Classifier To Predicting Primary Contributory Cause of Car Crashes in Chicago

Final Project Submission

Student name:

Nyairo Caroline Gesaka

Student pace:

Part time

Scheduled project review date/time:

Instructor name:

William Okomba and Noah Kandie

1. Business problem

1.1. Objective

The goal is to develop a machine learning classifier capable of predicting the primary contributory cause of car accidents in Chicago. By accurately identifying these causes, the City of Chicago and the Vehicle Safety Board can take targeted actions to enhance road safety and reduce the frequency and severity of car accidents.

1.2. Stakeholder

The primary stakeholders for this project are:

City of Chicago: The city's traffic management and road safety departments will utilize the insights to improve traffic safety measures.

Vehicle Safety Board: This board will use the data to formulate policies, launch safety campaigns, and improve vehicular regulations.

1.3. Business Problem

The primary business problem to be addressed is the high rate of car accidents in Chicago. By identifying and understanding the leading causes of these accidents, the city can implement specific interventions. These might include:

-Enhancing road infrastructure.

-Introducing stricter traffic regulations.

-Running educational campaigns targeting high-risk behaviors.

-Improving emergency response protocols.

2. Understand The Data

2.1.Data Source:

Chicago Car Crashes dataset(Traffic_Crashes_-_Crashes_20240516.csv).

2.2.Features:

CRASH_DATE: Useful for extracting temporal features like month and day.

POSTED_SPEED_LIMIT: Speed limits influence crash dynamics.

TRAFFIC_CONTROL_DEVICE: Type of traffic control device.

DEVICE_CONDITION: Condition of the traffic control device.

WEATHER_CONDITION: Weather conditions during the crash.

LIGHTING_CONDITION: Lighting conditions during the crash.

FIRST_CRASH_TYPE: Type of the first crash.

TRAFFICWAY_TYPE: Type of trafficway.

ALIGNMENT: Road alignment.

ROADWAY_SURFACE_COND: Roadway surface condition.

ROAD_DEFECT: Any road defects present.

REPORT_TYPE: Type of crash report.

CRASH_TYPE: Overall crash type.

DAMAGE: Damage severity.

CRASH_HOUR: Hour of the day when the crash occurred.

CRASH_DAY_OF_WEEK: Day of the week when the crash occurred.

CRASH_MONTH: Month when the crash occurred.

NUM_UNITS: Number of units (vehicles) involved.

LATITUDE: Latitude of the crash location.

LONGITUDE: Longitude of the crash location.

2.3.Target Variable:

Primary contributory cause of the accident (multi-class classification problem).

Hypothesis

The primary contributory cause of car accidents in Chicago can be accurately predicted using a machine learning classifier that analyzes various factors including:

-Weather Conditions Hypothesis: Adverse weather conditions (e.g., rain, snow, fog) significantly increase the likelihood of specific contributory causes of car accidents, such as slippery road surfaces and reduced visibility.

-Road Surface Condition Hypothesis: Poor road surface conditions (e.g., potholes, wet surfaces) are strongly associated with specific types of accidents, such as vehicle skidding and tire blowouts.

-Time of Day Hypothesis: The time of day (e.g., rush hour, nighttime) significantly influences the primary contributory causes, with factors like reduced visibility at night or higher traffic volumes during rush hour playing a crucial role.

In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import plotly.express as px #for Gospatial analysis
import math #to detect outliers
from scipy.stats import chi2_contingency # check multicolinearity(catego
from itertools import combinations
```

In [2]:

```
#Loading the dataset.This project uses Chicago car crash dataset.
df = pd.read_csv(r"C:\Users\Caro\Downloads\Traffic_Crashes_-_Crashes_20
```

In [3]:
```
#explore the datasets
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 455416 entries, 0 to 455415
Data columns (total 48 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   CRASH_RECORD_ID                455416 non-null  object 
 1   CRASH_DATE_EST_I               33893 non-null   object 
 2   CRASH_DATE                     455412 non-null  object 
 3   POSTED_SPEED_LIMIT             455412 non-null  float64
 4   TRAFFIC_CONTROL_DEVICE         455412 non-null  object 
 5   DEVICE_CONDITION               455412 non-null  object 
 6   WEATHER_CONDITION              455412 non-null  object 
 7   LIGHTING_CONDITION             455412 non-null  object 
 8   FIRST_CRASH_TYPE               455412 non-null  object 
 9   TRAFFICWAY_TYPE                455412 non-null  object 
 10  LANE_CNT                       105503 non-null  float64
 11  ALIGNMENT                      455412 non-null  object 
 12  ROADWAY_SURFACE_COND           455412 non-null  object 
 13  ROAD_DEFECT                    455412 non-null  object 
 14  REPORT_TYPE                    440668 non-null  object 
 15  CRASH_TYPE                     455412 non-null  object 
 16  INTERSECTION_RELATED_I         104186 non-null  object 
 17  NOT_RIGHT_OF_WAY_I             21117 non-null   object 
 18  HIT_AND_RUN_I                  142486 non-null  object 
 19  DAMAGE                         455412 non-null  object 
 20  DATE_POLICE_NOTIFIED           455412 non-null  object 
 21  PRIM_CONTRIBUTORY_CAUSE        455412 non-null  object 
 22  SEC_CONTRIBUTORY_CAUSE         455412 non-null  object 
 23  STREET_NO                      455412 non-null  float64
 24  STREET_DIRECTION               455409 non-null  object 
 25  STREET_NAME                    455411 non-null  object 
 26  BEAT_OF_OCCURRENCE             455409 non-null  float64
 27  PHOTOS_TAKEN_I                 6059 non-null    object 
 28  STATEMENTS_TAKEN_I             10403 non-null   object 
 29  DOORING_I                      1387 non-null    object 
 30  WORK_ZONE_I                    2626 non-null    object 
 31  WORK_ZONE_TYPE                 2014 non-null    object 
 32  WORKERS_PRESENT_I              699 non-null     object 
 33  NUM_UNITS                      455412 non-null  float64
 34  MOST_SEVERE_INJURY             454331 non-null  object 
 35  INJURIES_TOTAL                 454342 non-null  float64
 36  INJURIES_FATAL                 454342 non-null  float64
 37  INJURIES_INCAPACITATING        454342 non-null  float64
 38  INJURIES_NON_INCAPACITATING    454342 non-null  float64
 39  INJURIES_REPORTED_NOT_EVIDENT  454342 non-null  float64
 40  INJURIES_NO_INDICATION         454342 non-null  float64
 41  INJURIES_UNKNOWN               454342 non-null  float64
 42  CRASH_HOUR                     455412 non-null  float64
 43  CRASH_DAY_OF_WEEK              455412 non-null  object 
 44  CRASH_MONTH                    455411 non-null  float64
 45  LATITUDE                       449632 non-null  float64
 46  LONGITUDE                      449632 non-null  float64
 47  LOCATION                       449632 non-null  object 
dtypes: float64(16), object(32)
memory usage: 166.8+ MB
```

In [4]:
```
# check the first five elements
df.head()
```

Out[4]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DAT
0	23a79931ef555d54118f64dc9be2cf2dbf59636ce253f7...	NaN	09/05/202 07:05:00 P
1	2675c13fd0f474d730a5b780968b3cafc7c12d7adb661f...	NaN	09/22/202 06:45:00 P
2	5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4...	NaN	07/29/202 02:45:00 P
3	7ebf015016f83d09b321afd671a836d6b148330535d5df...	NaN	08/09/202 11:00:00 P
4	6c1659069e9c6285a650e70d6f9b574ed5f64c12888479...	NaN	08/18/202 12:50:00 P

5 rows × 48 columns

In [5]:
```
# check the last five elements
df.tail()
```

Out[5]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRAS
455411	853eaa096088487ee500121138be38cd31e5b93925ddaf...	NaN	09 06:(
455412	"error" : true	NaN	
455413	"message" : "Internal error"	NaN	
455414	"status" : 500	NaN	
455415	}	NaN	

5 rows × 48 columns

In [6]:
```
df.describe()
```

Out[6]:

	POSTED_SPEED_LIMIT	LANE_CNT	STREET_NO	BEAT_OF_OCCURRENCE	
count	455412.000000	1.055030e+05	455412.000000	455409.000000	4
mean	28.396285	1.795405e+01	3682.557390	1244.663990	
std	6.169818	3.903997e+03	2924.608106	704.793876	
min	0.000000	0.000000e+00	0.000000	111.000000	
25%	30.000000	2.000000e+00	1238.000000	714.000000	
50%	30.000000	2.000000e+00	3200.000000	1212.000000	
75%	30.000000	4.000000e+00	5569.000000	1822.000000	
max	99.000000	1.191625e+06	451100.000000	6100.000000	

In [7]:
```
# Check unique values for each column
for column in df.columns:
    unique_values = df[column].unique()
    print(f"Column '{column}' has {len(unique_values)} unique values")
    print(unique_values[::])  #all unique values for every column
    print("\n")
```

```
Column 'CRASH_RECORD_ID' has 455416 unique values
['23a79931ef555d54118f64dc9be2cf2dbf59636ce253f7a1179c4a1c091442a6eeab
8352220c7c56ca1ff7c4b4b0fc345c74e3e85ecb9d43deeb66b5f803d4a0'
 '2675c13fd0f474d730a5b780968b3cafc7c12d7adb661fa8a3093c0658d5a0d51b72
0fc9e031a1ddd83c761a8e2aa7283573557db246f4c9e956aaa58719cacf'
 '5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4106558b34b8a6d2b81af02
cf91b576ecd7ced08ffd10fcfd940a84f7613125b89d33636e6075064e22'
 ... '  "message" : "Internal error"' '  "status" : 500' '}']


Column 'CRASH_DATE_EST_I' has 3 unique values
[nan 'Y' 'N']


Column 'CRASH_DATE' has 346504 unique values
['09/05/2023 07:05:00 PM' '09/22/2023 06:45:00 PM'
 '07/29/2023 02:45:00 PM' ... '04/08/2018 05:00:00 AM'
 '09/28/2023 06:05:00 AM' nan]


Column 'POSTED_SPEED_LIMIT' has 44 unique values
[30. 50. 15. 25. 10. 35. 20. 55. 45.  5. 40.  0. 60.  3. 65. 39.  9. 2
2.
 70. 14. 34. 33.  6. 24.  1. 23. 99. 11. 26. 32. 12.  2.  7. 49.  8. 3
6.
 63. 29. 38. 16. 44. 62.  4. nan]


Column 'TRAFFIC_CONTROL_DEVICE' has 20 unique values
['TRAFFIC SIGNAL' 'NO CONTROLS' 'OTHER' 'UNKNOWN' 'OTHER WARNING SIGN'
 'STOP SIGN/FLASHER' 'PEDESTRIAN CROSSING SIGN' 'OTHER REG. SIGN' 'YIE
LD'
 'LANE USE MARKING' 'RAILROAD CROSSING GATE' 'FLASHING CONTROL SIGNAL'
 'SCHOOL ZONE' 'POLICE/FLAGMAN' 'DELINEATORS' 'OTHER RAILROAD CROSSIN
G'
 'RR CROSSING SIGN' 'NO PASSING' 'BICYCLE CROSSING SIGN' nan]


Column 'DEVICE_CONDITION' has 9 unique values
['FUNCTIONING PROPERLY' 'NO CONTROLS' 'FUNCTIONING IMPROPERLY' 'UNKNOW
N'
 'OTHER' 'NOT FUNCTIONING' 'MISSING' 'WORN REFLECTIVE MATERIAL' nan]


Column 'WEATHER_CONDITION' has 13 unique values
['CLEAR' 'SNOW' 'RAIN' 'UNKNOWN' 'CLOUDY/OVERCAST' 'FOG/SMOKE/HAZE'
 'BLOWING SNOW' 'FREEZING RAIN/DRIZZLE' 'OTHER' 'SEVERE CROSS WIND GAT
E'
 'SLEET/HAIL' 'BLOWING SAND, SOIL, DIRT' nan]


Column 'LIGHTING_CONDITION' has 7 unique values
['DUSK' 'DARKNESS, LIGHTED ROAD' 'DAYLIGHT' 'DARKNESS' 'UNKNOWN' 'DAW
N'
 nan]


Column 'FIRST_CRASH_TYPE' has 19 unique values
['ANGLE' 'REAR END' 'PARKED MOTOR VEHICLE' 'SIDESWIPE SAME DIRECTION'
 'PEDESTRIAN' 'FIXED OBJECT' 'TURNING' 'SIDESWIPE OPPOSITE DIRECTION'
 'REAR TO FRONT' 'HEAD ON' 'REAR TO SIDE' 'PEDALCYCLIST' 'OTHER OBJEC
```

```
T'
 'ANIMAL' 'REAR TO REAR' 'OTHER NONCOLLISION' 'OVERTURNED' 'TRAIN' na
n]


Column 'TRAFFICWAY_TYPE' has 21 unique values
['FIVE POINT, OR MORE' 'DIVIDED - W/MEDIAN BARRIER'
 'DIVIDED - W/MEDIAN (NOT RAISED)' 'NOT DIVIDED' 'OTHER' 'ONE-WAY'
 'PARKING LOT' 'T-INTERSECTION' 'RAMP' 'FOUR WAY' 'UNKNOWN' 'ALLEY'
 'UNKNOWN INTERSECTION TYPE' 'DRIVEWAY' 'TRAFFIC ROUTE' 'NOT REPORTED'
 'CENTER TURN LANE' 'L-INTERSECTION' 'Y-INTERSECTION' 'ROUNDABOUT' na
n]


Column 'LANE_CNT' has 33 unique values
[          nan 2.000000e+00 4.000000e+00 0.000000e+00 8.000000e+00
 6.000000e+00 3.000000e+00 1.000000e+00 1.100000e+01 2.200000e+01
 1.600000e+01 5.000000e+00 1.000000e+01 9.900000e+01 4.000000e+01
 9.000000e+00 1.300000e+01 1.400000e+01 7.000000e+00 2.000000e+01
 1.200000e+01 1.500000e+01 2.800000e+01 4.100000e+01 4.336340e+05
 2.500000e+01 3.000000e+01 3.500000e+01 1.191625e+06 6.000000e+01
 1.900000e+01 4.000000e+02 2.100000e+01]


Column 'ALIGNMENT' has 7 unique values
['STRAIGHT AND LEVEL' 'CURVE ON GRADE' 'CURVE, LEVEL' 'STRAIGHT ON GRA
DE'
 'STRAIGHT ON HILLCREST' 'CURVE ON HILLCREST' nan]


Column 'ROADWAY_SURFACE_COND' has 8 unique values
['DRY' 'SNOW OR SLUSH' 'WET' 'UNKNOWN' 'OTHER' 'ICE' 'SAND, MUD, DIRT'
nan]


Column 'ROAD_DEFECT' has 8 unique values
['NO DEFECTS' 'UNKNOWN' 'DEBRIS ON ROADWAY' 'OTHER' 'WORN SURFACE'
 'SHOULDER DEFECT' 'RUT, HOLES' nan]


Column 'REPORT_TYPE' has 4 unique values
['ON SCENE' 'NOT ON SCENE (DESK REPORT)' nan 'AMENDED']


Column 'CRASH_TYPE' has 3 unique values
['INJURY AND / OR TOW DUE TO CRASH' 'NO INJURY / DRIVE AWAY' nan]


Column 'INTERSECTION_RELATED_I' has 3 unique values
['Y' nan 'N']


Column 'NOT_RIGHT_OF_WAY_I' has 3 unique values
[nan 'Y' 'N']


Column 'HIT_AND_RUN_I' has 3 unique values
[nan 'Y' 'N']


Column 'DAMAGE' has 4 unique values
```

```
['OVER $1,500' '$501 - $1,500' '$500 OR LESS' nan]


Column 'DATE_POLICE_NOTIFIED' has 384044 unique values
['09/05/2023 07:05:00 PM' '09/22/2023 06:50:00 PM'
 '07/29/2023 02:45:00 PM' ... '05/12/2023 04:21:00 PM'
 '09/28/2023 06:08:00 AM' nan]


Column 'PRIM_CONTRIBUTORY_CAUSE' has 41 unique values
['UNABLE TO DETERMINE' 'FOLLOWING TOO CLOSELY'
 'FAILING TO REDUCE SPEED TO AVOID CRASH' 'FAILING TO YIELD RIGHT-OF-W
AY'
 'NOT APPLICABLE' 'WEATHER' 'IMPROPER BACKING'
 'IMPROPER TURNING/NO SIGNAL' 'DRIVING SKILLS/KNOWLEDGE/EXPERIENCE'
 'IMPROPER LANE USAGE'
 'VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)'
 'ROAD ENGINEERING/SURFACE/MARKING DEFECTS'
 'EQUIPMENT - VEHICLE CONDITION' 'IMPROPER OVERTAKING/PASSING'
 'RELATED TO BUS STOP' 'DISREGARDING TRAFFIC SIGNALS'
 'DRIVING ON WRONG SIDE/WRONG WAY' 'DISREGARDING ROAD MARKINGS'
 'DISTRACTION - FROM INSIDE VEHICLE' 'ANIMAL'
 'ROAD CONSTRUCTION/MAINTENANCE' 'TEXTING'
 'CELL PHONE USE OTHER THAN TEXTING' 'DISREGARDING OTHER TRAFFIC SIGN
S'
 'DISREGARDING STOP SIGN' 'EXCEEDING AUTHORIZED SPEED LIMIT'
 'OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRE
SSIVE MANNER'
 'DISTRACTION - FROM OUTSIDE VEHICLE' 'PHYSICAL CONDITION OF DRIVER'
 'EXCEEDING SAFE SPEED FOR CONDITIONS' 'DISREGARDING YIELD SIGN'
 'TURNING RIGHT ON RED'
 'UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)'
 'EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST'
 'HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)'
 'DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYE
R, ETC.)'
 'OBSTRUCTED CROSSWALKS' 'BICYCLE ADVANCING LEGALLY ON RED LIGHT'
 'PASSING STOPPED SCHOOL BUS' 'MOTORCYCLE ADVANCING LEGALLY ON RED LIG
HT'
 nan]


Column 'SEC_CONTRIBUTORY_CAUSE' has 41 unique values
['NOT APPLICABLE' 'FOLLOWING TOO CLOSELY'
 'OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRE
SSIVE MANNER'
 'DISTRACTION - FROM INSIDE VEHICLE' 'UNABLE TO DETERMINE' 'WEATHER'
 'FAILING TO YIELD RIGHT-OF-WAY' 'DRIVING SKILLS/KNOWLEDGE/EXPERIENCE'
 'FAILING TO REDUCE SPEED TO AVOID CRASH' 'IMPROPER OVERTAKING/PASSIN
G'
 'IMPROPER TURNING/NO SIGNAL' 'DISREGARDING STOP SIGN'
 'ROAD CONSTRUCTION/MAINTENANCE' 'IMPROPER LANE USAGE'
 'DISTRACTION - FROM OUTSIDE VEHICLE' 'DISREGARDING TRAFFIC SIGNALS'
 'DRIVING ON WRONG SIDE/WRONG WAY'
 'VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)'
 'EQUIPMENT - VEHICLE CONDITION' 'IMPROPER BACKING'
 'PHYSICAL CONDITION OF DRIVER' 'EXCEEDING SAFE SPEED FOR CONDITIONS'
 'BICYCLE ADVANCING LEGALLY ON RED LIGHT'
 'EXCEEDING AUTHORIZED SPEED LIMIT'
 'DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYE
R, ETC.)'
```

```
 'UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)'
 'ROAD ENGINEERING/SURFACE/MARKING DEFECTS'
 'EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST' 'ANIMAL'
 'DISREGARDING ROAD MARKINGS' 'DISREGARDING OTHER TRAFFIC SIGNS'
 'CELL PHONE USE OTHER THAN TEXTING'
 'HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)' 'TURNING RIGHT ON R
ED'
 'PASSING STOPPED SCHOOL BUS' 'RELATED TO BUS STOP'
 'MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT' 'DISREGARDING YIELD SIGN'
 'TEXTING' 'OBSTRUCTED CROSSWALKS' nan]


Column 'STREET_NO' has 11074 unique values
[5500. 7900. 2101. ... 8473. 3889.   nan]


Column 'STREET_DIRECTION' has 5 unique values
['S' 'W' 'E' 'N' nan]


Column 'STREET_NAME' has 1568 unique values
['WENTWORTH AVE' 'CHICAGO SKYWAY OB' 'ASHLAND AVE' ... 'MANILA AVE'
 'CARROLL DR' 'KINZUA AVE']


Column 'BEAT_OF_OCCURRENCE' has 277 unique values
[ 225.  411. 1235. 1650. 1654. 1655. 1652. 1131.  424. 1814. 2221.  23
2.
 1034. 1651.  222. 1211. 1653. 1712.  731. 1822. 2525.  423.  533.  33
1.
  925.  824. 1811. 1121.  434.  412. 1023. 2511. 1433. 1214.  914. 242
3.
 1012. 2222. 1613. 1513.  431.  822.  825. 1031.  924.  833. 2521. 191
3.
  912.  811.  523. 1215.  323. 2512. 1834.  813.  723. 1522. 2032. 101
1.
 1622.  913. 1411. 2024. 2213. 2223. 1924. 1925.  834.  221. 2232. 163
4.
 1934.  334.  433.  711. 2522.  621. 1414.  622.  932.  122. 2413. 142
4.
 1022. 1621.  313.  132. 2523.  324.  111. 1033.  231. 1412. 1531. 163
3.
 1623. 1722. 2212.  421. 1112.  524. 1224.  414. 1013. 2234.  735. 143
1.
  921.  114. 1914. 2411. 1631.  224.  815. 1233. 2532. 1823. 2022.  83
5.
 2513. 6100. 1912.  322. 1421. 1922. 2033. 1434. 1231. 1113.  614. 102
1.
 2023.  935.  321. 1135. 2432.  915. 1711.  623. 1733. 1123. 1221. 112
2.
  223. 1813. 2012. 1111.  312. 1533.  812. 1212.  311.  631. 2531. 103
2.
 1524.  432. 2533. 2524. 1724. 1222. 1234. 1911.  131. 1432.  923. 121
3.
 1732.  821. 1611. 1731. 1832.  234.  612. 2013.  611.   nan 1232.  91
1.
 1931. 2515. 1014. 1821.  934.  734. 2535.  511. 1225. 1413. 2514.  41
3.
  332. 1723. 2534. 1134.  215.  214.  722.  235. 1713.  712. 1632. 241
2.
  624. 1133.  832. 1935.  422.  931. 1921. 1624. 2431. 1422.  634.  12
```

```
3.
 1523.  112.  532.  124. 2233.  133. 2422. 1933.  726.  725.  233.  92
2.
  933.  814. 1532. 1125. 1115. 1824.  333.  733. 2211. 1124.  714. 181
2.
 2031. 2011.  513. 1512.  512.  724. 1423.  632. 2424. 1833.  613. 113
2.
 1932. 1614.  522.  213. 1915.  212.  314. 1831. 1923. 1511. 1612.  21
1.
 2433.  713.  823. 1024. 1114.  531.  633.  715.  121. 1223.  831.  73
2.
  113.]


Column 'PHOTOS_TAKEN_I' has 3 unique values
[nan 'Y' 'N']


Column 'STATEMENTS_TAKEN_I' has 3 unique values
[nan 'Y' 'N']


Column 'DOORING_I' has 3 unique values
[nan 'Y' 'N']


Column 'WORK_ZONE_I' has 3 unique values
[nan 'Y' 'N']


Column 'WORK_ZONE_TYPE' has 5 unique values
[nan 'CONSTRUCTION' 'UTILITY' 'UNKNOWN' 'MAINTENANCE']


Column 'WORKERS_PRESENT_I' has 3 unique values
[nan 'Y' 'N']


Column 'NUM_UNITS' has 16 unique values
[ 2.  4.  1.  3.  5.  6.  7.  8. 12. 16.  9. 11. 10. 14. 18. nan]


Column 'MOST_SEVERE_INJURY' has 6 unique values
['INCAPACITATING INJURY' 'NO INDICATION OF INJURY'
 'NONINCAPACITATING INJURY' 'FATAL' 'REPORTED, NOT EVIDENT' nan]


Column 'INJURIES_TOTAL' has 20 unique values
[ 3.  0.  1.  2.  5. nan  4.  6. 15.  7. 12.  9.  8. 10. 21. 11. 17. 1
4.
 19. 13.]


Column 'INJURIES_FATAL' has 5 unique values
[ 0.  1. nan  2.  3.]


Column 'INJURIES_INCAPACITATING' has 8 unique values
[ 1.  0.  5. nan  2.  3.  4.  6.]
```

```
Column 'INJURIES_NON_INCAPACITATING' has 18 unique values
[ 2.  0.  1.  5. nan  3.  4.  6.  7.  8. 11. 10. 21. 14. 12. 19.  9. 1
8.]


Column 'INJURIES_REPORTED_NOT_EVIDENT' has 14 unique values
[ 0.  2. nan  1.  3.  5.  4. 10.  6.  7. 11.  8.  9. 15.]


Column 'INJURIES_NO_INDICATION' has 45 unique values
[ 2.  1.  5.  3.  0.  4.  8.  7. nan  6. 15. 11. 20. 10. 18. 27.  9. 1
3.
 12. 26. 29. 14. 41. 16. 37. 22. 30. 50. 21. 17. 42. 46. 24. 61. 38. 1
9.
 48. 34. 31. 43. 45. 40. 36. 23. 28.]


Column 'INJURIES_UNKNOWN' has 2 unique values
[ 0. nan]


Column 'CRASH_HOUR' has 25 unique values
[19. 18. 14. 23. 12.  8. 17. 16. 15. 13. 10.  0.  6.  5. 22. 11. 21.
9.
 20.  1.  7.  2.  3.  4. nan]


Column 'CRASH_DAY_OF_WEEK' has 16 unique values
[3 6 7 4 2 1 5 '5' '2' '1' '7' '3' '6' '4' '{' nan]


Column 'CRASH_MONTH' has 13 unique values
[ 9.  7.  8. 11.  2.  1. 10. 12.  5.  6.  3.  4. nan]


Column 'LATITUDE' has 202996 unique values
[        nan 41.85412026 41.80978115 ... 41.89840746 41.75704136
 41.93763777]


Column 'LONGITUDE' has 202997 unique values
[         nan -87.66590234 -87.59421281 ... -87.72114211 -87.54764299
 -87.75990493]


Column 'LOCATION' has 203097 unique values
[nan 'POINT (-87.665902342962 41.854120262952)'
 'POINT (-87.594212812011 41.809781151018)' ...
 'POINT (-87.721142109818 41.898407459204)'
 'POINT (-87.547642985901 41.757041359914)'
 'POINT (-87.759904933396 41.937637767663)']
```

In [8]:
```
#check for duplicated rows
df[df.duplicated()]

#there are 0 dupllcated rows
```

Out[8]:

CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRA

0 rows × 48 columns

In [9]:
```
# check the shape of the data
df.shape
```

Out[9]: (455416, 48)

In [10]:
```
#Create a deep copy of the DataFrame
df_copy= df.copy(deep=True)
```

3. Data Preparation

3.1. Data Cleaning

In [11]:
```
#create another dataframe for cleaning
df_new = pd.read_csv(r"C:\Users\Caro\Downloads\Traffic_Crashes_-_Crashe
```

3.1.1. Handling missing values

In [12]:
```
#Check for missing values before dropping
df_new.isna().sum()
```

Out[12]:
```
CRASH_RECORD_ID                    0
CRASH_DATE_EST_I              421523
CRASH_DATE                         4
POSTED_SPEED_LIMIT                 4
TRAFFIC_CONTROL_DEVICE             4
DEVICE_CONDITION                   4
WEATHER_CONDITION                  4
LIGHTING_CONDITION                 4
FIRST_CRASH_TYPE                   4
TRAFFICWAY_TYPE                    4
LANE_CNT                      349913
ALIGNMENT                          4
ROADWAY_SURFACE_COND               4
ROAD_DEFECT                        4
REPORT_TYPE                    14748
CRASH_TYPE                         4
INTERSECTION_RELATED_I        351230
NOT_RIGHT_OF_WAY_I            434299
HIT_AND_RUN_I                 312930
DAMAGE                             4
DATE_POLICE_NOTIFIED               4
PRIM_CONTRIBUTORY_CAUSE            4
SEC_CONTRIBUTORY_CAUSE             4
STREET_NO                          4
STREET_DIRECTION                   7
STREET_NAME                        5
BEAT_OF_OCCURRENCE                 7
PHOTOS_TAKEN_I                449357
STATEMENTS_TAKEN_I            445013
DOORING_I                     454029
WORK_ZONE_I                   452790
WORK_ZONE_TYPE                453402
WORKERS_PRESENT_I             454717
NUM_UNITS                          4
MOST_SEVERE_INJURY              1085
INJURIES_TOTAL                  1074
INJURIES_FATAL                  1074
INJURIES_INCAPACITATING         1074
INJURIES_NON_INCAPACITATING     1074
INJURIES_REPORTED_NOT_EVIDENT   1074
INJURIES_NO_INDICATION          1074
INJURIES_UNKNOWN                1074
CRASH_HOUR                         4
CRASH_DAY_OF_WEEK                  4
CRASH_MONTH                        5
LATITUDE                        5784
LONGITUDE                       5784
LOCATION                        5784
dtype: int64
```

In [13]:
```
# List of columns to drop
columns_to_drop = ['CRASH_RECORD_ID','CRASH_DATE_EST_I','LANE_CNT','INT
                   'MOST_SEVERE_INJURY','INJURIES_TOTAL','INJURIES_FATA
                   'INJURIES_REPORTED_NOT_EVIDENT','INJURIES_NO_INDICAT
                   'PHOTOS_TAKEN_I','STATEMENTS_TAKEN_I','DOORING_I','W
                   'FIRST_CRASH_TYPE','TRAFFICWAY_TYPE','ALIGNMENT','RO
                   'BEAT_OF_OCCURRENCE','SEC_CONTRIBUTORY_CAUSE','DATE_

# Drop the specified columns
df_new = df_new.drop(columns=columns_to_drop)
```

In [14]:
```
#3. Impute numerical columns with mean
numerical_cols = ['POSTED_SPEED_LIMIT','NUM_UNITS','CRASH_HOUR','CRASH_
for col in numerical_cols:
    df_new[col].fillna(df_new[col].mean(), inplace=True)
```

In [15]:
```
# Store the original shape of the DataFrame
original_shape = df_new.shape[0]

# Convert 'CRASH_DATE' column to datetime format
df_new['CRASH_DATE'] = pd.to_datetime(df_new['CRASH_DATE'], errors='coe

# Find unique values in the 'CRASH_DATE' column that are not valid date
invalid_dates = df_new['CRASH_DATE'][~df_new['CRASH_DATE'].notnull() & 

# Drop rows with invalid dates
df_new = df_new[~df_new['CRASH_DATE'].isin(invalid_dates)]

# Check if any rows were dropped
dropped_rows = original_shape - df_new.shape[0]
print(f"Dropped {dropped_rows} rows with invalid dates.")
```

```
Dropped 4 rows with invalid dates.
```

In [16]:
```
#4. Impute categorical columns with mode
categorical_cols = [
    'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'WEATHER_CONDITION',
    'LIGHTING_CONDITION', 'ROADWAY_SURFACE_COND','CRASH_DAY_OF_WEEK',
    'CRASH_TYPE', 'PRIM_CONTRIBUTORY_CAUSE'
]
for col in categorical_cols:
    df_new[col].fillna(df_new[col].mode()[0], inplace=True)
```

In [17]:
```
#5. Fill Geospatial missing values with the mean of the column
df_new['LATITUDE'].fillna(df_new['LATITUDE'].mean(), inplace=True)
df_new['LONGITUDE'].fillna(df_new['LONGITUDE'].mean(), inplace=True)

# Recreate the LOCATION column
df_new['LOCATION'] = df_new.apply(
    lambda row: f"({row['LATITUDE']}, {row['LONGITUDE']})",
    axis=1
)
```

In [18]:
```
#Check for missing values after dropping
df_new.isna().sum()
```

Out[18]:
```
CRASH_DATE                 0
POSTED_SPEED_LIMIT         0
TRAFFIC_CONTROL_DEVICE     0
DEVICE_CONDITION           0
WEATHER_CONDITION          0
LIGHTING_CONDITION         0
ROADWAY_SURFACE_COND       0
CRASH_TYPE                 0
PRIM_CONTRIBUTORY_CAUSE    0
NUM_UNITS                  0
CRASH_HOUR                 0
CRASH_DAY_OF_WEEK          0
CRASH_MONTH                0
LATITUDE                   0
LONGITUDE                  0
LOCATION                   0
dtype: int64
```

In [19]:
```
#new shape after cleaning
df_new.shape
```

Out[19]:
```
(455412, 16)
```

In [20]:
```
#new columns after cleaning
df_new.columns
```

Out[20]:
```
Index(['CRASH_DATE', 'POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE',
       'DEVICE_CONDITION', 'WEATHER_CONDITION', 'LIGHTING_CONDITION',
       'ROADWAY_SURFACE_COND', 'CRASH_TYPE', 'PRIM_CONTRIBUTORY_CAUS
E',
       'NUM_UNITS', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
       'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
```

3.1.2. Data Type conversions

In [21]:

```
# Convert object columns to category to optimize memory and improve per
category_columns = [
    'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'WEATHER_CONDITION',
    'LIGHTING_CONDITION','ROADWAY_SURFACE_COND', 'CRASH_TYPE',
    'PRIM_CONTRIBUTORY_CAUSE', 'CRASH_DAY_OF_WEEK','LOCATION'
]
for col in category_columns:
    df_new[col] = df_new[col].astype('category')

#Crash-records will not be converted since its a unique identifier
```

In [22]:

```
# Convert numerical columns with integer values to reduce memory usage
integer_columns = [
    'POSTED_SPEED_LIMIT','NUM_UNITS',
    'CRASH_HOUR', 'CRASH_MONTH'
]
for col in integer_columns:
    df_new[col] = pd.to_numeric(df_new[col], downcast='integer', errors
```

In [23]:

```
# Select integer columns
integer_columns = df_new.select_dtypes(include='int64').columns

# Convert integer columns to int32
df_new[integer_columns] = df_new[integer_columns].astype('int32')

# Convert float columns to integer
float_columns = df_new.select_dtypes(include=['float']).columns
df_new[float_columns] = df_new[float_columns].astype(int)
```

In [24]:
```
# checking if the conversions were effective
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 455412 entries, 0 to 455411
Data columns (total 16 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   CRASH_DATE               455412 non-null  datetime64[ns]
 1   POSTED_SPEED_LIMIT       455412 non-null  int8
 2   TRAFFIC_CONTROL_DEVICE   455412 non-null  category
 3   DEVICE_CONDITION         455412 non-null  category
 4   WEATHER_CONDITION        455412 non-null  category
 5   LIGHTING_CONDITION       455412 non-null  category
 6   ROADWAY_SURFACE_COND     455412 non-null  category
 7   CRASH_TYPE               455412 non-null  category
 8   PRIM_CONTRIBUTORY_CAUSE  455412 non-null  category
 9   NUM_UNITS                455412 non-null  int8
 10  CRASH_HOUR               455412 non-null  int8
 11  CRASH_DAY_OF_WEEK        455412 non-null  category
 12  CRASH_MONTH              455412 non-null  int32
 13  LATITUDE                 455412 non-null  int32
 14  LONGITUDE                455412 non-null  int32
 15  LOCATION                 455412 non-null  category
dtypes: category(9), datetime64[ns](1), int32(3), int8(3)
memory usage: 28.3 MB
```

3.1.3. Feauture Engineering

In [25]:
```
# Convert 'CRASH_DATE' column to datetime format
df_new['CRASH_DATE'] = pd.to_datetime(df_new['CRASH_DATE'], errors='coe

# Check if 'CRASH_DATE' column is properly converted
print(df_new['CRASH_DATE'].dtype)

# Now, you can perform the feature engineering
# Feature engineering: Date and Time Features
df_new['CRASH_YEAR'] = df_new['CRASH_DATE'].dt.year
df_new['CRASH_MONTH'] = df_new['CRASH_DATE'].dt.month
df_new['CRASH_DAY'] = df_new['CRASH_DATE'].dt.day
df_new['CRASH_DAY_OF_WEEK'] = df_new['CRASH_DATE'].dt.dayofweek
df_new['CRASH_HOUR'] = df_new['CRASH_DATE'].dt.hour
```

```
datetime64[ns]
```

In [26]:

```
# Print the first few rows to verify the result
df_new[['CRASH_DATE', 'CRASH_YEAR', 'CRASH_MONTH', 'CRASH_DAY', 'CRASH_I
```

Out[26]:

	CRASH_DATE	CRASH_YEAR	CRASH_MONTH	CRASH_DAY	CRASH_DAY_OF_WEEK
0	2023-09-05 19:05:00	2023	9	5	1
1	2023-09-22 18:45:00	2023	9	22	4
2	2023-07-29 14:45:00	2023	7	29	5
3	2023-08-09 23:00:00	2023	8	9	2
4	2023-08-18 12:50:00	2023	8	18	4

In [27]:
```
# Interaction Features to capture relationships between different varial
# Create interaction feature: POSTED_SPEED_LIMIT * DEVICE_CONDITION
df_new['SPEED_DEVICE_CONDITION'] = df_new['POSTED_SPEED_LIMIT'] * df_new

# Create interaction feature: WEATHER_CONDITION * LIGHTING_CONDITION
df_new['WEATHER_LIGHTING_CONDITION'] = df_new['WEATHER_CONDITION'].cat.

df_new
```

Out[27]:

	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CO
0	2023-09-05 19:05:00	30	TRAFFIC SIGNAL	FUNC PR
1	2023-09-22 18:45:00	50	NO CONTROLS	NO CO
2	2023-07-29 14:45:00	30	TRAFFIC SIGNAL	FUNC PR
3	2023-08-09 23:00:00	30	NO CONTROLS	NO CO
4	2023-08-18 12:50:00	15	OTHER	FUNC PR
...	...	...	...	
455407	2018-04-13 18:08:00	30	NO CONTROLS	NO CO
455408	2022-08-12 19:31:00	15	NO CONTROLS	NO CO
455409	2018-04-08 05:00:00	20	NO CONTROLS	NOT FUNC
455410	2023-05-12 14:30:00	35	STOP SIGN/FLASHER	NO CO
455411	2023-09-28 06:05:00	30	TRAFFIC SIGNAL	FUNC PR

455412 rows × 20 columns

4. Exploratory Data Analysis

In [28]:

```
# 1. Summary Statistics
summary_stats = df_new.describe()
summary_stats
```

Out[28]:

	CRASH_DATE	POSTED_SPEED_LIMIT	NUM_UNITS	CRASH_HOUR	CRASI
count	455412	455412.000000	455412.000000	455412.000000	
mean	2020-08-01 19:04:37.228926464	28.396285	2.034367	13.201036	
min	2013-06-01 20:29:00	0.000000	1.000000	0.000000	
25%	2018-08-25 18:52:30	30.000000	2.000000	9.000000	
50%	2020-08-05 06:00:00	30.000000	2.000000	14.000000	
75%	2022-08-16 17:15:45	30.000000	2.000000	17.000000	
max	2024-05-15 23:42:00	99.000000	18.000000	23.000000	
std	NaN	6.169818	0.454028	5.571367	

4.1. Univarate Analysis

4.1.1 Primary Contributory Cause(Target Variable)

In [29]:
```
primary_cause_freq = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts()
print(primary_cause_freq)
```

```
PRIM_CONTRIBUTORY_CAUSE
UNABLE TO DETERMINE
178000
FAILING TO YIELD RIGHT-OF-WAY
50060
FOLLOWING TOO CLOSELY
44170
NOT APPLICABLE
24382
IMPROPER OVERTAKING/PASSING
22357
FAILING TO REDUCE SPEED TO AVOID CRASH
19062
IMPROPER BACKING
17763
IMPROPER LANE USAGE
16400
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
15097
IMPROPER TURNING/NO SIGNAL
15034
DISREGARDING TRAFFIC SIGNALS
8883
WEATHER
6841
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESS
IVE MANNER      5912
DISREGARDING STOP SIGN
4810
DISTRACTION - FROM INSIDE VEHICLE
3101
EQUIPMENT - VEHICLE CONDITION
2897
PHYSICAL CONDITION OF DRIVER
2626
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
2577
DRIVING ON WRONG SIDE/WRONG WAY
2433
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
2098
DISTRACTION - FROM OUTSIDE VEHICLE
1805
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
1139
EXCEEDING AUTHORIZED SPEED LIMIT
1035
DISREGARDING OTHER TRAFFIC SIGNS
973
ROAD CONSTRUCTION/MAINTENANCE
923
EXCEEDING SAFE SPEED FOR CONDITIONS
860
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
800
CELL PHONE USE OTHER THAN TEXTING
566
DISREGARDING ROAD MARKINGS
550
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
502
```

```
ANIMAL
418
TURNING RIGHT ON RED
393
RELATED TO BUS STOP
242
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER,
ETC.)              223
TEXTING
174
DISREGARDING YIELD SIGN
148
PASSING STOPPED SCHOOL BUS
60
OBSTRUCTED CROSSWALKS
45
BICYCLE ADVANCING LEGALLY ON RED LIGHT
42
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
11
Name: count, dtype: int64
```

In [56]:

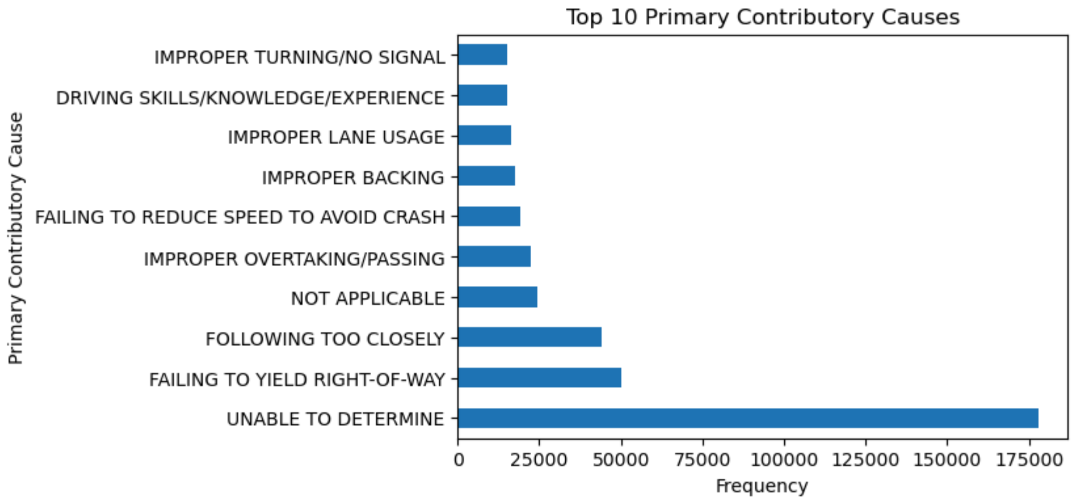
```
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge
top_10_causes
```

Out[56]:

```
PRIM_CONTRIBUTORY_CAUSE
UNABLE TO DETERMINE                                 178000
FAILING TO YIELD RIGHT-OF-WAY                        50060
FOLLOWING TOO CLOSELY                                44170
NOT APPLICABLE                                       24382
IMPROPER OVERTAKING/PASSING                          22357
FAILING TO REDUCE SPEED TO AVOID CRASH               19062
IMPROPER BACKING                                     17763
IMPROPER LANE USAGE                                  16400
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE                  15097
IMPROPER TURNING/NO SIGNAL                           15034
Name: count, dtype: int64
```

In [55]:

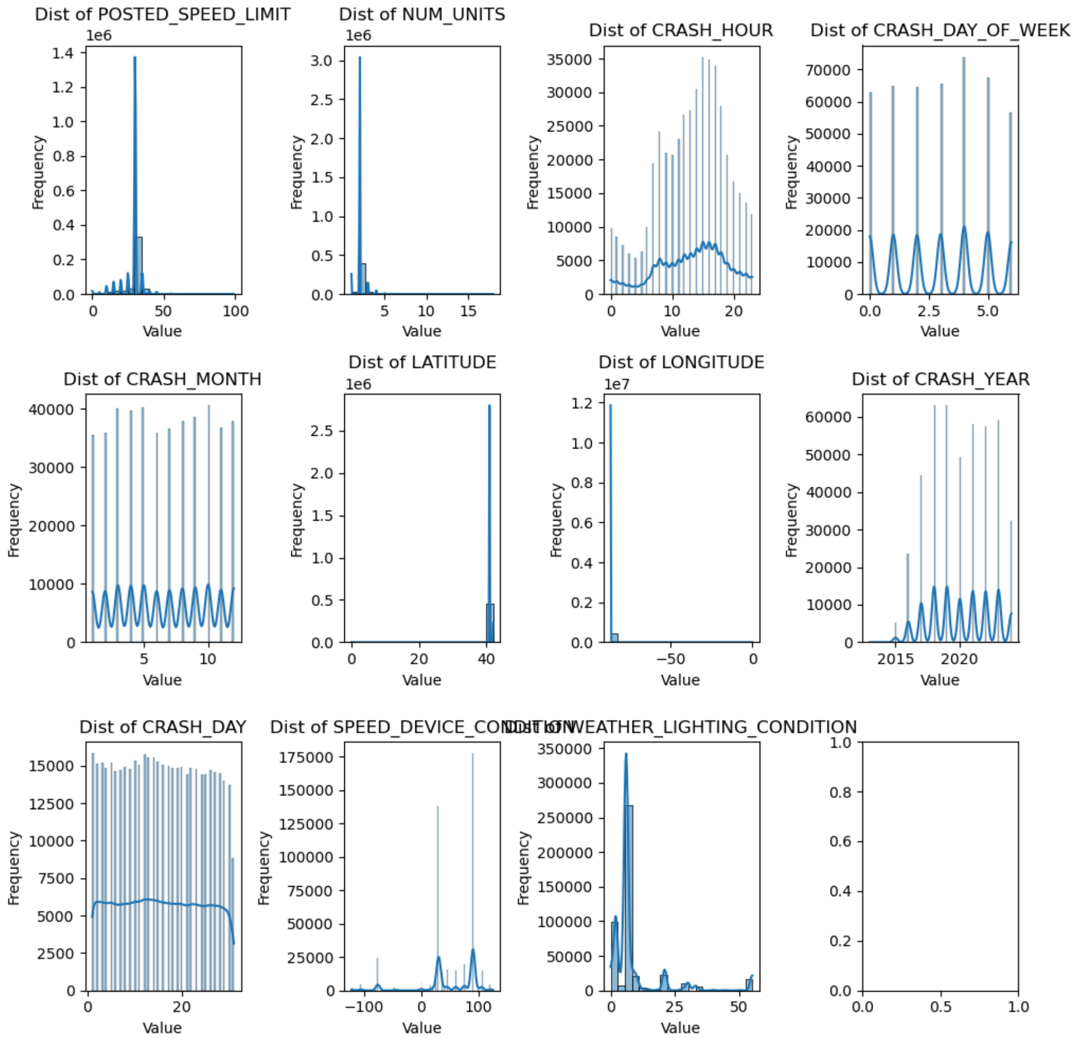
```
#Frequency of 10 contributory cause
# Get the top ten contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge

# Plot the top ten contributory causes
plt.figure(figsize=(6, 4))
top_10_causes.plot(kind='barh')
plt.title('Top 10 Primary Contributory Causes')
plt.ylabel('Primary Contributory Cause')
plt.xlabel('Frequency')
plt.show()
```



4.1.2. Numeric Variables

In [40]:

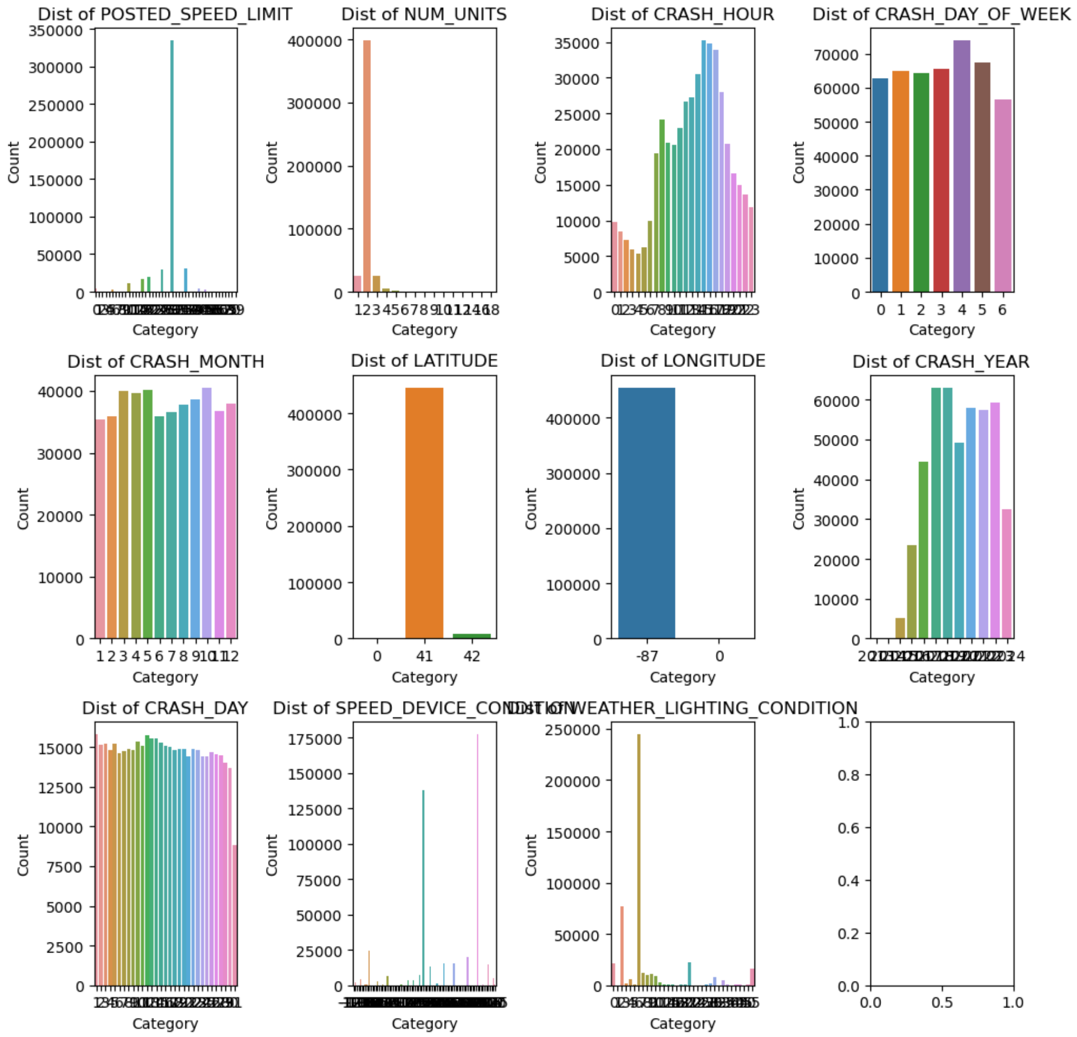
```
# distribution of numerical columns
fig, axs = plt.subplots(ncols=4, nrows=3, figsize=(10, 10))

for ax, col in zip(axs.ravel(), df_new.select_dtypes(include='number').
    sns.histplot(df_new[col], kde=True, ax=ax)
    ax.set_xlabel('Value')
    ax.set_ylabel('Frequency')
    ax.set_title(f'Dist of {col}')


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



4.1.3. Categorical Variables

In [41]:

```
# distribution of numerical columns
fig, axs = plt.subplots(ncols=4, nrows=3, figsize=(10, 10))

for ax, col in zip(axs.ravel(), df_new.select_dtypes(include='number').
    sns.countplot(data=df_new, x=col, ax=ax,)
    ax.set_xlabel('Category')
    ax.set_ylabel('Count')
    ax.set_title(f'Dist of {col}')



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

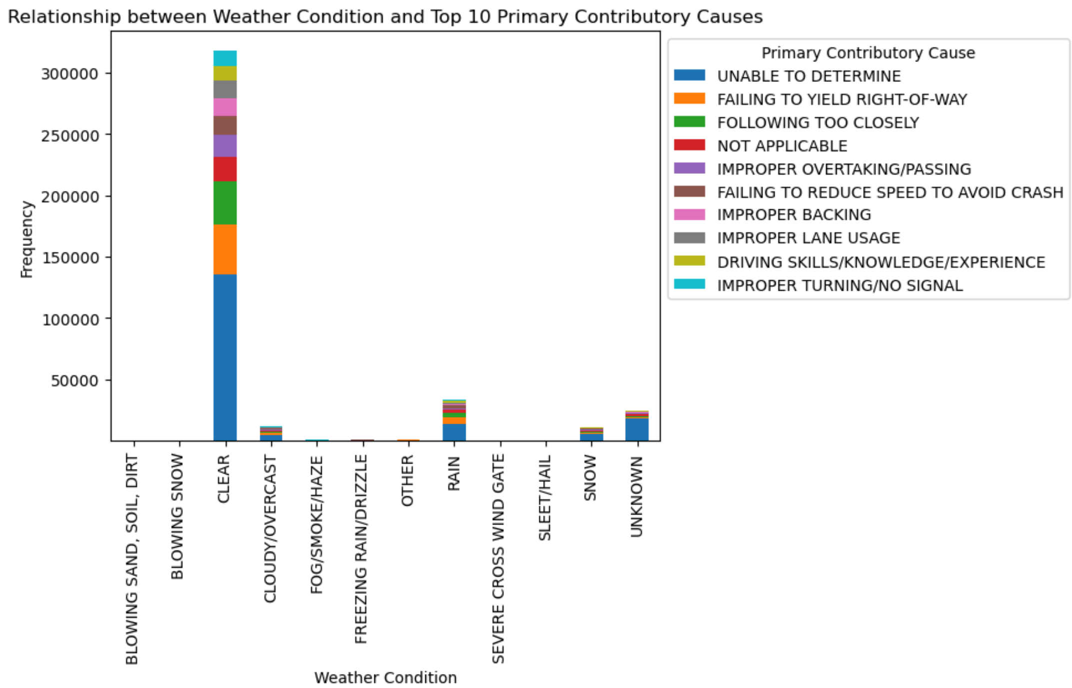


4.2 Bivaret Analysis

4.2.1. Categorical vs Categorical

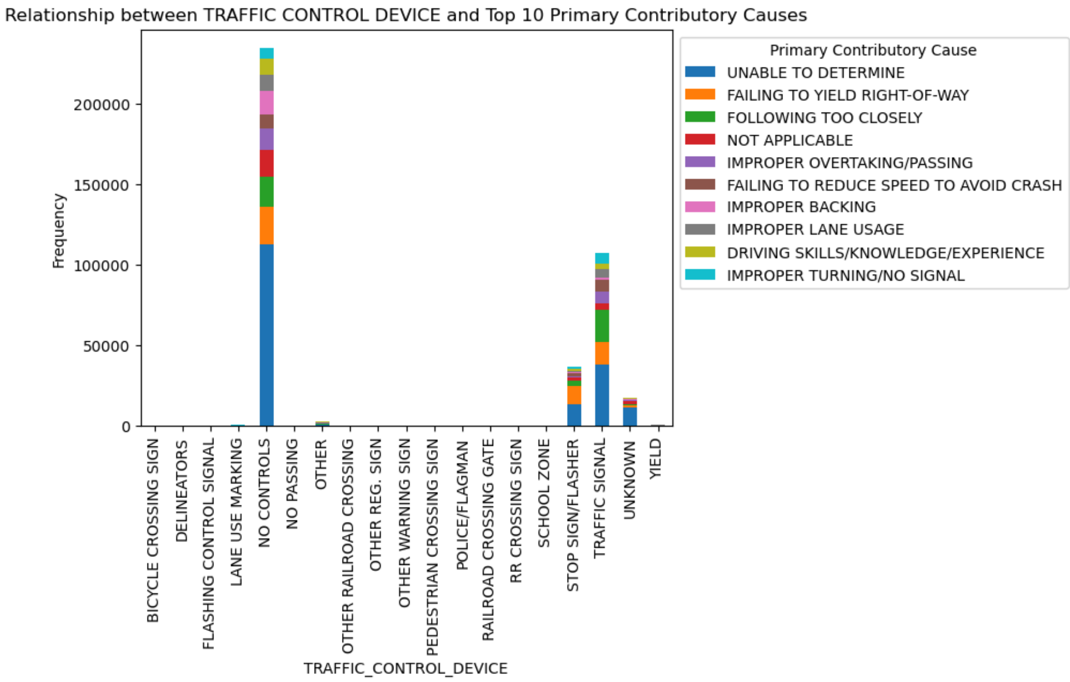
In [42]:

```
#Relationship between Weather Condition and Top 10 Primary Contributory
# Create a cross-tabulation table
cross_tab = pd.crosstab(df_new['WEATHER_CONDITION'], df_new['PRIM_CONTRI

# Calculate the top 10 primary contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge

# Filter the cross-tabulation table to include only the top 10 causes
cross_tab_top_10 = cross_tab[top_10_causes]

# Plot the clustered bar chart
cross_tab_top_10.plot(kind='bar', stacked=True)
plt.title('Relationship between Weather Condition and Top 10 Primary Co
plt.xlabel('Weather Condition')
plt.ylabel('Frequency')
plt.legend(title='Primary Contributory Cause', loc='upper left', bbox_t
plt.show()
```



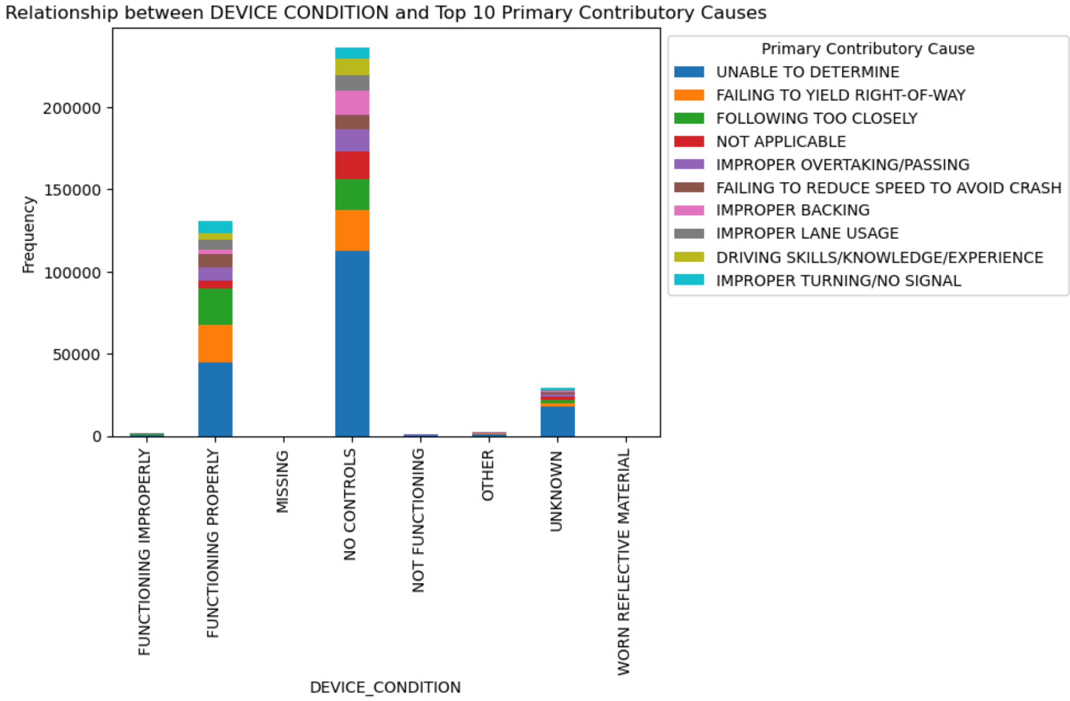
In [43]:

```
#Relationship between Weather Condition and Top 10 Primary Contributory
# Create a cross-tabulation table
cross_tab = pd.crosstab(df_new['TRAFFIC_CONTROL_DEVICE'], df_new['PRIM_(

# Calculate the top 10 primary contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge:

# Filter the cross-tabulation table to include only the top 10 causes
cross_tab_top_10 = cross_tab[top_10_causes]

# Plot the clustered bar chart
cross_tab_top_10.plot(kind='bar', stacked=True)
plt.title('Relationship between TRAFFIC CONTROL DEVICE and Top 10 Primar
plt.xlabel('TRAFFIC_CONTROL_DEVICE')
plt.ylabel('Frequency')
plt.legend(title='Primary Contributory Cause', loc='upper left', bbox_t(
plt.show()
```



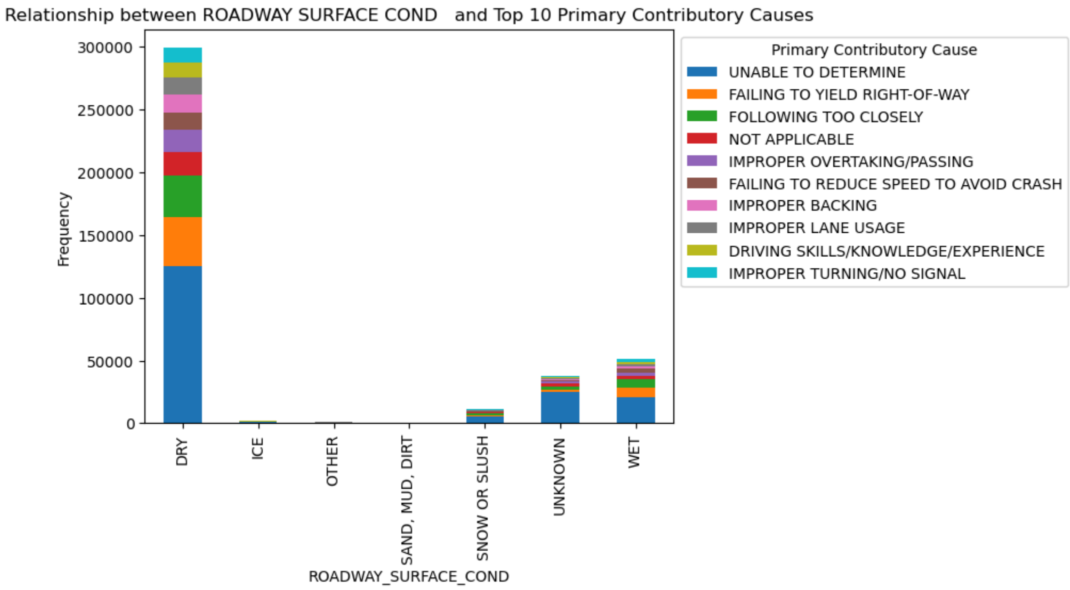
In [44]:

```
#Relationship between Weather Condition and Top 10 Primary Contributory
# Create a cross-tabulation table
cross_tab = pd.crosstab(df_new['DEVICE_CONDITION'], df_new['PRIM_CONTRI

# Calculate the top 10 primary contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge

# Filter the cross-tabulation table to include only the top 10 causes
cross_tab_top_10 = cross_tab[top_10_causes]

# Plot the clustered bar chart
cross_tab_top_10.plot(kind='bar', stacked=True)
plt.title('Relationship between DEVICE CONDITION and Top 10 Primary Con
plt.xlabel('DEVICE_CONDITION')
plt.ylabel('Frequency')
plt.legend(title='Primary Contributory Cause', loc='upper left', bbox_t
plt.show()
```



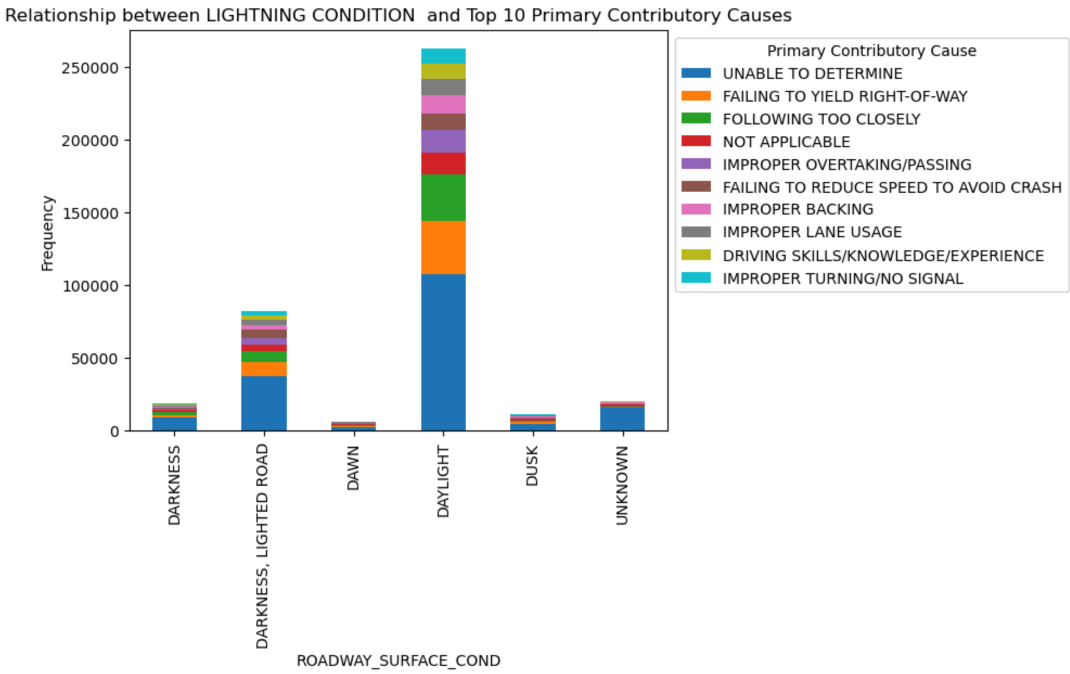
In [45]:

```
# Create a cross-tabulation table
cross_tab = pd.crosstab(df_new['ROADWAY_SURFACE_COND'], df_new['PRIM_CO
# Calculate the top 10 primary contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge
# Filter the cross-tabulation table to include only the top 10 causes
cross_tab_top_10 = cross_tab[top_10_causes]

# Plot the clustered bar chart
cross_tab_top_10.plot(kind='bar', stacked=True)
plt.title('Relationship between ROADWAY SURFACE COND   and Top 10 Primar
plt.xlabel('ROADWAY_SURFACE_COND')
plt.ylabel('Frequency')
plt.legend(title='Primary Contributory Cause', loc='upper left', bbox_t
plt.show()
```



In [46]:

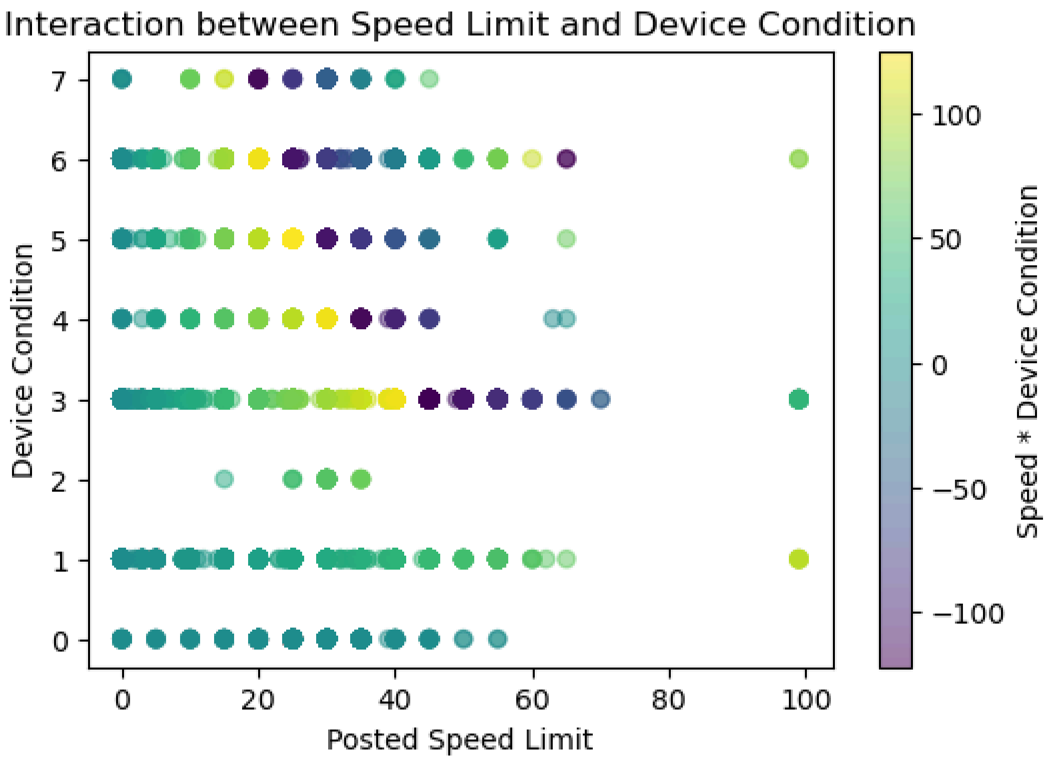
```
# Create a cross-tabulation table
cross_tab = pd.crosstab(df_new['LIGHTING_CONDITION'], df_new['PRIM_CONTI
# Calculate the top 10 primary contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge
# Filter the cross-tabulation table to include only the top 10 causes
cross_tab_top_10 = cross_tab[top_10_causes]
# Plot the clustered bar chart
cross_tab_top_10.plot(kind='bar', stacked=True)
plt.title('Relationship between LIGHTNING CONDITION  and Top 10 Primary
plt.xlabel('ROADWAY_SURFACE_COND')
plt.ylabel('Frequency')
plt.legend(title='Primary Contributory Cause', loc='upper left', bbox_t
plt.show()
```



4.2.2. Numerical vs Categorical

In [48]:

```
#Scatter Plot of Speed and Device Condition Interaction
plt.figure(figsize=(6, 4))
plt.scatter(df_new['POSTED_SPEED_LIMIT'], df_new['DEVICE_CONDITION'].ca
plt.colorbar(label='Speed * Device Condition')
plt.title('Interaction between Speed Limit and Device Condition')
plt.xlabel('Posted Speed Limit')
plt.ylabel('Device Condition')
plt.show()
```



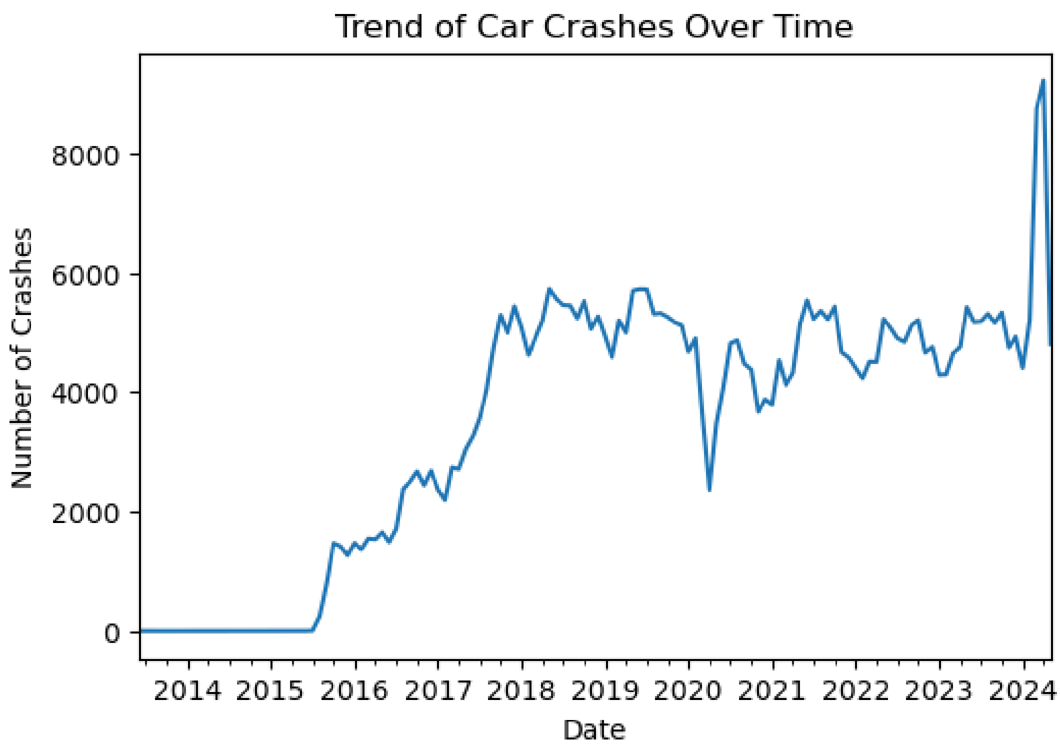
4.2.3. Categorical vs Date/Time

In [49]:

```
# Trend of car crash over time
df_new.assign(CRASH_DATE=pd.to_datetime(df_new['CRASH_DATE'])).resample
title='Trend of Car Crashes Over Time', xlabel='Date', ylabel='Number o
```

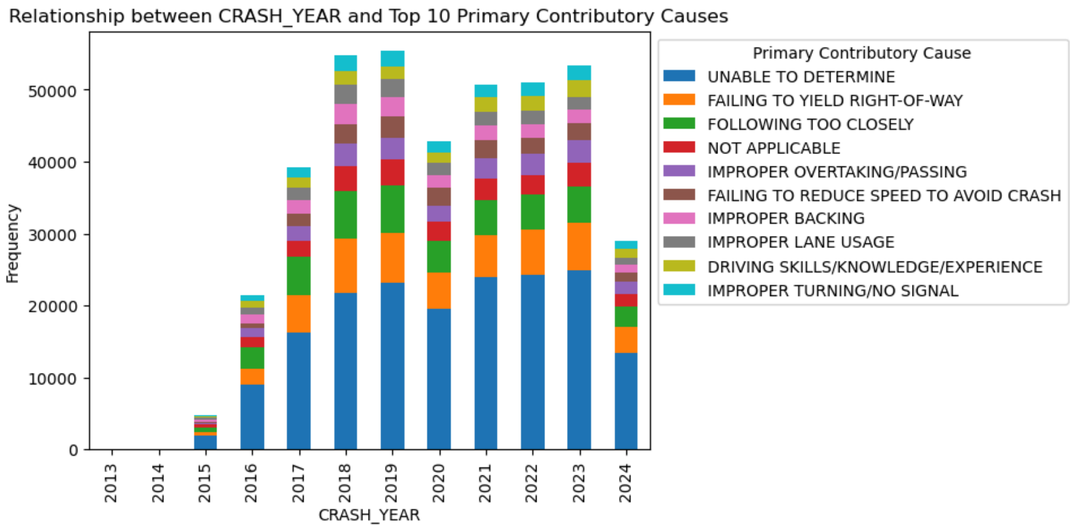
Out[49]:

```
<Axes: title={'center': 'Trend of Car Crashes Over Time'}, xlabel='Dat
e', ylabel='Number of Crashes'>
```



In [50]:

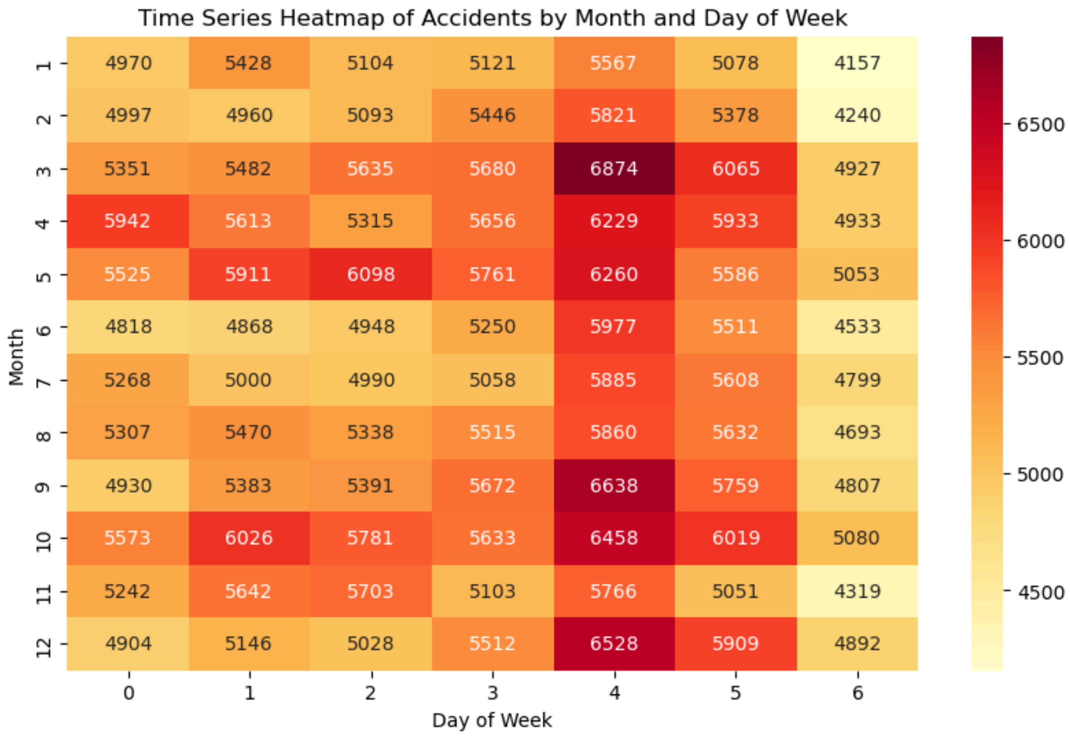
```
#Primmary contributory cause over time
#top 10 pmc
# Create a cross-tabulation table
cross_tab = pd.crosstab(df_new['CRASH_YEAR'], df_new['PRIM_CONTRIBUTORY_

# Calculate the top 10 primary contributory causes
top_10_causes = df_new['PRIM_CONTRIBUTORY_CAUSE'].value_counts().nlarge

# Filter the cross-tabulation table to include only the top 10 causes
cross_tab_top_10 = cross_tab[top_10_causes]

# Plot the clustered bar chart
cross_tab_top_10.plot(kind='bar', stacked=True)
plt.title('Relationship between CRASH_YEAR and Top 10 Primary Contribut
plt.xlabel('CRASH_YEAR')
plt.ylabel('Frequency')
plt.legend(title='Primary Contributory Cause', loc='upper left', bbox_t
plt.show()
```



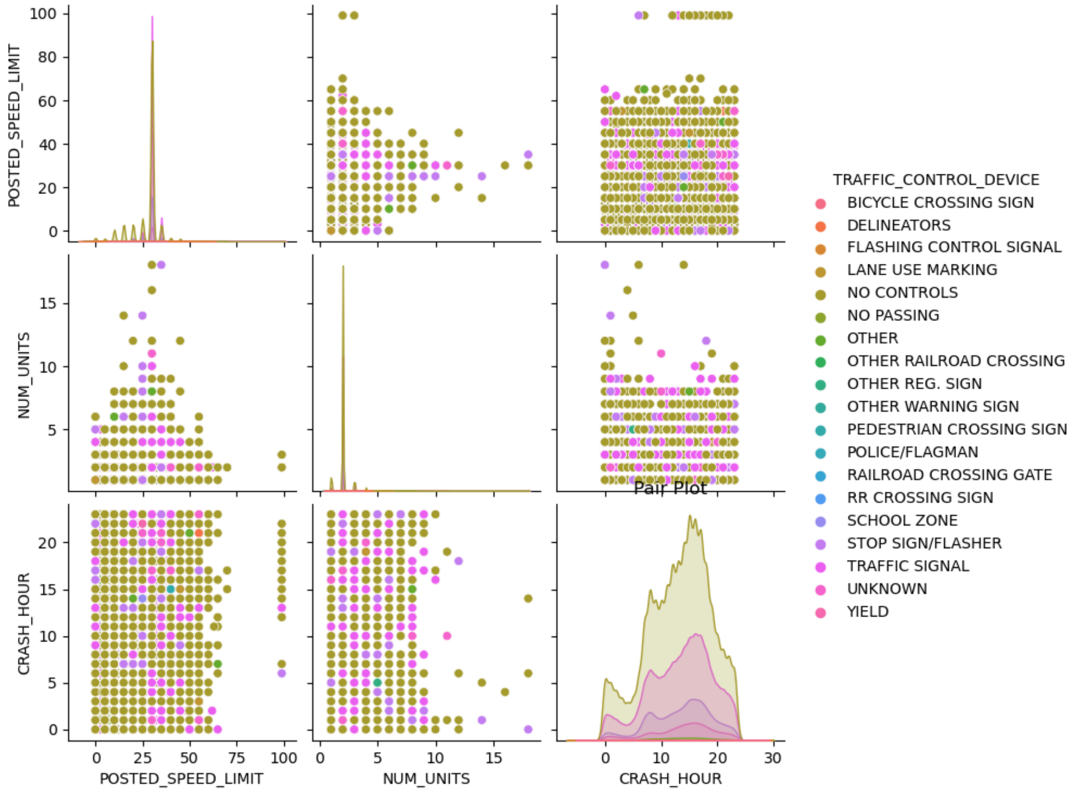
4.3. Multivaret Analysis

In [51]:

```
#Time Series Heatmap of Accidents by Month and Day of Week
# Create pivot table for heatmap
heatmap_data = df_new.pivot_table(index='CRASH_MONTH', columns='CRASH_D

# Plot heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, cmap='YlOrRd', annot=True, fmt='d')
plt.title('Time Series Heatmap of Accidents by Month and Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Month')
plt.show()
```



In [52]:
```
#using 3columns that can be ordinal or continous to plot a pair plot
# Pair plot
sns.pairplot(data=df_new, vars=['POSTED_SPEED_LIMIT', 'NUM_UNITS', 'CRA
plt.title('Pair Plot')
plt.show()
plt.tight_layout()
```

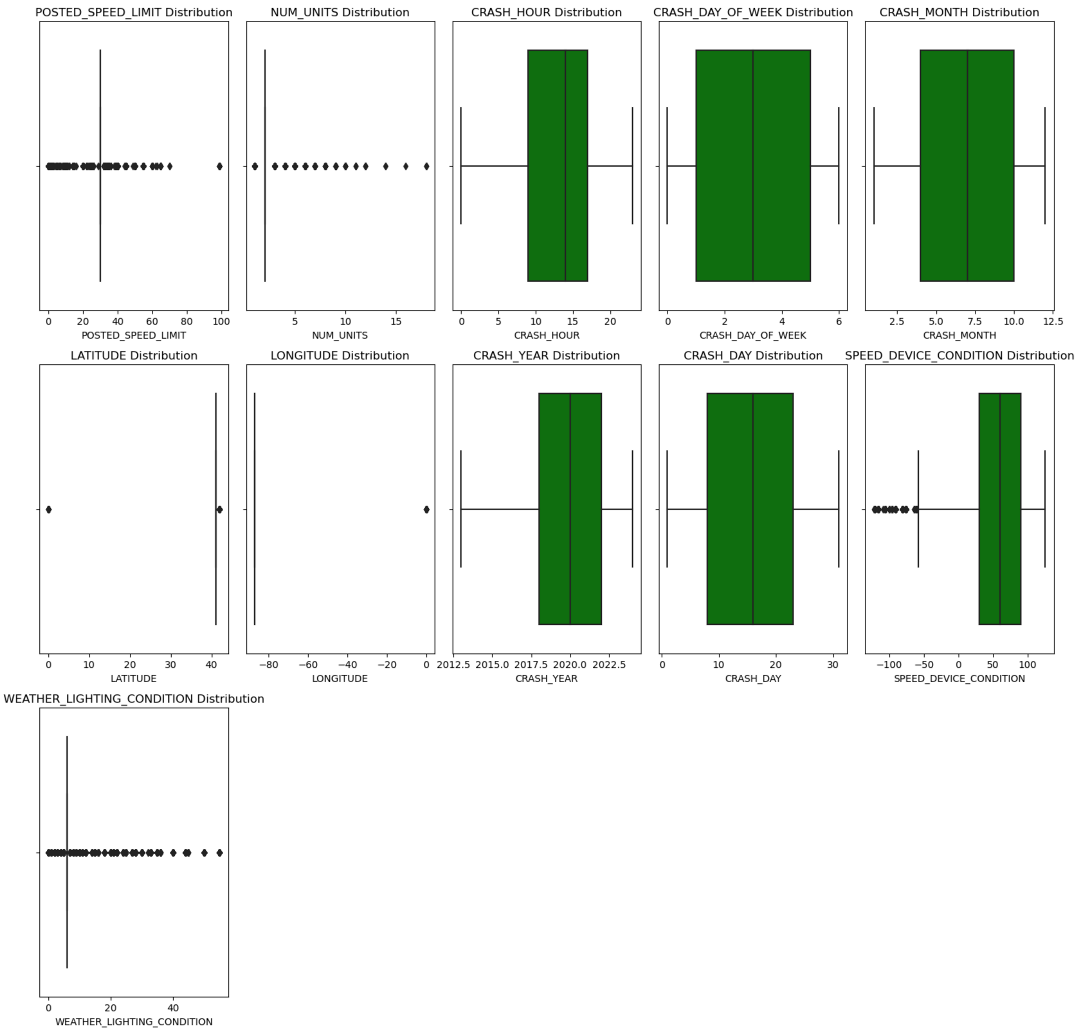



```
<Figure size 640x480 with 0 Axes>
```

4.3.1 Outliers Detection

In [53]:

```
# Detecting outliers in numerical columns
numerical_columns = df_new.select_dtypes(include=['int32', 'int8']).col
num_plots = len(numerical_columns)
num_cols = 5
num_rows = math.ceil(num_plots / num_cols)

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(15, 5

# Flatten the axes array for easy indexing
axes = axes.flatten()

for i, column in enumerate(numerical_columns):
    sns.boxplot(data=df_new, x=column, color='green', ax=axes[i])
    axes[i].set_title(f'{column} Distribution')
    axes[i].set_xlabel(column)

# Remove any empty subplots
for j in range(i + 1, num_rows * num_cols):
    fig.delaxes(axes[j])

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



In [54]:

```
# Save the cleaned
df_new.to_csv('CleanedNEW_Eda.csv', index=False)
```

5 . Data Preprocessing

5.1. Encoding

In [66]:

```
from sklearn.preprocessing import LabelEncoder

class DataEncoder:
    def __init__(self, data):
        self.df_new = pd.DataFrame(data)  # Convert the dictionary to a

    def label_encode(self, columns=None):
        # If columns is None, use all columns
        if columns is None:
            columns = self.df_new.columns

        # Apply label encoding to specified columns
        label_encoder = LabelEncoder()
        for col in columns:
            self.df_new[col] = label_encoder.fit_transform(self.df_new[

        return self.df_new

# Sample data dictionary
data = {

    'TRAFFIC_CONTROL_DEVICE': ['NO CONTROLS','TRAFFIC SIGNALS'],
    'DEVICE_CONDITION': ['NO CONTROLS','FUNCTIONING PROPERLY'],
    'WEATHER_CONDITION': ['CLEAR','RAIN'],
    'LIGHTING_CONDITION': ['DAYLIGHT','DARKNESS, LIGHTED ROAD'],
    'ROADWAY_SURFACE_COND': ['DRY','WET'],
}

# Create an instance of the DataEncoder
encoder = DataEncoder(data)

# Specify columns to encode (for demonstration, we'll encode all)
columns_to_encode = list(data.keys())

# Perform label encoding
df_new_encoded = encoder.label_encode(columns=columns_to_encode)

# Display the encoded DataFrame
df_new_encoded
```

Out[66]:

	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_(
0	0	1	0	
1	1	0	1	

In [67]:

```python
from sklearn.preprocessing import LabelEncoder

class DataEncoder:
    def __init__(self, data):
        self.df_new = pd.DataFrame(data)  # Convert the dictionary to a

    def label_encode(self, columns=None):
        # If columns is None, use all columns
        if columns is None:
            columns = self.df_new.columns

        # Apply label encoding to specified columns
        label_encoder = LabelEncoder()
        for col in columns:
            self.df_new[col] = label_encoder.fit_transform(self.df_new[

        return self.df_new

# Sample data dictionary
data = {
    'PRIM_CONTRIBUTORY_CAUSE': [
    'UNABLE TO DETERMINE' ,
    'FAILING TO YIELD RIGHT-OF-WAY',
    'FOLLOWING TOO CLOSELY' ,
    'NOT APPLICABLE',
    'IMPROPER OVERTAKING/PASSING',
    'FAILING TO REDUCE SPEED TO AVOID CRASH',
    'IMPROPER BACKING',
    'IMPROPER LANE USAGE',
    'DRIVING SKILLS/KNOWLEDGE/EXPERIENCE',
     'IMPROPER TURNING/NO SIGNAL']

}

# Create an instance of the DataEncoder
encoder = DataEncoder(data)

# Specify columns to encode (for demonstration, we'll encode all)
columns_to_encode = list(data.keys())

# Perform label encoding
df_new = encoder.label_encode(columns=columns_to_encode)

# Display the encoded DataFrame
df_new
```

Out[67]:

	PRIM_CONTRIBUTORY_CAUSE
0	9
1	2
2	3
3	8
4	6
5	1
6	4
7	5
8	0
9	7

5.5. Multicolinearity

5.5.1. Chi-Square

In [68]:

```
#Use chisquare to adress multicolinierity in categorical columns
class MulticollinearityHandler:
    def __init__(self, df_new_encoded, categorical_vars):
        self.df_new_encoded = df_new_encoded
        self.categorical_vars = categorical_vars
        self.results = []

    def chi_square_test(self, var1, var2):
        contingency_table = pd.crosstab(self.df_new_encoded[var1], self
        chi2, p, _, _ = chi2_contingency(contingency_table)
        return chi2, p

    def perform_tests(self):
        for var1, var2 in combinations(self.categorical_vars, 2):
            chi2, p = self.chi_square_test(var1, var2)
            self.results.append((var1, var2, chi2, p))

    def print_results(self):
        for var1, var2, chi2, p in self.results:
            print(f"Chi-square test between {var1} and {var2}:")
            print(f"Chi-square statistic: {chi2}")
            print(f"P-value: {p}\n")

    def suggest_drops(self, threshold=0.05):
        high_corr_pairs = [(var1, var2) for var1, var2, chi2, p in self
        variable_groups = {}

        # Grouping highly correlated variables
        for var1, var2 in high_corr_pairs:
            if var1 not in variable_groups and var2 not in variable_gro
                variable_groups[var1] = {var1, var2}
            elif var1 in variable_groups:
                variable_groups[var1].add(var2)
            elif var2 in variable_groups:
                variable_groups[var2].add(var1)

        # Suggest one variable to keep per group
        variables_to_keep = {min(group, key=len) for group in variable_

        # Variables to drop are those not suggested to keep
        variables_to_drop = set(self.categorical_vars) - variables_to_k

        return variables_to_drop

# implement
categorical_vars = ['TRAFFIC_CONTROL_DEVICE','DEVICE_CONDITION','LIGHTI
handler = MulticollinearityHandler(df, categorical_vars)

# Perform tests
handler.perform_tests()

# Print results
handler.print_results()

# Suggest variables to drop based on multicollinearity
variables_to_drop = handler.suggest_drops()
print(f"Suggested variables to drop: {variables_to_drop}")
```

```
Chi-square test between TRAFFIC_CONTROL_DEVICE and DEVICE_CONDITION:
Chi-square statistic: 690581.1689236385
P-value: 0.0

Chi-square test between TRAFFIC_CONTROL_DEVICE and LIGHTING_CONDITION:
Chi-square statistic: 42351.15985189342
P-value: 0.0

Chi-square test between TRAFFIC_CONTROL_DEVICE and WEATHER_CONDITION:
Chi-square statistic: 44299.47571747925
P-value: 0.0

Chi-square test between TRAFFIC_CONTROL_DEVICE and ROADWAY_SURFACE_CON
D:
Chi-square statistic: 42611.14510950528
P-value: 0.0

Chi-square test between DEVICE_CONDITION and LIGHTING_CONDITION:
Chi-square statistic: 38847.0180524664
P-value: 0.0

Chi-square test between DEVICE_CONDITION and WEATHER_CONDITION:
Chi-square statistic: 44012.587129293825
P-value: 0.0

Chi-square test between DEVICE_CONDITION and ROADWAY_SURFACE_COND:
Chi-square statistic: 53892.26917877303
P-value: 0.0

Chi-square test between LIGHTING_CONDITION and WEATHER_CONDITION:
Chi-square statistic: 209909.37617242814
P-value: 0.0

Chi-square test between LIGHTING_CONDITION and ROADWAY_SURFACE_COND:
Chi-square statistic: 137098.84655813643
P-value: 0.0

Chi-square test between WEATHER_CONDITION and ROADWAY_SURFACE_COND:
Chi-square statistic: 728998.8570421721
P-value: 0.0

Suggested variables to drop: {'TRAFFIC_CONTROL_DEVICE', 'LIGHTING_COND
ITION', 'ROADWAY_SURFACE_COND'}
```

5.2. Standardization

In [69]:

```
#Standardizing to scale numeric features to have a mean of 0 and a stand
#since i chose to keep outliers i chose standardization over scaling
class CustomRangeScaler:
    def __init__(self, range_dict):
        self.range_dict = range_dict

    def fit_transform(self, data):
        scaled_data = data.copy()
        for feature, (min_val, max_val) in self.range_dict.items():
            scaled_data[feature] = (scaled_data[feature] - min_val) / (
        return scaled_data

# Sample data (replace with your actual data loading step)
data = {
    'NUM_UNITS': [1, 18],
    'SPEED_DEVICE_CONTROL': [10, 360],
    'CRASH_DATE':[0,24],
}

# Define the desired range for each feature
range_dict = {
    'NUM_UNITS': [1, 18],
    'SPEED_DEVICE_CONTROL': [10, 360],
    'CRASH_DATE':[0,31]
}

# Scale the data
scaler = CustomRangeScaler(range_dict)
df_new_scaled = scaler.fit_transform(pd.DataFrame(data))

# Display the scaled DataFrame and original df
print("Scaled Data:")
df_new_scaled
```

```
Scaled Data:
```

Out[69]:

	NUM_UNITS	SPEED_DEVICE_CONTROL	CRASH_DATE
0	0.0	0.0	0.000000
1	1.0	1.0	0.774194

In [70]:
```
#Concatinanting the dataframes
#Make sure indexes are the same
df_new_encoded.index = df_new_scaled.index

# Concatenate the DataFrames
df_combined = pd.concat([df_new_encoded, df_new_scaled], axis=1)

# Remove duplicate columns if any
df_combined = df_combined.loc[:,~df_combined.columns.duplicated()]

# Display the combined DataFrame
df_combined
```

Out[70]:

	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION	WEATHER_CONDITION	LIGHTING_(
0	0	1	0	
1	1	0	1	

5.2.1.Variance Inflation Factor

In [71]:
```
from statsmodels.stats.outliers_influence import variance_inflation_fac

# Selecting numerical features for VIF calculation
numeric_features = df_combined.select_dtypes(include=['float64', 'int64

# Calculating VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = numeric_features.columns
vif_data["VIF"] = [variance_inflation_factor(numeric_features.values, i

# Display the VIF data
vif_data
```

Out[71]:

	Feature	VIF
0	NUM_UNITS	inf
1	SPEED_DEVICE_CONTROL	inf
2	CRASH_DATE	inf

In [72]:
```
#dropping all correlated variables
df_combined = df_combined.drop(columns=['SPEED_DEVICE_CONTROL','NUM_UNI
```

5.6. Splitting the Data

In [84]:
```
#Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Separate features and target
X = df[['PRIM_CONTRIBUTORY_CAUSE']]
y = df['PRIM_CONTRIBUTORY_CAUSE']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Verify shapes
print("X_train shape:", X_train_scaled.shape)
print("X_test shape:", X_test_scaled.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (8, 1)
X_test shape: (2, 1)
y_train shape: (8,)
y_test shape: (2,)
```

5.7. Scaling

In [85]:
```
# Scaling the features
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display scaled features
print("X_train_scaled:\n", X_train_scaled)
print("X_test_scaled:\n", X_test_scaled)
```

```
X_train_scaled:
 [[0. ]
 [1. ]
 [0.5]
 [1. ]
 [0. ]
 [1. ]
 [0. ]
 [1. ]]
X_test_scaled:
 [[1. ]
 [0.5]]
```

6.MODELLING

1. Logistic Regression Model(Baseline-Model)

In [87]:

```
#Logistic regression
#Train the model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classific

# Initialize the model
model = LogisticRegression()

# Train the model
model.fit(X_train_scaled, y_train)
```

Out[87]:

```
LogisticRegression()
```

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.

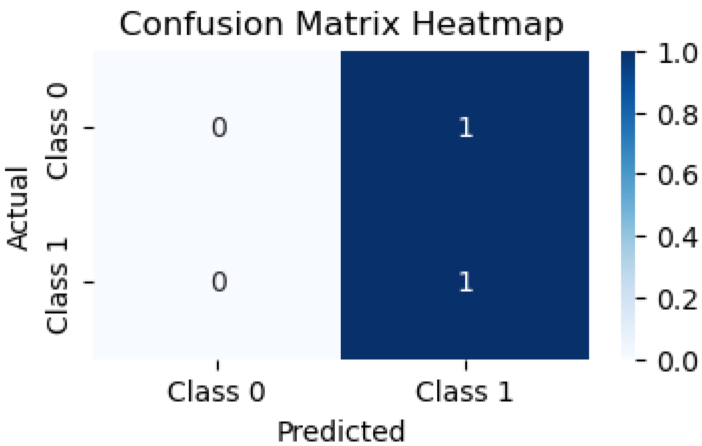
In [98]:

```
# Make predictions on the training data
y_pred_train = model.predict(X_train_scaled)

# Make predictions on the testing data
y_pred_test = model.predict(X_test_scaled)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)

#The model performs well on the training data but poorly on the test da
```

```
Training Accuracy: 0.875
Testing Accuracy: 0.5
```

In [104]:

```
# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)

# Create a heatmap
plt.figure(figsize=(4, 2))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix Heatmap')
plt.show()

print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```




```
Confusion Matrix:
 [[0 1]
 [0 1]]
Classification Report:
               precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           2       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

Hyperparameter tuning for logistic Regression Model

In [99]:

```
#import necessary libraries
from sklearn.model_selection import GridSearchCV, StratifiedKFold
# Hyperparameter tuning with adjusted number of splits
param_grid = {
    'C': [0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga']
}

# Use StratifiedKFold to ensure proper handling of small datasets
cv = StratifiedKFold(n_splits=2)  # Adjusted to 2 splits for this small

grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=cv, sco
grid_search.fit(X_train_scaled, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Cross-validation Score:", best_score)

best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test_scaled)
best_test_accuracy = accuracy_score(y_test, y_pred_best)
print("Best Testing Accuracy:", best_test_accuracy)
```

```
Best Parameters: {'C': 10, 'solver': 'liblinear'}
Best Cross-validation Score: 0.875
Best Testing Accuracy: 0.5
```

In [108]:

```
#Regularisation
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Regularization parameter
C = 1.0  # Inverse of regularization strength; smaller values specify s

# Create logistic regression model with L2 regularization (Ridge)
logreg_l2 = LogisticRegression(penalty='l2', C=C, solver='liblinear')

# Train the model
logreg_l2.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_l2 = logreg_l2.predict(X_test_scaled)

# Calculate accuracy on the test set
accuracy_l2 = accuracy_score(y_test, y_pred_l2)
print("Accuracy with L2 regularization:", accuracy_l2)
```

```
Accuracy with L2 regularization: 0.5
```

2.DecisionTreeClassifier

In [159]:

```
#Train the model
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Create a DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Train the classifier
clf.fit(X_train, y_train)
```

Out[159]:

```
DecisionTreeClassifier()
```

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.

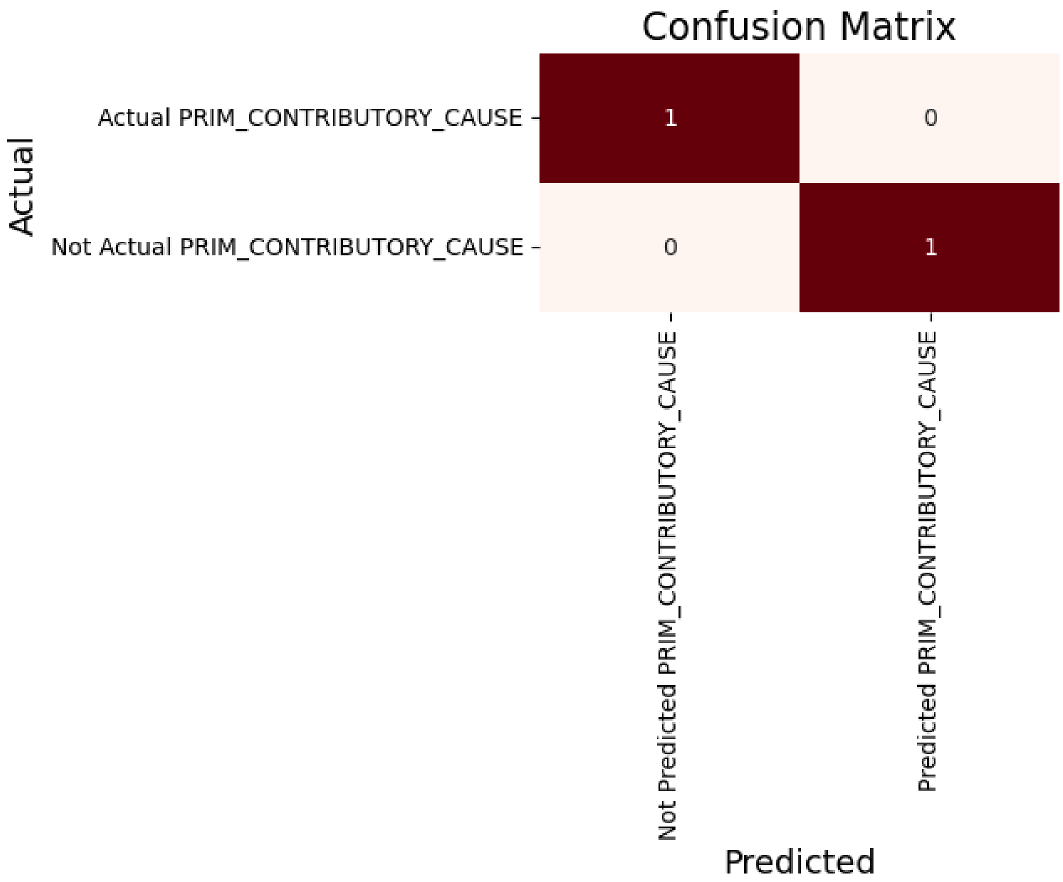
In [162]:

```
# Make predictions
y_pred_dec = clf.predict(X_test)

# Print the predictions
print("Predictions:", y_pred)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Predictions: [2 0]
Accuracy: 0.5
```

In [163]:

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_dec)

# Plot confusion matrix
plt.figure(figsize=(4,2))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', cbar=False,
            xticklabels=['Not Predicted PRIM_CONTRIBUTORY_CAUSE', 'Pred
            yticklabels=['Actual PRIM_CONTRIBUTORY_CAUSE', 'Not Actual 
plt.xlabel('Predicted', fontsize=14)
plt.ylabel('Actual', fontsize=14)
plt.title('Confusion Matrix', fontsize=16)
plt.show()
```



Hyperparameter tuning for logistic DecisionTreeClassifier

In [164]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# Define the parameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the Decision Tree Classifier
decision_tree = DecisionTreeClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=decision_tree, param_grid=param_gr

# Perform the grid search on the original training data
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Accuracy Score:", best_score)
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples
_leaf': 1, 'min_samples_split': 2}
Best Accuracy Score: 0.875
```

3. KNeighborsClassifier

In [165]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

#create knn object classifier
knn = KNeighborsClassifier(n_neighbors=5)

#Train the Classifier
knn.fit(X_train, y_train)
```

Out[165]:

```
KNeighborsClassifier()
```

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.

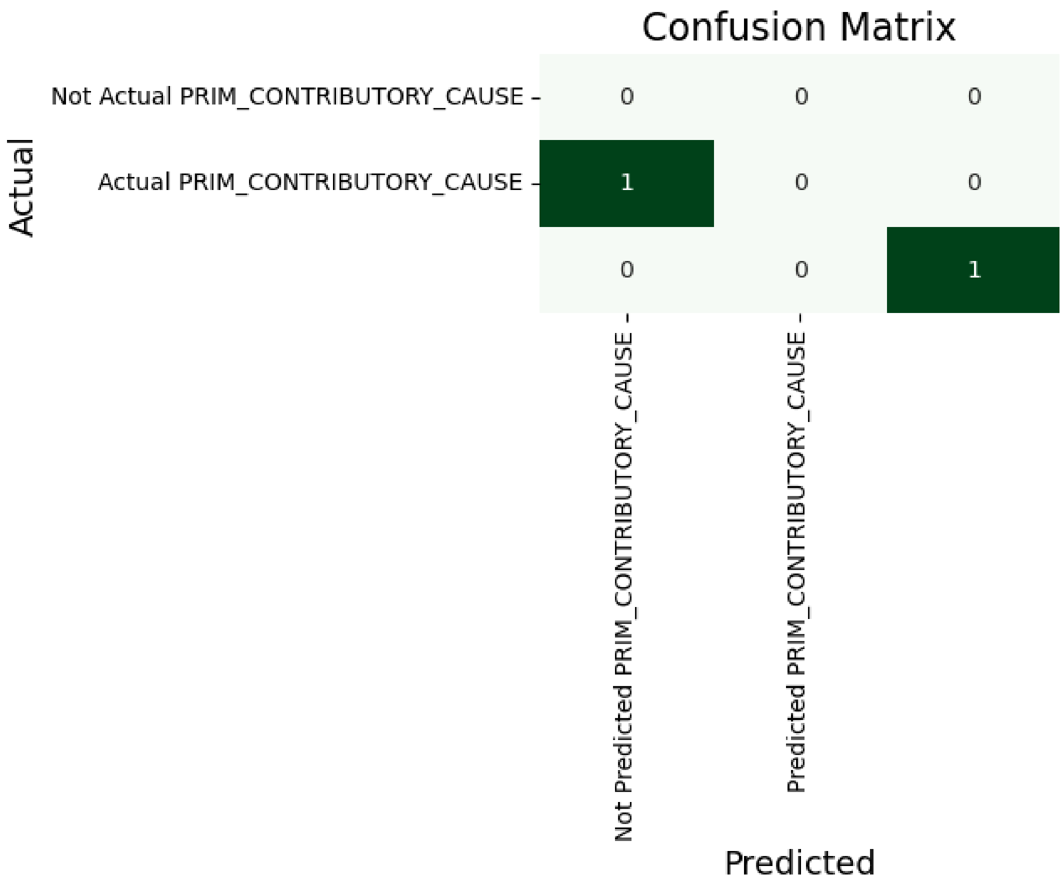
In [156]:

```
#Make predictions
y_pred_knn = knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

# Print the predictions
print("Predictions:", y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(cm)
```

```
Predictions: [2 0]
Accuracy: 0.5
Confusion Matrix:
[[0 0 0]
 [1 0 0]
 [0 0 1]]
```

In [158]:

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_knn)

# Plot confusion matrix
plt.figure(figsize=(4,2))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', cbar=False,
            xticklabels=['Not Predicted PRIM_CONTRIBUTORY_CAUSE', 'Pred
            yticklabels=['Not Actual PRIM_CONTRIBUTORY_CAUSE', 'Actual 
plt.xlabel('Predicted', fontsize=14)
plt.ylabel('Actual', fontsize=14)
plt.title('Confusion Matrix', fontsize=16)
plt.show()
```



4. RandomForestClassifier

In [143]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, conf

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier()

# Train the model on the training data
rf_model.fit(X_train, y_train)
```

Out[143]:

```
RandomForestClassifier()
```

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.

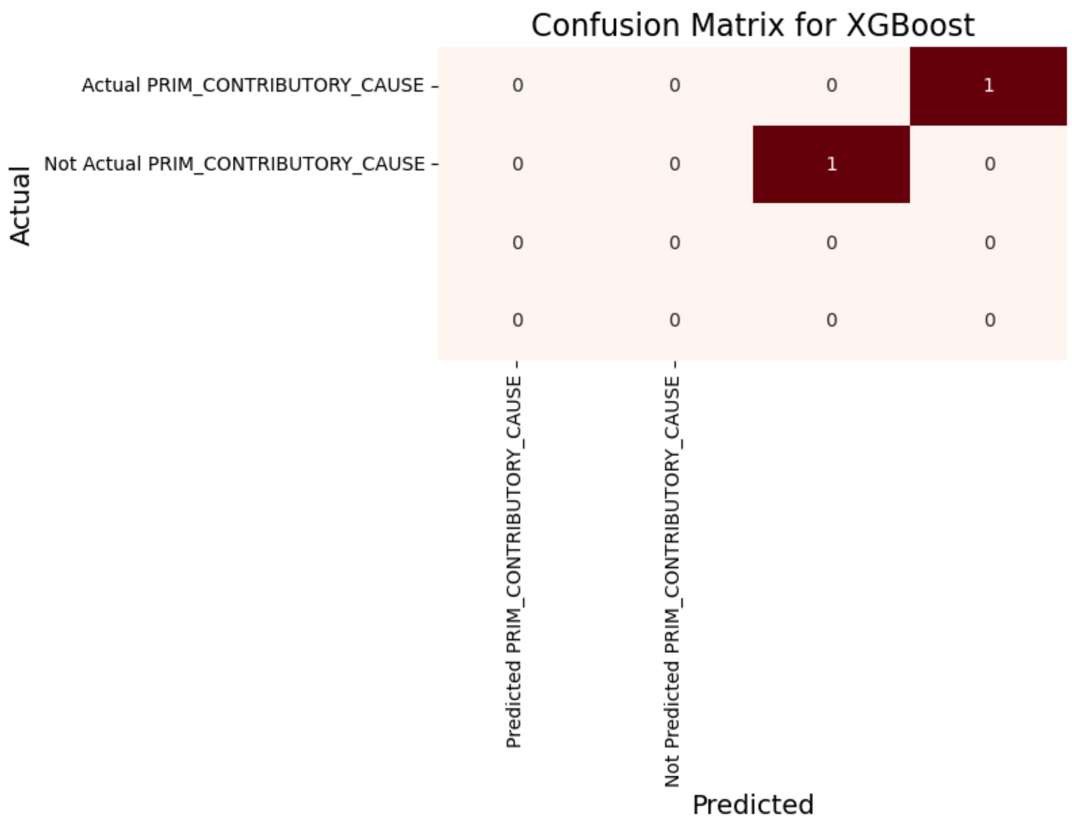
In [144]:

```
# Make predictions on the test data
rf_predictions = rf_model.predict(X_test)

# Evaluate the model
print("\nRandom Forest Classifier:")
print("Accuracy:", accuracy_score(y_test, rf_predictions))
print("Classification Report:\n", classification_report(y_test, rf_pred
print("Confusion Matrix:\n", confusion_matrix(y_test, rf_predictions))
```

```
Random Forest Classifier:
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           1       1.00      1.00      1.00         1
           2       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2

Confusion Matrix:
 [[1 0]
 [0 1]]
```

In [151]:

```
from sklearn.metrics import confusion_matrix
# Construct confusion matrix (example data)
cm = confusion_matrix(true_labels, predicted_labels)


# Plot confusion matrix
plt.figure(figsize=(6,3))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', cbar=False,
            xticklabels=['Predicted PRIM_CONTRIBUTORY_CAUSE', 'Not Pred
            yticklabels=['Actual PRIM_CONTRIBUTORY_CAUSE', 'Not Actual 
plt.xlabel('Predicted', fontsize=14)
plt.ylabel('Actual', fontsize=14)
plt.title('Confusion Matrix for XGBoost', fontsize=16)
plt.show()
```



7. MODEL EVALUATION

1.Logistic Regression Classifier(Baseline-Model)

Training Accuracy: The accuracy of the model on the training data is 0.875, which means that it correctly predicted 87.5% of the samples in the training set.

Testing Accuracy: The accuracy of the model on the testing data is 0.5, which means that it correctly predicted 50% of the samples in the testing set.

Confusion Matrix:

The confusion matrix shows the counts of true positive, true negative, false positive, and false negative predictions. In this case, the confusion matrix is: lua Copy code [[0 1] [0 1]] This indicates that the model correctly predicted one sample as class 2 (true positive), but misclassified one sample as class 2 when it actually belongs to class 1 (false positive). Classification Report:

For class 1: Precision: 0.00 (none of the predicted class 1 samples were actually class 1) Recall: 0.00 (none of the actual class 1 samples were correctly predicted) F1-score: 0.00 (harmonic mean of precision and recall) Support: 1 (one sample of class 1) For class 2: Precision: 0.50 (50% of the predicted class 2 samples were actually class 2) Recall: 1.00 (100% of the actual class 2 samples were correctly predicted) F1-score: 0.67 (harmonic

2.DecisionTreeClassifier

Predictions: The predictions made by the DecisionTreeClassifier model are [2, 0]. This suggests that the model predicted class 2 for the first sample and class 0 for the second sample.

Accuracy: The accuracy of the model on the testing dataset is 0.5. This means that the model correctly predicted 50% of the samples in the testing set.

Best Parameters: The best parameters found by grid search cross-validation for the DecisionTreeClassifier are:

criterion: 'gini' max_depth: None min_samples_leaf: 1 min_samples_split: 2 Best Accuracy Score: The best accuracy score achieved during grid search cross-validation is 0.875. This represents the mean accuracy of the model across different folds of the training data, using the best parameters.

Comparing the testing accuracy (0.5) with the best accuracy score from cross-validation (0.875), it seems that the model is not generalizing well to unseen data. This discrepancy suggests potential overfitting or issues with the model's performance on the testing set.

3.KNeighborsClassifier

3.KNeighborsClassifier Accuracy: The accuracy of the model on the testing dataset is 0.5. This means that the model correctly predicted 50% of the samples in the testing set.

Confusion Matrix:

The confusion matrix is a table that describes the performance of a classification model. It presents the counts of true positive, true negative, false positive, and false negative predictions. In this case, the confusion matrix is: lua Copy code [[0 0 0] [1 0 0] [0 0 1]] This indicates that the model correctly predicted one sample as class 0 (true negative) and one sample as class 2 (true positive), but misclassified one sample as class 1 when it actually belongs to class 0 (false positive). The confusion matrix provides insights into the performance of the model for each class. In this case, it seems that the model has issues correctly predicting samples, especially for classes 1 and 2. Further analysis and potentially model refinement are recommended to improve the model's performance. This could include adjusting hyperparameters, exploring different algorithms, or preprocessing the data to improve its quality.

4.RandomForestClassifier

Accuracy: The accuracy of the model on the testing dataset is 1.0. This means that the model correctly predicted all samples in the testing set, achieving 100% accuracy.

Classification Report:

The classification report provides a summary of precision, recall, and F1-score for each class, along with support (the number of samples in each class). For both classes 1 and 2, the precision, recall, and F1-score are all 1.0. This indicates perfect performance for both classes, with no false positives or false negatives. Confusion Matrix:

The confusion matrix is a table that describes the performance of a classification model. It presents the counts of true positive, true negative, false positive, and false negative predictions. In this case, the confusion matrix is: lua Copy code [[1 0] [0 1]] This indicates that the model correctly predicted one sample as class 1 (true positive) and one sample as class 2 (true positive), with no misclassifications.

CONCLUSION

Data Quality and Availability:This data had many columns,carefully handling of the multiclass data is key.

Feature Importance: Weather Condition,Lighting Condition,Device Control and Traffic Control Devices have proven to be among the factors that Facilitates primary car crashes.

Interpretability: By using RandomClassifier for prediction of car crashes one is sure of Accuracy.

RECOMMENDATION

Data Collaboration and Integration: Collaborate with relevant stakeholders, including local authorities, transportation departments, and law enforcement agencies, to access and integrate diverse datasets related to car accidents. This collaboration will enrich the analysis and enhance the predictive capabilities of the classifier.

Feature Engineering: Explore advanced feature engineering techniques to extract valuable insights from raw data, such as temporal patterns, spatial relationships, and interaction effects between variables. This process will improve the discriminatory power of the classifier and enhance its predictive accuracy.

Model Optimization: Continuously optimize the developed classifier by fine-tuning model parameters, exploring ensemble methods, and experimenting with advanced machine learning techniques. This iterative process will improve the model's performance and adaptability to changing patterns in car accidents over time.

Community Engagement: Engage with the community through educational campaigns, public forums, and feedback mechanisms to raise awareness about road safety and solicit input on potential interventions. This involvement will foster a collaborative approach to addressing the underlying causes of car accidents and promoting safer driving behaviors.

Policy Implementation: Translate the insights generated by the classifier into actionable policies and interventions aimed at reducing the frequency and severity of car accidents in Chicago. Work closely with policymakers, urban planners, and law enforcement agencies to implement targeted measures that address the identified risk factors effectively.

Monitoring and Evaluation: Establish a framework for monitoring and evaluating the impact of implemented interventions on road safety outcomes. Continuously track key performance indicators, such as accident rates, injury severity, and compliance with traffic regulations, to assess the effectiveness of interventions and inform future decision-making

In [ ]: