687552	1.0	2.00	3.0	5.00	26009.0

In [13]:
```
# Check the statistical measures of the numerical columns in the people dataset.
people_df.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
VEHICLE_ID	1976560.0	949940.890911	552469.374092	2.0	470890.75	942376.00	1431140.25	1912446.0
SEAT_NO	407740.0	4.164929	2.219211	1.0	3.00	3.00	6.00	12.0
AGE	1430970.0	37.931624	17.083221	-177.0	25.00	35.00	50.00	110.0
BAC_RESULT VALUE	2227.0	0.171311	0.103090	0.0	0.13	0.17	0.22	1.0

Categorical Columns

In [14]:
```
# Check the statistical measures of the categorical columns in the accidents dataset.
accidents_df.describe(include="O").T
```

Out[14]:

	count	unique	top	freq
CRASH_RECORD_ID	919074	919074	886cb19ca8f2a0aa6f4ee83653e40d0a4f725467291647...	1
CRASH_DATE_EST_I	67696	2	Y	59024
CRASH_DATE	919074	605232	02/06/2025 08:00:00 AM	36
TRAFFIC_CONTROL_DEVICE	919074	19	NO CONTROLS	519688
DEVICE_CONDITION	919074	8	NO CONTROLS	525927
WEATHER_CONDITION	919074	12	CLEAR	720777
LIGHTING_CONDITION	919074	6	DAYLIGHT	587729
FIRST_CRASH_TYPE	919074	18	PARKED MOTOR VEHICLE	212763
TRAFFICWAY_TYPE	919074	20	NOT DIVIDED	395093
ALIGNMENT	919074	6	STRAIGHT AND LEVEL	897315
ROADWAY_SURFACE_COND	919074	7	DRY	676829
ROAD_DEFECT	919074	7	NO DEFECTS	729926
REPORT_TYPE	890132	3	NOT ON SCENE (DESK REPORT)	499399
CRASH_TYPE	919074	2	NO INJURY / DRIVE AWAY	671391
INTERSECTION_RELATED_I	211256	2	Y	201172
NOT_RIGHT_OF_WAY_I	41746	2	Y	37918
HIT_AND_RUN_I	288212	2	Y	275875
DAMAGE	919074	3	OVER $1,500	580685
DATE_POLICE_NOTIFIED	919074	696379	10/12/2020 04:00:00 PM	12
PRIM_CONTRIBUTORY_CAUSE	919074	40	UNABLE TO DETERMINE	359789
SEC_CONTRIBUTORY_CAUSE	919074	40	NOT APPLICABLE	379217
STREET_DIRECTION	919070	4	W	329216
STREET_NAME	919073	1654	WESTERN AVE	25140
PHOTOS_TAKEN_I	12654	2	Y	9527
STATEMENTS_TAKEN_I	21306	2	Y	17446
DOORING_I	2878	2	Y	1930
WORK_ZONE_I	5083	2	Y	3917
WORK_ZONE_TYPE	3917	4	CONSTRUCTION	2715
WORKERS_PRESENT_I	1307	2	Y	1160
MOST_SEVERE_INJURY	917055	5	NO INDICATION OF INJURY	787697
LOCATION	912344	323732	POINT (-87.905309125103 41.976201139024)	1469

In [15]:
```
# Check the statistical measures of the categorical columns in the vehicles dataset.
vehicles_df.describe(include="O").T
```

Out[15]:

	count	unique	top	freq
CRASH_RECORD_ID	1874570	919074	376a09193852b2ec09fdbad0da0b82943e932ce95372c9...	18
CRASH_DATE	1874570	605258	02/06/2025 08:00:00 AM	72
UNIT_TYPE	1872290	9	DRIVER	1569342
CMRC_VEH_I	34849	2	Y	21894
MAKE	1830738	1399	CHEVROLET	210820
MODEL	1830593	2797	UNKNOWN	180803
LIC_PLATE_STATE	1663216	52	IL	1524505
VEHICLE_DEFECT	1830743	17	NONE	974751
VEHICLE_TYPE	1830743	22	PASSENGER	1147868
VEHICLE_USE	1830743	25	PERSONAL	1181893
TRAVEL_DIRECTION	1830743	9	N	429955
MANEUVER	1830743	28	STRAIGHT AHEAD	848593
TOWED_I	232969	2	Y	220313
FIRE_I	1538	2	Y	842
EXCEED_SPEED_LIMIT_I	2402	2	Y	1802
TOWED_BY	176584	15985	PRIVATE TOW	18145
TOWED_TO	108264	16803	UNKNOWN	15433
AREA_00_I	63325	2	Y	55894
AREA_01_I	504182	2	Y	481497
AREA_02_I	295187	2	Y	281068
AREA_03_I	180549	2	Y	172284
AREA_04_I	178456	2	Y	170451
AREA_05_I	277737	2	Y	266666
AREA_06_I	293799	2	Y	282502
AREA_07_I	276713	2	Y	264966
AREA_08_I	259353	2	Y	246170
AREA_09_I	134407	2	Y	127953
AREA_10_I	196932	2	Y	186965
AREA_11_I	377813	2	Y	356100
AREA_12_I	366712	2	Y	351838
AREA_99_I	213623	2	Y	205687
FIRST_CONTACT_POINT	1827580	23	FRONT	359777
USDOT_NO	10437	5588	0080806	242
CCMC_NO	2177	1664	UNK	185
ILCC_NO	1486	1030	UNK	190
COMMERCIAL_SRC	12303	4	SIDE OF TRUCK	5051
GVWR	10343	481	3	3114
CARRIER_NAME	18013	11333	CHICAGO TRANSIT AUTHORITY	726
CARRIER_STATE	16884	51	IL	10826
CARRIER_CITY	16572	2159	CHICAGO	4490
HAZMAT_PLACARDS_I	364	2	N	247
HAZMAT_NAME	64	39	FLAMMABLE	7
UN_NO	625	460	UNK	81
HAZMAT_PRESENT_I	13782	3	N	12612
HAZMAT_REPORT_I	13419	3	N	11449

	count	unique	top	freq
HAZMAT_REPORT_NO	1	1	JE106328	1
MCS_REPORT_I	13465	3	N	10589
MCS_REPORT_NO	7	7	JF	1
HAZMAT_VIO_CAUSE_CRASH_I	13593	3	N	12223
MCS_VIO_CAUSE_CRASH_I	13374	3	N	10893
IDOT_PERMIT_NO	1003	685	UNK	128
WIDE_LOAD_I	169	2	Y	101
TRAILER1_WIDTH	3058	3	0-96	1824
TRAILER2_WIDTH	371	3	0-96	203
VEHICLE_CONFIG	15602	8	TRACTOR/SEMI-TRAILER	6026
CARGO_BODY_TYPE	14918	9	VAN/ENCLOSED BOX	6336
LOAD_TYPE	14254	6	OTHER	7752
HAZMAT_OUT_OF_SERVICE_I	13043	2	N	13025
MCS_OUT_OF_SERVICE_I	13281	2	N	13229
HAZMAT_CLASS	1193	8	MISCELLANEOUS	1050

In [16]:
```
# Check the statistical measures of the categorical columns in the people dataset.
people_df.describe(include="O").T
```

Out[16]:

	count	unique	top	freq
PERSON_ID	2017954	2017954	O984309	1
PERSON_TYPE	2017954	6	DRIVER	1569342
CRASH_RECORD_ID	2017954	917069	31ecf6862c691ff12d3856213b902c146b07337b42a569...	61
CRASH_DATE	2017954	604160	12/29/2020 05:00:00 PM	72
CITY	1468809	14251	CHICAGO	1041777
STATE	1491186	52	IL	1407711
ZIPCODE	1352050	15182	60629	50446
SEX	1983854	3	M	1043210
DRIVERS_LICENSE_STATE	1181825	213	IL	1081120
DRIVERS_LICENSE_CLASS	981458	292	D	853823
SAFETY_EQUIPMENT	2012311	19	USAGE UNKNOWN	962191
AIRBAG_DEPLOYED	1978117	7	DID NOT DEPLOY	1001421
EJECTION	1992494	5	NONE	1855100
INJURY_CLASSIFICATION	2017195	5	NO INDICATION OF INJURY	1837913
HOSPITAL	325041	7824	REFUSED	106670
EMS_AGENCY	200590	8590	CFD	42225
EMS_RUN_NO	33538	1543	DNA	4800
DRIVER_ACTION	1606605	20	NONE	571337
DRIVER_VISION	1605956	14	NOT OBSCURED	796352
PHYSICAL_CONDITION	1607704	12	NORMAL	1039144
PEDPEDAL_ACTION	39572	23	CROSSING - WITH SIGNAL	8060
PEDPEDAL_VISIBILITY	39502	4	NO CONTRASTING CLOTHING	31168
PEDPEDAL_LOCATION	39573	8	IN ROADWAY	17096
BAC_RESULT	1607523	4	TEST NOT OFFERED	1584484
CELL_PHONE_USE	1160	2	Y	753

Gives a general view of what is going on in each column and the nature of the data in the columns. It is a crucial step in making the choice of which columns are appropriate for analysis and how to handle the data cleaning process.

2.4 SUMMARY OF THE DATASETS

2.4.1 The datasets have the following number of features and entries.

1. Accidents Dataframe: (Entries: 919074, Features: 48)
2. Vehicles Dataframe: (Entries: 1874570, Features: 71)
3. People Dataframe: (Entries: 2017954, Features: 29)

2.4.2 The datasets have sufficient entries and features for modeling and analysis.

2.4.3 The datasets have a similar column (CRASH_RECORD_ID) that can be used for merging.

2.4.4 The datasets will require quite a huge amount of cleaning for proper analysis and modeling.

3 DATA PREPARATION

Data preparation is a crucial step in any data analysis or machine learning project. It involves cleaning, transforming, and organizing raw data into a usable format for analysis or model building. Here are the key steps involved in data preparation:

1. Data Integration. (Merging)
2. Data Cleaning. (Removing unnecessary and personal identifying columns, Handling Missing Data, Removing Duplicates, Outlier Detection, Correcting Errors)
3. Data Transformation.(Feature Engineering, Normalization/Standardization, Encoding Categorical Variables, Handling Imbalanced Data)
4. Data Splitting. (Train and test split for modeling)

3.1 DATA INTEGRATION

3.1.1 MERGING

Perform a merge operation like inner or outer joins to combine data based on common columns.

In [17]:
```
# Merging the 3 datasets, check the number of rows and columns and returning the first 10 rows of the merged datafra
merged_df_1 = accidents_df.merge(vehicles_df, on="CRASH_RECORD_ID", how="inner")
merged_df = merged_df_1.merge(people_df, on="CRASH_RECORD_ID", how="inner")
print(merged_df.columns)
print( )
print(merged_df.shape)
print( )
merged_df.head(10)
```

```
Index(['CRASH_RECORD_ID', 'CRASH_DATE_EST_I', 'CRASH_DATE_x',
       'POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE',
       ...
       'EMS_RUN_NO', 'DRIVER_ACTION', 'DRIVER_VISION', 'PHYSICAL_CONDITION',
       'PEDPEDAL_ACTION', 'PEDPEDAL_VISIBILITY', 'PEDPEDAL_LOCATION',
       'BAC_RESULT', 'BAC_RESULT VALUE', 'CELL_PHONE_USE'],
      dtype='object', length=146)

(4237121, 146)
```

Out[17]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE_x	POSTED_SPEED_LIMIT	TRAFFIC_CONTRO
0	00a530520c84927490b597a6220ff3f2a3347472ef3086...	NaN	01/15/2025 01:50:00 PM	30	NC
1	204c3ca48ff3cdf0737e7f10b54ef5f8c49fbe30b5e53e...	NaN	12/04/2024 05:39:00 PM	30	
2	204c3ca48ff3cdf0737e7f10b54ef5f8c49fbe30b5e53e...	NaN	12/04/2024 05:39:00 PM	30	
3	204c3ca48ff3cdf0737e7f10b54ef5f8c49fbe30b5e53e...	NaN	12/04/2024 05:39:00 PM	30	
4	204c3ca48ff3cdf0737e7f10b54ef5f8c49fbe30b5e53e...	NaN	12/04/2024 05:39:00 PM	30	
5	0173d218723a1d608d77b7142ca9b9a4afd7463e824162...	Y	09/23/2023 10:39:00 AM	30	TRA
6	0173d218723a1d608d77b7142ca9b9a4afd7463e824162...	Y	09/23/2023 10:39:00 AM	30	TRA
7	0173d218723a1d608d77b7142ca9b9a4afd7463e824162...	Y	09/23/2023 10:39:00 AM	30	TRA
8	0173d218723a1d608d77b7142ca9b9a4afd7463e824162...	Y	09/23/2023 10:39:00 AM	30	TRA
9	0173d218723a1d608d77b7142ca9b9a4afd7463e824162...	Y	09/23/2023 10:39:00 AM	30	TRA

3.2 DATA CLEANING

This step involves identifying and correcting errors or inconsistencies in my dataset to ensure the data is accurate, complete, and reliable.

3.2.1 HANDLING DUPLICATES

Duplicate values can negatively impact data analysis, machine learning models and business decisions. They can lead to:

1. Distortion of statistical analysis and lead to incorrect conclusions.
2. If duplicates exist in both training and testing sets, the model may memorize rather than learn patterns, leading to overfitting

3. Duplicates increase dataset size unnecessarily, slowing down processing.
4. Duplicate records waste storage space and increase computational costs.
5. Duplicate data can inflate values in reports, leading to misleading graphs and charts.

In [18]:
```
# Checking for duplicates.
merged_df.duplicated().sum()
```

Out[18]:
```
0
```

There are 0 duplicated entries.

3.2.2 Dropping Unnecessary Columns.

Remove the columns which contain personal identifying information. Get rid of columns that would not help in analysis and modeling to reduce the number of features in the dataframe, that is, columns that are not related to the cause of the accident rather descriibe the activities after the accident.

In [19]:
```
# Dropping Columns.
merged_df.drop(
    ['CRASH_RECORD_ID', 'CRASH_DATE_EST_I', 'CRASH_DATE_x', 'REPORT_TYPE', 'LANE_CNT'
     'CRASH_TYPE', 'INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I', 'BEAT_OF_OCCURRENCE',
     'DAMAGE', 'DATE_POLICE_NOTIFIED', 'STREET_NAME', 'WORK_ZONE_I', 'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I'
     'BEAT_OF_OCCURRENCE', 'PHOTO_TAKEN_I', 'STATEMENTS_TAKEN_I', 'MOST_SEVERE_INJURY',
     'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING',
     'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN',
     'CRASH_HOUR', 'CRASH_UNIT_ID', 'CRASH_DATE_y', 'LOCATION','UNIT_NO', 'VEHICLE_ID_x',
     'CMRC_VEH_I', 'LIC_PLATE_STATE', 'TOWED_I', 'FIRE_I', 'TOWED_TO', 'TOWED_BY', 'AREA_00_I',
     'AREA_01_I', 'AREA_02_I', 'AREA_03_I', 'AREA_04_I', 'AREA_05_I', 'AREA_06_I',
     'AREA_07_I', 'AREA_08_I', 'AREA_09_I', 'AREA_10_I', 'AREA_11_I', 'AREA_12_I',
     'AREA_99_I',  'FIRST_CONTACT_POINT',  'CMV_ID', 'USDOT_NO', 'CCMC_NO', 'ILCC_NO', 'COMMERCIAL_SRC',
     'GVWR', 'CARRIER_NAME', 'CARRIER_STATE', 'CARRIER_CITY', 'HAZMAT_REPORT_I'
     'HAZMAT_NAME','HAZMAT_PRESENT_I', 'UN_NO', 'HAZMAT_REPORT_NO', 'MCS_REPORT_I',
     'MCS_REPORT_NO', 'IDOT_PERMIT_NO', 'TRAILER1_WIDTH', 'TRAILER2_WIDTH',
     'TRAILER1_LENGTH', 'TRAILER2_LENGTH', 'PERSON_TYPE'
     'PERSON_ID', 'VEHICLE_ID_y', 'CRASH_DATE', 'SEAT_NO', 'CITY', 'STATE',
     'ZIPCODE', 'EJECTION', 'INJURY_CLASSIFICATION',
     'HOSPITAL', 'EMS_AGENCY', 'EMS_RUN_NO', 'BAC_RESULT'],
    axis=1, #Ensures the columns only are being dropped
    inplace=True,
    errors='ignore'
)
print(merged_df.columns)
print( )
print(merged_df.shape)
print( )
merged_df.head(10)
```

```
Index(['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE', 'LANE_CNT', 'ALIGNMENT', 'ROADWAY_SURFACE_COND',
       'ROAD_DEFECT', 'CRASH_TYPE', 'PRIM_CONTRIBUTORY_CAUSE',
       'SEC_CONTRIBUTORY_CAUSE', 'STREET_NO', 'STREET_DIRECTION',
       'PHOTOS_TAKEN_I', 'DOORING_I', 'WORKERS_PRESENT_I', 'NUM_UNITS',
       'CRASH_DAY_OF_WEEK', 'CRASH_MONTH', 'LATITUDE', 'LONGITUDE',
       'UNIT_TYPE', 'NUM_PASSENGERS', 'MAKE', 'MODEL', 'VEHICLE_YEAR',
       'VEHICLE_DEFECT', 'VEHICLE_TYPE', 'VEHICLE_USE', 'TRAVEL_DIRECTION',
       'MANEUVER', 'OCCUPANT_CNT', 'EXCEED_SPEED_LIMIT_I', 'HAZMAT_PLACARDS_I',
       'HAZMAT_NAME', 'HAZMAT_REPORT_I', 'HAZMAT_VIO_CAUSE_CRASH_I',
       'MCS_VIO_CAUSE_CRASH_I', 'WIDE_LOAD_I', 'TOTAL_VEHICLE_LENGTH',
       'AXLE_CNT', 'VEHICLE_CONFIG', 'CARGO_BODY_TYPE', 'LOAD_TYPE',
       'HAZMAT_OUT_OF_SERVICE_I', 'MCS_OUT_OF_SERVICE_I', 'HAZMAT_CLASS',
       'PERSON_ID', 'PERSON_TYPE', 'SEX', 'AGE', 'DRIVERS_LICENSE_STATE',
       'DRIVERS_LICENSE_CLASS', 'SAFETY_EQUIPMENT', 'AIRBAG_DEPLOYED',
       'DRIVER_ACTION', 'DRIVER_VISION', 'PHYSICAL_CONDITION',
       'PEDPEDAL_ACTION', 'PEDPEDAL_VISIBILITY', 'PEDPEDAL_LOCATION',
       'BAC_RESULT VALUE', 'CELL_PHONE_USE'],
      dtype='object')

(4237121, 66)
```

Out[19]:

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_CONDITION	FIRST_C
0	30	NO CONTROLS	NO CONTROLS	CLEAR	DAYLIGHT	
1	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
2	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
3	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
4	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
5	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	UNKNOWN	UNKNOWN	F
6	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	UNKNOWN	UNKNOWN	F
7	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	UNKNOWN	UNKNOWN	F
8	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	UNKNOWN	UNKNOWN	F
9	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	UNKNOWN	UNKNOWN	F

3.2.3 Handling Missing Values

After dropping the columns, 66 columns remain. These are features that contribute to the accident occurence and can be modeled and analysed.

3.2.3.1 Numeric Columns

1. **LANE_CNT, OCCUPANT_CNT** - Fill with median (since lane count and occupants typically follow a skewed distribution).
2. **LATITUDE, LONGITUDE** - Use nearest valid values (e.g., forward or backward fill: ffill() or bfill()).
3. **NUM_PASSENGERS** - Fill with 0 because missing values mean there were no passengers.
4. **VEHICLE_YEAR** - Fill with median year to avoid bias from extreme values.
5. **TOTAL_VEHICLE_LENGTH, AXLE_CNT** - Fill with median if these features are important; otherwise, use 0 for unknown.
6. **AGE** - Fill with median.
7. **BAC_RESULT VALUE** - Fill with 0.00 as missing implies no test was conducted.

These values are denoted by 'NaN' which stands for "Not a Number". It is a special value used in computing to represent missing, undefined or unrepresentable data in numerical or data processing operations. These values are handled by:

Replacing NaN Values

1. For numeric columns: the mean for normally distributed data, the median if it is not normally distributed and '0' if the previous 2 might lead to loss of data integrity.
2. For categorical columns: the mode or 'Unknown' if mode might lead to loss of data integrity.

   Dropping the NaN values
   Get rid of the entries containing the NaN value if they are bound to affect analysis or modeling.

In [20]:
```
# Checking the columns with NaN values and how many we are dealing with.
merged_df.isna().sum()[merged_df.isna().sum() > 0]
```

Out[20]:
```
LANE_CNT                    3306376
STREET_DIRECTION                 14
PHOTOS_TAKEN_I              4174499
DOORING_I                   4223948
WORKERS_PRESENT_I           4231578
LATITUDE                      30310
LONGITUDE                     30310
UNIT_TYPE                      4349
NUM_PASSENGERS              3179356
MAKE                         100939
MODEL                        101324
VEHICLE_YEAR                 655246
VEHICLE_DEFECT               100927
VEHICLE_TYPE                 100927
VEHICLE_USE                  100927
TRAVEL_DIRECTION             100927
MANEUVER                     100927
OCCUPANT_CNT                 100927
EXCEED_SPEED_LIMIT_I        4230994
HAZMAT_PLACARDS_I           4236353
HAZMAT_NAME                 4236951
HAZMAT_REPORT_I             4208120
HAZMAT_VIO_CAUSE_CRASH_I    4207729
MCS_VIO_CAUSE_CRASH_I       4208161
WIDE_LOAD_I                 4236770
TOTAL_VEHICLE_LENGTH        4230673
AXLE_CNT                    4227668
VEHICLE_CONFIG              4203060
CARGO_BODY_TYPE             4204485
LOAD_TYPE                   4206436
HAZMAT_OUT_OF_SERVICE_I     4208955
MCS_OUT_OF_SERVICE_I        4208390
HAZMAT_CLASS                4234690
SEX                           72502
AGE                         1223786
DRIVERS_LICENSE_STATE       1762014
DRIVERS_LICENSE_CLASS       2180248
SAFETY_EQUIPMENT              12977
AIRBAG_DEPLOYED               85930
DRIVER_ACTION                873273
DRIVER_VISION                874699
PHYSICAL_CONDITION           870769
PEDPEDAL_ACTION             4151959
PEDPEDAL_VISIBILITY         4152115
PEDPEDAL_LOCATION           4151956
BAC_RESULT VALUE            4231865
CELL_PHONE_USE              4234631
dtype: int64
```

In [21]:
```
# Filling missing values in the numeric columns.
merged_df["LANE_CNT"].fillna(merged_df["LANE_CNT"].median(), inplace=True)
merged_df["OCCUPANT_CNT"].fillna(merged_df["OCCUPANT_CNT"].median(), inplace=True)
merged_df["LATITUDE"].fillna(method="ffill", inplace=True)
merged_df["LONGITUDE"].fillna(method="ffill", inplace=True)
merged_df["NUM_PASSENGERS"].fillna(0, inplace=True)
merged_df["VEHICLE_YEAR"].fillna(merged_df["VEHICLE_YEAR"].median(), inplace=True)
merged_df["TOTAL_VEHICLE_LENGTH"].fillna(merged_df["TOTAL_VEHICLE_LENGTH"].median(), inplace=True)
merged_df["AXLE_CNT"].fillna(merged_df["AXLE_CNT"].median(), inplace=True)
merged_df["AGE"].fillna(merged_df["AGE"].median(), inplace=True)
merged_df["BAC_RESULT VALUE"].fillna(0.00, inplace=True)
```

3.2.3.2 Categorical Columns

1. **VEHICLE_DEFECT, VEHICLE_TYPE, VEHICLE_USE, TRAVEL_DIRECTION, MANEUVER, STREET_DIRECTION, UNIT_TYPE, MAKE, MODEL, SEX, SAFETY_EQUIPMENT, AIRBAG_DEPLOYED, DRIVER_ACTION, DRIVER_VISION, PHYSICAL_CONDITION, DRIVERS_LICENSE_STATE, DRIVERS_LICENSE_CLASS** - Fill with "UNKNOWN" to keep category consistency.
2. **EXCEED_SPEED_LIMIT_I, HAZMAT_PLACARDS_I, HAZMAT_REPORT_I, HAZMAT_VIO_CAUSE_CRASH_I, MCS_VIO_CAUSE_CRASH_I, WIDE_LOAD_I, HAZMAT_OUT_OF_SERVICE_I, MCS_OUT_OF_SERVICE_I, CELL_PHONE_USE** - Fill with "NO" assuming missing means it was not applicable.
3. **PEDPEDAL_ACTION, PEDPEDAL_VISIBILITY, PEDPEDAL_LOCATION** - Fill with "NO_PEDESTRIAN" if missing means no pedestrian involved.

In [22]:
```
# Filling missing values in the categorical columns.
categorical_fill_unknown = ["VEHICLE_DEFECT", "VEHICLE_TYPE", "VEHICLE_USE", "TRAVEL_DIRECTION",
                            "MANEUVER", "DRIVERS_LICENSE_STATE", "DRIVERS_LICENSE_CLASS",
                            "SAFETY_EQUIPMENT", "AIRBAG_DEPLOYED", "DRIVER_ACTION", "DRIVER_VISION",
                            "PHYSICAL_CONDITION", "SEX", "STREET_DIRECTION", "UNIT_TYPE", "MAKE", "MODEL",
                            "HAZMAT_NAME", "VEHICLE_CONFIG", "CARGO_BODY_TYPE", "LOAD_TYPE", "HAZMAT_CLASS"]
for col in categorical_fill_unknown:
    merged_df[col].fillna("UNKNOWN", inplace=True)

categprical_fill_no = ["EXCEED_SPEED_LIMIT_I", "HAZMAT_PLACARDS_I", "HAZMAT_REPORT_I", "HAZMAT_VIO_CAUSE_CRASH_I",
                  "MCS_VIO_CAUSE_CRASH_I", "WIDE_LOAD_I", "HAZMAT_OUT_OF_SERVICE_I", "MCS_OUT_OF_SERVICE_I",
                  "CELL_PHONE_USE", "PHOTOS_TAKEN_I", "DOORING_I", "WORKERS_PRESENT_I"]
for col in categprical_fill_no:
    merged_df[col].fillna("NO", inplace=True)

pedestrian_fill = ["PEDPEDAL_ACTION", "PEDPEDAL_VISIBILITY", "PEDPEDAL_LOCATION"]
for col in pedestrian_fill:
    merged_df[col].fillna("NO_PEDESTRIAN", inplace=True)
```

In [23]:
```
# Recheck for any more missing values
merged_df.isna().sum().sum()
```

Out[23]:
```
0
```

After imputing the null values, the total null values have been reduced to 0.

3.2.4 Outlier Detection

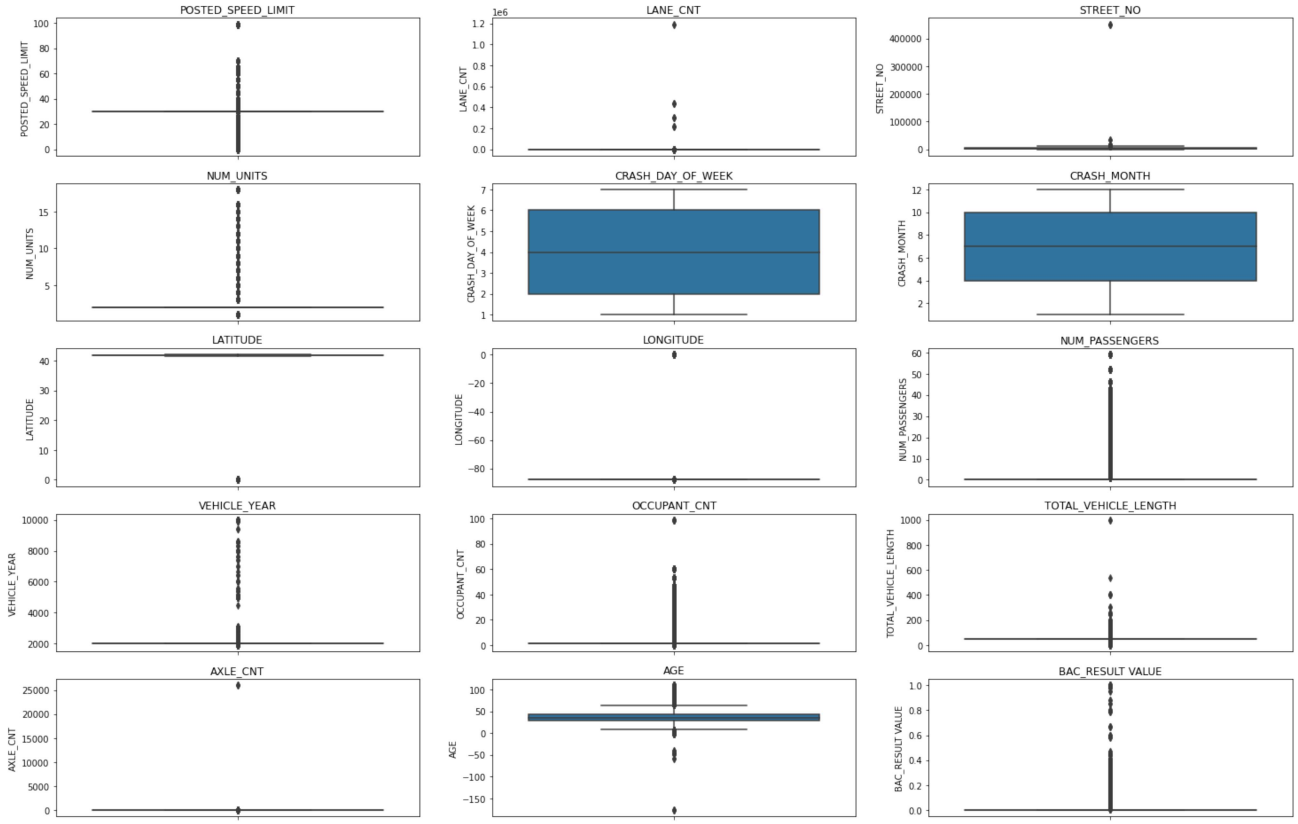
Outlier detection is crucial in data analysis and machine learning because outliers can significantly impact model performance and data interpretation.

3.2.4.1 Importance of outlier detection.

1. **Improves Model Accuracy** Outliers can distort statistical measures (mean, standard deviation) and affect model training thus affecting machine learning models, especially linear regression and clustering as they are sensitive to outliers.
2. **Prevents Model Overfitting** Models might learn patterns specific to outliers rather than general trends therefore reducing the ability of the model to generalize well on unseen data.
3. **Enhances Data Integrity** Outliers may indicate errors in data collection or entry identifying and handling them ensures data quality.
4. **Improves Data Distribution and Normality** Many statistical tests and ML algorithms assume normality thus removing extreme values helps meet these assumptions.
5. **Identifies Rare but Important Events I**n fraud detection, cybersecurity, or medical diagnostics, outliers can indicate critical incidents therefore instead of removing them, they might be the focus of the study.
6. **Affects Feature Scaling & Normalization** Outliers can impact Min-Max scaling and Standardization thus proper handling ensures fair feature scaling.

The best way to get outliers is by plotting a boxplot of the numerical columna as done below.

In [24]:
```
# Select numerical columns
numerical_cols = merged_df.select_dtypes(include=['int64', 'float64']).columns

# Set figure size
plt.figure(figsize=(25, 20))

# Create box plots for all numerical columns
for i, col in enumerate(numerical_cols, 1):
    plt.subplot((len(numerical_cols) // 3) + 1, 3, i)
    sns.boxplot(y=merged_df[col])
    plt.title(col)

plt.show()
```



From the above boxplots, there are extreme values in the following columns:

1. LONGITUDE and LATITUDE - seeing as the data is from chicago, a longitude and latitude of 0 is not expect as it is not within the USA.
2. TOTAL_VEHICLE_LENGTH- A vehicle length of above 200 feet is outrageuos as the maximum allowable length including trailers is 200 feet.
3. AXLE_CNT- The maximum possible axle count is 20 so any vlue above 20 is obvoiusly an error.
4. VEHICLE_YEAR- A Vehicle year that is not between 1950 and 2025 is an overbound entry and thus should be removed.
5. AGE- An age below 0 is impossible hence an error within the data.
6. STEET_NO - Chicago street numbers lie between 0 and 22000 so any value beyond this is not viable.

3.2.4.2 Dealing with outliers in the Longitude and Latitude columns

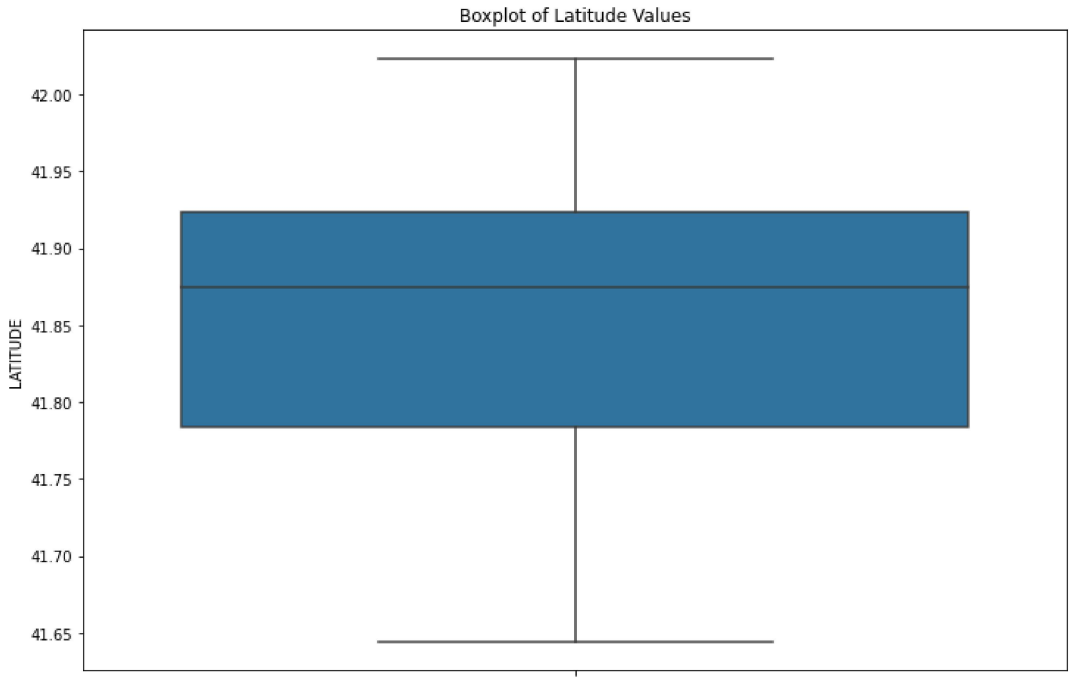
Drop the values that are 0 in the LONGITUDE and LATITUDE columns

In [25]:
```
#Dropping rows whose LONGITUDE and LATITUDE is 0
merged_df = merged_df[(merged_df['LATITUDE'] != 0) & (merged_df['LONGITUDE'] != 0)]
```

Checking the boxplot to see if there are outliers in the longitude and latitude columns.

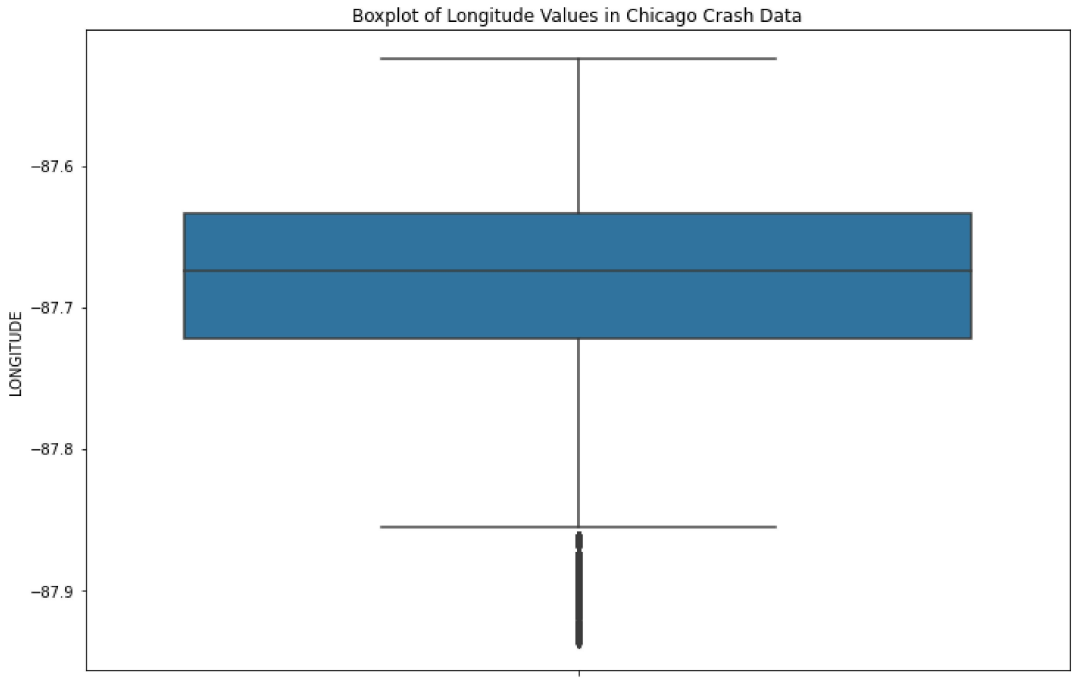
In [26]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['LATITUDE'])

# Add title
plt.title("Boxplot of Latitude Values")
```

Out[26]: Text(0.5, 1.0, 'Boxplot of Latitude Values')



In [27]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['LONGITUDE'])

# Add title
plt.title("Boxplot of Longitude Values in Chicago Crash Data")
```

Out[27]:
```
Text(0.5, 1.0, 'Boxplot of Longitude Values in Chicago Crash Data')
```



The longitudes and latitudes are within a reasonable range which is a proper reflection of the data.

3.2.4.3 Dealing with outliers in the TOTAL_VEHICLE_LENGTH column.

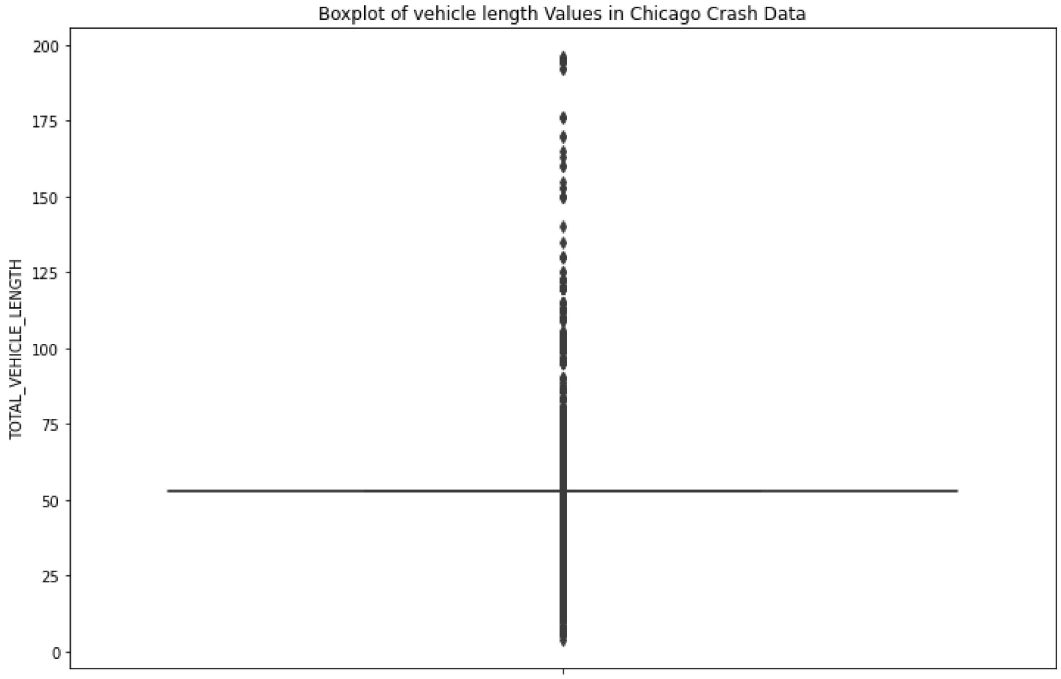
Ensure that the vehicle length is between 4 feet and 200 feet so that only realistic values remain.

In [28]:
```
#Drop the rows with unrealistic vehicle lengths
merged_df = merged_df[(merged_df['TOTAL_VEHICLE_LENGTH'] >= 4) & (merged_df['TOTAL_VEHICLE_LENGTH'] <=200)]
```

Checking the boxplot to see if the outliers have been solved.

In [29]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['TOTAL_VEHICLE_LENGTH'])

# Add title
plt.title("Boxplot of vehicle length Values in Chicago Crash Data")
```

Out[29]: Text(0.5, 1.0, 'Boxplot of vehicle length Values in Chicago Crash Data')



The values are within a reasonable range in relation to the dataset

3.2.4.4 Dealing with outliers in the AXLE_CNT column

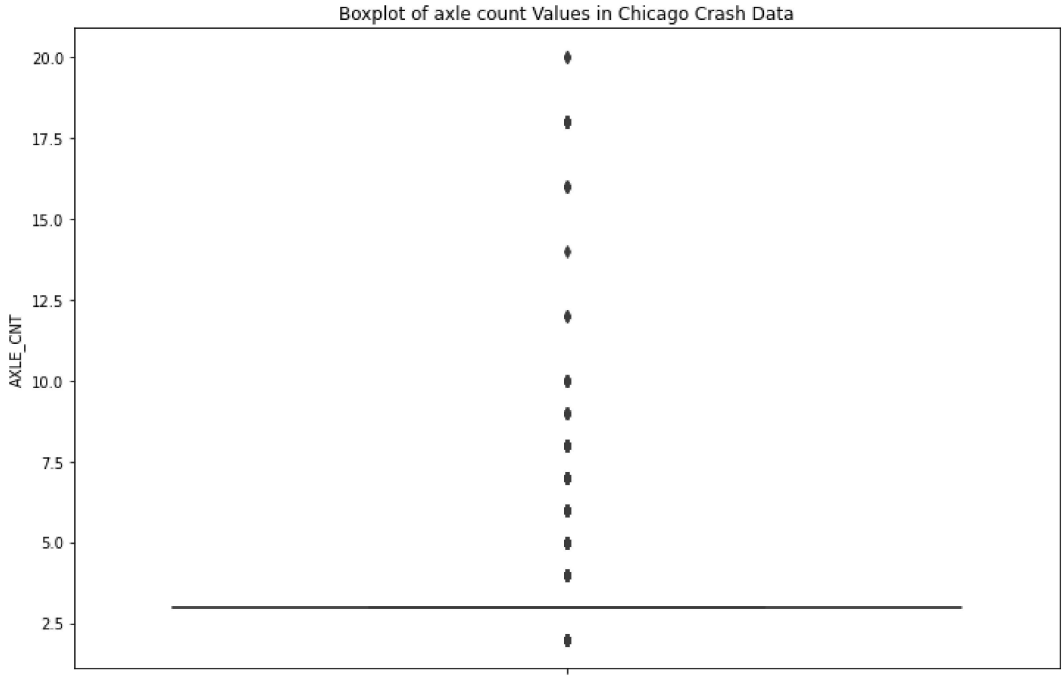
The expected range of axles is between 2 and 20. all values that do not lie between this range are erroneous and have to be dropped.

In [30]:
```
#Filter out axle counts that are not between 2 and 20
merged_df = merged_df[(merged_df['AXLE_CNT'] <= 24) & (merged_df["AXLE_CNT"] >= 2)]
```

Replot the boxplot to check if the remaining entries are within range.

In [31]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['AXLE_CNT'])

# Add title
plt.title("Boxplot of axle count Values in Chicago Crash Data")
```

Out[31]:
```
Text(0.5, 1.0, 'Boxplot of axle count Values in Chicago Crash Data')
```



3.2.4.5 Dealing with outliers in the VEHICLE_YEAR column.

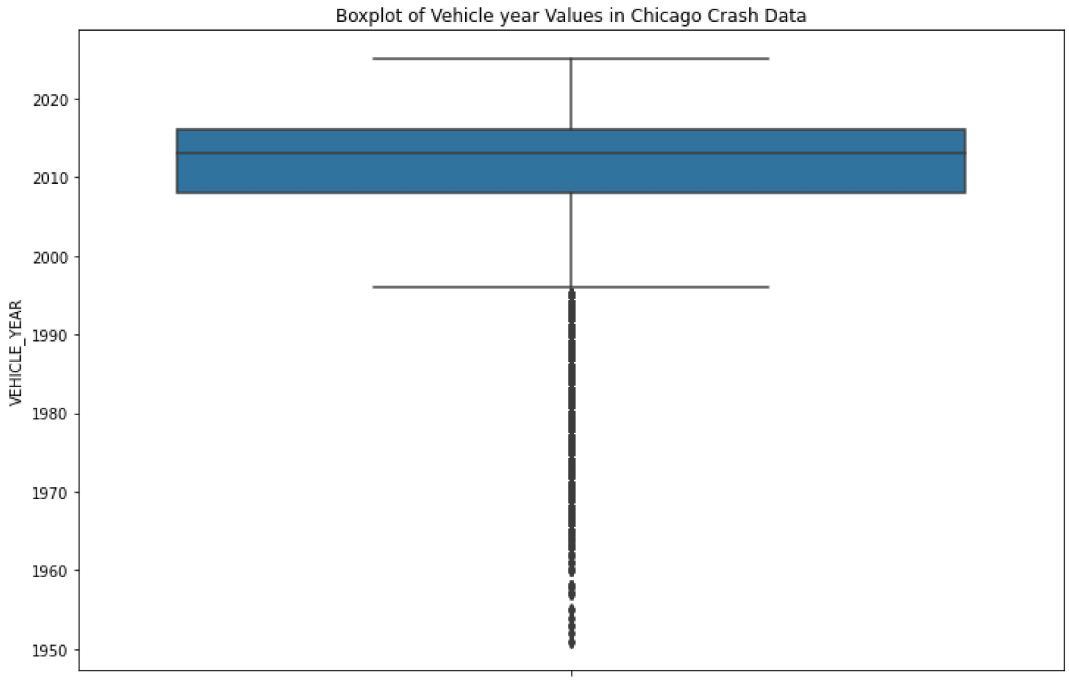
The expected range of vehicle manufacture years on chicago roads is between 1950 and 2025. Values beyond this range are beyond realistic expectation.

In [32]:
```
# Filter out vehicle years that are not between 1950 and 2025
merged_df = merged_df[(merged_df['VEHICLE_YEAR'] <= 2025) & (merged_df["VEHICLE_YEAR"] >= 1950)]
```

Plot the boxplot of the filtered years to confirm they are within the range.

In [33]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['VEHICLE_YEAR'])

# Add title
plt.title("Boxplot of Vehicle year Values in Chicago Crash Data")
```

Out[33]: Text(0.5, 1.0, 'Boxplot of Vehicle year Values in Chicago Crash Data')



The values are now within the acceptable range as shown by the above boxplot

3.2.4.6 Dealing with outliers in the AGE column.

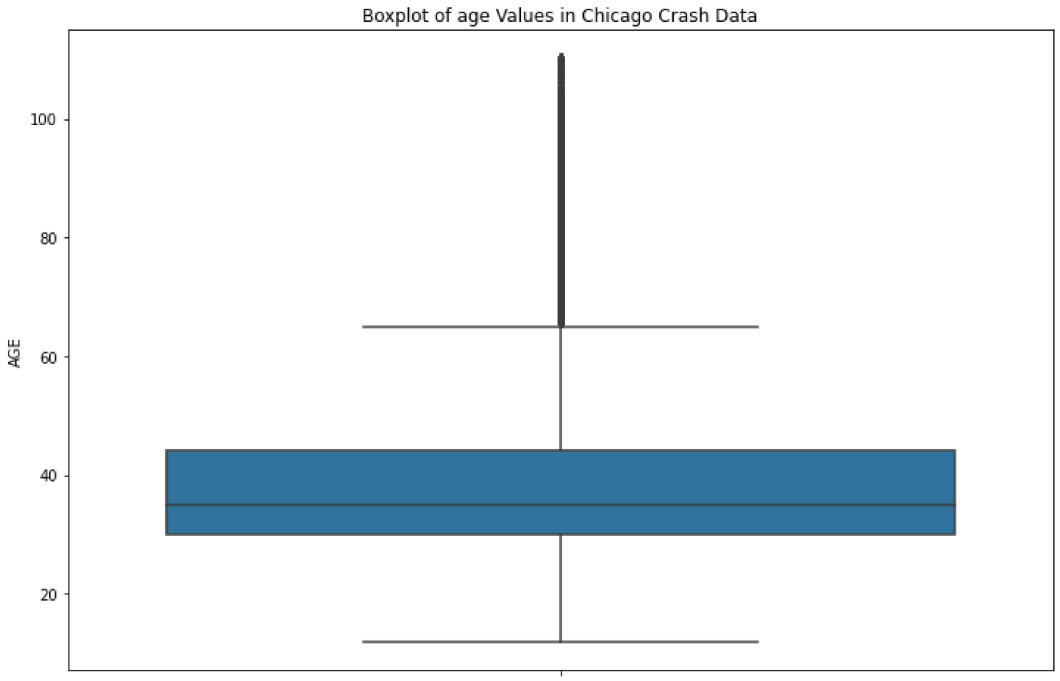
The above boxplot shows ages below 0. This is unrealistic and impossible. A realistic driving age might be above 12 years old thus the need to calibrate the data to fit this realistic range

In [34]:
```
#Ensure the ages are all positive
merged_df["AGE"] = merged_df['AGE'].abs()

# Filter out the entries with unrealistic ages
merged_df = merged_df[(merged_df['AGE'] <= 110) & (merged_df["AGE"] >= 12)]
```

Plot a boxplot to confirm ages are within the expected Range.

In [35]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['AGE'])

# Add title
plt.title("Boxplot of age Values in Chicago Crash Data")
```

Out[35]:
```
Text(0.5, 1.0, 'Boxplot of age Values in Chicago Crash Data')
```



The values are now within an acceptable range

3.2.4.7 Dealing with outliers in the STEET_NO column.

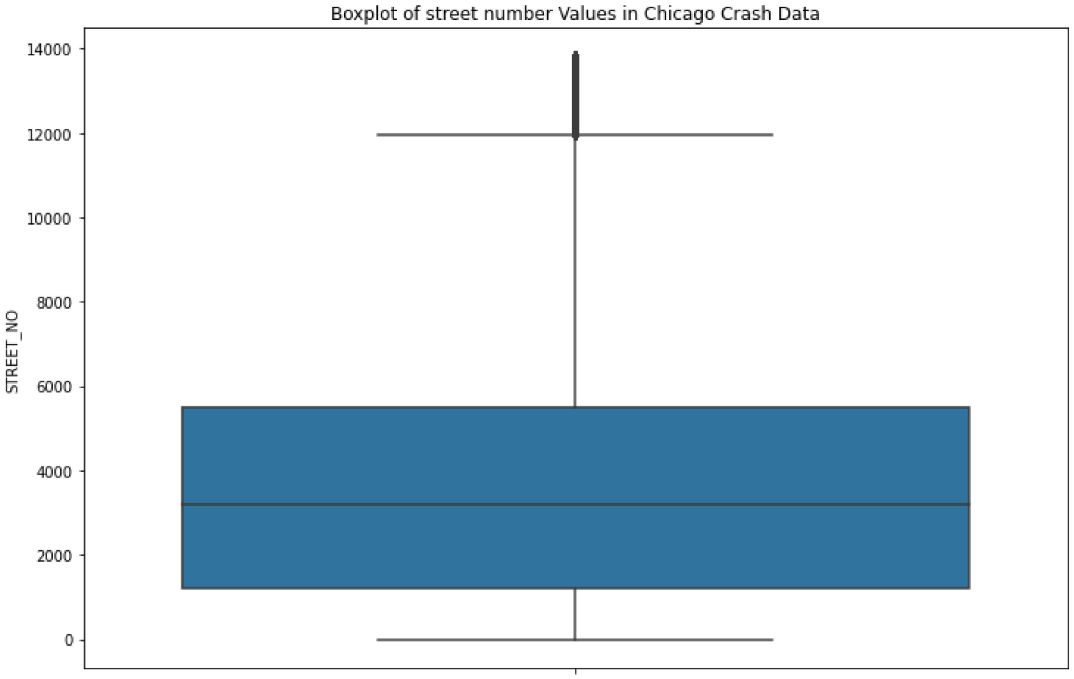
Chicago street numbers lie between 0 and 22000. Any value above or below this range is an error which needs to be filtered out.

In [36]:
```
# Filter out the entries with unrealistic street numbers
merged_df = merged_df[(merged_df['STREET_NO'] <= 22000) & (merged_df["STREET_NO"] >= 0)]
```

Polt the boxplot of the Cleaned column.

In [37]:
```
# Set figure size
plt.figure(figsize=(12, 8))

# Create boxplot
sns.boxplot(y=merged_df['STREET_NO'])

# Add title
plt.title("Boxplot of street number Values in Chicago Crash Data")
```

Out[37]: Text(0.5, 1.0, 'Boxplot of street number Values in Chicago Crash Data')



3.2.5 Saving the clean data

In [38]:
```
#Saving to a csv file.
merged_df.to_csv("Clean_Chicago_crash_data.csv", index=False, encoding="utf-8")
```

4 DATA TRANSFORMATION.

4.1 Feature Engineering

Feature Engineering is the process of transforming raw data into meaningful features that improve the performance of machine learning models. It helps models learn patterns more effectively by creating, modifying, or selecting relevant features.

4.1.1 Creating A New Feature

Combine existing columns ie MAKE + MODEL → MAKE_MODEL.

In [39]:
```
#Combine make and model
merged_df['MAKE_MODEL'] = merged_df['MAKE'] + " " + merged_df['MODEL']
# Drop the individual make and model columns.
merged_df.drop(columns=['MAKE', 'MODEL'], inplace=True)
```

4.1.2 Dropping rows whose primary contributory cause was not determined

In [40]:
```
# Drop rows where PRIM_CONTRIBUTORY_CAUSE is "Unable to determine"
merged_df = merged_df[merged_df['PRIM_CONTRIBUTORY_CAUSE'] != "UNABLE TO DETERMINE"]

# Verify the change
print(merged_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts())
```

```
FAILING TO YIELD RIGHT-OF-WAY                                          521286
FOLLOWING TOO CLOSELY                                                  466482
IMPROPER OVERTAKING/PASSING                                            219270
FAILING TO REDUCE SPEED TO AVOID CRASH                                 206099
NOT APPLICABLE                                                         186570
IMPROPER LANE USAGE                                                    152587
IMPROPER TURNING/NO SIGNAL                                             151232
IMPROPER BACKING                                                       134338
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE                                    121550
DISREGARDING TRAFFIC SIGNALS                                           111129
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE MANNER     59506
WEATHER                                                                 55838
DISREGARDING STOP SIGN                                                  53816
DISTRACTION - FROM INSIDE VEHICLE                                       30509
DRIVING ON WRONG SIDE/WRONG WAY                                         26264
EQUIPMENT - VEHICLE CONDITION                                           23343
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)                    22246
PHYSICAL CONDITION OF DRIVER                                            21541
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)      20414
DISTRACTION - FROM OUTSIDE VEHICLE                                      15551
DISREGARDING OTHER TRAFFIC SIGNS                                         9950
EXCEEDING AUTHORIZED SPEED LIMIT                                         9630
EXCEEDING SAFE SPEED FOR CONDITIONS                                      7635
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST                        6117
CELL PHONE USE OTHER THAN TEXTING                                        5804
DISREGARDING ROAD MARKINGS                                               5085
ROAD CONSTRUCTION/MAINTENANCE                                            5073
ROAD ENGINEERING/SURFACE/MARKING DEFECTS                                 3969
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)                          3923
TURNING RIGHT ON RED                                                     3824
RELATED TO BUS STOP                                                      2357
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER, ETC.)      2044
ANIMAL                                                                   1799
TEXTING                                                                  1613
DISREGARDING YIELD SIGN                                                  1263
PASSING STOPPED SCHOOL BUS                                                777
OBSTRUCTED CROSSWALKS                                                     466
BICYCLE ADVANCING LEGALLY ON RED LIGHT                                    370
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT                                 100
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
```

4.1.3 Dropping rows where MAKE_MODEL is unknown

In [41]:
```
# Drop rows where 'MAKE_MODEL' contains 'UNKNOWN'
merged_df = merged_df[~merged_df['MAKE_MODEL'].str.contains('UNKNOWN', case=False, na=False)]

# Reset the index after dropping rows
merged_df.reset_index(drop=True, inplace=True)
```

4.1.4 Combine entries TOYOTA CAMRY and TOYOTA MOTOR COMPANY, LTD. CAMRY into a unified value of TOYOTA CAMRY

In [42]:
```
# Define patterns to standardize to 'TOYOTA CAMRY'
merged_df['MAKE_MODEL'] = merged_df['MAKE_MODEL'].str.replace(
    r'TOYOTA.*CAMRY',
    'TOYOTA CAMRY',
    case=False,
    regex=True
)

# Optional: Reset index after changes
merged_df.reset_index(drop=True, inplace=True)
```

Type *Markdown* and LaTeX: \alpha^2

In [43]:
```
# Make a copy to be used for EDA

project_df = merged_df.copy()

#rewturn the new first 10 rows of the new df
project_df.head(10)
```

Out[43]:

	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_CONDITION	FIRST_C
0	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
1	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
2	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
3	30	UNKNOWN	UNKNOWN	RAIN	DARKNESS, LIGHTED ROAD	
4	25	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DARKNESS, LIGHTED ROAD	
5	25	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DARKNESS, LIGHTED ROAD	
6	25	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DARKNESS, LIGHTED ROAD	
7	25	TRAFFIC SIGNAL	FUNCTIONING PROPERLY	CLEAR	DARKNESS, LIGHTED ROAD	
8	30	NO CONTROLS	NO CONTROLS	CLEAR	DARKNESS, LIGHTED ROAD	
9	30	NO CONTROLS	NO CONTROLS	CLEAR	DARKNESS, LIGHTED ROAD	

4.1.5 Normalization/Standardization

Both Normalization and Standardization are feature scaling techniques that adjust the range or distribution of numerical data, helping machine learning models perform better. Since this is done for proper modeling the data is divided to target and predictor variables (y and X).

In [44]:
```
# Define the target (y) and predictors (X).
y = merged_df['PRIM_CONTRIBUTORY_CAUSE']
X = merged_df.drop(columns='PRIM_CONTRIBUTORY_CAUSE')
```

4.1.5.1 Standardizing the numerical columns

In [45]:
```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform only numerical columns
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])
```

The numeric columns have been standardized and are ready for modeling.

4.1.6 Encoding Categorical Variables

Encoding is the process of converting categorical data (e.g., labels, text) into numerical format so that machine learning models can understand and process it. the target is encoded by label encoding while the predictors are encoded by One Hot Encoding. However, in our case where One Hot Encoding would require significant memory resources that are not available to us, we use frequency encoding.

4.1.6.1 Apply Label Encoding to target variable

In [46]:
```
#Initialize the encoder
le = LabelEncoder()

#Fit and transform the target variable
y = le.fit_transform(y)
```

4.1.6.2 Apply Frequency Encoding to categorical columns in the predictor variables

In [47]:
```
# Select only categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Frequency Encoding for categorical columns
for col in categorical_cols:
    freq = X[col].value_counts(normalize=True)  # Get frequency of each category
    X[col] = X[col].map(freq)  # Map frequencies to the original column
```

4.1.7 Handling Imbalanced Data

Imbalanced data occurs when one or more classes in a classification problem have significantly fewer instances than others. This might lead to misleading accuracy in the model.

Importance of Handling Class Imbalance

1. Bias in Model Predictions. Imbalanced data leads models to favor majority classes, ignoring minority classes.
2. Misleading Accuracy Metrics. High overall accuracy can be misleading if the model simply predicts the majority class.
3. Improved Model Performance. Proper handling improves the model's ability to generalize across all classes.

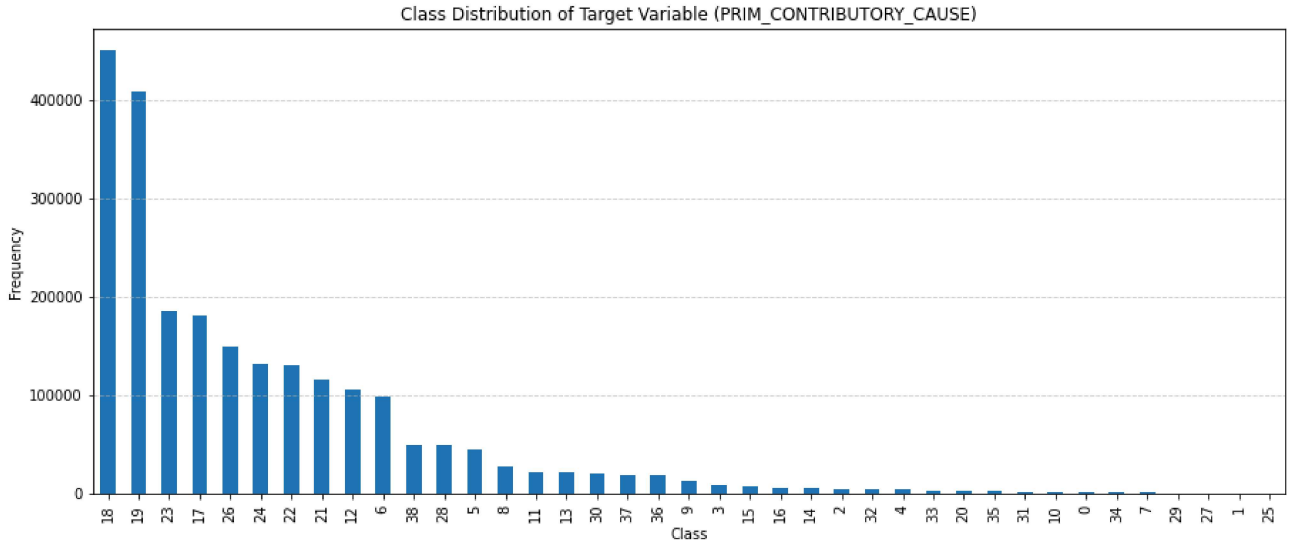
4.1.7.1 Checking for class imbalance

In [48]:
```
# Convert array to Pandas Series
y = pd.Series(y, name='PRIM_CONTRIBUTORY_CAUSE')

# Check class distribution
class_counts = y.value_counts()
print(class_counts)

# Calculate class ratios
class_ratios = (class_counts / len(y)) * 100

# Display class ratios
print(class_ratios)
```

```
18    449905
19    408120
23    184850
17    180786
26    149796
24    132558
22    130643
21    116167
12    105363
6      98102
38     49165
28     48892
5      45630
8      27591
11     22178
13     21344
30     19830
37     18828
36     18533
9      13743
3       8306
15      7345
16      6320
14      5203
2       5016
32      4341
4       4129
33      3433
20      3279
35      3125
31      1858
10      1820
0       1504
34      1419
7        963
29       516
27       344
1        177
25        82
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
18    19.550853
19    17.735064
23     8.032752
17     7.856148
26     6.509462
24     5.760376
22     5.677159
21     5.048097
12     4.578603
6      4.263073
38     2.136490
28     2.124627
5      1.982875
8      1.198981
11     0.963756
13     0.927514
30     0.861723
37     0.818180
36     0.805361
9      0.597209
3      0.360941
15     0.319181
16     0.274639
14     0.226099
2      0.217973
32     0.188640
4      0.179428
33     0.149183
20     0.142491
35     0.135798
31     0.080740
10     0.079089
0      0.065357
34     0.061663
7      0.041848
29     0.022423
27     0.014949
1      0.007692
```

```
25     0.003563
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: float64
```

In [49]:
```
# Plot class distribution as a bar chart
plt.figure(figsize=(15, 6))
class_counts.plot(kind='bar')
plt.title('Class Distribution of Target Variable (PRIM_CONTRIBUTORY_CAUSE)')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.xticks(rotation=90)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



No class in the target variable covers more than 50% of the data, therefore the data is fairly balanced.

4.2 DATA SPLITTING

Data splitting is the process of dividing a dataset into separate parts for training and evaluating a machine learning model. It helps to assess model performance on unseen data, ensuring generalization and reducing overfitting.

In [50]:
```
# Perform a 70% training and 30% testing split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify= y)
```

5 EXPLORATORY DATA ANALYSIS

EDA is the process of analyzing and visualizing the dataset to understand its structure, patterns, and relationships.

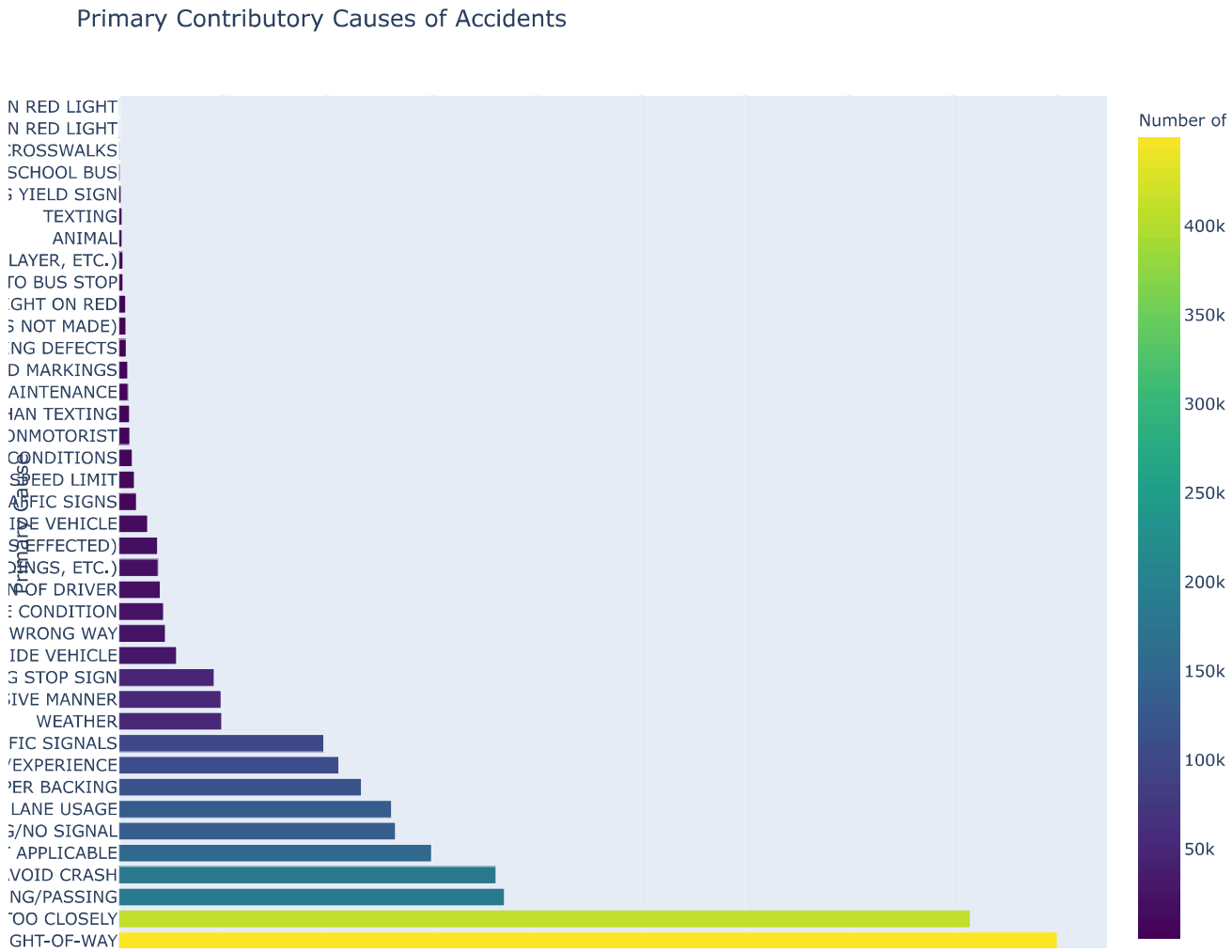
5.1 What are the most common contributory causes of car accidents?

5.1.1 Univariate analysis

5.1.1.1 Most common primary cause of accidents

In [51]:

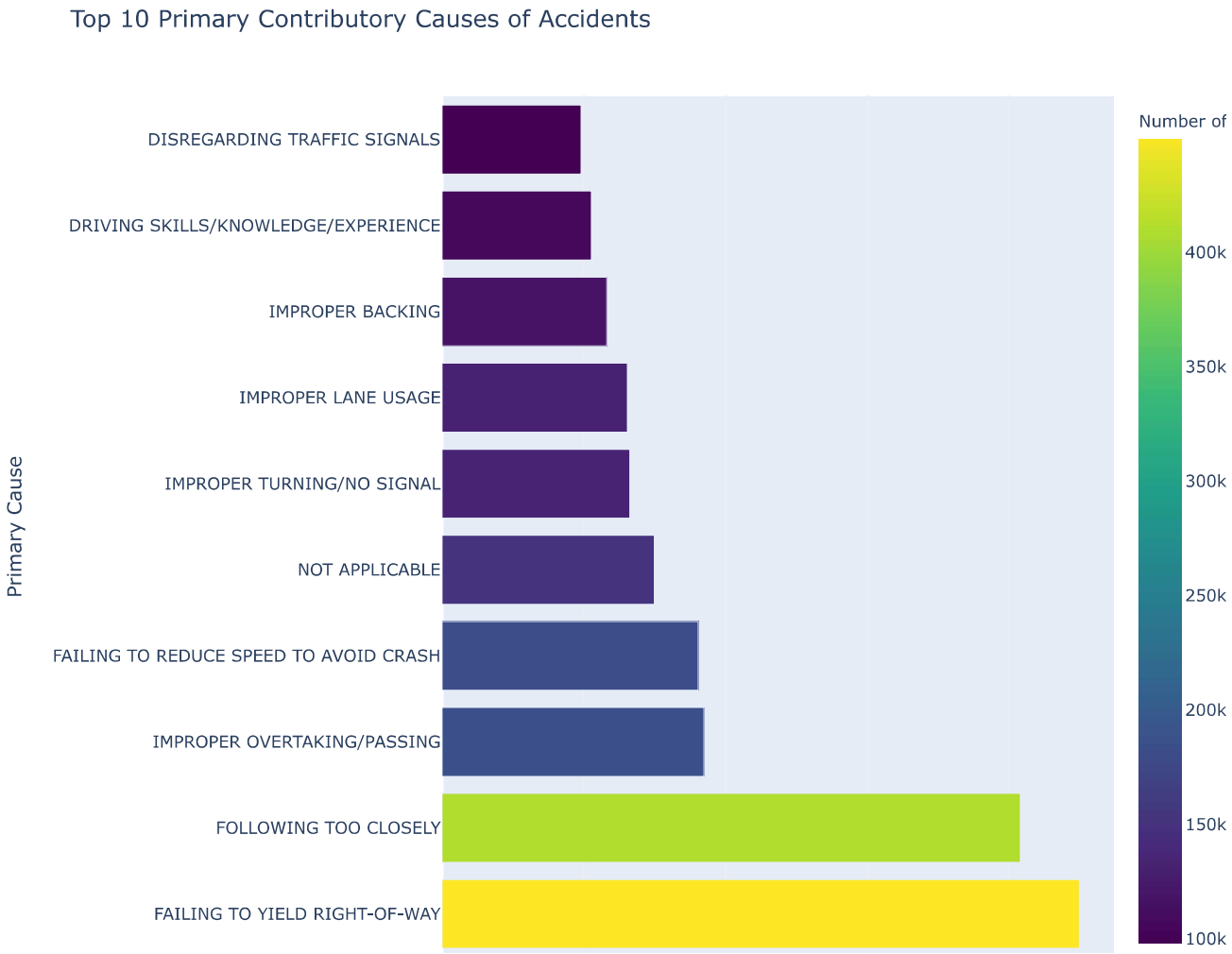
```
# Calculate the number of occurrences per category
primary_cause_count = project_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts().reset_index()
primary_cause_count.columns = ['Primary Cause', 'Number of Occurrences']

# Create the bar plot using Plotly
fig = px.bar(
    primary_cause_count,
    x='Number of Occurrences',
    y='Primary Cause',
    orientation='h',  # Horizontal bars
    title='Primary Contributory Causes of Accidents',
    color='Number of Occurrences',
    color_continuous_scale='Viridis',
    height=800,
)

# Update layout
fig.update_layout(
    xaxis_title='Number of Occurrences',
    yaxis_title='Primary Cause',
    showlegend=False
)

# Save the image
plt.savefig('Primary_Cause_of_Accidents.png')

# Show the plot
fig.show()
```




```
<Figure size 432x288 with 0 Axes>
```

The top ten are as shown below

In [52]:
```
# Calculate the number of occurrences per category and get the top 10
top_10_causes = project_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlargest(10).reset_index()
top_10_causes.columns = ['Primary Cause', 'Number of Occurrences']

# Create the bar plot using Plotly
fig = px.bar(
    top_10_causes,
    x='Number of Occurrences',
    y='Primary Cause',
    orientation='h',  # Horizontal bars
    title='Top 10 Primary Contributory Causes of Accidents',
    color='Number of Occurrences',
    color_continuous_scale='Viridis',
    height=800,
)

# Update layout
fig.update_layout(
    xaxis_title='Number of Occurrences',
    yaxis_title='Primary Cause',
    showlegend=False
)

# Save the image
plt.savefig('Top_10_Primary_Cause_of_Accidents.png')

# Show the plot
fig.show()
```



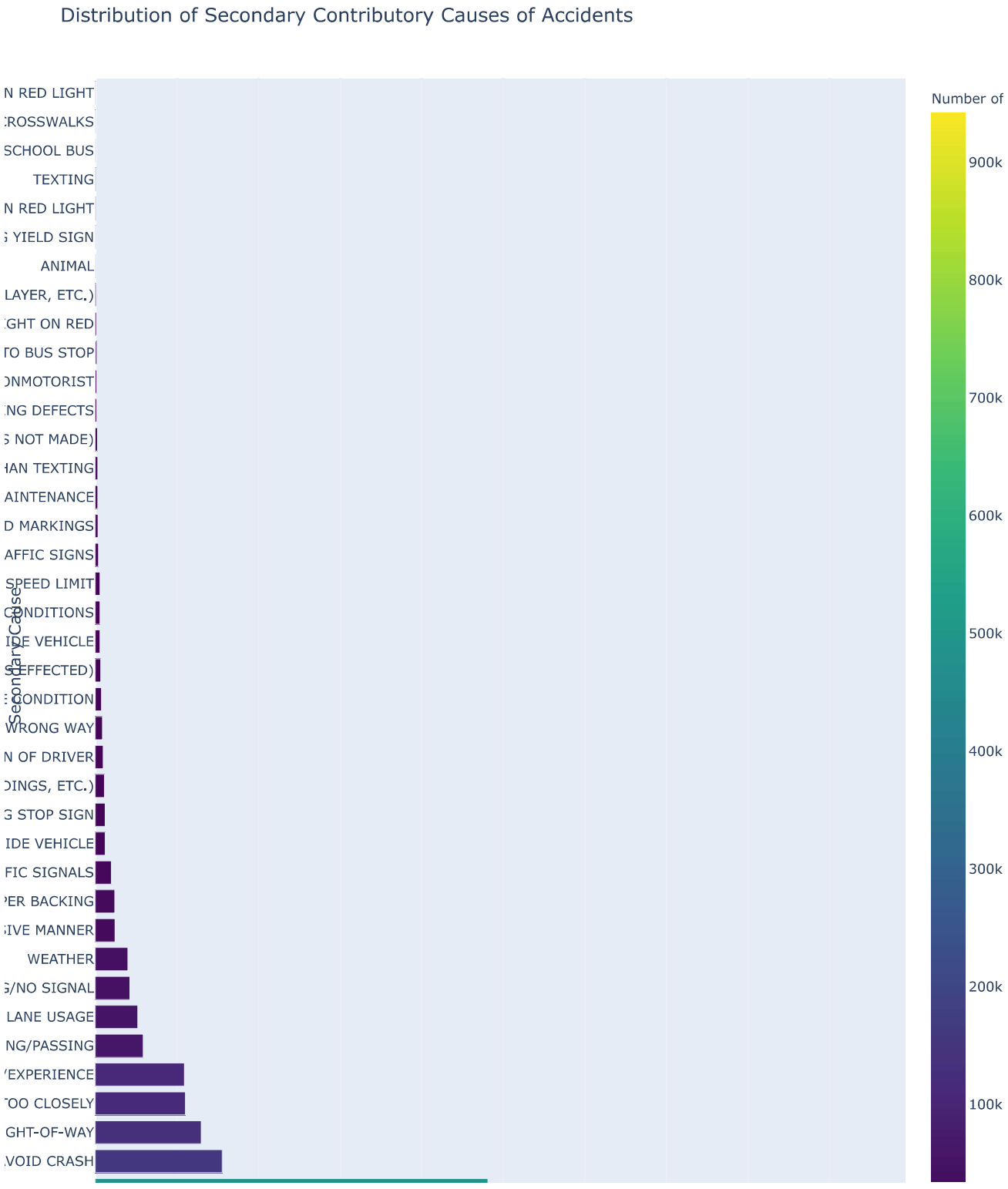
```
<Figure size 432x288 with 0 Axes>
```

Insight - Most accidents in Chicago are caused by failing to yield right of way followed by folowing too closely.

5.1.1.2 Most common secondary cause of accidents

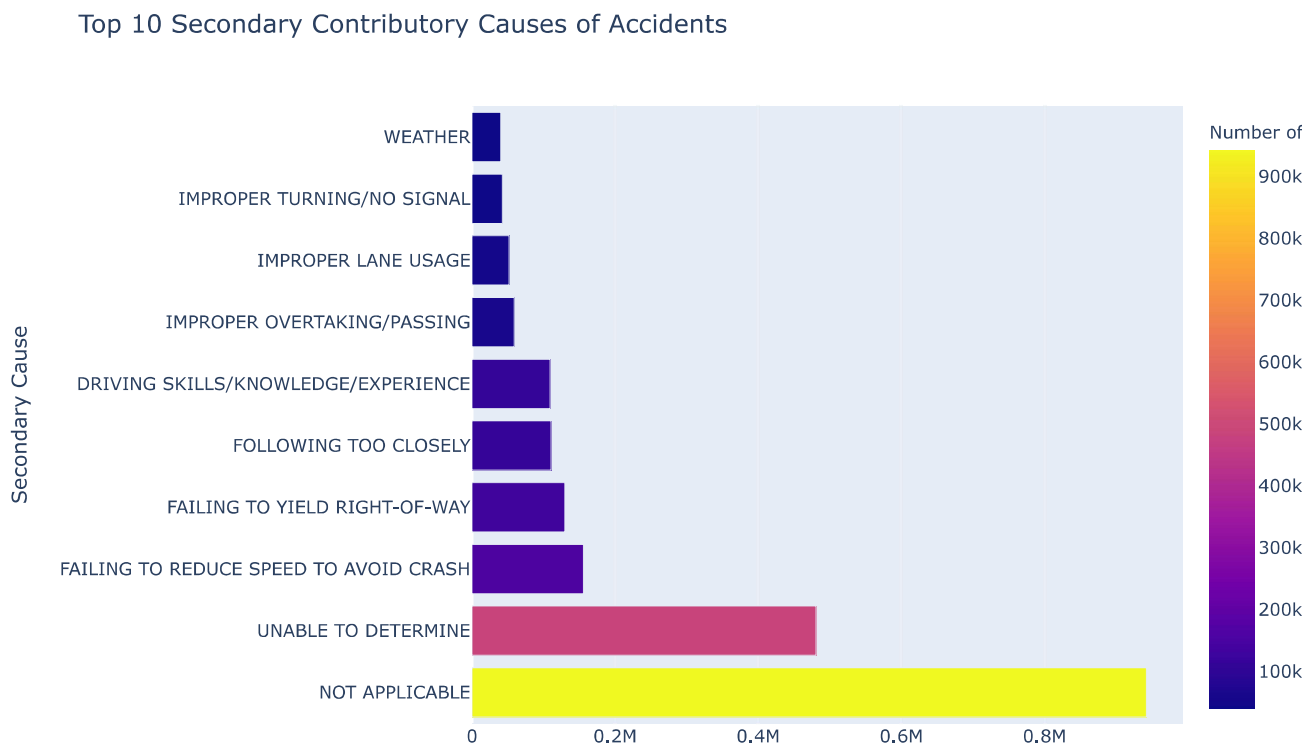
In [53]:

```
# Calculate the number of occurrences per category
secondary_cause_counts = project_df['SEC_CONTRIBUTORY_CAUSE'].value_counts().reset_index()
secondary_cause_counts.columns = ['Secondary Cause', 'Number of Occurrences']

# Plot for all secondary contributory causes
fig_total = px.bar(
    secondary_cause_counts,
    x='Number of Occurrences',
    y='Secondary Cause',
    orientation='h',
    title='Distribution of Secondary Contributory Causes of Accidents',
    color='Number of Occurrences',
    color_continuous_scale='Viridis',
    height=1200,
)

fig_total.update_layout(
    xaxis_title='Number of Occurrences',
    yaxis_title='Secondary Cause',
    showlegend=False
)

# Show the plot
fig_total.show()
```



The top ten are as shown below

In [54]:
```
# Calculate the number of occurrences per category and get the top 10
top_10_secondary_causes = project_df['SEC_CONTRIBUTORY_CAUSE'].value_counts().nlargest(10).reset_index()
top_10_secondary_causes.columns = ['Secondary Cause', 'Number of Occurrences']

# Create the bar plot using Plotly
fig = px.bar(
    top_10_secondary_causes,
    x='Number of Occurrences',
    y='Secondary Cause',
    orientation='h',  # Horizontal bars
    title='Top 10 Secondary Contributory Causes of Accidents',
    color='Number of Occurrences',
    color_continuous_scale='Plasma',
    height=600,
)

# Update layout
fig.update_layout(
    xaxis_title='Number of Occurrences',
    yaxis_title='Secondary Cause',
    showlegend=False
)

# Save the image
plt.savefig('Top_10_Secondary_Cause_of_Accidents.png')

# Show the plot
fig.show()
```



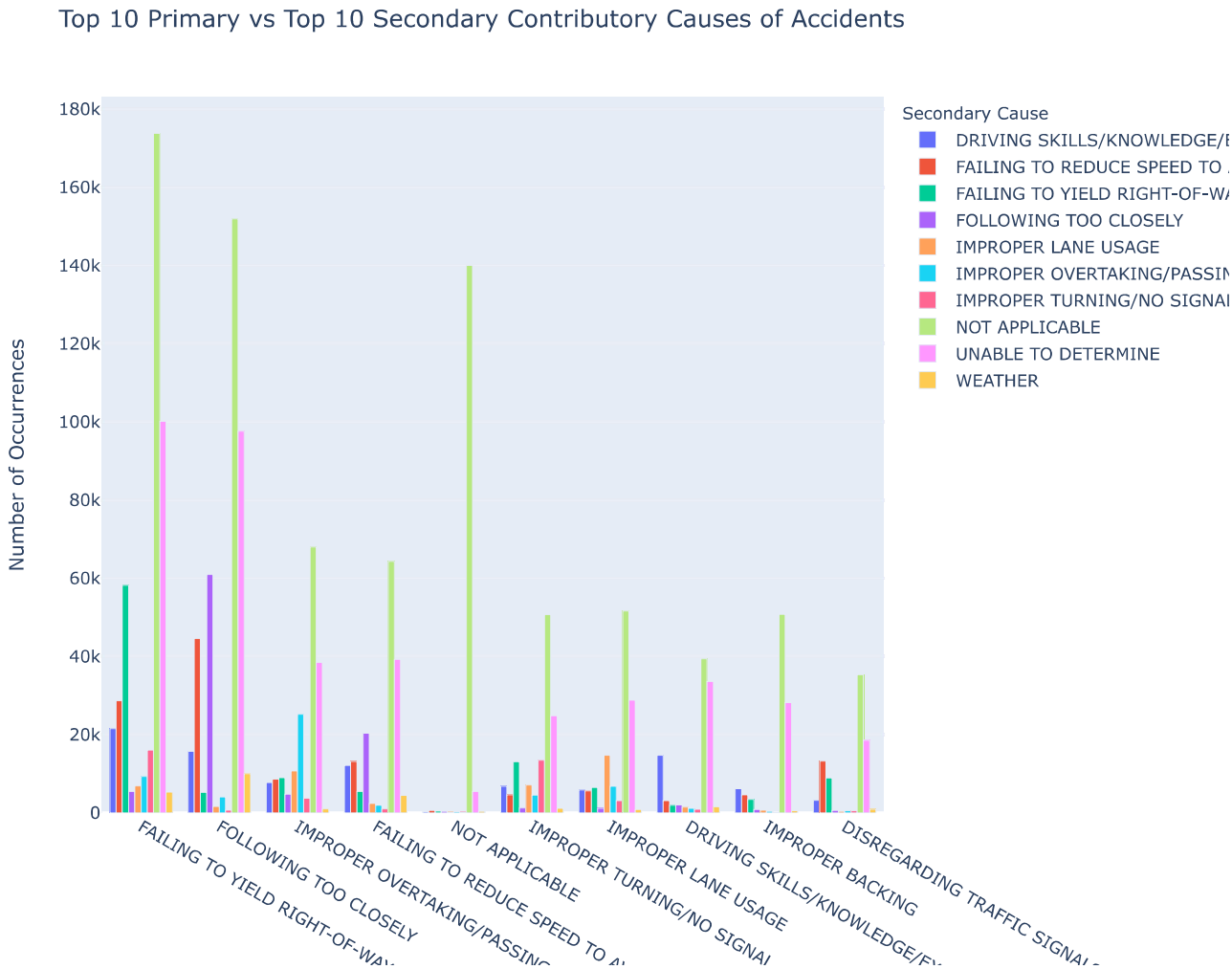

```
<Figure size 432x288 with 0 Axes>
```

Insight - Most accidents do not have a secondary cause or the secondary cause is usually not determined.

5.1.2 Bivariate Analysis

5.1.2.1 Primary and secondary causes

In [55]:
```
# Get the top 10 Primary and Secondary Contributory Causes
top_10_primary = project_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlargest(10).index
top_10_secondary = project_df['SEC_CONTRIBUTORY_CAUSE'].value_counts().nlargest(10).index

# Filter the DataFrame to include only rows with top 10 primary and secondary causes
top_10_df = project_df[
    (project_df['PRIM_CONTRIBUTORY_CAUSE'].isin(top_10_primary)) &
    (project_df['SEC_CONTRIBUTORY_CAUSE'].isin(top_10_secondary))
]

# Count the combinations of Primary and Secondary Causes
primary_secondary_counts = top_10_df.groupby(['PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE']).size().reset_in

# Create a bar plot
fig = px.bar(
    primary_secondary_counts,
    x='PRIM_CONTRIBUTORY_CAUSE',
    y='Count',
    color='SEC_CONTRIBUTORY_CAUSE',
    barmode='group',
    title='Top 10 Primary vs Top 10 Secondary Contributory Causes of Accidents',
    labels={
        'PRIM_CONTRIBUTORY_CAUSE': 'Primary Cause',
        'Count': 'Number of Occurrences',
        'SEC_CONTRIBUTORY_CAUSE': 'Secondary Cause'
    },
    height=800
)

# Update layout for better visibility
fig.update_layout(
    xaxis_title='Primary Cause',
    yaxis_title='Number of Occurrences',
    legend_title='Secondary Cause',
    xaxis={'categoryorder':'total descending'}
)

# Save the image
plt.savefig('Top_10_Primary_and_Secondary_Cause_of_Accidents.png')

# Show the plot
fig.show()
```




```
<Figure size 432x288 with 0 Axes>
```

Insight - In the top 10 primary causes, we can see that there was mostly no secondary cause or it wasn't determined.

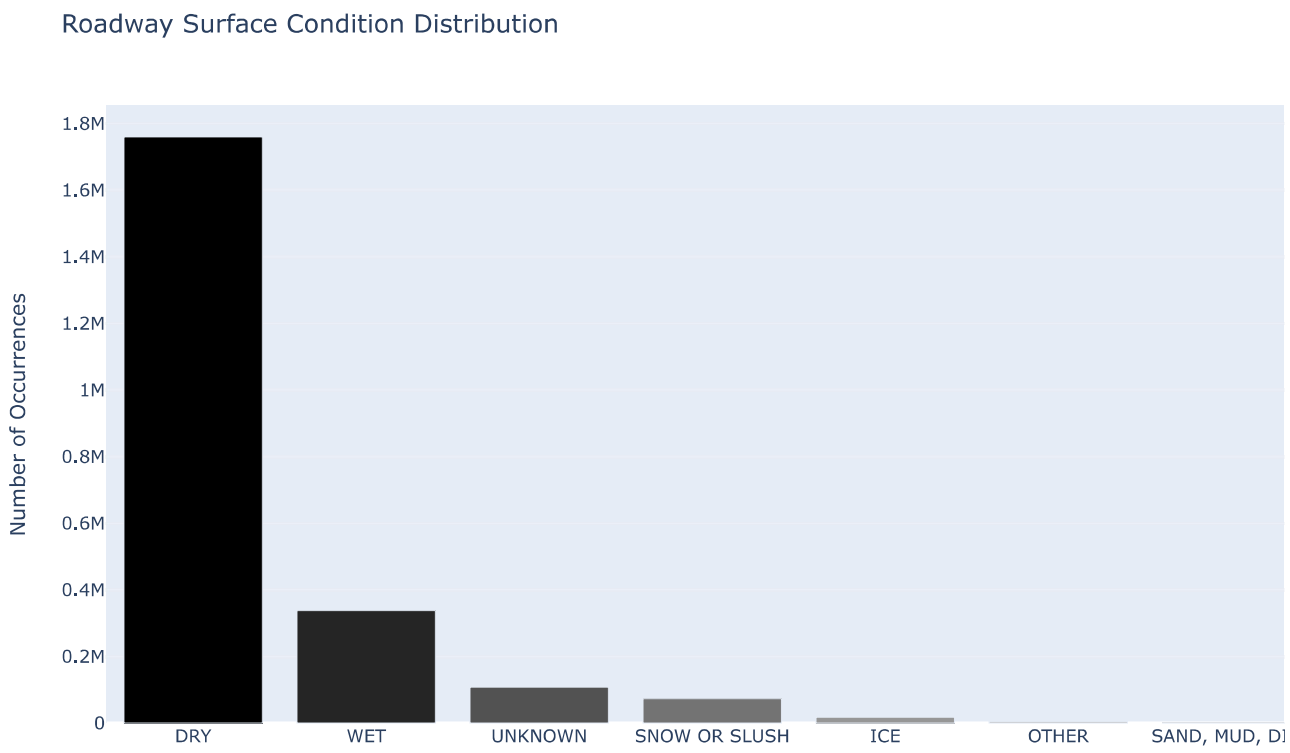
5.2 How do road and weather conditions impact the likelihood of different accident causes?

5.2.1 Univariate Analysis

5.2.1.1 Plot for Road Conditions

In [56]:

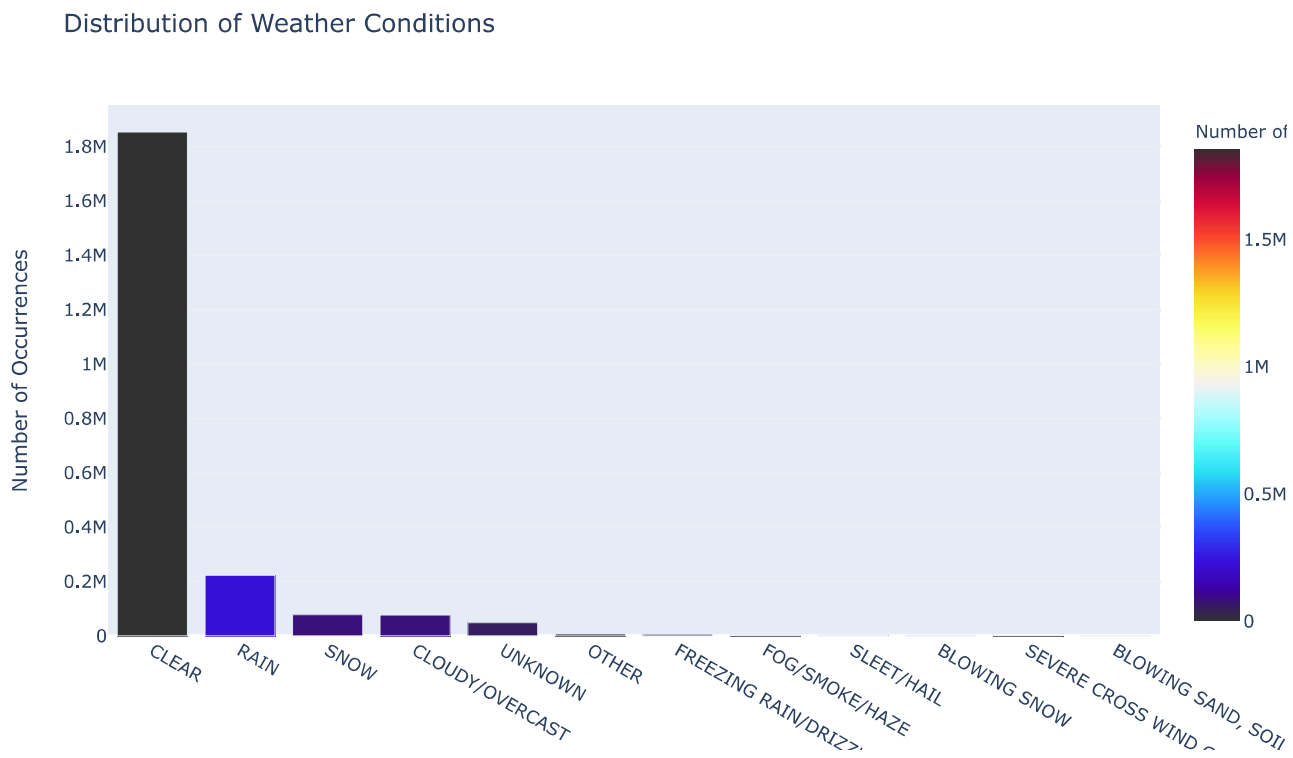
```
import plotly.express as px

# Count the occurrences of each Roadway Surface Condition
road_surface_counts = project_df['ROADWAY_SURFACE_COND'].value_counts().reset_index()
road_surface_counts.columns = ['Roadway Surface Condition', 'Count']

# Sort by Count to ensure the largest is first
road_surface_counts = road_surface_counts.sort_values(by='Count', ascending=False)

# Create the bar plot using the reversed Greys color palette
fig = px.bar(
    road_surface_counts,
    x='Roadway Surface Condition',
    y='Count',
    color='Roadway Surface Condition',
    color_discrete_sequence=px.colors.sequential.Greys[::-1],  # Reverse the Greys palette
    title='Roadway Surface Condition Distribution',
    labels={
        'Roadway Surface Condition': 'Road Surface Condition',
        'Count': 'Number of Occurrences'
    },
    height=600
)

# Update layout for better readability
fig.update_layout(
    xaxis_title='Road Surface Condition',
    yaxis_title='Number of Occurrences',
    xaxis={'categoryorder':'total descending'},
    showlegend=False  # Hide legend since color is repetitive
)


fig.show()
```



5.2.1.2 Insight - Most accidents occur on dry surfaces indicating more careless driving on dry surfaces.

5.2.1.3 Plot for Weather Conditions

In [57]:
```
# Count occurrences of each weather condition
weather_conditions_counts = project_df['WEATHER_CONDITION'].value_counts().reset_index()
weather_conditions_counts.columns = ['Weather Condition', 'Number of Occurrences']

# Create a bar plot for Weather Conditions
fig_weather = px.bar(
    weather_conditions_counts,
    x='Weather Condition',
    y='Number of Occurrences',
    title='Distribution of Weather Conditions',
    color='Number of Occurrences',
    color_continuous_scale='Edge',
    height=600
)

fig_weather.update_layout(
    xaxis_title='Weather Condition',
    yaxis_title='Number of Occurrences',
    showlegend=False,
    xaxis={'categoryorder':'total descending'}
)

# Save the image
plt.savefig('Weather_Condition.png')

fig_weather.show()
```



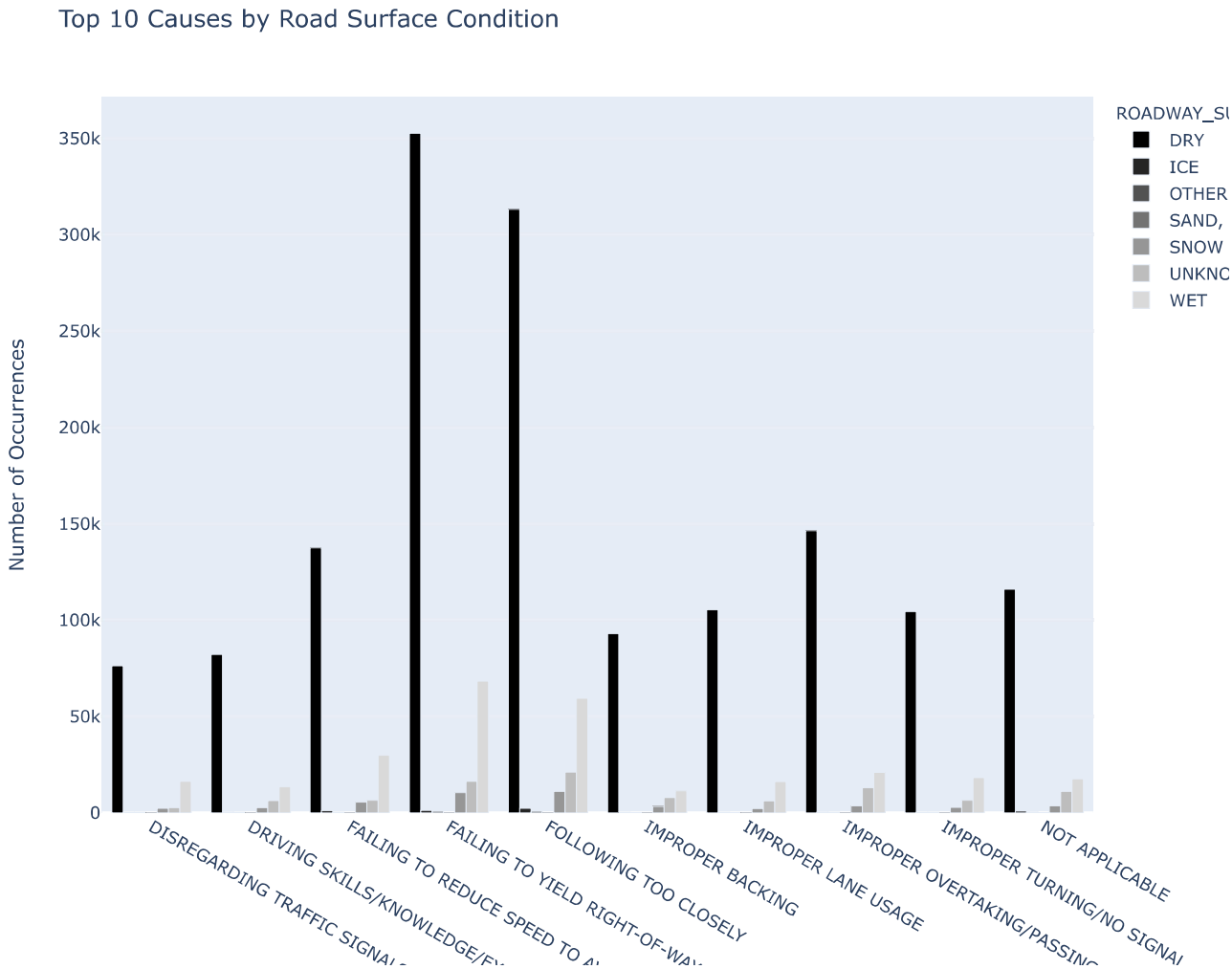

```
<Figure size 432x288 with 0 Axes>
```

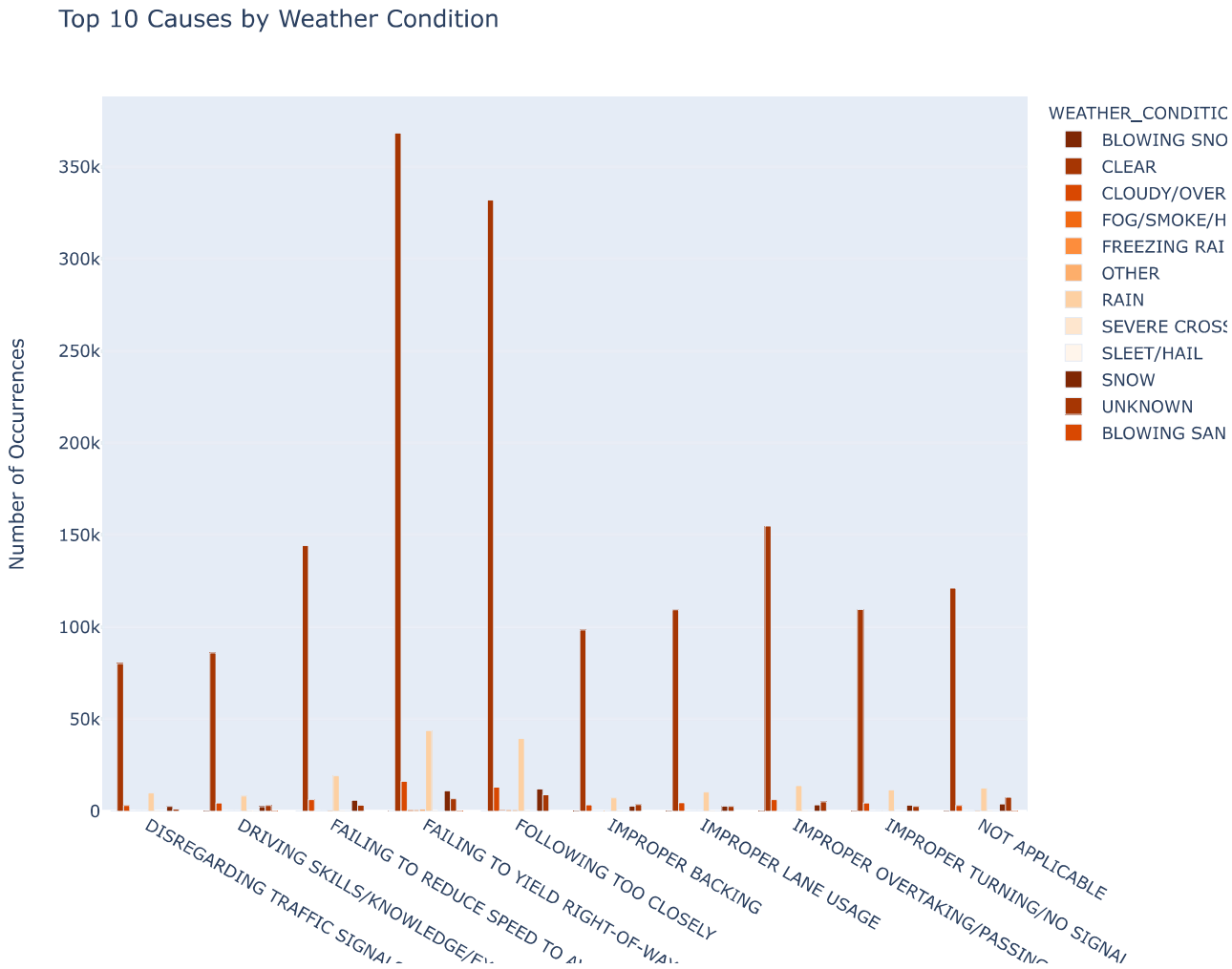
5.2.1.4 Insight - Most accidents occur on clear weather indicating more careless driving in clear conditions.

5.2.2 Bivariate Analysis

5.2.2.1 Plot a bar chart of the top 10 primary contributory causes against road surface and weather conditions.

In [58]:
```
# Get the top 10 primary contributory causes
top_10_causes = project_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlargest(10).index

# Filter the data for the top 10 causes
filtered_df = project_df[project_df['PRIM_CONTRIBUTORY_CAUSE'].isin(top_10_causes)]

# Group by cause and road surface condition
road_surface_df = filtered_df.groupby(['PRIM_CONTRIBUTORY_CAUSE', 'ROADWAY_SURFACE_COND']).size().reset_index(name=

# Group by cause and weather condition
weather_df = filtered_df.groupby(['PRIM_CONTRIBUTORY_CAUSE', 'WEATHER_CONDITION']).size().reset_index(name='Count')

# Plot for Road Surface Condition
fig1 = px.bar(
    road_surface_df,
    x='PRIM_CONTRIBUTORY_CAUSE',
    y='Count',
    color='ROADWAY_SURFACE_COND',
    title='Top 10 Causes by Road Surface Condition',
    color_discrete_sequence=px.colors.sequential.Greys[::-1],  # Darkest for the largest value
    height=800
)

fig1.update_layout(barmode='group', xaxis_title='Primary Cause', yaxis_title='Number of Occurrences')
fig1.show()

# Plot for Weather Condition
fig2 = px.bar(
    weather_df,
    x='PRIM_CONTRIBUTORY_CAUSE',
    y='Count',
    color='WEATHER_CONDITION',
    title='Top 10 Causes by Weather Condition',
    color_discrete_sequence=px.colors.sequential.Oranges[::-1],
    height=800  # Darkest for the largest value
)

fig2.update_layout(barmode='group', xaxis_title='Primary Cause', yaxis_title='Number of Occurrences')

# Save the image
plt.savefig('Top_10_Causes_Against_Road_surface_and_Weather_Condition.png')


fig2.show()
```


Top 10 Causes by Road Surface Condition
Number of Occurrences
350k
300k
250k
200k
150k
100k
50k
0
ROADWAY_SU
DRY
ICE
OTHER
SAND,
SNOW
UNKNC
WET
DISREGARDING TRAFFIC SIGNAL
DRIVING SKILLS/KNOWLEDGE/EX
FAILING TO REDUCE SPEED TO A
FAILING TO YIELD RIGHT-OF-WAY
FOLLOWING TOO CLOSELY
IMPROPER BACKING
IMPROPER LANE USAGE
IMPROPER OVERTAKING/PASSING
IMPROPER TURNING/NO SIGNAL
NOT APPLICABLE




```
<Figure size 432x288 with 0 Axes>
```

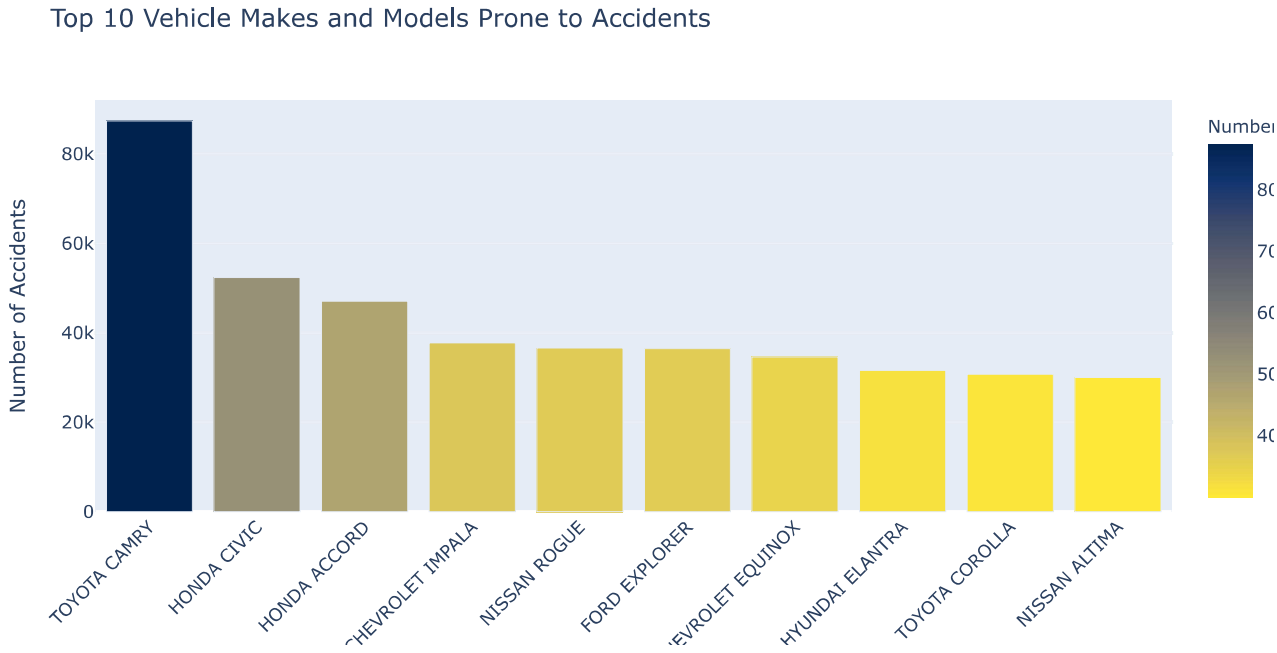
5.2.2.2 Insight - we can see the most common causes of accidents also come about on dry roads and clears weather.

5.3 Do certain car models or vehicle types have a higher risk of specific accident causes?

5.3.1 Univariate Analysis

5.3.1.1 Plot top 10 Vehicle make and model prone to accidents

In [59]:
```
# Count occurrences of each vehicle type
vehicle_count_df = project_df['MAKE_MODEL'].value_counts().reset_index(name='Count').rename(columns={'index': 'MAKE_

# Get the top 10 most accident-prone vehicle types
top_vehicles_df = vehicle_count_df.head(10)

# Plot using Plotly
fig = px.bar(
    top_vehicles_df,
    x='MAKE_MODEL',
    y='Count',
    title='Top 10 Vehicle Makes and Models Prone to Accidents',
    labels={'MAKE_MODEL': 'Vehicle Make and Model', 'Count': 'Number of Accidents'},
    color='Count',
    color_continuous_scale=px.colors.sequential.Cividis[::-1]  # Darker shades for higher counts
)
fig.update_layout(xaxis={'categoryorder': 'total descending'}, xaxis_tickangle=-45)

plt.savefig('Top_10_Vehicle_models.png')

fig.show()
```




```
<Figure size 432x288 with 0 Axes>
```

5.3.1.2 Insight the top 10 vehicles most prone to accidents are

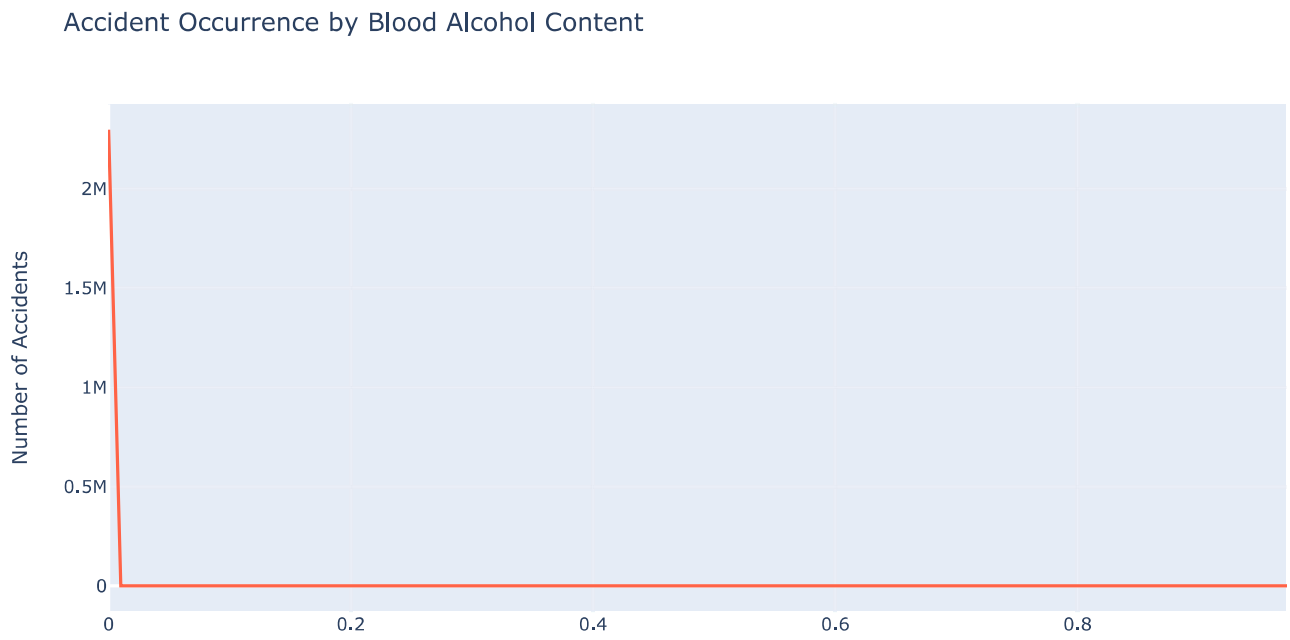
1. TOYOTA CAMRY with 87,396 accidents reported
2. HONDA CIVIC with 52,375 accidents reported
3. HONDA ACCORD with 47,035 accidents reported
4. CHEVROLET IMPALA with 37,739 accidents reported
5. NISSAN ROGUE with 36,619 accidents reported
6. FORD EXPLORER with 36,474 accidents reported
7. CHEVROLET EQUINOX with 34,638 accidents reported
8. HYUNDAI ELANTRA with 31,584 accidents reported
9. TOYOTA COROLLA with 30,719 accidents reported
10. NISSAN ALTIMA with 30,009 accidents reported

5.4 Is blood alcohol content a major factor in accident occurence?

5.4.1 Univariate Analysis

5.4.1.1 Plot BAC against accident frequency

In [60]:
```
# Grouping accidents by BAC levels
bac_df = project_df.groupby('BAC_RESULT VALUE').size().reset_index(name='Count')

# Sorting by BAC level for a better line plot
bac_df = bac_df.sort_values(by='BAC_RESULT VALUE')

# Creating the line plot
fig = px.line(
    bac_df,
    x='BAC_RESULT VALUE',
    y='Count',
    title='Accident Occurrence by Blood Alcohol Content',
    labels={'BAC_RESULT VALUE': 'Blood Alcohol Content', 'Count': 'Number of Accidents'},
    color_discrete_sequence=['#FF6347']  # Using a red shade to highlight risk
)

plt.savefig('Blood_Alcohol_Content_at_time_of_accident.png')

fig.show()
```

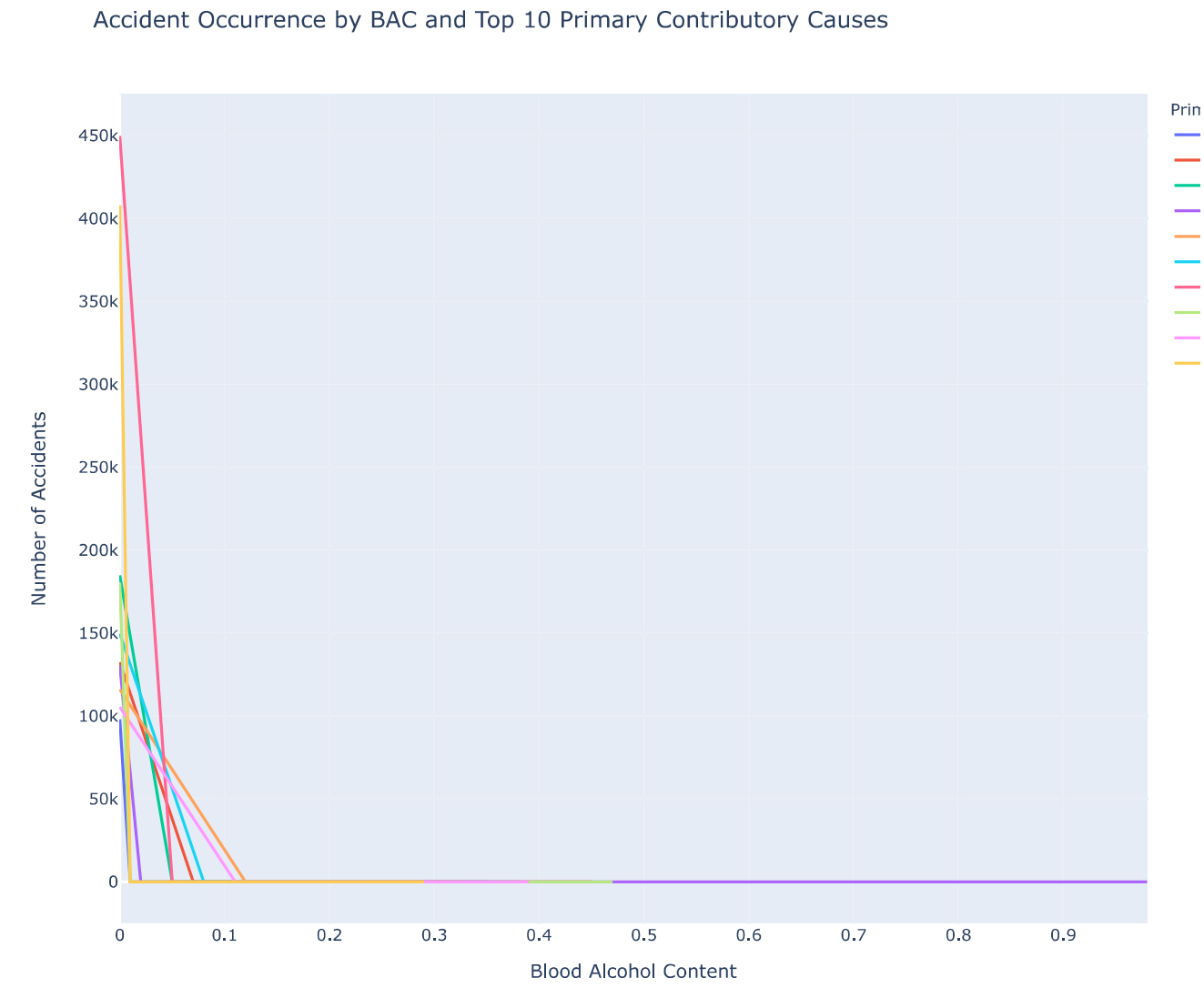



```
<Figure size 432x288 with 0 Axes>
```

5.4.1.2 Insight - Blood Alcohol Content at time of most accidents was below the general allowable limit so it is not a huge contributory factor

5.4.1.3 Plot BAC against PRIM_CONTIBUTORY_CAUSE.

In [61]:
```
# Get the top 10 primary causes by occurrence
top_10_causes = project_df['PRIM_CONTRIBUTORY_CAUSE'].value_counts().head(10).index

# Filter the DataFrame to include only the top 10 causes
top_bac_cause_df = project_df[project_df['PRIM_CONTRIBUTORY_CAUSE'].isin(top_10_causes)]

# Group by BAC level and Primary Contributory Cause
bac_cause_df = top_bac_cause_df.groupby(['BAC_RESULT VALUE', 'PRIM_CONTRIBUTORY_CAUSE']).size().reset_index(name='C

# Sort by BAC level for better line progression
bac_cause_df = bac_cause_df.sort_values(by='BAC_RESULT VALUE')

# Create the line plot
fig = px.line(
    bac_cause_df,
    x='BAC_RESULT VALUE',
    y='Count',
    color='PRIM_CONTRIBUTORY_CAUSE',
    title='Accident Occurrence by BAC and Top 10 Primary Contributory Causes',
    height=800,
    width=1200,
    labels={
        'BAC_RESULT VALUE': 'Blood Alcohol Content',
        'Count': 'Number of Accidents',
        'PRIM_CONTRIBUTORY_CAUSE': 'Primary Cause'
    }
)

# Save the plot as an image
plt.savefig('Blood_Alcohol_Content_vs_Top10_PRIM_CONTRIBUTORY_CAUSE.png')

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



<Figure size 432x288 with 0 Axes>

5.4.1.4 Insight - BAC generallly has a negative correlation to the top 10 primary accident causes therefore likely has very little effect on accident occurence.

6 MODELING

Modeling is a crucial phase in any data science project, where the goal is to develop a mathematical or computational representation of the underlying patterns within the data. This phase involves selecting the appropriate algorithms, training the models, and evaluating their performance to ensure they generalize well to unseen data.

6.0.1 Why is Modeling Important?

1. Prediction and Inference: Models help in predicting future outcomes and inferring relationships within the data.
2. Decision-Making: Accurate models support data-driven decisions.
3. Optimization: Models can be used to optimize processes or outcomes.

6.1 Modeling primary accident cause using a decision tree classifier.

6.1.1 Begin with a base decision tree classifier model using Gini impurity.

In [ ]:
```
# Create Pipeline for Decision Tree with Gini
base_decision_tree_pipeline_gini = Pipeline([
    ('classifier', DecisionTreeClassifier(criterion='gini', random_state=42))  # Model
])

# Train the model with Gini
base_decision_tree_pipeline_gini.fit(X_train, y_train)
```

Out[63]:
```
Pipeline(steps=[('classifier', DecisionTreeClassifier(random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Check the Precision, Recall, F1_score, AUC and Accuracy.

In [ ]:
```
# Predictions
y_pred_decision_tree_gini = base_decision_tree_pipeline_gini.predict(X_test)

# Evaluate the model with Gini
accuracy_decision_tree_gini = accuracy_score(y_test, y_pred_decision_tree_gini)
# Precision, Recall, F1-Score, and AUC for Gini
precision_gini = precision_score(y_test, y_pred_decision_tree_gini, average='weighted')
recall_gini = recall_score(y_test, y_pred_decision_tree_gini, average='weighted')
f1_gini = f1_score(y_test, y_pred_decision_tree_gini, average='weighted')
auc_gini = roc_auc_score(pd.get_dummies(y_test), pd.get_dummies(y_pred_decision_tree_gini), multi_class='ovr')
print(f"Precision (Gini): {precision_gini*100:.2f}%")
print(f"Recall (Gini): {recall_gini*100:.2f}%")
print(f"F1-Score (Gini): {f1_gini*100:.2f}%")
print(f"AUC (Gini): {auc_gini*100:.2f}%")
print(f"Base Decision Tree (Gini) Accuracy: {accuracy_decision_tree_gini * 100:.2f}%")
```

```
Precision (Gini): 69.56%
Recall (Gini): 69.50%
F1-Score (Gini): 69.51%
AUC (Gini): 77.38%
Base Decision Tree (Gini) Accuracy: 69.50%
```

The base metrics are:

1. Precision (Gini): 69.56%
2. Recall (Gini): 69.50%
3. F1-Score (Gini): 69.51%
4. AUC (Gini): 77.38%
5. Base Decision Tree (Gini) Accuracy: 69.50%

From the above metrics we can conclude:

1. Moderate Predictive Performance The Precision, Recall, and F1-Score are all around 69-70%, indicating moderate performance. The model correctly identifies the primary contributory cause of a car accident in about 70% of the cases. However, this also means there's a 30% error rate, which may not be sufficient for critical decision-making scenarios (e.g., safety interventions).
2. Balanced Precision and Recall Precision (69.56%) and Recall (69.50%) are nearly identical, suggesting a balanced trade-off between false positives and false negatives. This indicates that the model is equally good at identifying the true causes and avoiding incorrect predictions. However, this balance comes at the cost of lower overall accuracy, reflecting moderate reliability.
3. F1-Score Indicates Scope for Improvement An F1-Score of 69.51% confirms that the model maintains a balance between precision and recall. However, the score is relatively low, suggesting the need for further improvement in the model's predictive power. The model may be missing some key patterns or suffering from underfitting due to its simplicity.
4. AUC Indicates Limited Discrimination Ability An AUC of 77.38% is moderately good but indicates that the model struggles to distinguish between classes. It suggests that the model is only somewhat effective at ranking predictions, leading to less confidence in decision boundaries. This could be problematic in identifying the exact cause of an accident, especially in complex scenarios.
5. Underfitting and Model Complexity The model's relatively low accuracy (69.50%) and moderate AUC suggest underfitting. This is likely due to the simplicity of the base Decision Tree. It may not be capturing the complex interactions between features e.g. car type, weather conditions, and road characteristics.

6.1.2 Proceed to test the decision tree classifier with Entropy impurity.

In [ ]:
```
# Create Pipeline for Decision Tree with Entropy
base_dt_pipeline_entropy = Pipeline([
    ('classifier', DecisionTreeClassifier(criterion='entropy', random_state=42))  # Model
])

# Train the model with Entropy
base_dt_pipeline_entropy.fit(X_train, y_train)
```

Out[119]:
```
Pipeline(steps=[('classifier',
                 DecisionTreeClassifier(criterion='entropy', random_state=42))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Check the performance metrics for the entropy impurity decision tree.

In [ ]:
```
# Predictions
y_pred_decision_tree_entropy = base_dt_pipeline_entropy.predict(X_test)

# Evaluate the model with Entropy
accuracy_dt_entropy = accuracy_score(y_test, y_pred_decision_tree_entropy)
precision_entropy = precision_score(y_test, y_pred_decision_tree_entropy, average='weighted')
recall_entropy = recall_score(y_test, y_pred_decision_tree_entropy, average='weighted')
f1_entropy = f1_score(y_test, y_pred_decision_tree_entropy, average='weighted')
auc_entropy = roc_auc_score(pd.get_dummies(y_test), pd.get_dummies(y_pred_decision_tree_entropy), multi_class='ovr'
print(f"Precision (Entropy): {precision_entropy*100:.2f}%")
print(f"Recall (Entropy): {recall_entropy*100:.2f}%")
print(f"F1-Score (Entropy): {f1_entropy*100:.2f}%")
print(f"AUC (Entropy): {auc_entropy*100:.2f}%")
print(f"Base Decision Tree (Entropy) Accuracy: {accuracy_dt_entropy * 100:.2f}%")
```

```
Precision (Entropy): 77.17%
Recall (Entropy): 77.19%
F1-Score (Entropy): 77.16%
AUC (Entropy): 84.02%
Base Decision Tree (Entropy) Accuracy: 77.19%
```

The decision tree with entropy has better metrics. To be specific, precision and accuracy have improved sifgnificantly meaning the entropy model is performng better in predicting on unseen data.

1. Precision (Entropy): 77.17%
2. Recall (Entropy): 77.19%

3. F1-Score (Entropy): 77.16%
4. AUC (Entropy): 84.02%
5. Base Decision Tree (Entropy) Accuracy: 77.19%.

These metrics imply:

1. Improved Predictive Performance The Precision, Recall, and F1-Score are all above 77%, indicating good performance. The model correctly identifies the primary contributory cause of a car accident in about 77% of the cases. This demonstrates a higher accuracy and reliability compared to the Gini-based Decision Tree.
2. Balanced Precision and Recall Precision (77.17%) and Recall (77.19%) are nearly identical, indicating a balanced trade-off between false positives and false negatives. The model is equally effective at identifying true causes and avoiding incorrect predictions. This balance is crucial for scenarios where both false positives and false negatives are costly, such as accident prevention.
3. Higher F1-Score Indicates Robustness An F1-Score of 77.16% suggests a good balance between precision and recall, confirming consistent model performance. This is a significant improvement over the Gini-based Decision Tree, reflecting better overall predictive power. The model is effectively capturing patterns in the data, leading to more reliable predictions.
4. AUC Indicates Strong Discrimination Ability An AUC of 84.02% shows the model has a good ability to distinguish between classes. This indicates that the model can confidently rank predictions, leading to more accurate decision boundaries. It also suggests that the model is less likely to be confused by overlapping class distributions, making it more robust.
5. Entropy Split Criterion Advantage The improved performance is likely due to the Entropy split criterion, which tends to be more sensitive to mixed classes. Entropy focuses on information gain, making it better at identifying informative splits and capturing complex patterns in the data. This may explain why the Entropy-based model outperforms the Gini-based model in both accuracy and AUC.
6. Indication of Better Generalization The relatively high AUC and balanced Precision and Recall indicate that the model is generalizing well to unseen data. It is less prone to overfitting compared to a deeper, more complex tree, maintaining consistent performance.
7. Good Baseline but Potential for Improvement Although the model shows good performance, further improvements can be explored.

Hyperparameter tune it to boost peformance.

6.1.3. Begin with GridsearchCV hyperparameter tuning

In [ ]:
```
# Hyperparameter space for GridSearchCV
param_grid_entropy = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4],
    'classifier__splitter': ['best', 'random']
}

# Grid Search with Stratified K-Fold cross-validation
grid_search_entropy = GridSearchCV(
    base_dt_pipeline_entropy,
    param_grid=param_grid_entropy,
    cv=StratifiedKFold(n_splits=5),
    n_jobs=-1,
    scoring='accuracy',
    verbose=2
)

# Fitting the model
grid_search_entropy.fit(X_train, y_train)

# Best Parameters
print('Best Parameters for Entropy:', grid_search_entropy.best_params_)

# Best Model
best_dt_entropy = grid_search_entropy.best_estimator_

# Predict on test set
y_pred_entropy = best_dt_entropy.predict(X_test)

# Evaluate the model
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
precision_entropy = precision_score(y_test, y_pred_entropy, average='weighted')
recall_entropy = recall_score(y_test, y_pred_entropy, average='weighted')
f1_entropy = f1_score(y_test, y_pred_entropy, average='weighted')
auc_entropy = roc_auc_score(pd.get_dummies(y_test), pd.get_dummies(y_pred_entropy), multi_class='ovr')
print(f"Accuracy (Decision Tree - Entropy): {accuracy_entropy * 100:.2f}%")
print(f"Precision (Decision Tree - Entropy): {precision_entropy * 100:.2f}%)")
print(f"Recall (Decision Tree - Entropy): {recall_entropy * 100:.2f}%")
print(f"F1-Score (Decision Tree - Entropy): {f1_entropy * 100:.2f}%")
print(f"AUC (Decision Tree - Entropy): {auc_entropy * 100:.2f}%")
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_split.py:805: UserWarning:

The least populated class in y has only 1 members, which is less than n_splits=5.


Best Parameters for Entropy: {'classifier__max_depth': None, 'classifier__min_samples_leaf': 1, 'classifier__min_s
amples_split': 2, 'classifier__splitter': 'random'}

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.


Accuracy (Decision Tree - Entropy): 74.74%
Precision (Decision Tree - Entropy): 74.81%)
Recall (Decision Tree - Entropy): 74.74%
F1-Score (Decision Tree - Entropy): 74.75%
AUC (Decision Tree - Entropy): 50.01%
```

1. Accuracy (Decision Tree - Entropy): 74.74%
2. Precision (Decision Tree - Entropy): 74.81%)
3. Recall (Decision Tree - Entropy): 74.74%
4. F1-Score (Decision Tree - Entropy): 74.75%
5. (Decision Tree - Entropy): 50.01%

Metrics from grid search hyperparameter tuning shows:

1. Consistent but Moderate Performance Accuracy, Precision, Recall, and F1-Score are all approximately 74.7%, showing consistent but moderate predictive power. The model is correctly predicting the contributory cause in about three-quarters of the cases, but the

performance is not exceptional. This indicates the model is not overfitting, but it also suggests limited improvement despite hyperparameter tuning.

2. Balanced Precision and Recall Precision and Recall are nearly identical, which means the model is: Equally good at identifying true positive causes and avoiding false positives. Not biased towards any particular class, maintaining balanced predictions. This balance is beneficial for applications where both false positives and false negatives are equally undesirable.
3. Moderate F1-Score The F1-Score of 74.75% reflects the harmonic mean of Precision and Recall, confirming a balanced performance. It indicates that the model is not favoring precision over recall or vice versa. However, the moderate F1-Score suggests there is still room for improvement in overall predictive capability.
4. Low AUC Indicates Poor Class Separation An AUC of 50.01% is just marginally better than random guessing (which would be 50%). This indicates the model has very limited ability to distinguish between classes. It suggests that the decision tree is struggling to define clear decision boundaries, leading to overlapping class predictions.
5. Impact of Hyperparameter Tuning Despite using Grid Search for hyperparameter tuning, the model did not achieve a significant improvement in performance. This may indicate that the Entropy split criterion is not sufficiently capturing complex patterns in the dataset. Alternatively, it suggests the need for more advanced model architectures or additional feature engineering.
6. Possible Causes for Low AUC Imbalanced Classes: If the dataset is imbalanced, the model might be biased towards majority classes, leading to poor AUC. Insufficient Complexity: Decision Trees may lack the complexity needed to model intricate patterns in the data. Feature Interactions: The model might be missing important feature interactions, leading to poor class separability.

The gridsearch hyperparameter tuned model metrics are showing decreased performance from the original model.

In [ ]:
```
# Hyperparameter space for RandomizedSearchCV
param_dist_entropy = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4],
    'classifier__splitter': ['best', 'random']
}

# Randomized Search with Stratified K-Fold cross-validation
random_search_entropy = RandomizedSearchCV(
    base_dt_pipeline_entropy,
    param_distributions=param_dist_entropy,
    n_iter=20,  # Number of random combinations to try
    cv=StratifiedKFold(n_splits=5),
    n_jobs=-1,
    random_state=42
)
# Fitting the model
random_search_entropy.fit(X_train, y_train)
print('Best Parameters for Entropy:', random_search_entropy.best_params_)

# Best Model
best_dt_entropy_random =random_search_entropy.best_estimator_

# Predict on test set
y_pred_entropy = best_dt_entropy_random.predict(X_test)

# Evaluate the model
accuracy_entropy = accuracy_score(y_test, y_pred_entropy)
precision_entropy = precision_score(y_test, y_pred_entropy, average='weighted')
recall_entropy = recall_score(y_test, y_pred_entropy, average='weighted')
f1_entropy = f1_score(y_test, y_pred_entropy, average='weighted')
auc_entropy = roc_auc_score(pd.get_dummies(y_test), pd.get_dummies(y_pred_entropy), multi_class='ovr')
print(f"Accuracy (Decision Tree - Entropy): {accuracy_entropy * 100:.2f}%")
print(f"Precision (Decision Tree - Entropy): {precision_entropy * 100:.2f}%)")
print(f"Recall (Decision Tree - Entropy): {recall_entropy * 100:.2f}%")
print(f"F1-Score (Decision Tree - Entropy): {f1_entropy * 100:.2f}%")
print(f"AUC (Decision Tree - Entropy): {auc_entropy * 100:.2f}%")
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_split.py:805: UserWarning:

The least populated class in y has only 1 members, which is less than n_splits=5.


Best Parameters for Entropy: {'classifier__splitter': 'best', 'classifier__min_samples_split': 2, 'classifier__min
_samples_leaf': 1, 'classifier__max_depth': None}
Accuracy (Decision Tree - Entropy): 77.19%
Precision (Decision Tree - Entropy): 77.17%)
Recall (Decision Tree - Entropy): 77.19%
F1-Score (Decision Tree - Entropy): 77.16%
AUC (Decision Tree - Entropy): 84.02%
```

1. Accuracy (Decision Tree - Entropy): 77.19%

2. Precision (Decision Tree - Entropy): 77.17%)
3. Recall (Decision Tree - Entropy): 77.19%
4. F1-Score (Decision Tree - Entropy): 77.16%
5. AUC (Decision Tree - Entropy): 84.02%

Maintained same performance metrics as Base Decision tree classifier with entropy.

6.1.4.1 Model Strength

1. **Strengths**: High Accuracy and balanced Precision and Recall make it reliable and consistent. Good AUC performance shows effective class discrimination. Interpretable model with a clear decision-making process, suitable for stakeholder communication.

6.1.5 Test a logistic regression Classifier

In [ ]:
```
# Logistic Regression
lr_model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
y_prob_lr = lr_model.predict_proba(X_test)  # For AUC calculation

# Precision, Recall, F1-Score, and AUC
precision_lr = precision_score(y_test, y_pred_lr, average='weighted')
recall_lr = recall_score(y_test, y_pred_lr, average='weighted')
f1_lr = f1_score(y_test, y_pred_lr, average='weighted')
auc_lr = roc_auc_score(pd.get_dummies(y_test), y_prob_lr, multi_class='ovr')

# Display Metrics
print(f"Precision (Logistic Regression): {precision_lr * 100:.2f}%")
print(f"Recall (Logistic Regression): {recall_lr * 100:.2f}%")
print(f"F1-Score (Logistic Regression): {f1_lr * 100:.2f}%")
print(f"AUC (Logistic Regression): {auc_lr * 100:.2f}%")

# Accuracy (for reference)
lr_accuracy = accuracy_score(y_test, y_pred_lr)
print(f'Logistic Regression Accuracy: {lr_accuracy * 100:.2f}%')
```

```
Training LogisticRegression...

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning:

'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinom
ial'. Leave it to its default value to avoid this warning.

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.


Precision (Logistic Regression): 33.08%
Recall (Logistic Regression): 38.17%
F1-Score (Logistic Regression): 31.87%
AUC (Logistic Regression): 82.58%
Logistic Regression Accuracy: 38.17%
```

Logistic Regression is not the best choice due to its low accuracy and F1-Score. Threshold tuning and class imbalance handling could slightly improve performance but are unlikely to close the gap with advanced models.

In [ ]:
```
# Random Forest
rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
y_prob_rf = rf_model.predict_proba(X_test)  # For AUC calculation

# Precision, Recall, F1-Score, and AUC
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')
auc_rf = roc_auc_score(pd.get_dummies(y_test), y_prob_rf, multi_class='ovr')

# Display Metrics
print(f"Precision (RandomForest): {precision_rf * 100:.2f}%")
print(f"Recall (RandomForest): {recall_rf * 100:.2f}%")
print(f"F1-Score (RandomForest): {f1_rf * 100:.2f}%")
print(f"AUC (RandomForest): {auc_rf * 100:.2f}%")

# Accuracy (for reference)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Accuracy: {rf_accuracy * 100:.2f}%')
```

```
Training RandomForest...

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter t
o control this behavior.


Precision (RandomForest): 57.99%
Recall (RandomForest): 50.57%
F1-Score (RandomForest): 44.79%
AUC (RandomForest): 92.58%
Random Forest Accuracy: 50.57%
```

The moderate accuracy and low F1-Score indicate Random Forests are not effectively handling class imbalance or complex decision boundaries. However, the high AUC suggests that optimizing the thresholds and tuning hyperparameters could significantly improve performance. The model could be deployed as an initial baseline but XGBoost may offer better accuracy and reliability.

In [ ]:
```
# XGBoost
xgb_model = XGBClassifier( eval_metric='logloss')
xgb_model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning:

[19:27:20] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.
```

Out[113]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
xgb_preds = xgb_model.predict(X_test)
xgb_probs = xgb_model.predict_proba(X_test)[:, 1]
```

In [ ]:

```
# XGBoost Metrics
precision_xgb = precision_score(y_test, xgb_preds, average='weighted', zero_division=0)
recall_xgb = recall_score(y_test, xgb_preds, average='weighted')
f1_xgb = f1_score(y_test, xgb_preds, average='weighted')
auc_xgb = roc_auc_score(y_test, xgb_model.predict_proba(X_test), multi_class='ovr')
accuracy_xgb = accuracy_score(y_test, xgb_preds)

print(f"Precision (XGBoost): {precision_xgb * 100:.2f}%")
print(f"Recall (XGBoost): {recall_xgb * 100:.2f}%")
print(f"F1-Score (XGBoost): {f1_xgb * 100:.2f}%")
print(f"AUC (XGBoost): {auc_xgb * 100:.2f}%")
print(f"XGBoost Accuracy: {accuracy_xgb * 100:.2f}%")
```

```
Precision (XGBoost): 76.83%
Recall (XGBoost): 76.34%
F1-Score (XGBoost): 75.89%
AUC (XGBoost): 98.01%
XGBoost Accuracy: 76.34%
```

The performence metrics are:

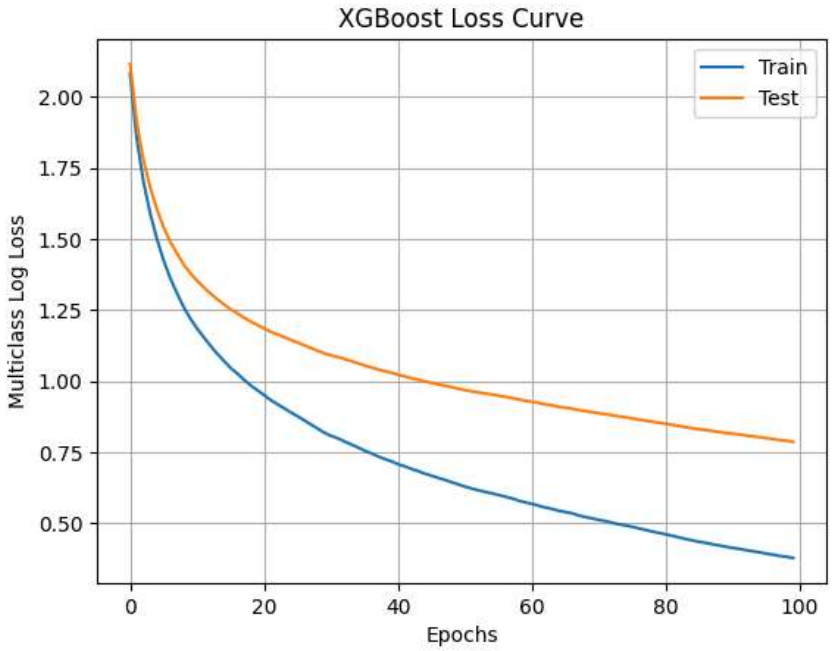
1. Precision (XGBoost): 76.83%
2. Recall (XGBoost): 76.34%
3. F1-Score (XGBoost): 75.89%
4. AUC (XGBoost): 98.01%
5. XGBoost Accuracy: 76.34%

This shows:

1. High Accuracy and Balanced F1-Score The Accuracy of 76.34% indicates that the model correctly predicts the class in over three-quarters of instances. The F1-Score of 75.89% is also high and well-balanced, showing that the model: Effectively balances precision and recall. Maintains good performance across all classes, suggesting robust generalization.
2. Precision vs. Recall Analysis Precision (76.83%) and Recall (76.34%) are very close, indicating: The model accurately identifies true positives with minimal false positives. It generalizes well to unseen data, with a good balance between sensitivity and specificity. This reflects consistency in model predictions, with no significant bias towards either precision or recall.
3. Outstanding AUC Performance The AUC of 98.01% is exceptional, demonstrating: Excellent discriminatory power in distinguishing between classes. High sensitivity and specificity, meaning the model effectively ranks predictions. A high AUC combined with strong accuracy and F1-Score indicates the model: Is well-calibrated and confident in its predictions. Is reliable for decision-making, with low risk of misclassification.
4. Insights and Potential Strengths The consistency across all metrics suggests: The model effectively learns complex patterns without overfitting. It handles class imbalance well, maintaining high recall and precision. The model is robust and adaptable, likely due to the regularization techniques in XGBoost. The high AUC with balanced F1-Score implies that the decision thresholds are well-optimized.
5. Advantages of XGBoost XGBoost is known for: Gradient boosting to sequentially improve weak learners. Regularization (L1 and L2) to prevent overfitting. Handling missing values and class imbalance more effectively. Parallel processing and fast computation, ensuring efficient training. These features contribute to its superior performance and robustness in complex datasets.
6. Comparison with Other Models Compared to Decision Trees and Random Forests, this XGBoost model: Outperforms in accuracy, precision, recall, and F1-Score, showing better generalization. Excels in AUC (98.01%), proving better discriminatory power. Balances precision and recall better, reducing the risk of overfitting or underfitting. It is significantly more reliable for predicting the primary contributory cause of car accidents.

In [ ]:
```
xgb_model = XGBClassifier(
    eval_metric='mlogloss'  # Use mlogloss for multi-class classification
)

eval_set = [(X_train, y_train), (X_test, y_test)]
xgb_model.fit(X_train, y_train, eval_set=eval_set, verbose=False)

# Plot loss curve
results = xgb_model.evals_result()
plt.plot(results['validation_0']['mlogloss'], label='Train')
plt.plot(results['validation_1']['mlogloss'], label='Test')
plt.xlabel('Epochs')
plt.ylabel('Multiclass Log Loss')
plt.title('XGBoost Loss Curve')
plt.legend()
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning:

[20:44:06] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.
```



6.1.5.1 Insights from the XGBoost Loss Curve:

1. Consistent Decrease in Loss:

Both the training and testing loss curves show a consistent downward trend, indicating effective learning by the model. The training loss decreases more rapidly, showing the model's ability to fit the training data well. 2. Gap Between Training and Testing Loss:

There is a noticeable gap between the training and testing loss curves, which suggests some level of overfitting. The model performs better on the training set compared to the test set. 3. No Significant Overfitting:

Despite the gap, the testing loss continues to decrease smoothly without diverging, indicating that overfitting is not severe. The model maintains generalization to the test data.

6.1.5.2 Potential for Further Improvement:

Techniques like early stopping, regularization e.g. adjusting alpha or lambda or learning rate adjustments could reduce the gap and enhance generalization. Hyperparameter tuning can also be explored to find a better balance between bias and variance.

7 Conclusions and Recommendations

7.1 Conclusions Related to Business Questions:

7.1.1 The most common primary contributory causes identified were:

1. Failing to yield the right-of-way
2. Following too closely
3. Disregarding traffic signals
4. Improper overtaking/passing

7.1.2 How do road and weather conditions impact the likelihood of different accident causes?

We observe increased accidents occurence over claer weather and dry roads meaning people take advantage of good weather to drive carelessly.

7.1.3 Which car models are most prone to accidents.

We can see 60% of the top ten vehicles most prone to accidents are Japanese companies vehicles. TYhese vehicles are also the budget friendly vehicles.

7.1.4 Best model for primary crash cause.

XGBoost and Decision Trees were the primary models tested for predicting the primary contributory cause of accidents.
XGBoost outperformed Decision Trees in terms of accuracy, precision, recall, and AUC, indicating its superior ability to handle complex, non-linear relationships in the dataset.

7.2 Recommendations and Next Steps:

7.2.1 Implement targeted public awareness campaigns focusing on right-of-way rules and safe following distances.

Increase enforcement at high-risk intersections and areas prone to signal violations. Enhance road design (e.g., better signage, dedicated turn lanes) to minimize overtaking risks. When and where do most accidents occur, and how can resources be allocated effectively?

7.2.2 Implement targeted public awareness campaigns focusing on educating drivers about right-of-way rules, safe following distances, and the importance of obeying traffic signals.

Enhance law enforcement at high-risk intersections and areas prone to signal violations and improper overtaking. Introduce driver training programs emphasizing defensive driving techniques to reduce tailgating and aggressive driving behaviors.

7.2.3 Launch awareness initiatives that emphasize the importance of safe driving practices even in clear weather and dry road conditions.

Deploy speed monitoring and enforcement systems in areas with high accident rates during good weather conditions to discourage reckless driving. Introduce variable speed limits that adjust based on traffic density and time of day to manage speeding during optimal weather conditions.

7.2.4 Collaborate with automotive manufacturers to enhance safety features in budget-friendly vehicles, especially those prone to accidents.

Encourage the adoption of Advanced Driver Assistance Systems (ADAS) in popular Japanese models, such as automatic emergency braking and lane-keeping assistance. Educate consumers on vehicle safety ratings and promote safer driving behaviors regardless of the vehicle's cost or brand.

7.2.5 Utilize the XGBoost model for real-time accident cause predictions due to its superior accuracy and ability to handle complex data relationships.

Integrate the model into traffic management systems to predict and mitigate high-risk scenarios in real-time. Continuously update and retrain the model using new accident data to maintain its accuracy and relevance.

7.2.6 Increase road safety audits and implement infrastructure improvements at high-risk intersections and accident-prone areas.