Primary Cause for Car Crashes in Chicago

Author: Aziza Gulyamova

Overview

Improving traffic safety is a major concern for any metropolitan city. The main approach to prevent the incidents and traffic crashes is to be able to predict this kind of situations and be ready to take action.

My goal for this project is to build a classification model that will be able to analyze the primary causes for car crashes. Being able to design a model that can accurately predict what kind of behavior did the crash had will allow the city to effectively act against it. If we know the cause for an incident, a city can then plan appropriately as to what measures should be taken to prevent them from happening again. In this project, I will be looking at car crash data from the city of Chicago.

Data

The Traffic Crashes and Traffic Crashes - People data comes from the Chicago Data Portal, an open data source maintained by the city of Chicago. The datasets contain all traffic crashes reports going back to 2017. Each crash incident has a unique crash record ID and report number associated with it, which allows for cross-referencing on the dashboards provided for the datasets.

The Crashes dataset contains a number of details related to the incident, such as location/time information, conditions of the road and traffic safety device functionality. The most important detail available is the primary contributory cause for the crash.

The People dataset contains a details associated with people involvesd in crash, like drivers, passangers and pedastrians. The data provider is mostly related to injuries of the person, personal information and after crash acrtivities. Links to the datasets:

Crashes Dataset: [https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if (https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if)](https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if)

People Dataset: [https://data.cityofchicago.org/Transportation/Traffic-Crashes-People/u6pd-qa9d (https://data.cityofchicago.org/Transportation/Traffic-Crashes-People/u6pd-qa9d)](https://data.cityofchicago.org/Transportation/Traffic-Crashes-People/u6pd-qa9d)

Plan of Analysis

▼ **Data Cleaning**

- Import Packages
- Upload Dataset
- Explore Crashes Dataset
- Clean Crashes Dataset
- Explore People Dataset
- Clean People Dataset
- Merge Datasets
- Clean Merged Dataset

▼ **Exploration Analysis**

- Descriptive Analysis
- Analysis of Data as a Whole
- Crashes with No Injury
- Crashes with Fatal Injury
- Binning of Primary Cause Categories

Train Test Split

▼ **Logistic Regression Classifier**

- **Model 1: Logistic Regression with All Features**
- Model Evaluation
- Grid Search for Best C - Value
- Model Summary

▼ **K Nearest Neighbors Classifier**

- **Model 2: KNN with All Features**
- Model Evaluation
- Model Summary

▼ **Decision Tree Classifier**

- Model Evaluation
- Model Summary

Modeling Conclusion

Evaluation of Final Model

Recommendations Based on Final Model

Next Step

Data Cleaning

Before proceeding to any analysis and modeling, I will need to upload necessary packages and upload dataset. After that, the data needs to bee explored and cleaned from unnecessary columns and observations.

Import Packages

In [1]:
```
import pandas as pd
import numpy as np

# packages for visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# packages for transformation
from imblearn.over_sampling import SMOTE, SMOTENC
from imblearn.pipeline import Pipeline, make_pipeline

#packages for preproccesing
from sklearn.model_selection import train_test_split, cross_val_score, KI
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardSc
from sklearn.compose import ColumnTransformer

# packages for metrics and evaluation
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, clas
mean_squared_error, make_scorer, precision_score, recall_score, accuracy_

# packages for classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier


# Import additional files with statistical functions
import sys
import os

module_path = os.path.abspath(os.path.join('../src'))
if module_path not in sys.path:
    sys.path.append(module_path)

import explore_data as ed
import model_functions as mf
```

The **ModelHistory** class below will help to **store the information of each model that was built**. I will contain the **name of the model, accuracy scores and additional notes.**

In [2]:
```
class ModelHistory:

    def __init__(self, random_state=2021):
        self.scorer = 'accuracy'
        self.history = pd.DataFrame(columns=['Name', 'Accuracy_Score', 'I

    def report(self, pipeline, X, y, name, notes='', cv=10,):
        kf = KFold(n_splits=cv, random_state=2021, shuffle=True)
        scores = cross_val_score(pipeline, X, y,
                                 scoring=self.scorer, cv=kf)
        self.log_report(name, scores.mean(), notes)
        print('Average Score:', scores.mean())
        return scores

    def log_report(self, name, av_score, notes):
        frame = pd.DataFrame([[name, av_score, notes]], columns=['Name',
        self.history = self.history.append(frame)
        self.history = self.history.reset_index(drop=True)
        self.history = self.history.sort_values('Accuracy_Score')

    def print_error(self, name, error):
        print('{} has an average error of ${:.2f}'.format(name, error))
```

Upload Datasets

First, I upload Datasets into following variables:

- **crashes:** Traffic Crashes - Crashes
- **people:** Traffic Crashes. - People

*** All Datasets are uploaded as string variables in order to keep leading zeros**

In [3]:
```
crashes = pd.read_csv("../data/Traffic_Crashes_-_Crashes.csv", dtype=str
people = pd.read_csv("../data/Traffic_Crashes_-_People.csv", dtype=str)
states = pd.read_csv("../data/state_abbrev.csv", dtype=str) # file that :
```

Explore Crashes Dataset

Now, explore Crashes Data and check following information:

- What columns do we have in each of the datasets
- Are there any missing values in tables
- Are there duplicates in data

In [4]:
```
ed.show_info(crashes)
```

```
Lenght of Dataset: 496205
                               missing_values_% Data_type
CRASH_RECORD_ID                        0.000000    object
RD_NO                                  0.719259    object
CRASH_DATE_EST_I                      92.480930    object
CRASH_DATE                             0.000000    object
POSTED_SPEED_LIMIT                     0.000000    object
TRAFFIC_CONTROL_DEVICE                 0.000000    object
DEVICE_CONDITION                       0.000000    object
WEATHER_CONDITION                      0.000000    object
LIGHTING_CONDITION                     0.000000    object
FIRST_CRASH_TYPE                       0.000000    object
TRAFFICWAY_TYPE                        0.000000    object
LANE_CNT                              59.902661    object
ALIGNMENT                              0.000000    object
ROADWAY_SURFACE_COND                   0.000000    object
ROAD_DEFECT                            0.000000    object
REPORT_TYPE                            2.448182    object
CRASH_TYPE                             0.000000    object
INTERSECTION_RELATED_I                77.427676    object
NOT_RIGHT_OF_WAY_I                    95.279572    object
HIT_AND_RUN_I                         70.402354    object
DAMAGE                                 0.000000    object
DATE_POLICE_NOTIFIED                   0.000000    object
PRIM_CONTRIBUTORY_CAUSE                0.000000    object
SEC_CONTRIBUTORY_CAUSE                 0.000000    object
STREET_NO                              0.000000    object
STREET_DIRECTION                       0.000605    object
STREET_NAME                            0.000202    object
BEAT_OF_OCCURRENCE                     0.001008    object
PHOTOS_TAKEN_I                        98.746284    object
STATEMENTS_TAKEN_I                    97.978860    object
DOORING_I                             99.682389    object
WORK_ZONE_I                           99.359740    object
WORK_ZONE_TYPE                        99.495168    object
WORKERS_PRESENT_I                     99.845628    object
NUM_UNITS                              0.000000    object
MOST_SEVERE_INJURY                     0.204149    object
INJURIES_TOTAL                         0.201933    object
INJURIES_FATAL                         0.201933    object
INJURIES_INCAPACITATING                0.201933    object
INJURIES_NON_INCAPACITATING            0.201933    object
INJURIES_REPORTED_NOT_EVIDENT          0.201933    object
INJURIES_NO_INDICATION                 0.201933    object
```

```
INJURIES_UNKNOWN                      0.201933    object

CRASH_HOUR                            0.000000    object
CRASH_DAY_OF_WEEK                     0.000000    object
CRASH_MONTH                           0.000000    object
LATITUDE                              0.558035    object
LONGITUDE                             0.558035    object
LOCATION                              0.558035    object
```

Check for **duplicated values** in the **CRASH_RECORD_ID** column, since it's a column with **unique ID** numbers per observation.

In [5]:
```
crashes.CRASH_RECORD_ID.duplicated().sum()
```

Out[5]:
```
0
```

Investigate Columns with more than 50% Missing Values in Crashes Dataset

As it seems, the **crashes datase**t have mostly missing values for **columns with "_I"** ending. I will first looks through what values those columns have and **drop them if necessary**.

In [6]:
```
# The "missing_values" function returns list of column names with missin
crashes_m_vals = ed.missing_values(crashes, 60, 100)
# The "values" function prints out the value_counts for the list of colu
ed.values(crashes, crashes_m_vals)
```

```
Y    32428
N     4882
Name: CRASH_DATE_EST_I, dtype: int64


Y    106712
N      5293
Name: INTERSECTION_RELATED_I, dtype: int64


Y    21367
N     2056
Name: NOT_RIGHT_OF_WAY_I, dtype: int64


Y    140483
N      6382
Name: HIT_AND_RUN_I, dtype: int64


Y    4830
N    1391
Name: PHOTOS_TAKEN_I, dtype: int64
```

```
Y    8154
N    1875
Name: STATEMENTS_TAKEN_I, dtype: int64


Y    1072
N     504
Name: DOORING_I, dtype: int64


Y    2505
N     672
Name: WORK_ZONE_I, dtype: int64


CONSTRUCTION    1770
UNKNOWN          330
MAINTENANCE      249
UTILITY          156
Name: WORK_ZONE_TYPE, dtype: int64


Y    687
N     79
Name: WORKERS_PRESENT_I, dtype: int64
```

Result: Most of the columns appears to **have "YES" and "NO" values**. Considering that the dataset is **missing substantial amount of observations** for this columns, I will be **dropping them from the dataset.**

I will take a look to **"LANE_CNT" column**, since it is **missing almost 60% of values**

In [7]:
```python
crashes['LANE_CNT'].value_counts(normalize = True)
```

Out[7]:
```
2          0.458050
4          0.249174
1          0.163566
3          0.043586
0          0.040354
6          0.022617
5          0.009745
8          0.009585
7          0.000925
10         0.000814
99         0.000543
9          0.000332
11         0.000151
12         0.000146
20         0.000075
22         0.000065
16         0.000035
15         0.000035
14         0.000025
30         0.000025
40         0.000020
60         0.000015
21         0.000015
100        0.000010
25         0.000010
400        0.000005
17         0.000005
44         0.000005
28         0.000005
13         0.000005
45         0.000005
299679     0.000005
35         0.000005
19         0.000005
1191625    0.000005
218474     0.000005
24         0.000005
902        0.000005
433634     0.000005
80         0.000005
41         0.000005
Name: LANE_CNT, dtype: float64
```

Result: Considering that **"LANE_CNT" column represents the count of throught lines** according to dataset description, some of the values shown above are **misleading and unrelatable**, thus I will be **dropping this column.**

Clean Crashes Dataset

I will drop columns stated above from Crash dataset

In [8]:
```
crashes_drop_cols = ed.missing_values(crashes, 50, 100)
crashes.drop(columns = crashes_drop_cols,axis = 1, inplace = True)
ed.show_info(crashes)
```

```
Lenght of Dataset: 496205
                              missing_values_% Data_type
CRASH_RECORD_ID                       0.000000    object
RD_NO                                 0.719259    object
CRASH_DATE                            0.000000    object
POSTED_SPEED_LIMIT                    0.000000    object
TRAFFIC_CONTROL_DEVICE                0.000000    object
DEVICE_CONDITION                      0.000000    object
WEATHER_CONDITION                     0.000000    object
LIGHTING_CONDITION                    0.000000    object
FIRST_CRASH_TYPE                      0.000000    object
TRAFFICWAY_TYPE                       0.000000    object
ALIGNMENT                             0.000000    object
ROADWAY_SURFACE_COND                  0.000000    object
ROAD_DEFECT                           0.000000    object
REPORT_TYPE                           2.448182    object
CRASH_TYPE                            0.000000    object
DAMAGE                                0.000000    object
DATE_POLICE_NOTIFIED                  0.000000    object
PRIM_CONTRIBUTORY_CAUSE               0.000000    object
SEC_CONTRIBUTORY_CAUSE                0.000000    object
STREET_NO                             0.000000    object
STREET_DIRECTION                      0.000605    object
STREET_NAME                           0.000202    object
BEAT_OF_OCCURRENCE                    0.001008    object
NUM_UNITS                             0.000000    object
MOST_SEVERE_INJURY                    0.204149    object
INJURIES_TOTAL                        0.201933    object
INJURIES_FATAL                        0.201933    object
INJURIES_INCAPACITATING               0.201933    object
INJURIES_NON_INCAPACITATING           0.201933    object
INJURIES_REPORTED_NOT_EVIDENT         0.201933    object
INJURIES_NO_INDICATION                0.201933    object
INJURIES_UNKNOWN                      0.201933    object
CRASH_HOUR                            0.000000    object
CRASH_DAY_OF_WEEK                     0.000000    object
CRASH_MONTH                           0.000000    object
LATITUDE                              0.558035    object
LONGITUDE                             0.558035    object
LOCATION                              0.558035    object
```

Drop Observations

Since, some if the columns still have a small amount of missing values, I will drop those observations.

In [9]:
```
crashes.dropna(inplace = True)
ed.show_info(crashes)
```

```
Lenght of Dataset: 476858
                              missing_values_% Data_type
CRASH_RECORD_ID                            0.0    object
RD_NO                                      0.0    object
CRASH_DATE                                 0.0    object
POSTED_SPEED_LIMIT                         0.0    object
TRAFFIC_CONTROL_DEVICE                     0.0    object
DEVICE_CONDITION                           0.0    object
WEATHER_CONDITION                          0.0    object
LIGHTING_CONDITION                         0.0    object
FIRST_CRASH_TYPE                           0.0    object
TRAFFICWAY_TYPE                            0.0    object
ALIGNMENT                                  0.0    object
ROADWAY_SURFACE_COND                       0.0    object
ROAD_DEFECT                                0.0    object
REPORT_TYPE                                0.0    object
CRASH_TYPE                                 0.0    object
DAMAGE                                     0.0    object
DATE_POLICE_NOTIFIED                       0.0    object
PRIM_CONTRIBUTORY_CAUSE                    0.0    object
SEC_CONTRIBUTORY_CAUSE                     0.0    object
STREET_NO                                  0.0    object
STREET_DIRECTION                           0.0    object
STREET_NAME                                0.0    object
BEAT_OF_OCCURRENCE                         0.0    object
NUM_UNITS                                  0.0    object
MOST_SEVERE_INJURY                         0.0    object
INJURIES_TOTAL                             0.0    object
INJURIES_FATAL                             0.0    object
INJURIES_INCAPACITATING                    0.0    object
INJURIES_NON_INCAPACITATING                0.0    object
INJURIES_REPORTED_NOT_EVIDENT              0.0    object
INJURIES_NO_INDICATION                     0.0    object
INJURIES_UNKNOWN                           0.0    object
CRASH_HOUR                                 0.0    object
CRASH_DAY_OF_WEEK                          0.0    object
CRASH_MONTH                                0.0    object
LATITUDE                                   0.0    object
LONGITUDE                                  0.0    object
LOCATION                                   0.0    object
```

Convert to Numeric

Now, I will **convert columns** that suppose to be numeric **to int variables**

In [10]:
```
# The "conversion" function takes in df and list of columns to convert t
list_of_cols = ['POSTED_SPEED_LIMIT', 'STREET_NO', 'BEAT_OF_OCCURRENCE',
                'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITA
                'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EV
                'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN', 'CRASH_HOU
                'CRASH_DAY_OF_WEEK', 'CRASH_MONTH']
ed.conversion(crashes, list_of_cols, int)
ed.show_info(crashes)
```

```
Lenght of Dataset: 476858
                              missing_values_% Data_type
CRASH_RECORD_ID                            0.0    object
RD_NO                                      0.0    object
CRASH_DATE                                 0.0    object
POSTED_SPEED_LIMIT                         0.0     int64
TRAFFIC_CONTROL_DEVICE                     0.0    object
DEVICE_CONDITION                           0.0    object
WEATHER_CONDITION                          0.0    object
LIGHTING_CONDITION                         0.0    object
FIRST_CRASH_TYPE                           0.0    object
TRAFFICWAY_TYPE                            0.0    object
ALIGNMENT                                  0.0    object
ROADWAY_SURFACE_COND                       0.0    object
ROAD_DEFECT                                0.0    object
REPORT_TYPE                                0.0    object
CRASH_TYPE                                 0.0    object
DAMAGE                                     0.0    object
DATE_POLICE_NOTIFIED                       0.0    object
PRIM_CONTRIBUTORY_CAUSE                    0.0    object
SEC_CONTRIBUTORY_CAUSE                     0.0    object
STREET_NO                                  0.0     int64
STREET_DIRECTION                           0.0    object
STREET_NAME                                0.0    object
BEAT_OF_OCCURRENCE                         0.0     int64
NUM_UNITS                                  0.0     int64
MOST_SEVERE_INJURY                         0.0    object
INJURIES_TOTAL                             0.0     int64
INJURIES_FATAL                             0.0     int64
INJURIES_INCAPACITATING                    0.0     int64
INJURIES_NON_INCAPACITATING                0.0     int64
INJURIES_REPORTED_NOT_EVIDENT              0.0     int64
INJURIES_NO_INDICATION                     0.0     int64
INJURIES_UNKNOWN                           0.0     int64
CRASH_HOUR                                 0.0     int64
CRASH_DAY_OF_WEEK                          0.0     int64
CRASH_MONTH                                0.0     int64
LATITUDE                                   0.0    object
LONGITUDE                                  0.0    object
LOCATION                                   0.0    object
```

First, I will look into **CRASH_DATE** column to see what dates are included in dataset

In [11]:
```
crashes.CRASH_DATE.value_counts()
```

Out[11]:
```
12/29/2020 05:00:00 PM    29
11/10/2017 10:30:00 AM    26
11/10/2017 10:00:00 AM    20
01/12/2019 02:30:00 PM    18
01/12/2019 03:00:00 PM    18
                          ..
07/03/2020 08:09:00 PM     1
04/18/2017 09:15:00 PM     1
11/26/2020 02:40:00 AM     1
01/17/2020 10:45:00 PM     1
12/15/2018 09:25:00 AM     1
Name: CRASH_DATE, Length: 313432, dtype: int64
```

Since the column contains **exact date,** I will **separate the year of each the crash**

In [12]:
```
crashes['CRASH_DATE'] = pd.to_datetime(crashes['CRASH_DATE'])
crashes['CRASH_YEAR'] = crashes['CRASH_DATE'].dt.year

crashes[crashes.columns[30:]]
```

Out[12]:

	INJURIES_NO_INDICATION	INJURIES_UNKNOWN	CRASH_HOUR	CRASH_DAY_OF_WEEK
0	3	0	17	4
1	3	0	16	6
2	3	0	10	6
3	3	0	1	7
5	2	0	22	5
...	...	...	...	...
496200	2	0	7	3
496201	2	0	17	4
496202	2	0	16	4
496203	4	0	15	4
496204	2	0	16	4

476858 rows × 9 columns

Drop the "CRASH_DATE" column and put the **"CRASH_YEAR", "CRASH_MONTH", "CRASH_TIME" columns in front.**

In [13]:
```
crashes.drop(columns = "CRASH_DATE",axis = 1, inplace = True)

crashes.columns.get_loc("CRASH_YEAR")
```

Out[13]: 37

In [14]:
```
cols = list(crashes.columns)
cols = cols[:2] + [cols[37]] + cols[2:37]
crashes = crashes[cols]
```

In [15]:
```
crashes.columns.get_loc("CRASH_MONTH")
```

Out[15]: 34

In [16]:
```
cols = list(crashes.columns)
cols = cols[:3] + [cols[34]] + cols[3:34] + cols[35:]
crashes = crashes[cols]
```

In [17]:
```
crashes.columns.get_loc("CRASH_HOUR")
```

Out[17]: 33

In [18]:
```
cols = list(crashes.columns)
cols = cols[:4] + [cols[33]] + cols[4:33] + cols[34:]
crashes = crashes[cols]
```

In [19]:
```
crashes.columns.get_loc("CRASH_DAY_OF_WEEK")
```

Out[19]: 34

In [20]:
```
cols = list(crashes.columns)
cols = cols[:5] + [cols[34]] + cols[5:34] + cols[35:]
crashes = crashes[cols]
crashes.head()
```

Out[20]:

	CRASH_RECORD_ID	RD_NO	CRASH_YEAR	CRASH_MONTI
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	JC343143	2019	
1	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	JA329216	2017	
2	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	JD292400	2020	
3	f8960f698e870ebdc60b521b2a141a5395556bc3704191...	JD293602	2020	
5	00e47f189660cd8ba1e85fc63061bf1d8465184393f134...	JC194776	2019	

5 rows × 38 columns

In [21]:
```
ed.show_info(crashes)
```

```
Lenght of Dataset: 476858
                                missing_values_% Data_type
CRASH_RECORD_ID                              0.0    object
RD_NO                                        0.0    object
CRASH_YEAR                                   0.0     int64
CRASH_MONTH                                  0.0     int64
CRASH_HOUR                                   0.0     int64
CRASH_DAY_OF_WEEK                            0.0     int64
POSTED_SPEED_LIMIT                           0.0     int64
TRAFFIC_CONTROL_DEVICE                       0.0    object
DEVICE_CONDITION                             0.0    object
WEATHER_CONDITION                            0.0    object
LIGHTING_CONDITION                           0.0    object
FIRST_CRASH_TYPE                             0.0    object
TRAFFICWAY_TYPE                              0.0    object
ALIGNMENT                                    0.0    object
ROADWAY_SURFACE_COND                         0.0    object
ROAD_DEFECT                                  0.0    object
REPORT_TYPE                                  0.0    object
CRASH_TYPE                                   0.0    object
DAMAGE                                       0.0    object
DATE_POLICE_NOTIFIED                         0.0    object
PRIM_CONTRIBUTORY_CAUSE                      0.0    object
SEC_CONTRIBUTORY_CAUSE                       0.0    object
STREET_NO                                    0.0     int64
STREET_DIRECTION                             0.0    object
STREET_NAME                                  0.0    object
BEAT_OF_OCCURRENCE                           0.0     int64
NUM_UNITS                                    0.0     int64
MOST_SEVERE_INJURY                           0.0    object
INJURIES_TOTAL                               0.0     int64
INJURIES_FATAL                               0.0     int64
INJURIES_INCAPACITATING                      0.0     int64
INJURIES_NON_INCAPACITATING                  0.0     int64
INJURIES_REPORTED_NOT_EVIDENT                0.0     int64
INJURIES_NO_INDICATION                       0.0     int64
INJURIES_UNKNOWN                             0.0     int64
LATITUDE                                     0.0    object
LONGITUDE                                    0.0    object
LOCATION                                     0.0    object
```

In [22]:
```
crashes.CRASH_YEAR.value_counts()
```

Out[22]:
```
2018    115129
2019    112972
2020     88665
2017     81672
2016     43749
2021     24906
2015      9758
2014         6
2013         1
Name: CRASH_YEAR, dtype: int64
```

To narrow down the dataset, I will **keep the observations only from 2019-2020**, since these are most recent years (as 2021 is still in progress)

In [23]:
```
crashes = crashes[(crashes['CRASH_YEAR'] == 2019) | (crashes['CRASH_YEAR
ed.show_info(crashes)
```

```
Lenght of Dataset: 201637
                              missing_values_% Data_type
CRASH_RECORD_ID                            0.0    object
RD_NO                                      0.0    object
CRASH_YEAR                                 0.0     int64
CRASH_MONTH                                0.0     int64
CRASH_HOUR                                 0.0     int64
CRASH_DAY_OF_WEEK                          0.0     int64
POSTED_SPEED_LIMIT                         0.0     int64
TRAFFIC_CONTROL_DEVICE                     0.0    object
DEVICE_CONDITION                           0.0    object
WEATHER_CONDITION                          0.0    object
LIGHTING_CONDITION                         0.0    object
FIRST_CRASH_TYPE                           0.0    object
TRAFFICWAY_TYPE                            0.0    object
ALIGNMENT                                  0.0    object
ROADWAY_SURFACE_COND                       0.0    object
ROAD_DEFECT                                0.0    object
REPORT_TYPE                                0.0    object
CRASH_TYPE                                 0.0    object
DAMAGE                                     0.0    object
DATE_POLICE_NOTIFIED                       0.0    object
PRIM_CONTRIBUTORY_CAUSE                    0.0    object
SEC_CONTRIBUTORY_CAUSE                     0.0    object
STREET_NO                                  0.0     int64
STREET_DIRECTION                           0.0    object
STREET_NAME                                0.0    object
BEAT_OF_OCCURRENCE                         0.0     int64
NUM_UNITS                                  0.0     int64
MOST_SEVERE_INJURY                         0.0    object
INJURIES_TOTAL                             0.0     int64
INJURIES_FATAL                             0.0     int64
INJURIES_INCAPACITATING                    0.0     int64
INJURIES_NON_INCAPACITATING                0.0     int64
INJURIES_REPORTED_NOT_EVIDENT              0.0     int64
INJURIES_NO_INDICATION                     0.0     int64
INJURIES_UNKNOWN                           0.0     int64
LATITUDE                                   0.0    object
LONGITUDE                                  0.0    object
LOCATION                                   0.0    object
```

Categorical Features of Crashes Dataset - Part I

Now, I will look into **what values some the object type features have.**

In [24]:
```
list_of_feat = ['TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'TRAFFICWA
ed.values(crashes, list_of_feat)
```

```
NO CONTROLS                 115130
TRAFFIC SIGNAL               56271

STOP SIGN/FLASHER            20734
UNKNOWN                       6543
OTHER                         1321
YIELD                          308
OTHER REG. SIGN                264
OTHER WARNING SIGN             198
PEDESTRIAN CROSSING SIGN       191
LANE USE MARKING               131
RAILROAD CROSSING GATE         124
FLASHING CONTROL SIGNAL        109
POLICE/FLAGMAN                  84
DELINEATORS                     66
SCHOOL ZONE                     54
OTHER RAILROAD CROSSING         48
RR CROSSING SIGN                37
BICYCLE CROSSING SIGN           14
NO PASSING                      10
Name: TRAFFIC_CONTROL_DEVICE, dtype: int64


NO CONTROLS                 116423
FUNCTIONING PROPERLY         70461
UNKNOWN                      11379
OTHER                         1729
FUNCTIONING IMPROPERLY         895
NOT FUNCTIONING                655
WORN REFLECTIVE MATERIAL        71
MISSING                         24
Name: DEVICE_CONDITION, dtype: int64


NOT DIVIDED                        87543
DIVIDED - W/MEDIAN (NOT RAISED)    31721
ONE-WAY                            26103
PARKING LOT                        13664
FOUR WAY                           13308
DIVIDED - W/MEDIAN BARRIER         11304
OTHER                               4872
ALLEY                               3456
T-INTERSECTION                      2798
UNKNOWN                             1991
CENTER TURN LANE                    1537
UNKNOWN INTERSECTION TYPE            841
DRIVEWAY                             720
RAMP                                 621
FIVE POINT, OR MORE                  342
Y-INTERSECTION                       339
TRAFFIC ROUTE                        246
NOT REPORTED                         102
ROUNDABOUT                            83
L-INTERSECTION                        46
```

```
Name: TRAFFICWAY_TYPE, dtype: int64
```

```
STRAIGHT AND LEVEL        196549
STRAIGHT ON GRADE           2675
CURVE, LEVEL                1514
STRAIGHT ON HILLCREST        546
CURVE ON GRADE               264
CURVE ON HILLCREST            89
Name: ALIGNMENT, dtype: int64
```

The **TRAFFIC_CONTROL_DEVICE** column have multiple values that have common category: signal, sign. I will narrow down those values to one category

In [25]:
```
crashes['TRAFFIC_CONTROL_DEVICE'] = crashes['TRAFFIC_CONTROL_DEVICE'].ap

crashes['TRAFFIC_CONTROL_DEVICE'] = crashes['TRAFFIC_CONTROL_DEVICE'].ap
crashes['TRAFFIC_CONTROL_DEVICE'].value_counts()
```

Out[25]:
```
NO CONTROLS                 115130
SIGNAL                       56380
SIGN                         21438
UNKNOWN                       6543
OTHER                         1321
YIELD                          308
LANE USE MARKING               131
RAILROAD CROSSING GATE         124
POLICE/FLAGMAN                  84
DELINEATORS                     66
SCHOOL ZONE                     54
OTHER RAILROAD CROSSING         48
NO PASSING                      10
Name: TRAFFIC_CONTROL_DEVICE, dtype: int64
```

TRAFFICWAY_TYPE column has a values with different types of intersection, I will reframe them all as one: intersection type.

In [26]:
```
crashes['TRAFFICWAY_TYPE'] = crashes['TRAFFICWAY_TYPE'].apply(lambda x:
crashes['TRAFFICWAY_TYPE'].value_counts()
```

Out[26]:
```
NOT DIVIDED                        87543
DIVIDED - W/MEDIAN (NOT RAISED)    31721
ONE-WAY                            26103
PARKING LOT                        13664
FOUR WAY                           13308
DIVIDED - W/MEDIAN BARRIER         11304
OTHER                               4872
INTERSECTION                        4024
ALLEY                               3456
UNKNOWN                             1991
CENTER TURN LANE                    1537
DRIVEWAY                             720
RAMP                                 621
FIVE POINT, OR MORE                  342
TRAFFIC ROUTE                        246
NOT REPORTED                         102
ROUNDABOUT                            83
Name: TRAFFICWAY_TYPE, dtype: int64
```

ALIGNMENT column has various types of two categories: straight and curved. I will bin all of the values onto those two categories and create **binary column: 1** represents **STRAIGHT ALIGNMENT, 0** represents **CURVED ALIGNMENT**

In [27]:
```
crashes['ALIGNMENT'] = crashes['ALIGNMENT'].apply(lambda x: 1 if 'STRAIGH
crashes.rename(columns={'ALIGNMENT': 'STRAIGHT_ALIGNMENT'}, inplace = Tru
crashes['STRAIGHT_ALIGNMENT'].value_counts()
```

Out[27]:
```
1    199770
0      1867
Name: STRAIGHT_ALIGNMENT, dtype: int64
```

Categorical Features of Crashes Dataset - Part II

In [28]:
```
list_of_feat = ['WEATHER_CONDITION', 'LIGHTING_CONDITION', 'ROADWAY_SURF
ed.values(crashes, list_of_feat)
```

```
CLEAR                       159395
RAIN                         18099
UNKNOWN                       8634
SNOW                          7640
CLOUDY/OVERCAST               6102
OTHER                          633
FREEZING RAIN/DRIZZLE          453
SLEET/HAIL                     321
FOG/SMOKE/HAZE                 251
BLOWING SNOW                    70
SEVERE CROSS WIND GATE          37
BLOWING SAND, SOIL, DIRT         2
Name: WEATHER_CONDITION, dtype: int64


DAYLIGHT                  129948
DARKNESS, LIGHTED ROAD     44739
DARKNESS                    9986
UNKNOWN                     7298
DUSK                        6108
DAWN                        3558
Name: LIGHTING_CONDITION, dtype: int64


DRY                 151175
WET                  27835
UNKNOWN              13274
SNOW OR SLUSH         7211
ICE                   1636
OTHER                  432
SAND, MUD, DIRT         74
Name: ROADWAY_SURFACE_COND, dtype: int64


NO DEFECTS          167364
UNKNOWN              30137
RUT, HOLES            1909
OTHER                  999
WORN SURFACE           649
SHOULDER DEFECT        421
DEBRIS ON ROADWAY      158
Name: ROAD_DEFECT, dtype: int64
```

WEATHER_CONDITION column have multiple categories tht can be combined:

In [29]:
```
crashes['WEATHER_CONDITION'] = crashes['WEATHER_CONDITION'].apply(lambda
crashes['WEATHER_CONDITION'] = crashes['WEATHER_CONDITION'].apply(lambda

crashes['WEATHER_CONDITION'] = crashes['WEATHER_CONDITION'].apply(lambda

crashes['WEATHER_CONDITION'].value_counts()
```

Out[29]:
```
CLEAR               159395
RAIN                 18552
UNKNOWN               8634
SNOW                  8031
CLOUDY/OVERCAST       6102
OTHER                  672
FOG/SMOKE/HAZE         251
Name: WEATHER_CONDITION, dtype: int64
```

LIGHTING_CONDITION column categories can also be categorized:

In [30]:
```
crashes['LIGHTING_CONDITION'] = crashes['LIGHTING_CONDITION'].apply(lamb
crashes['LIGHTING_CONDITION'] = crashes['LIGHTING_CONDITION'].apply(lamb
                                                                    ('RO
                                                                     'DU

crashes['LIGHTING_CONDITION'].value_counts()
```

Out[30]:
```
LIGHT          129948
SOME_LIGHT      54405
DARKNESS         9986
UNKNOWN          7298
Name: LIGHTING_CONDITION, dtype: int64
```

ROADWAY_SURFACE_COND column can also be **binarized: 1** for **clean** (dry, no defect) **road condition, 0** for **defected** (ice, wet, sand, etc)

In [31]:
```
crashes['ROADWAY_SURFACE_COND'] = crashes['ROADWAY_SURFACE_COND'].apply(

crashes.rename(columns={'ROADWAY_SURFACE_COND': 'GOOD_ROADWAY_SUFACE'}, 
crashes['GOOD_ROADWAY_SUFACE'].value_counts()
```

Out[31]:
```
1    164449
0     37188
Name: GOOD_ROADWAY_SUFACE, dtype: int64
```

ROAD_DEFECT will be also binarized: **1 for defect, 0 for no defect**

In [32]:
```
crashes['ROAD_DEFECT'] = crashes['ROAD_DEFECT'].apply(lambda x: 0 if ('N(
crashes['ROAD_DEFECT'].value_counts()
```

Out[32]:
```
0    197501
1      4136
Name: ROAD_DEFECT, dtype: int64
```

Categorical Features of Crashes Dataset - Part III

In [33]:
```
list_of_feat = ['FIRST_CRASH_TYPE', 'REPORT_TYPE', 'CRASH_TYPE', 'DAMAGE
ed.values(crashes, list_of_feat)
```

```
PARKED MOTOR VEHICLE            47729
REAR END                        45040
SIDESWIPE SAME DIRECTION        28807
TURNING                         28575
ANGLE                           20881
FIXED OBJECT                    10421
PEDESTRIAN                       5052
PEDALCYCLIST                     3119
SIDESWIPE OPPOSITE DIRECTION     2812
REAR TO FRONT                    2397
OTHER OBJECT                     2184
HEAD ON                          1580
REAR TO SIDE                     1499
OTHER NONCOLLISION                695
REAR TO REAR                      542
ANIMAL                            158
OVERTURNED                        128
TRAIN                              18
Name: FIRST_CRASH_TYPE, dtype: int64


NOT ON SCENE (DESK REPORT)    102303
ON SCENE                       99334
Name: REPORT_TYPE, dtype: int64


NO INJURY / DRIVE AWAY              144582
INJURY AND / OR TOW DUE TO CRASH     57055
Name: CRASH_TYPE, dtype: int64


OVER $1,500      120646
$501 - $1,500     56002
$500 OR LESS      24989
Name: DAMAGE, dtype: int64
```

For **FIRST_CRASH_TYPE** column, I will combine two sideswipe categories into one:

In [34]:
```
crashes['FIRST_CRASH_TYPE'] = crashes['FIRST_CRASH_TYPE'].apply(lambda x
crashes['FIRST_CRASH_TYPE'].value_counts()
```

Out[34]:
```
PARKED MOTOR VEHICLE    47729
REAR END                45040
SIDESWIPE               31619
TURNING                 28575
ANGLE                   20881
FIXED OBJECT            10421
PEDESTRIAN               5052
PEDALCYCLIST             3119
REAR TO FRONT            2397
OTHER OBJECT             2184
HEAD ON                  1580
REAR TO SIDE             1499
OTHER NONCOLLISION        695
REAR TO REAR              542
ANIMAL                    158
OVERTURNED                128
TRAIN                      18
Name: FIRST_CRASH_TYPE, dtype: int64
```

REPORT_TYPE column will become binary: **1 for DESK REPORT TYPE** and **0 for ON SCENE:**

In [35]:
```
crashes['REPORT_TYPE'] = crashes['REPORT_TYPE'].apply(lambda x: 1 if 'DE
crashes.rename(columns={'REPORT_TYPE': 'DESK_REPORT_TYPE'}, inplace = Tr
crashes['DESK_REPORT_TYPE'].value_counts()
```

Out[35]:
```
1    102303
0     99334
Name: DESK_REPORT_TYPE, dtype: int64
```

I will leave two other columns: **CRASH_TYPE** and **DAMAGE** as they are.

Categorical Features of Crashes Dataset - Part IV

In [36]:
```
list_of_feat = ['PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'MO
ed.values(crashes, list_of_feat)
```

```
UNABLE TO DETERMINE
76425
FAILING TO YIELD RIGHT-OF-WAY
21679
FOLLOWING TOO CLOSELY
19710
NOT APPLICABLE
11038
```

FAILING TO REDUCE SPEED TO AVOID CRASH
10000
IMPROPER OVERTAKING/PASSING
9197
IMPROPER BACKING
8324
IMPROPER LANE USAGE
7163
IMPROPER TURNING/NO SIGNAL
6725
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
6106
DISREGARDING TRAFFIC SIGNALS
4193
WEATHER
3219
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESS
IVE MANNER 2668
DISREGARDING STOP SIGN
2430
DISTRACTION - FROM INSIDE VEHICLE
1533
EQUIPMENT - VEHICLE CONDITION
1442
PHYSICAL CONDITION OF DRIVER
1321
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
1289
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
1234
DRIVING ON WRONG SIDE/WRONG WAY
1053
DISTRACTION - FROM OUTSIDE VEHICLE
929
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
594
DISREGARDING OTHER TRAFFIC SIGNS
459
ROAD CONSTRUCTION/MAINTENANCE
434
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
380
CELL PHONE USE OTHER THAN TEXTING
308
EXCEEDING SAFE SPEED FOR CONDITIONS
263
DISREGARDING ROAD MARKINGS
263
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
219
EXCEEDING AUTHORIZED SPEED LIMIT
194
ANIMAL
194

TURNING RIGHT ON RED
151

RELATED TO BUS STOP
147
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER, ETC.)   116
DISREGARDING YIELD SIGN
85
TEXTING
81
OBSTRUCTED CROSSWALKS
27
PASSING STOPPED SCHOOL BUS
26
BICYCLE ADVANCING LEGALLY ON RED LIGHT
12
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
6
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64

NOT APPLICABLE
87117
UNABLE TO DETERMINE
71051
FAILING TO REDUCE SPEED TO AVOID CRASH
8197
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
5772
FAILING TO YIELD RIGHT-OF-WAY
5392
FOLLOWING TOO CLOSELY
4813
IMPROPER OVERTAKING/PASSING
2771
IMPROPER LANE USAGE
2585
WEATHER
2295
IMPROPER TURNING/NO SIGNAL
1853
IMPROPER BACKING
1477
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE MANNER   1314
DISREGARDING TRAFFIC SIGNALS
777
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
649
PHYSICAL CONDITION OF DRIVER
626
DISTRACTION - FROM INSIDE VEHICLE
621

```
DISREGARDING STOP SIGN
500

EQUIPMENT - VEHICLE CONDITION
431
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
384
DRIVING ON WRONG SIDE/WRONG WAY
380
DISTRACTION - FROM OUTSIDE VEHICLE
366
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
245
ROAD CONSTRUCTION/MAINTENANCE
219
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
219
DISREGARDING OTHER TRAFFIC SIGNS
211
DISREGARDING ROAD MARKINGS
208
EXCEEDING SAFE SPEED FOR CONDITIONS
188
RELATED TO BUS STOP
170
EXCEEDING AUTHORIZED SPEED LIMIT
150
CELL PHONE USE OTHER THAN TEXTING
147
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
105
ANIMAL
80
TURNING RIGHT ON RED
69
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER,
ETC.)            56
DISREGARDING YIELD SIGN
50
OBSTRUCTED CROSSWALKS
43
BICYCLE ADVANCING LEGALLY ON RED LIGHT
35
TEXTING
34
PASSING STOPPED SCHOOL BUS
20
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
17
Name: SEC_CONTRIBUTORY_CAUSE, dtype: int64


NO INDICATION OF INJURY     172911
NONINCAPACITATING INJURY     16392
```

```
REPORTED, NOT EVIDENT          8521
INCAPACITATING INJURY          3625

FATAL                           188
Name: MOST_SEVERE_INJURY, dtype: int64
```

I will keep those columns as they are for now and will investigate them deeper later on during EDA

Explore People Dataset

Now, explore People Data and check following information:

- What columns do we have in each of the datasets
- Are there any missing values in tables
- Are there duplicates in data

In [37]:
```
ed.show_info(people)
```

```
Lenght of Dataset: 1095613
                       missing_values_% Data_type
PERSON_ID                      0.000000    object
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
RD_NO                          0.706545    object
VEHICLE_ID                     1.971317    object
CRASH_DATE                     0.000000    object
SEAT_NO                       79.582298    object
CITY                          26.154491    object
STATE                         25.322445    object
ZIPCODE                       32.571903    object
SEX                            1.485287    object
AGE                           28.550136    object
DRIVERS_LICENSE_STATE         40.752072    object
DRIVERS_LICENSE_CLASS         48.649204    object
SAFETY_EQUIPMENT               0.296090    object
AIRBAG_DEPLOYED                1.885246    object
EJECTION                       1.230909    object
INJURY_CLASSIFICATION          0.052573    object
HOSPITAL                      81.850708    object
EMS_AGENCY                    88.482977    object
EMS_RUN_NO                    98.128080    object
DRIVER_ACTION                 20.609832    object
DRIVER_VISION                 20.636301    object
PHYSICAL_CONDITION            20.552239    object
PEDPEDAL_ACTION               98.148890    object
PEDPEDAL_VISIBILITY           98.152815    object
PEDPEDAL_LOCATION             98.148799    object
BAC_RESULT                    20.508154    object
BAC_RESULT VALUE              99.871579    object
CELL_PHONE_USE                99.894397    object
```

Check for **duplicates in CRASH_RECORD_ID** column also.

In [38]:
```
people.CRASH_RECORD_ID.duplicated().sum()
```

Out[38]:
```
600634
```

Eploration Results:

- **People dataset** has significant amount of missing values in following columns:

SEAT_NO, CITY, STATE, ZIPCODE, AGE, DRIVERS_LICENSE_STATE, DRIVERS_LICENSE_CLASS, 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

- **CRASH_RECORD_ID** has **563894 duplicated values** due to the fact that there were **multiple people involved in each crash.**

Investigate Columns with more than 90% Missing Values in People Dataset

In [39]:
```
# The "missing_values" function returns list of column names with missin
people_m_vals = ed.missing_values(people, 90, 100)
# The "values" function prints out the value_counts for the list of colu
ed.values(people, people_m_vals)
```

```
DNA                      3657
NONE                     1446
99                        986
REFUSED                   536
55                        301
                         ...
REFUSED BY MOTHER           1
AMB. 28                     1
CDF AMB68                   1
EMS #35                     1
06267                       1
Name: EMS_RUN_NO, Length: 998, dtype: int64


CROSSING - WITH SIGNAL                                  4189
WITH TRAFFIC                                            3277
UNKNOWN/NA                                              2678
OTHER ACTION                                            2635
NO ACTION                                               1042
CROSSING - AGAINST SIGNAL                               1016
CROSSING - NO CONTROLS (NOT AT INTERSECTION)             907
NOT AT INTERSECTION                                      887
CROSSING - NO CONTROLS (AT INTERSECTION)                 755
AGAINST TRAFFIC                                          679
STANDING IN ROADWAY                                      492
CROSSING - CONTROLS PRESENT (NOT AT INTERSECTION)        446
TURNING LEFT                                             262
PARKED VEHICLE                                           243
ENTER FROM DRIVE/ALLEY                                   207
INTOXICATED PED/PEDAL                                    153
```

```
WORKING IN ROADWAY                                         140
TURNING RIGHT                                              135
PLAYING IN ROADWAY                                          86
TO/FROM DISABLED VEHICLE                                    15
PLAYING/WORKING ON VEHICLE                                  15
SCHOOL BUS (WITHIN 50 FT.)                                  12
WAITING FOR SCHOOL BUS                                      10
Name: PEDPEDAL_ACTION, dtype: int64
```

```
NO CONTRASTING CLOTHING    15971
CONTRASTING CLOTHING        2684
OTHER LIGHT SOURCE USED     1072
REFLECTIVE MATERIAL          511
Name: PEDPEDAL_VISIBILITY, dtype: int64
```

```
IN ROADWAY         9358
IN CROSSWALK       6653
UNKNOWN/NA         1659
BIKEWAY             955
NOT IN ROADWAY      904
BIKE LANE           371
DRIVEWAY ACCESS     304
SHOULDER             78
Name: PEDPEDAL_LOCATION, dtype: int64
```

```
0.0000    141
0.1700     97
0.1800     97
0.2100     83
0.1400     81
0.2000     73
0.1600     68
0.1500     65
0.1900     64
0.2300     59
0.2200     57
0.1300     52
0.1200     52
0.1100     50
0.2400     43
0.2600     29
0.2700     28
0.2500     26
0.1000     25
0.0900     24
0.2800     21
0.0300     15
0.3000     15
0.0700     14
0.0800     14
0.0400     14
```

```
0.2900     12
0.3300     12
0.0500      9
0.3200      8

0.0600      7
0.0200      7
0.3100      6
0.3800      6
0.3500      5
0.3400      4
0.0100      3
0.4500      2
0.3600      2
0.3900      2
0.6000      2
0.4400      2
0.4000      1
0.8000      1
0.5800      1
0.8800      1
1.0000      1
0.7900      1
0.9500      1
0.6700      1
0.4100      1
0.9900      1
0.4700      1
Name: BAC_RESULT VALUE, dtype: int64
```

```
Y    752
N    405
Name: CELL_PHONE_USE, dtype: int64
```

Result: Most of the **columns** that are **missing more than 90%** of data in **People dataset** are the columns related to **pedastrian/biker activities**. Rest of the columns are about **EMS run number, cellphone usage and alcohol concentration in blood.** Since this values **can't be reproduced**, I will be **dropping** them.

Investigate Columns with ~80% Missing Values in People Dataset

In [40]:
```
people_m_vals = ed.missing_values(people, 80,90)
ed.values(people, people_m_vals)
```

```
REFUSED                                         66970
DNA                                             27791
NONE                                            16949
99                                               6814
DECLINED                                         4235
                                                ...
GOING TO HAVE WIFE DRIVE HIM TO HIS DOCTOR          1
LUTHERN GEN HOSPITAL                                1
GRANDFATHER REFUSED                                 1
ILLINOIS MASONIC MEDICAL                            1
COMMUNITY CENTRAL HOSPITAL                          1
Name: HOSPITAL, Length: 5135, dtype: int64


DNA                       24055
CFD                       20951
REFUSED                   12760
NONE                       8483
99                         6552
                          ...
CPD 71                        1
BURBANK 210                   1
CHICAG FIRE DEPARTMENT        1
CFD AMBULANCE # 70            1
CFD #107 CFD #38              1
Name: EMS_AGENCY, Length: 6160, dtype: int64
```

Result: The columns above represent the **Hospital**, the injured were taken and the **EMS agency** that took them there. Considering, I **cannot forge** the values, the columns **will be dropped.**

Investigate Columns with ~40% Missing Values in People Dataset

In [41]:
```
people_m_vals = ed.missing_values(people, 40,50)
ed.values(people, people_m_vals)
```

```
IL    597121
XX     14049
IN     10641
WI      3418
MI      2814
       ...
ES         1
BJ         1
NZ         1
DH         1
CL         1
Name: DRIVERS_LICENSE_STATE, Length: 184, dtype: int64


D     490052
A      19116
C      15465
B      15038
DM      9164
       ...
UP         1
TA         1
GV         1
D6         1
?          1
Name: DRIVERS_LICENSE_CLASS, Length: 235, dtype: int64
```

Result: Since the "DRIVERS_LICENSE_STATE" column can be usefull in modeling, I will keep it for now in dataset. But as for "DRIVERS_LICENSE_CLASS", I will be dropping the column, due to the large amount of innacurate classes.

- I am only aware of 8 legal drivel license classes.

Clean People Dataset

Drop Columns

I will drop columns stated above from People dataset

In [42]:
```
people_drop_cols = ed.missing_values(people, 48, 100)
people.drop(columns = people_drop_cols,axis = 1, inplace = True)
ed.show_info(people)
```

```
Lenght of Dataset: 1095613
                       missing_values_% Data_type
PERSON_ID                      0.000000    object
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
RD_NO                          0.706545    object
VEHICLE_ID                     1.971317    object
CRASH_DATE                     0.000000    object
CITY                          26.154491    object
STATE                         25.322445    object
ZIPCODE                       32.571903    object
SEX                            1.485287    object
AGE                           28.550136    object
DRIVERS_LICENSE_STATE         40.752072    object
SAFETY_EQUIPMENT               0.296090    object
AIRBAG_DEPLOYED                1.885246    object
EJECTION                       1.230909    object
INJURY_CLASSIFICATION          0.052573    object
DRIVER_ACTION                 20.609832    object
DRIVER_VISION                 20.636301    object
PHYSICAL_CONDITION            20.552239    object
BAC_RESULT                    20.508154    object
```

I will also drop columns that are **already present** in Crashes Dataset or are **irrelevant to the modeling: PERSON_ID, RD_NO, CRASH_DATE, ZIPCODE, CITY, STATE.**

In [43]:
```
columns = ['PERSON_ID', 'RD_NO', 'ZIPCODE', 'CITY', 'STATE']
people.drop(columns = columns,axis = 1, inplace = True)
ed.show_info(people)
```

```
Lenght of Dataset: 1095613
                       missing_values_% Data_type
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
VEHICLE_ID                     1.971317    object
CRASH_DATE                     0.000000    object
SEX                            1.485287    object
AGE                           28.550136    object
DRIVERS_LICENSE_STATE         40.752072    object
SAFETY_EQUIPMENT               0.296090    object
AIRBAG_DEPLOYED                1.885246    object
EJECTION                       1.230909    object
INJURY_CLASSIFICATION          0.052573    object
DRIVER_ACTION                 20.609832    object
DRIVER_VISION                 20.636301    object
PHYSICAL_CONDITION            20.552239    object
BAC_RESULT                    20.508154    object
```

In [44]:
```
people['CRASH_DATE'] = pd.to_datetime(people['CRASH_DATE'])
people['CRASH_YEAR'] = people['CRASH_DATE'].dt.year
people = people[(people['CRASH_YEAR'] == 2019) | (people['CRASH_YEAR'] ==
people.drop(columns = ["CRASH_DATE", "CRASH_YEAR"],axis = 1, inplace = Tr

ed.show_info(people)
```

```
Lenght of Dataset: 465973
                       missing_values_% Data_type
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
VEHICLE_ID                     2.202703    object
SEX                            1.662543    object
AGE                           27.981879    object
DRIVERS_LICENSE_STATE         40.789917    object
SAFETY_EQUIPMENT               0.349376    object
AIRBAG_DEPLOYED                2.064712    object
EJECTION                       1.360594    object
INJURY_CLASSIFICATION          0.033049    object
DRIVER_ACTION                 21.138778    object
DRIVER_VISION                 21.175261    object
PHYSICAL_CONDITION            21.068173    object
BAC_RESULT                    21.098862    object
```

I will **drop observations** from colums that are **missing less than 10%** of data.

In [45]:
```
people_m_vals = ed.missing_values(people, 0, 10)
people.dropna(subset=people_m_vals, inplace = True)
ed.show_info(people)
```

```
Lenght of Dataset: 448227
                       missing_values_% Data_type
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
VEHICLE_ID                     0.000000    object
SEX                            0.000000    object
AGE                           27.309377    object
DRIVERS_LICENSE_STATE         38.519099    object
SAFETY_EQUIPMENT               0.000000    object
AIRBAG_DEPLOYED                0.000000    object
EJECTION                       0.000000    object
INJURY_CLASSIFICATION          0.000000    object
DRIVER_ACTION                 20.067510    object
DRIVER_VISION                 20.067734    object
PHYSICAL_CONDITION            20.067957    object
BAC_RESULT                    20.068180    object
```

Convert to Numeric

Since **AGE** is only one column that needs to be **converted to int** and still has NaN values, I will **drop observations with NaN values** and then convert.

In [46]:
```
people.dropna(subset = ['AGE'], inplace = True)
people['AGE'] = people.AGE.astype(int)
ed.show_info(people)
```

```
Lenght of Dataset: 325819
                       missing_values_% Data_type
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
VEHICLE_ID                     0.000000    object
SEX                            0.000000    object
AGE                            0.000000     int64
DRIVERS_LICENSE_STATE         21.085020    object
SAFETY_EQUIPMENT               0.000000    object
AIRBAG_DEPLOYED                0.000000    object
EJECTION                       0.000000    object
INJURY_CLASSIFICATION          0.000000    object
DRIVER_ACTION                 18.423112    object
DRIVER_VISION                 18.423112    object
PHYSICAL_CONDITION            18.423419    object
BAC_RESULT                    18.423726    object
```

Categorical Features of People Dataset - Part I

In [47]:
```
list_of_feat = ['PERSON_TYPE', 'SEX', 'SAFETY_EQUIPMENT', 'AIRBAG_DEPLOYI
ed.values(people, list_of_feat)
```

```
DRIVER                 265769
PASSENGER               60026
NON-CONTACT VEHICLE        24
Name: PERSON_TYPE, dtype: int64


M    187332
F    137961
X       526
Name: SEX, dtype: int64


SAFETY BELT USED                        195302
USAGE UNKNOWN                           115170
NONE PRESENT                              8614
SAFETY BELT NOT USED                      2152
CHILD RESTRAINT - FORWARD FACING          1166
CHILD RESTRAINT - REAR FACING              660
CHILD RESTRAINT USED                       643
CHILD RESTRAINT - TYPE UNKNOWN             628
HELMET NOT USED                            484
DOT COMPLIANT MOTORCYCLE HELMET            396
```

```
BOOSTER SEAT                              318
CHILD RESTRAINT NOT USED                  103

NOT DOT COMPLIANT MOTORCYCLE HELMET        58
SHOULD/LAP BELT USED IMPROPERLY            57
CHILD RESTRAINT USED IMPROPERLY            34
HELMET USED                                14
WHEELCHAIR                                 12
STRETCHER                                   8
Name: SAFETY_EQUIPMENT, dtype: int64


DID NOT DEPLOY                            203148
NOT APPLICABLE                             80175
DEPLOYED, FRONT                            14635
DEPLOYMENT UNKNOWN                         13854
DEPLOYED, COMBINATION                       9924
DEPLOYED, SIDE                              3905
DEPLOYED OTHER (KNEE, AIR, BELT, ETC.)       178
Name: AIRBAG_DEPLOYED, dtype: int64


NONE                     319735
UNKNOWN                    5345
TOTALLY EJECTED             477
TRAPPED/EXTRICATED          157
PARTIALLY EJECTED           105
Name: EJECTION, dtype: int64
```

For **PERSON_TYPE** column I will **drop observations** that have value of **"NON-CONTACT VEHICLE" and "PASSANGER""**. Also, I will remove the duplicated values from the dataset, such as duplicated crash id values. Since the goal of the project does not require passangers information.

In [48]:
```python
index_ptype = people[~(people['PERSON_TYPE'] == "DRIVER")].index
people.drop(index_ptype, inplace=True)
people['PERSON_TYPE'].value_counts()
```

Out[48]:
```
DRIVER    265769
Name: PERSON_TYPE, dtype: int64
```

In [49]:
```python
people.drop_duplicates(subset ="CRASH_RECORD_ID",
                     keep = 'first', inplace = True)
```

In [50]:
```
people.SEX.value_counts()
```

Out[50]:
```
M    102012
F     65265
X       248
Name: SEX, dtype: int64
```

As for **SEX** column, I will **binarize** it too: **1 for Male and 0 for Female**. The values of **X** will be dropped, assuming that X means "no answer provided"

In [51]:
```
index_ptype = people[(people['SEX'] == "X")].index
people.drop(index_ptype, inplace=True)
people['SEX'].value_counts()
```

Out[51]:
```
M    102012
F     65265
Name: SEX, dtype: int64
```

In [52]:
```
people['SEX'] = people['SEX'].apply(lambda x: 1 if 'M' in x  else 0)
people.rename(columns={'SEX': 'MALE_PERSON'}, inplace = True)
people['MALE_PERSON'].value_counts()
```

Out[52]:
```
1    102012
0     65265
Name: MALE_PERSON, dtype: int64
```

For **AIRBAG_DEPLOYED** column I will **binarize** it too: **1 for deployed and 0 for not deployed.**

In [53]:
```
people['AIRBAG_DEPLOYED'] = people['AIRBAG_DEPLOYED'].apply(lambda x: 1 
people['AIRBAG_DEPLOYED'].value_counts()
```

Out[53]:
```
0    152495
1     14782
Name: AIRBAG_DEPLOYED, dtype: int64
```

The **EJECTION** column will be binarized as: **1 for ejected and 0 for not.**

In [54]:
```
people['EJECTION'] = people['EJECTION'].apply(lambda x: 1 if 'EJECTED' in
people['EJECTION'].value_counts()
```

Out[54]:
```
0    166916
1       361
Name: EJECTION, dtype: int64
```

Categorical Features of People Dataset - Part II

In [55]:
```
list_of_feat = ['INJURY_CLASSIFICATION', 'DRIVER_ACTION', 'DRIVER_VISION
ed.values(people, list_of_feat)
```

```
NO INDICATION OF INJURY     155053
NONINCAPACITATING INJURY      7099
REPORTED, NOT EVIDENT         3653
INCAPACITATING INJURY         1368
FATAL                          104
Name: INJURY_CLASSIFICATION, dtype: int64


NONE                                 44689
UNKNOWN                              33457
FAILED TO YIELD                      21871
OTHER                                19199
FOLLOWED TOO CLOSELY                 14666
IMPROPER BACKING                      6543
IMPROPER TURN                         6489
IMPROPER LANE CHANGE                  5501
TOO FAST FOR CONDITIONS               3946
DISREGARDED CONTROL DEVICES           3910
IMPROPER PASSING                      3815
IMPROPER PARKING                       824
WRONG WAY/SIDE                         741
OVERCORRECTED                          538
CELL PHONE USE OTHER THAN TEXTING      375
EVADING POLICE VEHICLE                 371
EMERGENCY VEHICLE ON CALL              222
TEXTING                                100
LICENSE RESTRICTIONS                    10
STOPPED SCHOOL BUS                      10
Name: DRIVER_ACTION, dtype: int64


NOT OBSCURED              103350
UNKNOWN                    57604
OTHER                       2564
MOVING VEHICLES             1390
PARKED VEHICLES              997
WINDSHIELD (WATER/ICE)       697
BLINDED - SUNLIGHT           392
TREES, PLANTS                123
BUILDINGS                     84
HILLCREST                     20
BLOWING MATERIALS             20
BLINDED - HEADLIGHTS          20
EMBANKMENT                    12
SIGNBOARD                      4
Name: DRIVER_VISION, dtype: int64


NORMAL                          134239
UNKNOWN                          27379
IMPAIRED - ALCOHOL                1671
FATIGUED/ASLEEP                    966
REMOVED BY EMS                     753
```

```
OTHER                                698
EMOTIONAL                            628

ILLNESS/FAINTED                      344
IMPAIRED - DRUGS                     215
HAD BEEN DRINKING                    201
IMPAIRED - ALCOHOL AND DRUGS         138
MEDICATED                             45
Name: PHYSICAL_CONDITION, dtype: int64


TEST NOT OFFERED                     163263
TEST REFUSED                           2512
TEST TAKEN                              813
TEST PERFORMED, RESULTS UNKNOWN         689
Name: BAC_RESULT, dtype: int64
```

I will **drop** the **INJURY_CLASSIFICATION** column, because I already have the **same data in Crashes dataset**.

In [56]:
```
people.drop(columns = "INJURY_CLASSIFICATION",axis = 1, inplace = True)
```

As for the **rest of the columns**, I will **keep them** as they are for now.

In [57]:
```
ed.show_info(people)
```

```
Lenght of Dataset: 167277
                       missing_values_% Data_type
PERSON_TYPE                    0.000000    object
CRASH_RECORD_ID                0.000000    object
VEHICLE_ID                     0.000000    object
MALE_PERSON                    0.000000     int64
AGE                            0.000000     int64
DRIVERS_LICENSE_STATE          4.032832    object
SAFETY_EQUIPMENT               0.000000    object
AIRBAG_DEPLOYED                0.000000     int64
EJECTION                       0.000000     int64
DRIVER_ACTION                  0.000000    object
DRIVER_VISION                  0.000000    object
PHYSICAL_CONDITION             0.000000    object
BAC_RESULT                     0.000000    object
```

In [58]:
```
ed.show_info(crashes)
```

```
Lenght of Dataset: 201637
                                          missing_values_% Data_type
CRASH_RECORD_ID                                        0.0    object
RD_NO                                                  0.0    object
CRASH_YEAR                                             0.0     int64
CRASH_MONTH                                            0.0     int64
CRASH_HOUR                                             0.0     int64
CRASH_DAY_OF_WEEK                                      0.0     int64
POSTED_SPEED_LIMIT                                     0.0     int64
TRAFFIC_CONTROL_DEVICE                                 0.0    object
DEVICE_CONDITION                                       0.0    object
WEATHER_CONDITION                                      0.0    object
LIGHTING_CONDITION                                     0.0    object
FIRST_CRASH_TYPE                                       0.0    object
TRAFFICWAY_TYPE                                        0.0    object
STRAIGHT_ALIGNMENT                                     0.0     int64
GOOD_ROADWAY_SUFACE                                    0.0     int64
ROAD_DEFECT                                            0.0     int64
DESK_REPORT_TYPE                                       0.0     int64
CRASH_TYPE                                             0.0    object
DAMAGE                                                 0.0    object
DATE_POLICE_NOTIFIED                                   0.0    object
PRIM_CONTRIBUTORY_CAUSE                                0.0    object
SEC_CONTRIBUTORY_CAUSE                                 0.0    object
STREET_NO                                              0.0     int64
STREET_DIRECTION                                       0.0    object
STREET_NAME                                            0.0    object
BEAT_OF_OCCURRENCE                                     0.0     int64
NUM_UNITS                                              0.0     int64
MOST_SEVERE_INJURY                                     0.0    object
INJURIES_TOTAL                                         0.0     int64
INJURIES_FATAL                                         0.0     int64
INJURIES_INCAPACITATING                                0.0     int64
INJURIES_NON_INCAPACITATING                            0.0     int64
INJURIES_REPORTED_NOT_EVIDENT                          0.0     int64
INJURIES_NO_INDICATION                                 0.0     int64
INJURIES_UNKNOWN                                       0.0     int64
LATITUDE                                               0.0    object
LONGITUDE                                              0.0    object
LOCATION                                               0.0    object
```

Merge Datasets

I will **merge two datasets** and create one combined. The **People dataframe has a duplicates** of the **CRASH_RECORD_ID**, since in one car crash there could be **multiple injured people**.

In [59]:
```
df = pd.merge(left=crashes, right=people, left_on='CRASH_RECORD_ID', rig
ed.show_info(df)
```

```
Lenght of Dataset: 160657
                                 missing_values_% Data_type
CRASH_RECORD_ID                          0.000000    object
RD_NO                                    0.000000    object
CRASH_YEAR                               0.000000     int64
CRASH_MONTH                              0.000000     int64
CRASH_HOUR                               0.000000     int64
CRASH_DAY_OF_WEEK                        0.000000     int64
POSTED_SPEED_LIMIT                       0.000000     int64
TRAFFIC_CONTROL_DEVICE                   0.000000    object
DEVICE_CONDITION                         0.000000    object
WEATHER_CONDITION                        0.000000    object
LIGHTING_CONDITION                       0.000000    object
FIRST_CRASH_TYPE                         0.000000    object
TRAFFICWAY_TYPE                          0.000000    object
STRAIGHT_ALIGNMENT                       0.000000     int64
GOOD_ROADWAY_SUFACE                      0.000000     int64
ROAD_DEFECT                              0.000000     int64
DESK_REPORT_TYPE                         0.000000     int64
CRASH_TYPE                               0.000000    object
DAMAGE                                   0.000000    object
DATE_POLICE_NOTIFIED                     0.000000    object
PRIM_CONTRIBUTORY_CAUSE                  0.000000    object
SEC_CONTRIBUTORY_CAUSE                   0.000000    object
STREET_NO                                0.000000     int64
STREET_DIRECTION                         0.000000    object
STREET_NAME                              0.000000    object
BEAT_OF_OCCURRENCE                       0.000000     int64
NUM_UNITS                                0.000000     int64
MOST_SEVERE_INJURY                       0.000000    object
INJURIES_TOTAL                           0.000000     int64
INJURIES_FATAL                           0.000000     int64
INJURIES_INCAPACITATING                  0.000000     int64
INJURIES_NON_INCAPACITATING              0.000000     int64
INJURIES_REPORTED_NOT_EVIDENT            0.000000     int64
INJURIES_NO_INDICATION                   0.000000     int64
INJURIES_UNKNOWN                         0.000000     int64
LATITUDE                                 0.000000    object
LONGITUDE                                0.000000    object
LOCATION                                 0.000000    object
PERSON_TYPE                              0.000000    object
VEHICLE_ID                               0.000000    object
MALE_PERSON                              0.000000     int64
AGE                                      0.000000     int64
DRIVERS_LICENSE_STATE                    4.029703    object
SAFETY_EQUIPMENT                         0.000000    object
AIRBAG_DEPLOYED                          0.000000     int64
EJECTION                                 0.000000     int64
DRIVER_ACTION                            0.000000    object
DRIVER_VISION                            0.000000    object
PHYSICAL_CONDITION                       0.000000    object
BAC_RESULT                               0.000000    object
```

Clean Merged Dataset

Drop Rows with Missing Values

In [60]:
```
df.dropna(inplace = True)
ed.show_info(df)
```

```
Lenght of Dataset: 154183
                               missing_values_% Data_type
CRASH_RECORD_ID                             0.0    object
RD_NO                                       0.0    object
CRASH_YEAR                                  0.0     int64
CRASH_MONTH                                 0.0     int64
CRASH_HOUR                                  0.0     int64
CRASH_DAY_OF_WEEK                           0.0     int64
POSTED_SPEED_LIMIT                          0.0     int64
TRAFFIC_CONTROL_DEVICE                      0.0    object
DEVICE_CONDITION                            0.0    object
WEATHER_CONDITION                           0.0    object
LIGHTING_CONDITION                          0.0    object
FIRST_CRASH_TYPE                            0.0    object
TRAFFICWAY_TYPE                             0.0    object
STRAIGHT_ALIGNMENT                          0.0     int64
GOOD_ROADWAY_SUFACE                         0.0     int64
ROAD_DEFECT                                 0.0     int64
DESK_REPORT_TYPE                            0.0     int64
CRASH_TYPE                                  0.0    object
DAMAGE                                      0.0    object
DATE_POLICE_NOTIFIED                        0.0    object
PRIM_CONTRIBUTORY_CAUSE                     0.0    object
SEC_CONTRIBUTORY_CAUSE                      0.0    object
STREET_NO                                   0.0     int64
STREET_DIRECTION                            0.0    object
STREET_NAME                                 0.0    object
BEAT_OF_OCCURRENCE                          0.0     int64
NUM_UNITS                                   0.0     int64
MOST_SEVERE_INJURY                          0.0    object
INJURIES_TOTAL                              0.0     int64
INJURIES_FATAL                              0.0     int64
INJURIES_INCAPACITATING                     0.0     int64
INJURIES_NON_INCAPACITATING                 0.0     int64
INJURIES_REPORTED_NOT_EVIDENT               0.0     int64
INJURIES_NO_INDICATION                      0.0     int64
INJURIES_UNKNOWN                            0.0     int64
LATITUDE                                    0.0    object
LONGITUDE                                   0.0    object
LOCATION                                    0.0    object
PERSON_TYPE                                 0.0    object
VEHICLE_ID                                  0.0    object
MALE_PERSON                                 0.0     int64
```

```
AGE                                        0.0     int64
DRIVERS_LICENSE_STATE                      0.0    object
SAFETY_EQUIPMENT                           0.0    object
AIRBAG_DEPLOYED                            0.0     int64
EJECTION                                   0.0     int64
DRIVER_ACTION                              0.0    object
DRIVER_VISION                              0.0    object
PHYSICAL_CONDITION                         0.0    object
BAC_RESULT                                 0.0    object
```

Now I will see if I can drop **SEC_CONTRIBUTORY_CAUSE**, but first I'll try to replace **PRIM_CONTRIBUTORY_CAUSE** that are not determined by **PRIM_CONTRIBUTORY_CAUSE** values if they exist.

In [61]:
```
df.PRIM_CONTRIBUTORY_CAUSE.value_counts()
```

Out[61]:
```
UNABLE TO DETERMINE
47740
FAILING TO YIELD RIGHT-OF-WAY
19484
FOLLOWING TOO CLOSELY
18125
FAILING TO REDUCE SPEED TO AVOID CRASH
8485
IMPROPER OVERTAKING/PASSING
8049
NOT APPLICABLE
7459
IMPROPER BACKING
6109
IMPROPER TURNING/NO SIGNAL
6066
IMPROPER LANE USAGE
6038
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
4681
DISREGARDING TRAFFIC SIGNALS
3855
WEATHER
2890
DISREGARDING STOP SIGN
2174
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESS
IVE MANNER      1678
DISTRACTION - FROM INSIDE VEHICLE
1403
EQUIPMENT - VEHICLE CONDITION
1313
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
1182
PHYSICAL CONDITION OF DRIVER
1162
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
```

```
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
1162
DISTRACTION - FROM OUTSIDE VEHICLE
847
DRIVING ON WRONG SIDE/WRONG WAY
819
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
565
DISREGARDING OTHER TRAFFIC SIGNS
416
ROAD CONSTRUCTION/MAINTENANCE
403
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
353
CELL PHONE USE OTHER THAN TEXTING
273
DISREGARDING ROAD MARKINGS
228
EXCEEDING SAFE SPEED FOR CONDITIONS
221
ANIMAL
169
EXCEEDING AUTHORIZED SPEED LIMIT
138
TURNING RIGHT ON RED
135
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
135
RELATED TO BUS STOP
124
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER,
ETC.)           105
DISREGARDING YIELD SIGN
75
TEXTING
70
OBSTRUCTED CROSSWALKS
25
PASSING STOPPED SCHOOL BUS
17
BICYCLE ADVANCING LEGALLY ON RED LIGHT
6
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
4
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
```

In [62]:
```
ex, row in df.iterrows():
df.loc[index,'PRIM_CONTRIBUTORY_CAUSE'] == 'UNABLE TO DETERMINE':
 if (df.loc[index,'SEC_CONTRIBUTORY_CAUSE'] != 'UNABLE TO DETERMINE') & (
     df.loc[index,'PRIM_CONTRIBUTORY_CAUSE'] = df.loc[index,'SEC_CONTRIBU
```

In [63]:
```
df.PRIM_CONTRIBUTORY_CAUSE.value_counts()
```

```
Out[63]: UNABLE TO DETERMINE
46361
FAILING TO YIELD RIGHT-OF-WAY
19601
FOLLOWING TOO CLOSELY
18257
FAILING TO REDUCE SPEED TO AVOID CRASH
8625
IMPROPER OVERTAKING/PASSING
8102
NOT APPLICABLE
7459
IMPROPER BACKING
6153
IMPROPER LANE USAGE
6136
IMPROPER TURNING/NO SIGNAL
6092
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
4952
DISREGARDING TRAFFIC SIGNALS
3877
WEATHER
3043
DISREGARDING STOP SIGN
2190
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESS
IVE MANNER     1689
DISTRACTION - FROM INSIDE VEHICLE
1418
EQUIPMENT - VEHICLE CONDITION
1339
PHYSICAL CONDITION OF DRIVER
1226
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
1193
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
1192
DISTRACTION - FROM OUTSIDE VEHICLE
858
DRIVING ON WRONG SIDE/WRONG WAY
828
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
573
DISREGARDING OTHER TRAFFIC SIGNS
428
ROAD CONSTRUCTION/MAINTENANCE
413
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
354
CELL PHONE USE OTHER THAN TEXTING
278
DISREGARDING ROAD MARKINGS
233
```

```
EXCEEDING SAFE SPEED FOR CONDITIONS

224
ANIMAL
173
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
173
EXCEEDING AUTHORIZED SPEED LIMIT
141
TURNING RIGHT ON RED
139
RELATED TO BUS STOP
136
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER,
ETC.)                106
DISREGARDING YIELD SIGN
83
TEXTING
70
OBSTRUCTED CROSSWALKS
32
PASSING STOPPED SCHOOL BUS
17
BICYCLE ADVANCING LEGALLY ON RED LIGHT
11
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
8
```

In [64]:
```
df.drop(['SEC_CONTRIBUTORY_CAUSE'],axis = 1, inplace = True)
```

Due to the fact that observations with **undetermiate cause** will not be helpful in modeling, I will **drop them** too.

In [65]:
```
index_ptype = df[~(df['PRIM_CONTRIBUTORY_CAUSE'] != 'NOT APPLICABLE')].in
df.drop(index_ptype, inplace=True)

index_ptype = df[~(df['PRIM_CONTRIBUTORY_CAUSE'] != 'UNABLE TO DETERMINE
df.drop(index_ptype, inplace=True)
```

In [66]:
```
df.PRIM_CONTRIBUTORY_CAUSE.value_counts()
```

Out[66]:
```
FAILING TO YIELD RIGHT-OF-WAY
19601
FOLLOWING TOO CLOSELY
18257
FAILING TO REDUCE SPEED TO AVOID CRASH
8625
IMPROPER OVERTAKING/PASSING
8102
IMPROPER BACKING
6153
IMPROPER LANE USAGE
```

6136

IMPROPER TURNING/NO SIGNAL
6092
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
4952
DISREGARDING TRAFFIC SIGNALS
3877
WEATHER
3043
DISREGARDING STOP SIGN
2190
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESS
IVE MANNER 1689
DISTRACTION - FROM INSIDE VEHICLE
1418
EQUIPMENT - VEHICLE CONDITION
1339
PHYSICAL CONDITION OF DRIVER
1226
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
1193
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
1192
DISTRACTION - FROM OUTSIDE VEHICLE
858
DRIVING ON WRONG SIDE/WRONG WAY
828
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
573
DISREGARDING OTHER TRAFFIC SIGNS
428
ROAD CONSTRUCTION/MAINTENANCE
413
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
354
CELL PHONE USE OTHER THAN TEXTING
278
DISREGARDING ROAD MARKINGS
233
EXCEEDING SAFE SPEED FOR CONDITIONS
224
ANIMAL
173
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
173
EXCEEDING AUTHORIZED SPEED LIMIT
141
TURNING RIGHT ON RED
139
RELATED TO BUS STOP
136
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER,
ETC.) 106

```
DISREGARDING YIELD SIGN

83
TEXTING
70
OBSTRUCTED CROSSWALKS
32
PASSING STOPPED SCHOOL BUS
17
BICYCLE ADVANCING LEGALLY ON RED LIGHT
11
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
8
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
```

In [67]:
```
ed.show_info(df)
```

```
Lenght of Dataset: 100363
                              missing_values_% Data_type
CRASH_RECORD_ID                            0.0    object
RD_NO                                      0.0    object
CRASH_YEAR                                 0.0     int64
CRASH_MONTH                                0.0     int64
CRASH_HOUR                                 0.0     int64
CRASH_DAY_OF_WEEK                          0.0     int64
POSTED_SPEED_LIMIT                         0.0     int64
TRAFFIC_CONTROL_DEVICE                     0.0    object
DEVICE_CONDITION                           0.0    object
WEATHER_CONDITION                          0.0    object
LIGHTING_CONDITION                         0.0    object
FIRST_CRASH_TYPE                           0.0    object
TRAFFICWAY_TYPE                            0.0    object
STRAIGHT_ALIGNMENT                         0.0     int64
GOOD_ROADWAY_SUFACE                        0.0     int64
ROAD_DEFECT                                0.0     int64
DESK_REPORT_TYPE                           0.0     int64
CRASH_TYPE                                 0.0    object
DAMAGE                                     0.0    object
DATE_POLICE_NOTIFIED                       0.0    object
PRIM_CONTRIBUTORY_CAUSE                    0.0    object
STREET_NO                                  0.0     int64
STREET_DIRECTION                           0.0    object
STREET_NAME                                0.0    object
BEAT_OF_OCCURRENCE                         0.0     int64
NUM_UNITS                                  0.0     int64
MOST_SEVERE_INJURY                         0.0    object
INJURIES_TOTAL                             0.0     int64
INJURIES_FATAL                             0.0     int64
INJURIES_INCAPACITATING                    0.0     int64
INJURIES_NON_INCAPACITATING                0.0     int64
INJURIES_REPORTED_NOT_EVIDENT              0.0     int64
INJURIES_NO_INDICATION                     0.0     int64
INJURIES_UNKNOWN                           0.0     int64
LATITUDE                                   0.0    object
```

```
LATITUDE                                        0.0     object
LONGITUDE                                       0.0     object

LOCATION                                        0.0     object
PERSON_TYPE                                     0.0     object
VEHICLE_ID                                      0.0     object
MALE_PERSON                                     0.0      int64
AGE                                             0.0      int64
DRIVERS_LICENSE_STATE                           0.0     object
SAFETY_EQUIPMENT                                0.0     object
AIRBAG_DEPLOYED                                 0.0      int64
EJECTION                                        0.0      int64
DRIVER_ACTION                                   0.0     object
DRIVER_VISION                                   0.0     object
PHYSICAL_CONDITION                              0.0     object
BAC_RESULT                                      0.0     object
```

In [68]:

```
df.DRIVERS_LICENSE_STATE.value_counts()
```

Out[68]:

```
IL     93742
IN      1705
WI       590
MI       515
FL       372
       ...  
ZD         1
SX         1
BU         1
LD         1
DR         1
Name: DRIVERS_LICENSE_STATE, Length: 121, dtype: int64
```

In [69]:

```
states.head()
```

Out[69]:

	State	Abbreviation	Unnamed: 2
0	Alabama	AL	NaN
1	Alaska	AK	NaN
2	Arizona	AZ	NaN
3	Arkansas	AR	NaN
4	California	CA	NaN

In [70]:

```
states_list = states['Abbreviation'].tolist()
states_index = df[~(df['DRIVERS_LICENSE_STATE'].isin(states_list))].inde
df.drop(states_index, inplace=True)
```

In [71]:

```
ed.show_info(df)
```

```
Lenght of Dataset: 99909
                               missing_values_% Data_type
CRASH_RECORD_ID                                 0.0     object
```

```
CRASH_RECORD_ID                          0.0    object
RD_NO                                    0.0    object
CRASH_YEAR                               0.0     int64
CRASH_MONTH                              0.0     int64
CRASH_HOUR                               0.0     int64
CRASH_DAY_OF_WEEK                        0.0     int64
POSTED_SPEED_LIMIT                       0.0     int64
TRAFFIC_CONTROL_DEVICE                   0.0    object
DEVICE_CONDITION                         0.0    object
WEATHER_CONDITION                        0.0    object
LIGHTING_CONDITION                       0.0    object
FIRST_CRASH_TYPE                         0.0    object
TRAFFICWAY_TYPE                          0.0    object
STRAIGHT_ALIGNMENT                       0.0     int64
GOOD_ROADWAY_SUFACE                      0.0     int64
ROAD_DEFECT                              0.0     int64
DESK_REPORT_TYPE                         0.0     int64
CRASH_TYPE                               0.0    object
DAMAGE                                   0.0    object
DATE_POLICE_NOTIFIED                     0.0    object
PRIM_CONTRIBUTORY_CAUSE                  0.0    object
STREET_NO                                0.0     int64
STREET_DIRECTION                         0.0    object
STREET_NAME                              0.0    object
BEAT_OF_OCCURRENCE                       0.0     int64
NUM_UNITS                                0.0     int64
MOST_SEVERE_INJURY                       0.0    object
INJURIES_TOTAL                           0.0     int64
INJURIES_FATAL                           0.0     int64
INJURIES_INCAPACITATING                  0.0     int64
INJURIES_NON_INCAPACITATING              0.0     int64
INJURIES_REPORTED_NOT_EVIDENT            0.0     int64
INJURIES_NO_INDICATION                   0.0     int64
INJURIES_UNKNOWN                         0.0     int64
LATITUDE                                 0.0    object
LONGITUDE                                0.0    object
LOCATION                                 0.0    object
PERSON_TYPE                              0.0    object
VEHICLE_ID                               0.0    object
MALE_PERSON                              0.0     int64
AGE                                      0.0     int64
DRIVERS_LICENSE_STATE                    0.0    object
SAFETY_EQUIPMENT                         0.0    object
AIRBAG_DEPLOYED                          0.0     int64
EJECTION                                 0.0     int64
DRIVER_ACTION                            0.0    object
DRIVER_VISION                            0.0    object
PHYSICAL_CONDITION                       0.0    object
BAC_RESULT                               0.0    object
```

Exploration Analysis

In [ ]:

```

```

In [ ]:

```

```

Descriptive Analysis

First, I will perform descriptive analysis to get the mean, standart deviation and value counts of the numeric columns.

In [72]:

```
df.describe()
```

Out[72]:

	CRASH_YEAR	CRASH_MONTH	CRASH_HOUR	CRASH_DAY_OF_WEEK	POSTED_SPEED_L
count	99909.000000	99909.000000	99909.000000	99909.000000	99909.00
mean	2019.423035	6.468426	13.381397	4.157133	29.16
std	0.494043	3.444740	5.430082	1.958839	5.28
min	2019.000000	1.000000	0.000000	1.000000	0.00
25%	2019.000000	3.000000	10.000000	2.000000	30.00
50%	2019.000000	7.000000	14.000000	4.000000	30.00
75%	2020.000000	9.000000	17.000000	6.000000	30.00
max	2020.000000	12.000000	23.000000	7.000000	70.00

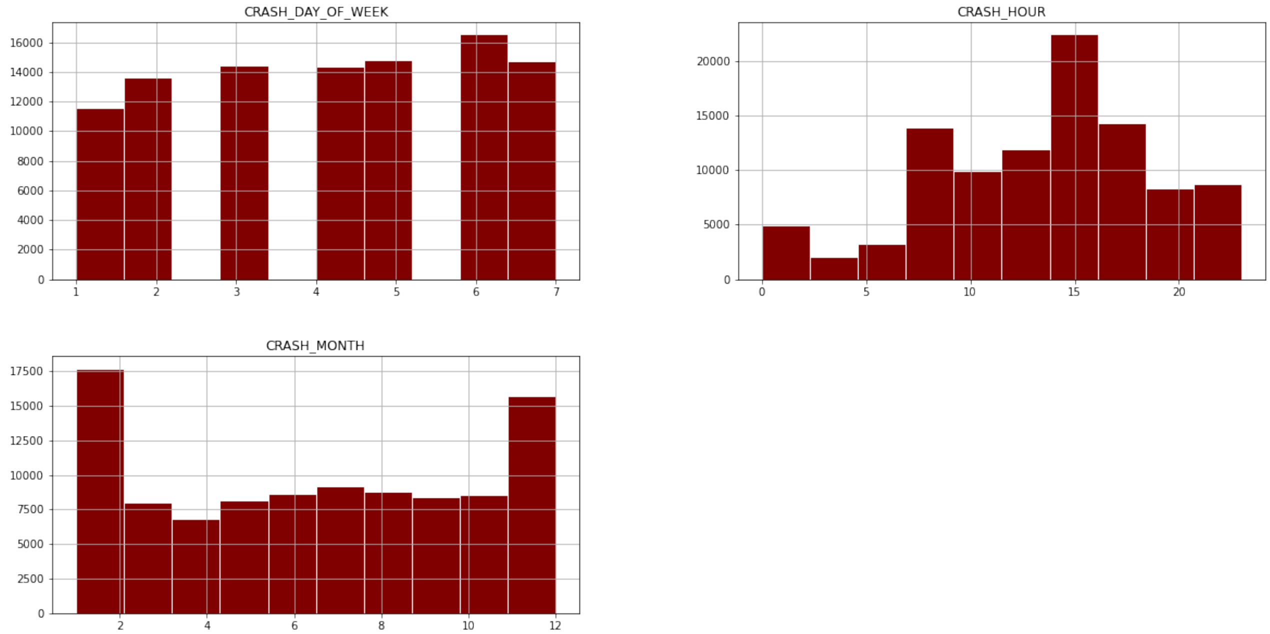
8 rows × 23 columns

- The describe function does not give a lot of useful information, since the most of the numeric columns represent binary categorical data.

Plot Histogram for Numeric Features

I will plot a histograms for features like **CRASH_MONTH, CRASH_HOUR, CRASH_DAY_OF_WEEK** to see what is the most common value for those features.

In [73]:
```
cols_cont =['CRASH_MONTH', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK']

df[cols_cont].hist(figsize = (20,10), edgecolor="w", facecolor='maroon')
```

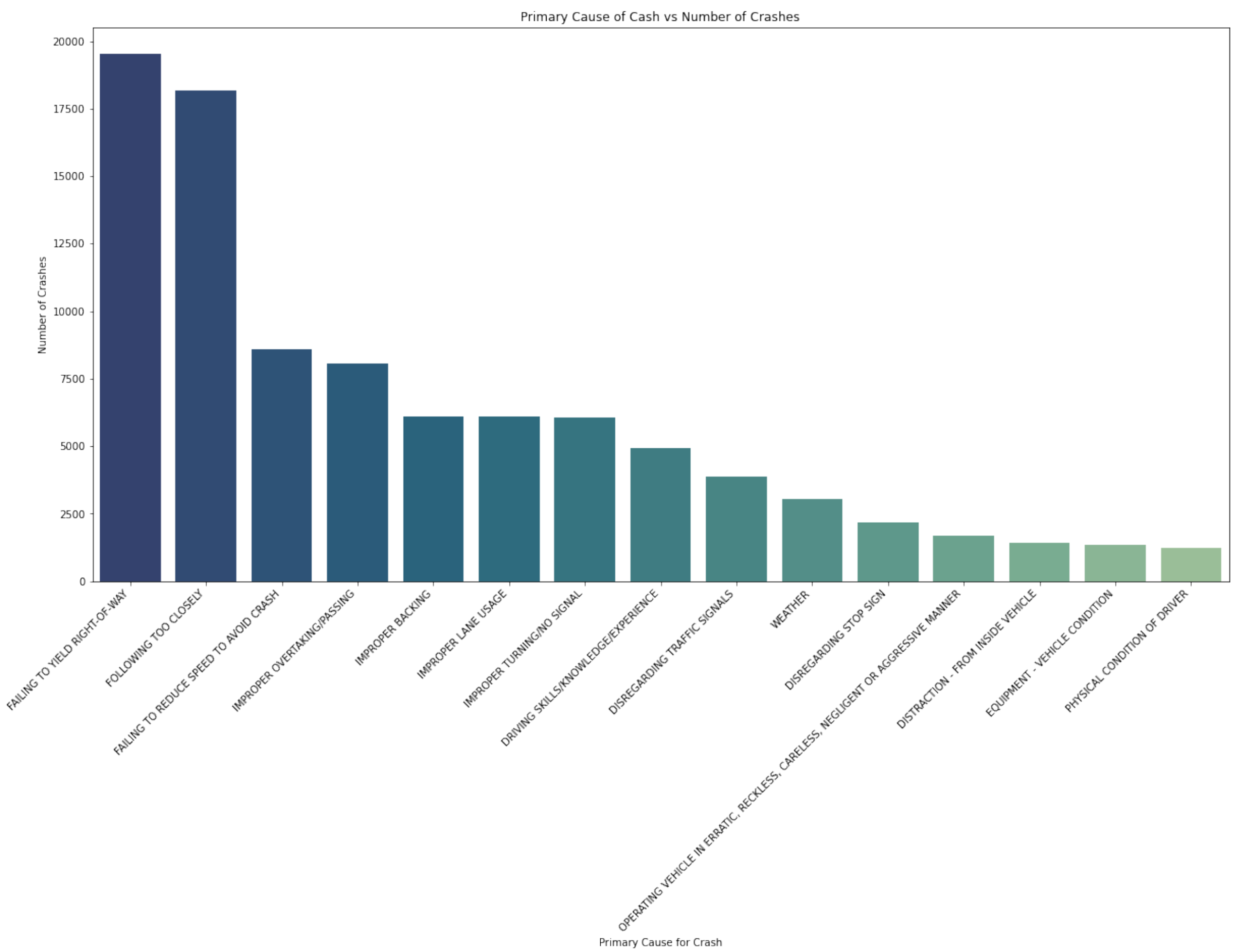


- The highest amount of crashes happened on **Saturday**
- Large amount of crashes happened during afternoon **(around 3 PM)**. My guess would be the reason is traffic.
- And majority of crashes are during **January and December**, probably due to the weather condition.

Primary Cause of Crash vs. Number of Crashes

Now I will investigate what causes of crash have the highest amount....

```
In [74]: plt.figure(figsize =(20,10))
         plt.xticks(rotation=45, horizontalalignment='right')

         ax = sns.countplot(x="PRIM_CONTRIBUTORY_CAUSE", data=df,
                            order = df['PRIM_CONTRIBUTORY_CAUSE'].value_counts().i
                            palette = 'crest_r')

         plt.xlabel('Primary Cause for Crash')
         plt.ylabel('Number of Crashes')
         plt.title('Primary Cause of Cash vs Number of Crashes')
         plt.show()
```



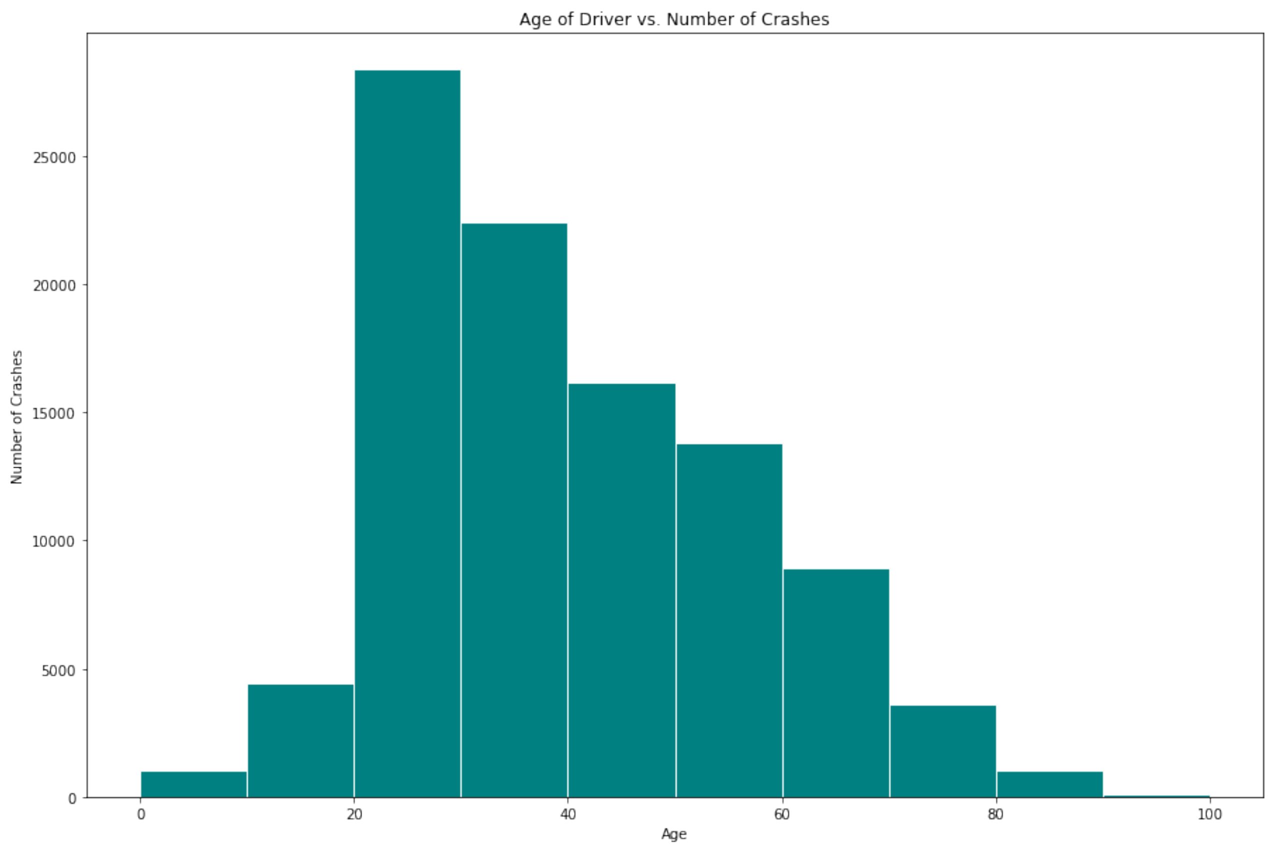
As the graph shows, most of the crashes happened because of following causes: **failing to yield, following too closely.**

Driver's Age vs. Number of Crashes

Let's see at what age drivers were at thye moment of the crashes.

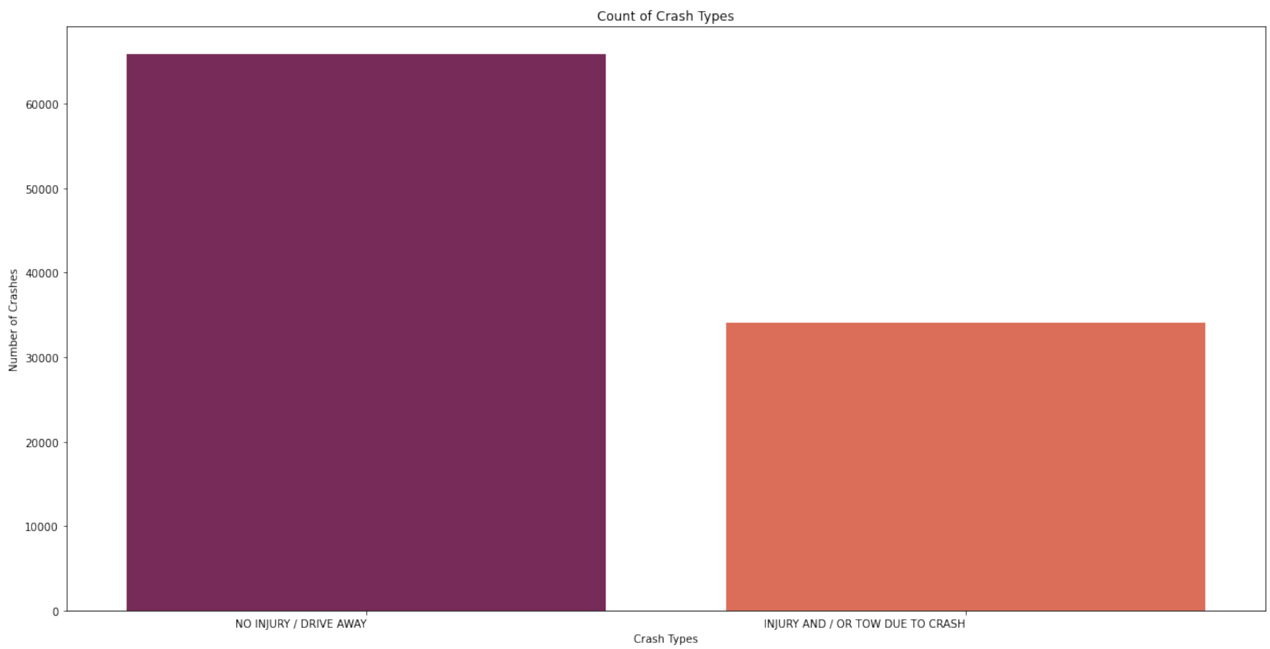
In [124]:
```
fig, ax = plt.subplots(figsize =(15,10))

ax.hist(df['AGE'], edgecolor="w", facecolor='teal')

# Set title
ax.set_title("Age of Driver vs. Number of Crashes")

# adding labels
ax.set_xlabel('Age')
ax.set_ylabel('Number of Crashes')
```

Out[124]:
```
Text(0, 0.5, 'Number of Crashes')
```



Crash Types Frequency

In [76]:
```
plt.figure(figsize =(20,10))
plt.xticks(horizontalalignment='right')

ax = sns.countplot(x="CRASH_TYPE", data=df,
                   order = df['CRASH_TYPE'].value_counts().index,
                   palette = 'rocket')


plt.xlabel('Crash Types')
plt.ylabel('Number of Crashes')
plt.title('Count of Crash Types')
plt.show()
```



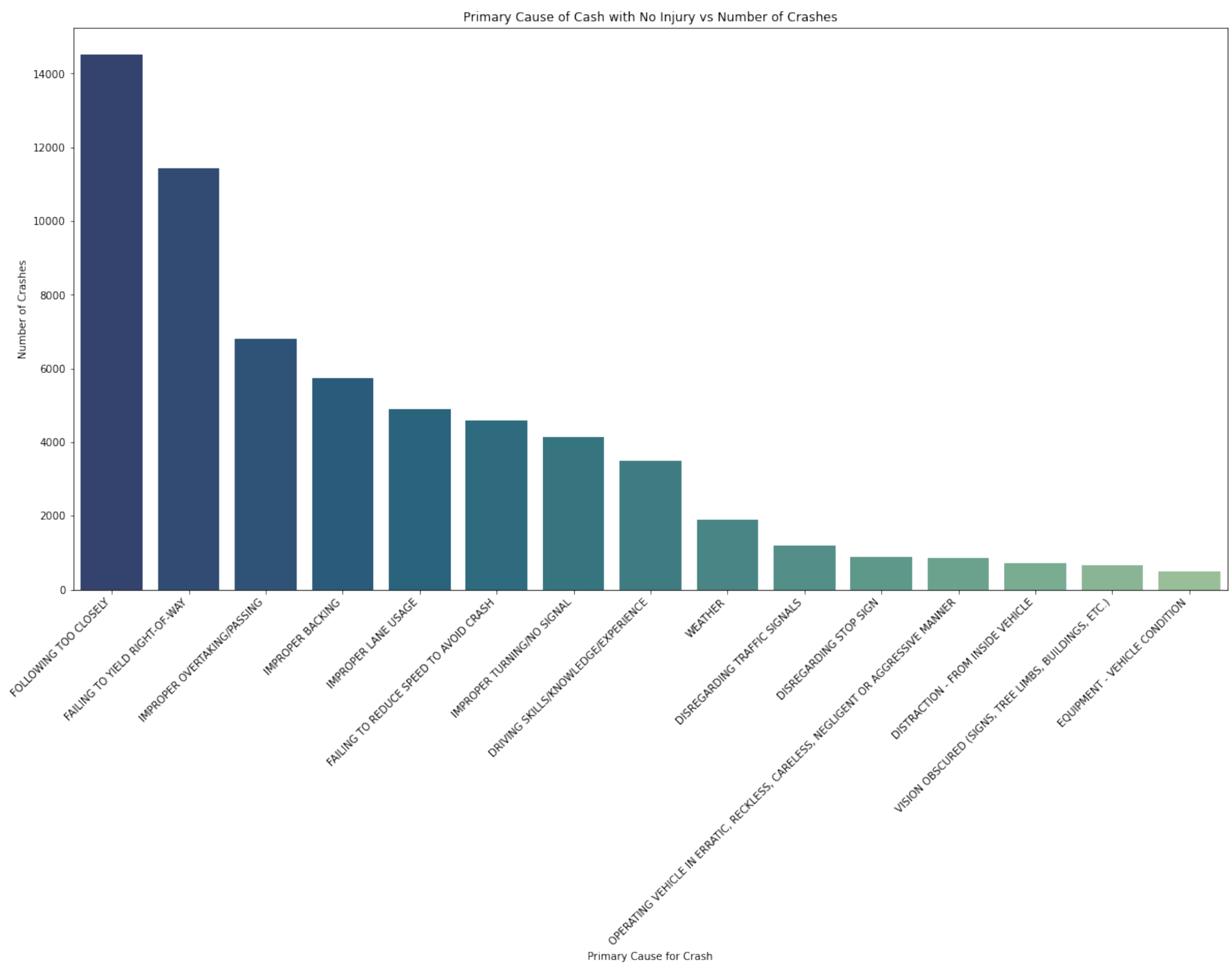
Exploration of Crashes with No Injury

Now, I will look into the observations where the **crash resulted in no injury** and see what were the **main causes and reasons** for those crashes.

In [77]:
```
df_no_injury = df[df['CRASH_TYPE'] == 'NO INJURY / DRIVE AWAY']
```

Primary Cause of the Crashes

In [78]:
```
plt.figure(figsize =(20,10))
plt.xticks(rotation=45, horizontalalignment='right')

ax = sns.countplot(x="PRIM_CONTRIBUTORY_CAUSE", data=df_no_injury,
                   order = df_no_injury['PRIM_CONTRIBUTORY_CAUSE'].value_
                   palette = 'crest_r')

plt.xlabel('Primary Cause for Crash')
plt.ylabel('Number of Crashes')
plt.title('Primary Cause of Cash with No Injury vs Number of Crashes')
plt.show()
```



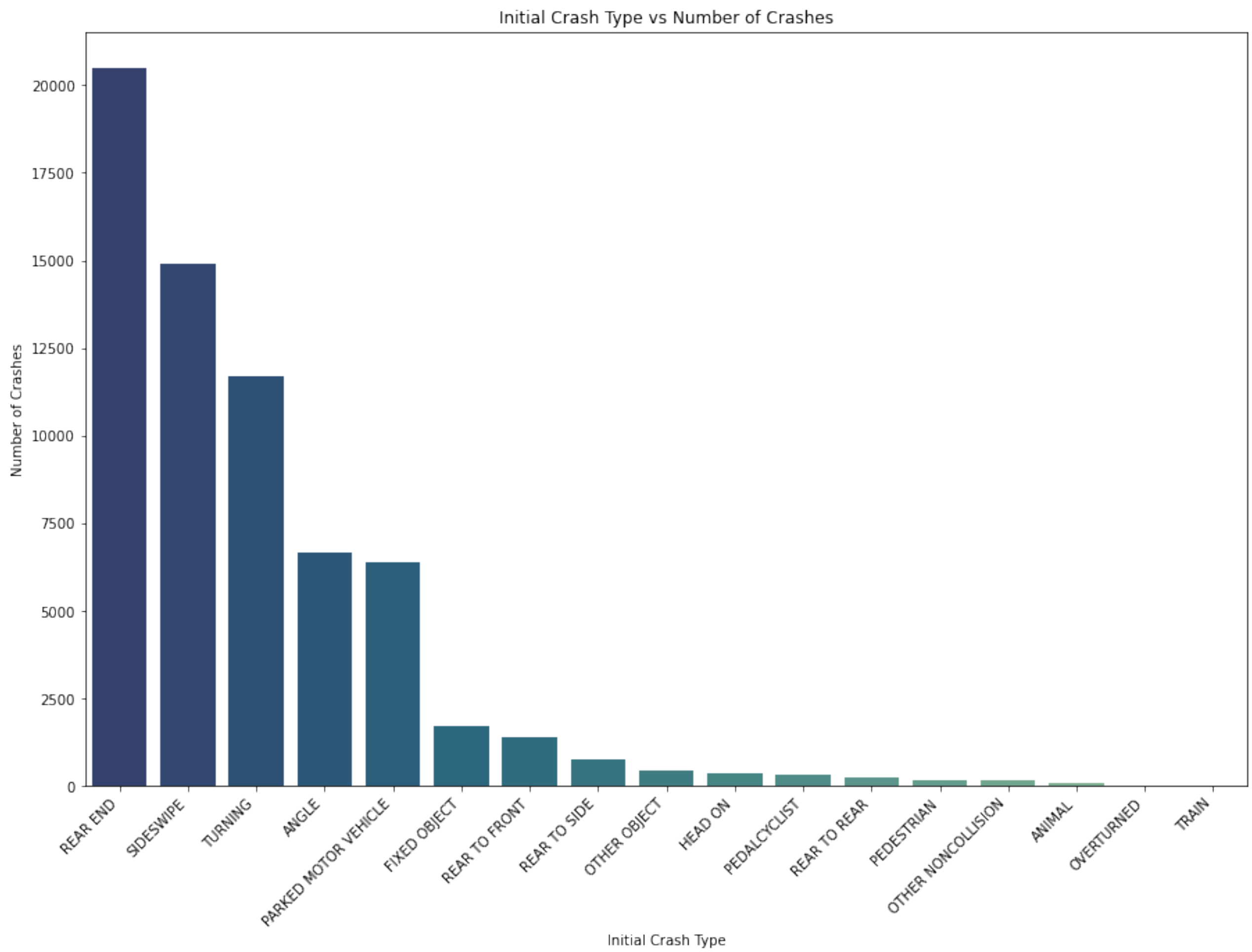
As it is shown on graph, the majority of **no-injury crashes** were because of **following too closely or failing to yield.**

EDA Result:

- Most of the Crashes happened during **light time and clear weather.**
- The **roadway surface** for most of the cases was **good**.
- Vast majority of the crashes happened on **roads with no defect.**

Inital State of the Crashes

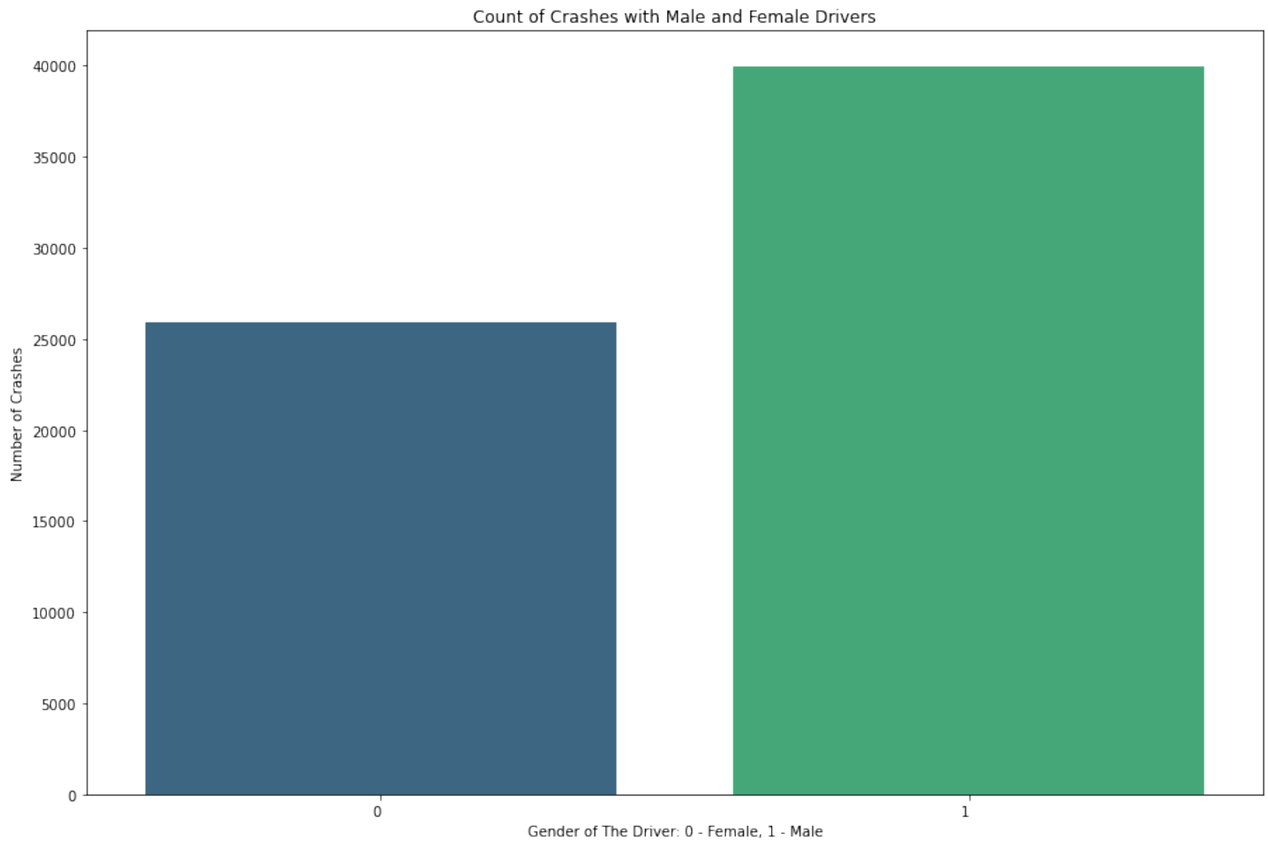
In [79]:
```
plt.figure(figsize =(15,10))
plt.xticks(rotation=45, horizontalalignment='right')

ax = sns.countplot(x="FIRST_CRASH_TYPE", data=df_no_injury,
                   order = df_no_injury['FIRST_CRASH_TYPE'].value_counts
plt.xlabel('Initial Crash Type')
plt.ylabel('Number of Crashes')
plt.title('Initial Crash Type vs Number of Crashes')
plt.show()
```



Majority of crashes happened on the **rear end, sideswipe or while turning.**

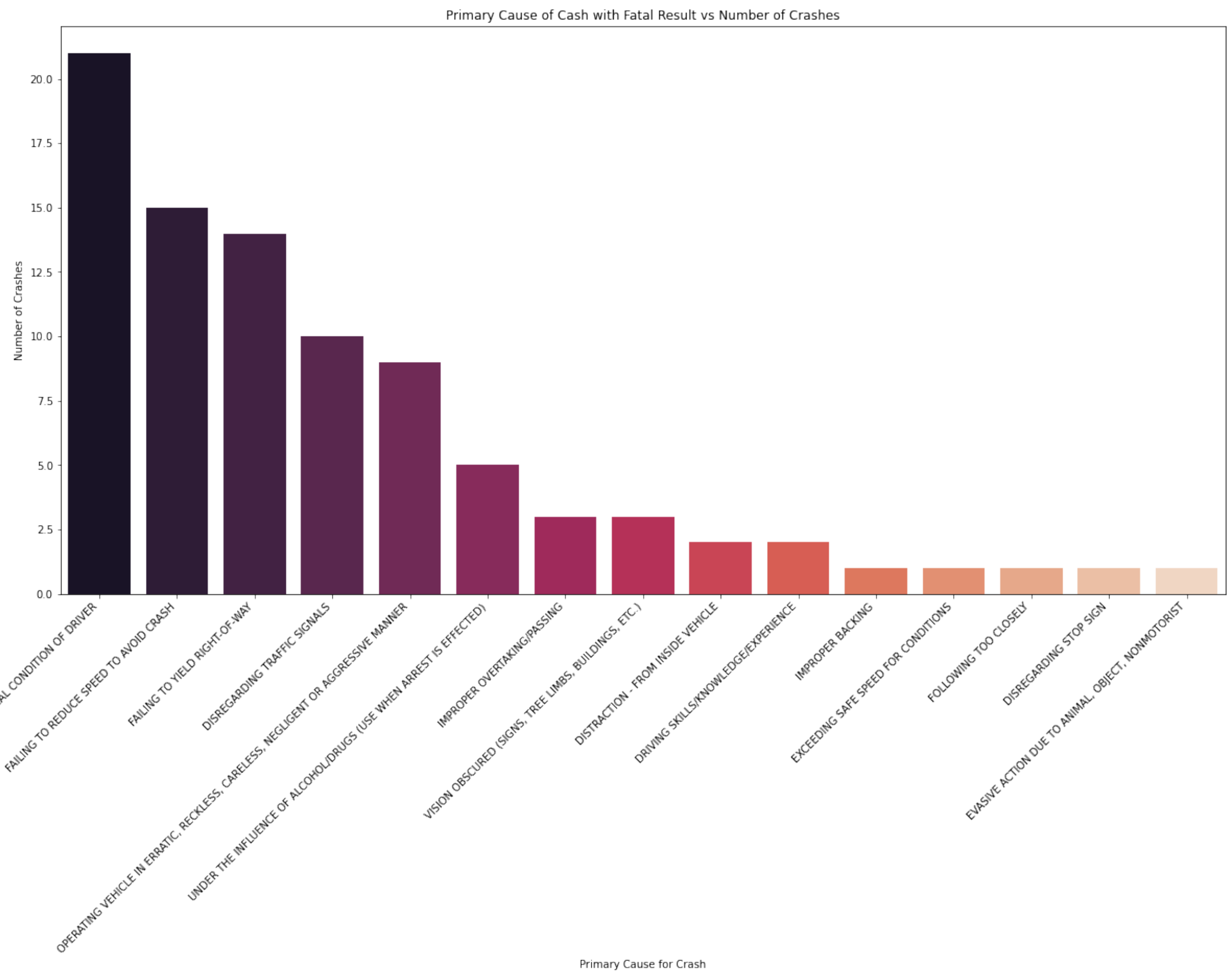
Driver's Gender

In [80]:
```
plt.figure(figsize =(15,10))
plt.xticks(horizontalalignment='right')

ax = sns.countplot(x="MALE_PERSON", data=df_no_injury, palette = 'viridi

plt.ylabel('Number of Crashes')
plt.xlabel('Gender of The Driver: 0 - Female, 1 - Male')
plt.title('Count of Crashes with Male and Female Drivers')
plt.show()
```



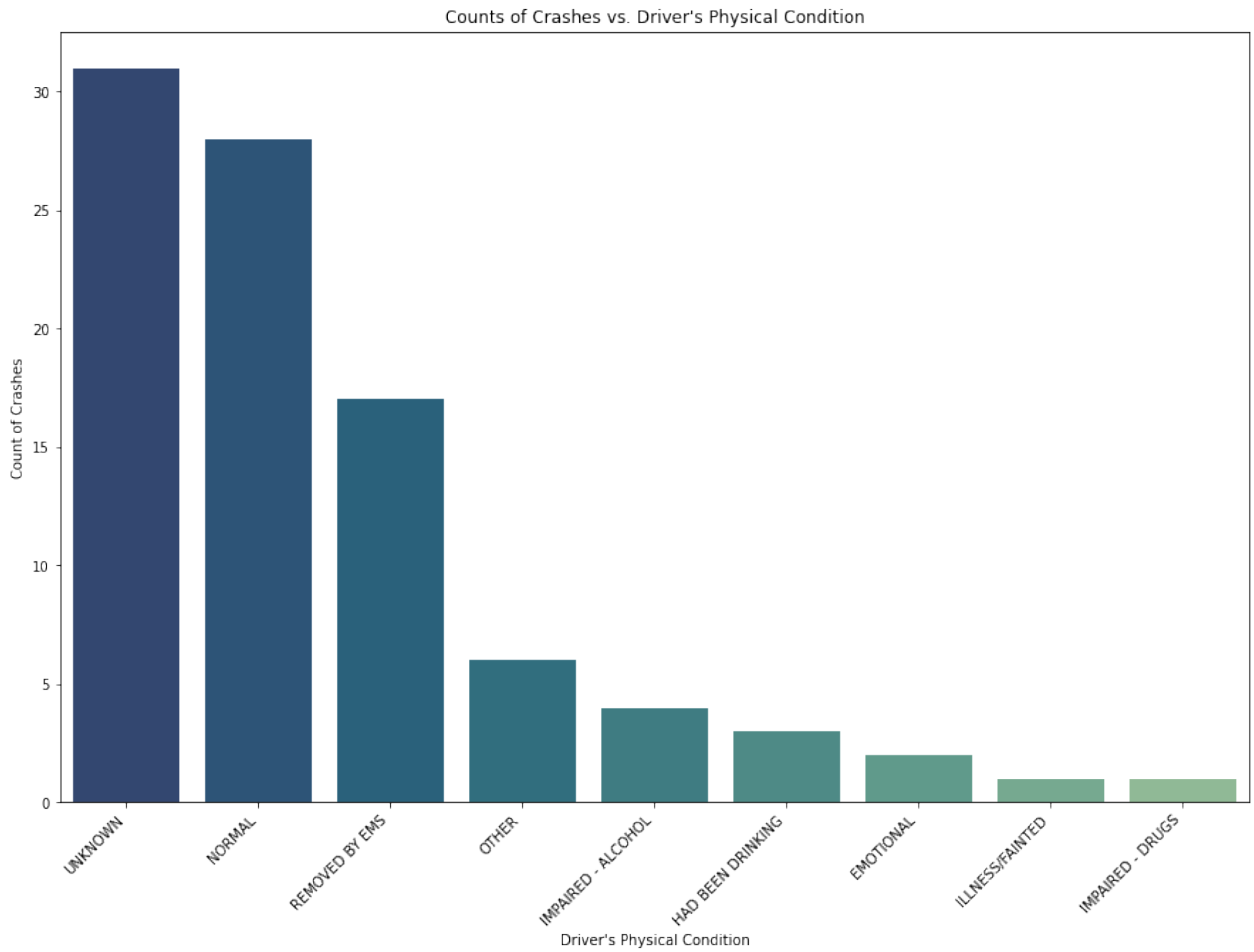
Crashes Resulted in Fatal Injury

In [81]:
```
df_fatal = df[df['MOST_SEVERE_INJURY'] == 'FATAL']

plt.figure(figsize =(20,10))
plt.xticks(rotation=45, horizontalalignment='right')

ax = sns.countplot(x="PRIM_CONTRIBUTORY_CAUSE", data=df_fatal,
                   order = df_fatal['PRIM_CONTRIBUTORY_CAUSE'].value_cour
                   palette = 'rocket')

plt.xlabel('Primary Cause for Crash')
plt.ylabel('Number of Crashes')
plt.title('Primary Cause of Cash with Fatal Result vs Number of Crashes'
plt.show()
```



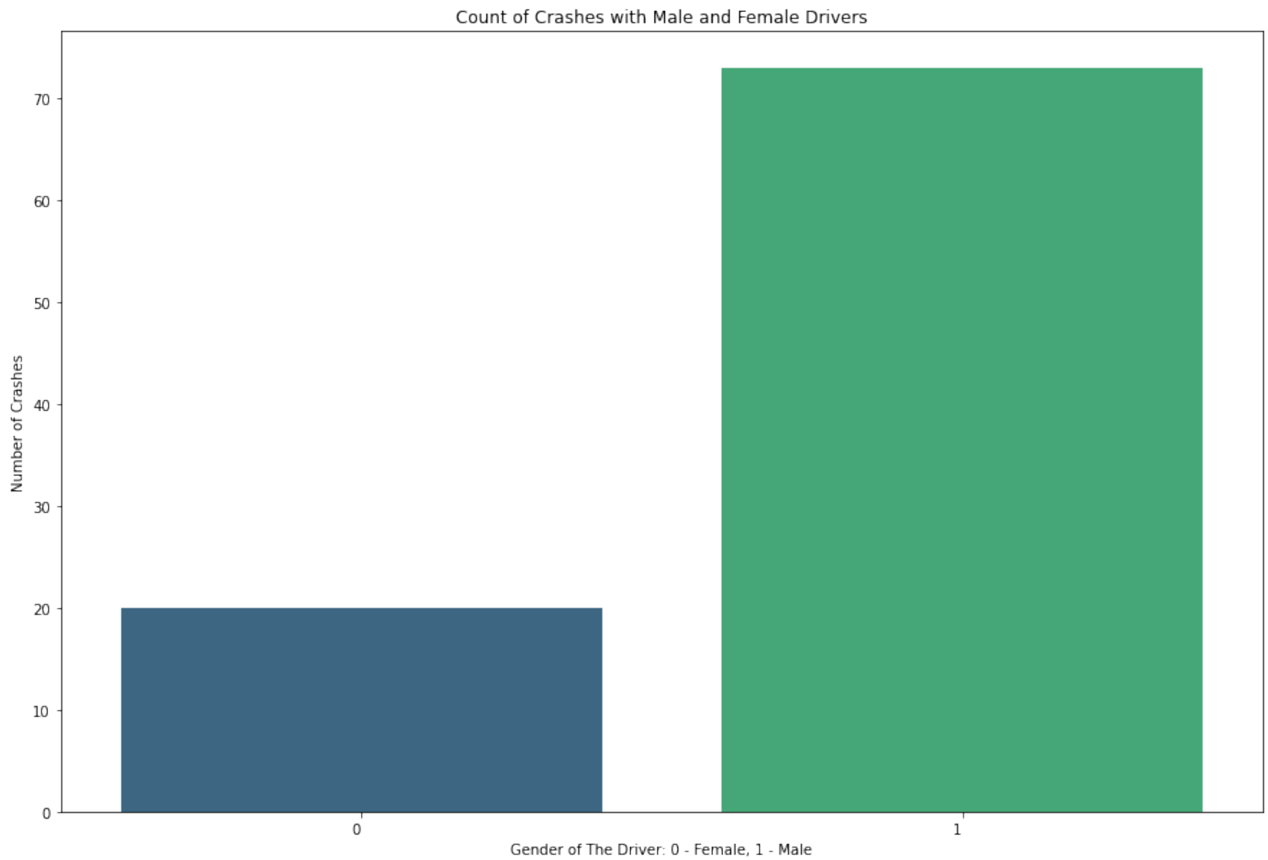
Since majority of crashes were caused by **Physical Condition of the driver**, let's see into that column.

In [82]:
```
plt.figure(figsize =(15,10))
plt.xticks(rotation=45, horizontalalignment='right')

ax = sns.countplot(x="PHYSICAL_CONDITION", data=df_fatal,
                   order = df_fatal['PHYSICAL_CONDITION'].value_counts()
plt.xlabel('Driver\'s Physical Condition')
plt.ylabel('Count of Crashes')
plt.title('Counts of Crashes vs. Driver\'s Physical Condition')
plt.show()
```



Driver's Gender of Crash with Fatal Injury

In [83]:
```
plt.figure(figsize =(15,10))
plt.xticks(horizontalalignment='right')

ax = sns.countplot(x="MALE_PERSON", data=df_fatal, palette = 'viridis')

plt.ylabel('Number of Crashes')
plt.xlabel('Gender of The Driver: 0 - Female, 1 - Male')
plt.title('Count of Crashes with Male and Female Drivers')
plt.show()
```



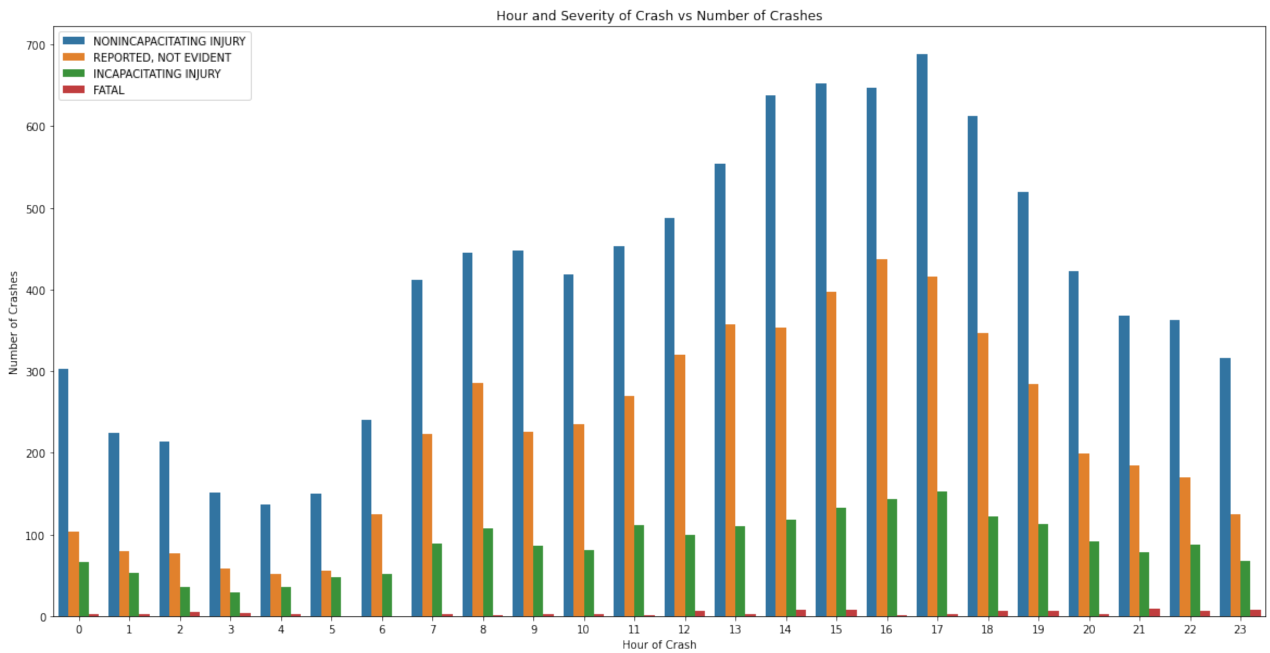
The graph above shows that most of the drivers involved in accident with fatal injury were male drivers.

EDA Result:

- Most of the Crashes with Fatal Injury happened during **light time or with some light, and weather was clear.**
- The **roadway surface** for most of the cases was **good**.
- Vast majority of the crashes happened on **roads with no defect.**

Severity of Injury and Hour of Crash

```
In [84]: df_injury = df[df['MOST_SEVERE_INJURY'] != 'NO INDICATION OF INJURY']


plt.figure(figsize =(20,10))
ax = sns.countplot(x="CRASH_HOUR", hue = 'MOST_SEVERE_INJURY',data=df_in
plt.xlabel('Hour of Crash')
plt.ylabel('Number of Crashes')
plt.title('Hour and Severity of Crash vs Number of Crashes')
plt.legend(loc = 'upper left')
plt.show()
```

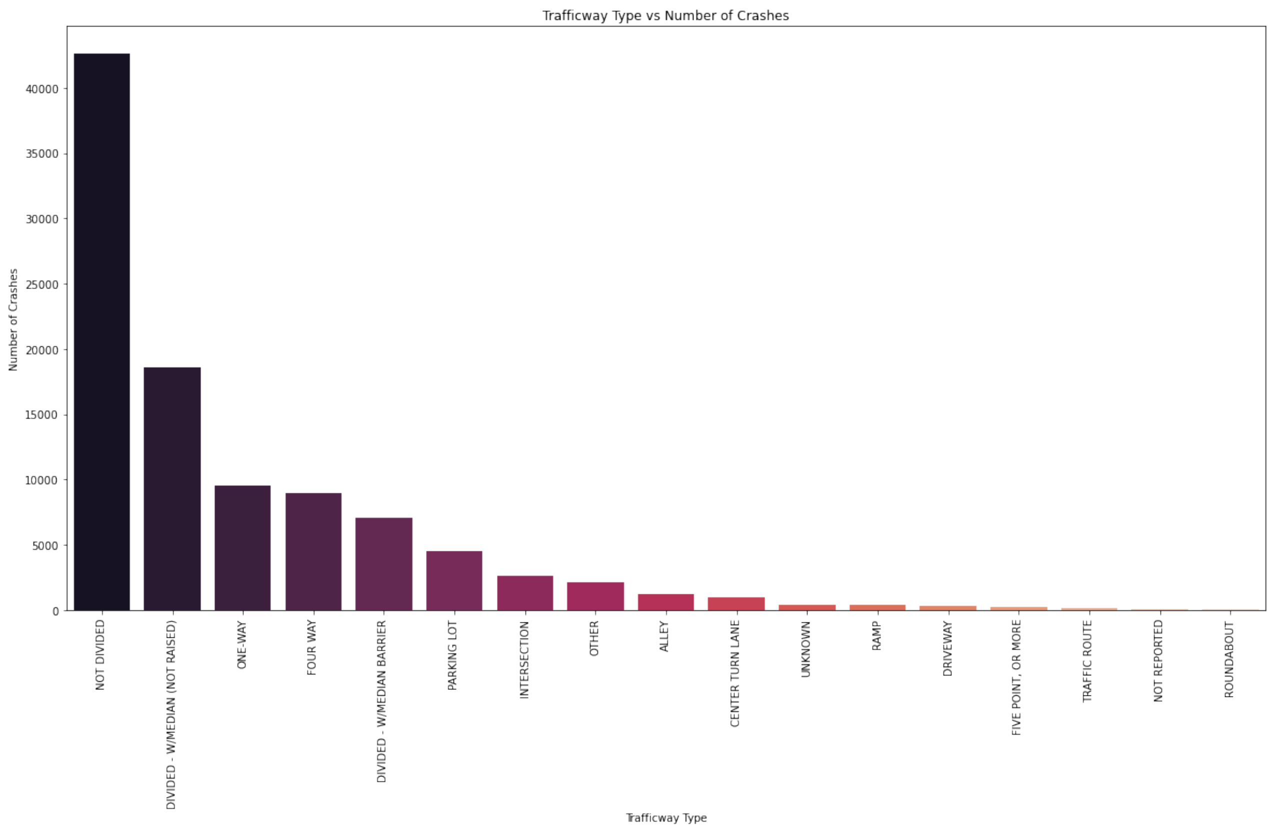


As the graph shows, the **peak hours** for the crashes with injury are from **1 PM to 5 PM.**

Trafficway Type and Number of Crashes

Lastly, I will look into the Trafficway type of the place where crash happened.

In [85]:
```
plt.figure(figsize =(20,10))
plt.xticks(rotation=90)

ax = sns.countplot(x="TRAFFICWAY_TYPE", data=df,
                   order = df['TRAFFICWAY_TYPE'].value_counts().index,
                   palette = 'rocket')
plt.xlabel('Trafficway Type')
plt.ylabel('Number of Crashes')
plt.title('Trafficway Type vs Number of Crashes')
plt.show()
```



For most of the crashes, the **trafficway was not divided.**

Binning Primary Contributory Cause for the Crash

The ease the work of the models, I will **regroup the types of causes** and create less categorized column with following categories: **Improper/Agressive Driving, Irresponsible Behavior and External/Other Causes.**

In [86]:
```
df.PRIM_CONTRIBUTORY_CAUSE.value_counts().to_dict()
```

Out[86]:
```
{'FAILING TO YIELD RIGHT-OF-WAY': 19532,
 'FOLLOWING TOO CLOSELY': 18186,
 'FAILING TO REDUCE SPEED TO AVOID CRASH': 8589,
 'IMPROPER OVERTAKING/PASSING': 8074,
 'IMPROPER BACKING': 6115,
 'IMPROPER LANE USAGE': 6104,
 'IMPROPER TURNING/NO SIGNAL': 6065,
 'DRIVING SKILLS/KNOWLEDGE/EXPERIENCE': 4915,
 'DISREGARDING TRAFFIC SIGNALS': 3858,
 'WEATHER': 3030,
 'DISREGARDING STOP SIGN': 2185,
 'OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRE
SSIVE MANNER': 1669,
 'DISTRACTION - FROM INSIDE VEHICLE': 1413,
 'EQUIPMENT - VEHICLE CONDITION': 1335,
 'PHYSICAL CONDITION OF DRIVER': 1219,
 'VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)': 1189,
 'UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)':
1181,
 'DISTRACTION - FROM OUTSIDE VEHICLE': 853,
 'DRIVING ON WRONG SIDE/WRONG WAY': 824,
 'ROAD ENGINEERING/SURFACE/MARKING DEFECTS': 573,
 'DISREGARDING OTHER TRAFFIC SIGNS': 422,
 'ROAD CONSTRUCTION/MAINTENANCE': 413,
 'EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST': 354,
 'CELL PHONE USE OTHER THAN TEXTING': 275,
 'DISREGARDING ROAD MARKINGS': 229,
 'EXCEEDING SAFE SPEED FOR CONDITIONS': 222,
 'ANIMAL': 173,
 'HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)': 172,
 'EXCEEDING AUTHORIZED SPEED LIMIT': 141,
 'TURNING RIGHT ON RED': 139,
 'RELATED TO BUS STOP': 136,
 'DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER
, ETC.)': 104,
 'DISREGARDING YIELD SIGN': 82,
 'TEXTING': 70,
 'OBSTRUCTED CROSSWALKS': 32,
 'PASSING STOPPED SCHOOL BUS': 17,
 'BICYCLE ADVANCING LEGALLY ON RED LIGHT': 11,
 'MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT': 8}
```

In [87]:
```
agress_dict = {'FAILING TO YIELD RIGHT-OF-WAY': 'AGRESSIVE/IMPROPER DRIVI
               'FAILING TO REDUCE SPEED TO AVOID CRASH': 'AGRESSIVE/IMPRO
               'IMPROPER OVERTAKING/PASSING': 'AGRESSIVE/IMPROPER DRIVIN
               'IMPROPER TURNING/NO SIGNAL': 'AGRESSIVE/IMPROPER DRIVING
               'IMPROPER LANE USAGE': 'AGRESSIVE/IMPROPER DRIVING',
               'IMPROPER BACKING': 'AGRESSIVE/IMPROPER DRIVING',
               'TURNING RIGHT ON RED':'AGRESSIVE/IMPROPER DRIVING',
               'RELATED TO BUS STOP': 'AGRESSIVE/IMPROPER DRIVING',
               'EXCEEDING AUTHORIZED SPEED LIMIT': 'AGRESSIVE/IMPROPER DI
```

```python
df['PRIM_CONTRIBUTORY_CAUSE'] = df['PRIM_CONTRIBUTORY_CAUSE'].map(agress_
df['PRIM_CONTRIBUTORY_CAUSE'].value_counts()
```

Out[87]:

```
AGRESSIVE/IMPROPER DRIVING
54895
FOLLOWING TOO CLOSELY
18186
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
4915
DISREGARDING TRAFFIC SIGNALS
3858
WEATHER
3030
DISREGARDING STOP SIGN
2185
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESS
IVE MANNER      1669
DISTRACTION - FROM INSIDE VEHICLE
1413
EQUIPMENT - VEHICLE CONDITION
1335
PHYSICAL CONDITION OF DRIVER
1219
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
1189
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
1181
DISTRACTION - FROM OUTSIDE VEHICLE
853
DRIVING ON WRONG SIDE/WRONG WAY
824
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
573
DISREGARDING OTHER TRAFFIC SIGNS
422
ROAD CONSTRUCTION/MAINTENANCE
413
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
354
CELL PHONE USE OTHER THAN TEXTING
275
DISREGARDING ROAD MARKINGS
229
EXCEEDING SAFE SPEED FOR CONDITIONS
222
ANIMAL
173
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
172
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER,
ETC.)           104
DISREGARDING YIELD SIGN
82
```

```
TEXTING
70
OBSTRUCTED CROSSWALKS
32
PASSING STOPPED SCHOOL BUS
17
BICYCLE ADVANCING LEGALLY ON RED LIGHT
11
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
8
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
```

In [88]:

```
irresp_dict = {'OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGI
               'EQUIPMENT - VEHICLE CONDITION': 'IRRESPONSIBLE BEHAVIOR'
               'UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS
               'PHYSICAL CONDITION OF DRIVER':'IRRESPONSIBLE BEHAVIOR',
               'CELL PHONE USE OTHER THAN TEXTING': 'IRRESPONSIBLE BEHAVI
               'HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)':'IRRESPO
               'DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE
               'TEXTING': 'IRRESPONSIBLE BEHAVIOR', 'PASSING STOPPED SCHO
               'DISTRACTION - FROM INSIDE VEHICLE': 'IRRESPONSIBLE BEHAVI
               'DISREGARDING TRAFFIC SIGNALS': 'IRRESPONSIBLE BEHAVIOR',
               'DISREGARDING STOP SIGN': 'IRRESPONSIBLE BEHAVIOR',
               'DISREGARDING OTHER TRAFFIC SIGNS': 'IRRESPONSIBLE BEHAVIO
               'DISREGARDING YIELD SIGN': 'IRRESPONSIBLE BEHAVIOR',
               'DISREGARDING ROAD MARKINGS': 'IRRESPONSIBLE BEHAVIOR',
               'FOLLOWING TOO CLOSELY': 'IRRESPONSIBLE BEHAVIOR',
               'DRIVING ON WRONG SIDE/WRONG WAY': 'IRRESPONSIBLE BEHAVIOR
               'TURNING RIGHT ON RED':'IRRESPONSIBLE BEHAVIOR'}

df['PRIM_CONTRIBUTORY_CAUSE'] = df['PRIM_CONTRIBUTORY_CAUSE'].map(irresp_
df['PRIM_CONTRIBUTORY_CAUSE'].value_counts()
```

Out[88]:

```
AGRESSIVE/IMPROPER DRIVING                               54895
IRRESPONSIBLE BEHAVIOR                                   33241
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE                       4915
WEATHER                                                   3030
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)      1189
DISTRACTION - FROM OUTSIDE VEHICLE                         853
ROAD ENGINEERING/SURFACE/MARKING DEFECTS                   573
ROAD CONSTRUCTION/MAINTENANCE                              413
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST          354
EXCEEDING SAFE SPEED FOR CONDITIONS                        222
ANIMAL                                                     173
OBSTRUCTED CROSSWALKS                                       32
BICYCLE ADVANCING LEGALLY ON RED LIGHT                      11
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT                    8
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
```

In [89]:
```
external_dict = {'DRIVING SKILLS/KNOWLEDGE/EXPERIENCE': 'EXTERNAL/OTHER F
                 'WEATHER': 'EXTERNAL/OTHER FACTORS',
                 'VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)':
                 'DISTRACTION - FROM OUTSIDE VEHICLE': 'EXTERNAL/OTHER FA
                 'ROAD ENGINEERING/SURFACE/MARKING DEFECTS': 'EXTERNAL/OT
                 'ROAD CONSTRUCTION/MAINTENANCE': 'EXTERNAL/OTHER FACTORS
                 'EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST': 'EX
                 'EXCEEDING SAFE SPEED FOR CONDITIONS': 'EXTERNAL/OTHER F
                 'ANIMAL': 'EXTERNAL/OTHER FACTORS',
                 'OBSTRUCTED CROSSWALKS': 'EXTERNAL/OTHER FACTORS',
                 'BICYCLE ADVANCING LEGALLY ON RED LIGHT': 'EXTERNAL/OTHE
                 'MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT': 'EXTERNAL/O

df['PRIM_CONTRIBUTORY_CAUSE'] = df['PRIM_CONTRIBUTORY_CAUSE'].map(externa
df['PRIM_CONTRIBUTORY_CAUSE'].value_counts()
```

Out[89]:
```
AGRESSIVE/IMPROPER DRIVING    54895
IRRESPONSIBLE BEHAVIOR        33241
EXTERNAL/OTHER FACTORS        11773
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: int64
```

In [90]:
```
df.PRIM_CONTRIBUTORY_CAUSE.value_counts(normalize = True)
```

Out[90]:
```
AGRESSIVE/IMPROPER DRIVING    0.549450
IRRESPONSIBLE BEHAVIOR        0.332713
EXTERNAL/OTHER FACTORS        0.117837
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: float64
```

Now the target variable is classified into three categories with following ratios:
Agressive/Improper Driving - 55%, Irresponsible Behavior - 34% and External Causes - 11%

Drop Needless Columns

In [91]:
```
df.drop(columns = ['LOCATION', 'STREET_NAME','STREET_DIRECTION','STREET_I
                   'RD_NO','BEAT_OF_OCCURRENCE', 'NUM_UNITS', 'VEHICLE_II
                   'AIRBAG_DEPLOYED','EJECTION', 'DATE_POLICE_NOTIFIED']
```

In [92]:
```
ed.show_info(df)
```

```
Lenght of Dataset: 99909
                                      missing_values_% Data_type
CRASH_RECORD_ID                                    0.0    object
CRASH_YEAR                                         0.0     int64
CRASH_MONTH                                        0.0     int64
CRASH_HOUR                                         0.0     int64
CRASH_DAY_OF_WEEK                                  0.0     int64
POSTED_SPEED_LIMIT                                 0.0     int64
TRAFFIC_CONTROL_DEVICE                             0.0    object
DEVICE_CONDITION                                   0.0    object
WEATHER_CONDITION                                  0.0    object
LIGHTING_CONDITION                                 0.0    object
FIRST_CRASH_TYPE                                   0.0    object
TRAFFICWAY_TYPE                                    0.0    object
STRAIGHT_ALIGNMENT                                 0.0     int64
GOOD_ROADWAY_SUFACE                                0.0     int64
ROAD_DEFECT                                        0.0     int64
DESK_REPORT_TYPE                                   0.0     int64
CRASH_TYPE                                         0.0    object
DAMAGE                                             0.0    object
PRIM_CONTRIBUTORY_CAUSE                            0.0    object
MOST_SEVERE_INJURY                                 0.0    object
INJURIES_TOTAL                                     0.0     int64
INJURIES_FATAL                                     0.0     int64
INJURIES_INCAPACITATING                            0.0     int64
INJURIES_NON_INCAPACITATING                        0.0     int64
INJURIES_REPORTED_NOT_EVIDENT                      0.0     int64
INJURIES_NO_INDICATION                             0.0     int64
INJURIES_UNKNOWN                                   0.0     int64
LATITUDE                                           0.0    object
LONGITUDE                                          0.0    object
PERSON_TYPE                                        0.0    object
MALE_PERSON                                        0.0     int64
AGE                                                0.0     int64
DRIVERS_LICENSE_STATE                              0.0    object
SAFETY_EQUIPMENT                                   0.0    object
DRIVER_ACTION                                      0.0    object
DRIVER_VISION                                      0.0    object
PHYSICAL_CONDITION                                 0.0    object
BAC_RESULT                                         0.0    object
```

Modeling

Since all of the cleaning and explorations were performed, now I will start building models.

Train - Test Split

To prevent models from **overfitting** and to be able to **accurately evaluate** model I will split the data to **X as a features and y as a target variable.**

In [93]:
```
df.drop(columns = ['CRASH_RECORD_ID', 'PERSON_TYPE', 'DRIVERS_LICENSE_STA
                   'SAFETY_EQUIPMENT', 'LATITUDE', 'LONGITUDE'], inplace
```

In [94]:
```
X = df.drop("PRIM_CONTRIBUTORY_CAUSE", axis=1)
y = df['PRIM_CONTRIBUTORY_CAUSE']
```

Now, split the data to **test - train sets** and then **split the train set into test and train sets too.**

In [95]:
```
X_all, X_hold, y_all, y_hold = train_test_split(X, y, random_state=2021)
```

In [96]:
```
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, random_
```

Let's check **how balanced is** my **target classes:**

In [97]:
```
pd.Series(y_train).value_counts(normalize=True)
```

Out[97]:
```
AGRESSIVE/IMPROPER DRIVING    0.549521
IRRESPONSIBLE BEHAVIOR        0.333250
EXTERNAL/OTHER FACTORS        0.117228
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: float64
```

Looking at the ratios, I can say that the **data is mostly imbalanced.** Thus, I will use **SMOTE to resample training sets.**

Model 1: Logistic Regression

The Baseline Model I chose was Logistic Regression Classifier.

ColumnTransformer for Categorical Features

Before I start any modeling, I will use **ColumnTransformer** and **OneHotEncode** all categorical features of the dataframe:

In [98]:
```
cols = X_train.select_dtypes(include='object').columns
indices = []
for col in cols:
    indices.append(X_train.columns.get_loc(col))
indices
```

Out[98]: `[5, 6, 7, 8, 9, 10, 15, 16, 17, 27, 28, 29, 30]`

In [99]:
```
transformer = ColumnTransformer(transformers=[('categorical', OneHotEnco
```

Build a new **pipeline** with transformer and perform the **cross-validation:**

In [100]:
```
categ_pipeline = make_pipeline(transformer, StandardScaler(with_mean = Fa
                               LogisticRegression(multi_class='multinomia
                                                  C=0.01, random_state=2(
```

In [101]:
```
history = ModelHistory()
history.report(categ_pipeline,X_train, y_train, 'Logistic Regression - mu
               'Regression with Numeric and Categorical Features')
```

```
Average Score: 0.7279618560311703
```

Out[101]:
```
array([0.72384342, 0.73469751, 0.7297153 , 0.72224199, 0.72793594,
       0.73078292, 0.72313167, 0.73096085, 0.73055704, 0.72575191])
```

Model Results: The average **accuracy score for the Model 1 is 0.7254.**

In [102]:
```
categ_pipeline.fit(X_train, y_train)
categ_pipeline.score(X_test, y_test)
```

Out[102]: `0.7233758607804409`

Predict on test set.

In [103]:
```
train_preds = categ_pipeline.predict(X_train)
test_preds = categ_pipeline.predict(X_test)
```

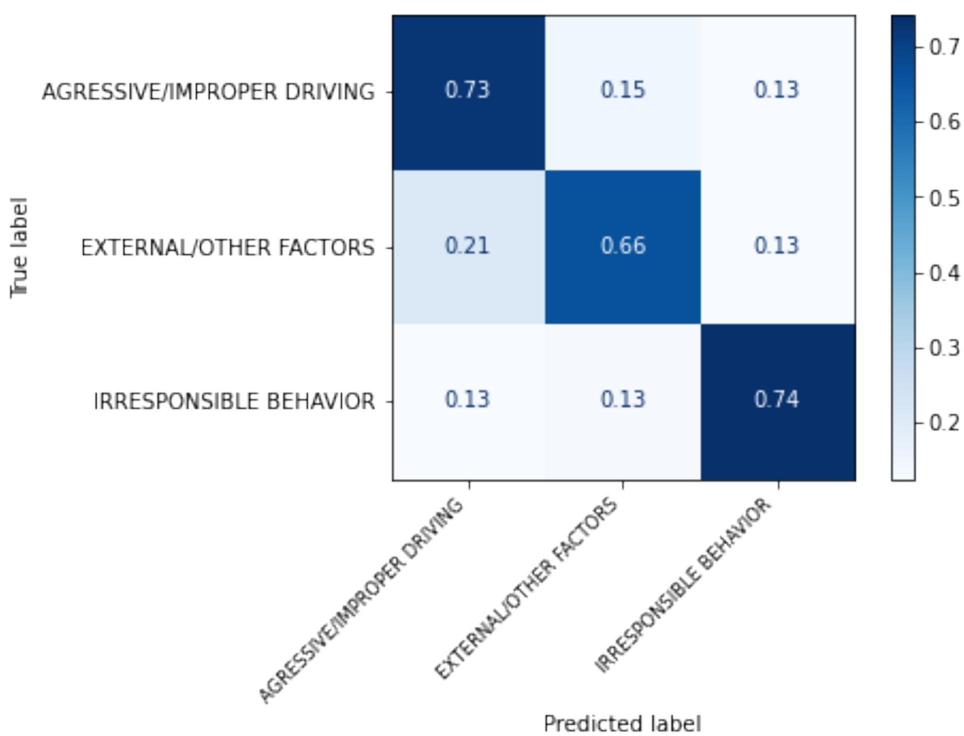
Check the Metrics of the Model 1:

In [104]:
```
mf.print_metrics(y_train,train_preds)
mf.print_metrics(y_test,test_preds)
```

```
Precision Score: 0.7674185856581323
Recall Score: 0.7296167123385174
Accuracy Score: 0.7296167123385174
F1 Score: 0.7416141958070329


Precision Score: 0.7623146336194836
Recall Score: 0.7233758607804409
Accuracy Score: 0.7233758607804409
F1 Score: 0.7357184668940655
```

In [105]:
```
plt.figure(figsize =(10,10))
plot_confusion_matrix(categ_pipeline, X_test, y_test,
                      cmap=plt.cm.Blues,normalize='true')

plt.xticks(rotation=45, horizontalalignment='right', fontsize='small')
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



Model 1 Results:

Based on the **Scores of the Training and Test Sets**, there is **no overfitting** - The Accuracy Score for **Training Model is 0.7266,** while the same score for **Testing Model is 0.7284.** In terms of confusion matrix, this model performed much better, considering **the diagonal of true values is high.**

Model 2: KNN with All Features

In order to see if the metrics can be improved and coefficients of the matrix be higher, I will build some other models with different classifiers.

In [106]:
```
cat_knn_pipeline = make_pipeline(transformer, StandardScaler(with_mean =
```

In [107]:
```
history.report(cat_knn_pipeline,X_train, y_train, 'KNN - Defaults',
               'KNN with All Features')
```

```
Average Score: 0.6818394947493223
```

Out[107]:
```
array([0.66530249, 0.68131673, 0.67580071, 0.68078292, 0.68042705,
       0.68398577, 0.68647687, 0.68469751, 0.68891262, 0.69069229])
```

In [108]:
```
cat_knn_pipeline.fit(X_train, y_train)
cat_knn_pipeline.score(X_train, y_train)
```

Out[108]:
```
0.7824833623972384
```

In [109]:
```
cat_knn_pipeline.score(X_test, y_test)
```

Out[109]:
```
0.6804569476325202
```

Result: The **Recall Score for Test Set is 0.4493.**

Predict on trainig and test sets.

In [110]:
```
train_preds = cat_knn_pipeline.predict(X_train)
test_preds = cat_knn_pipeline.predict(X_test)
```

Check the Metrics of the Model 2:

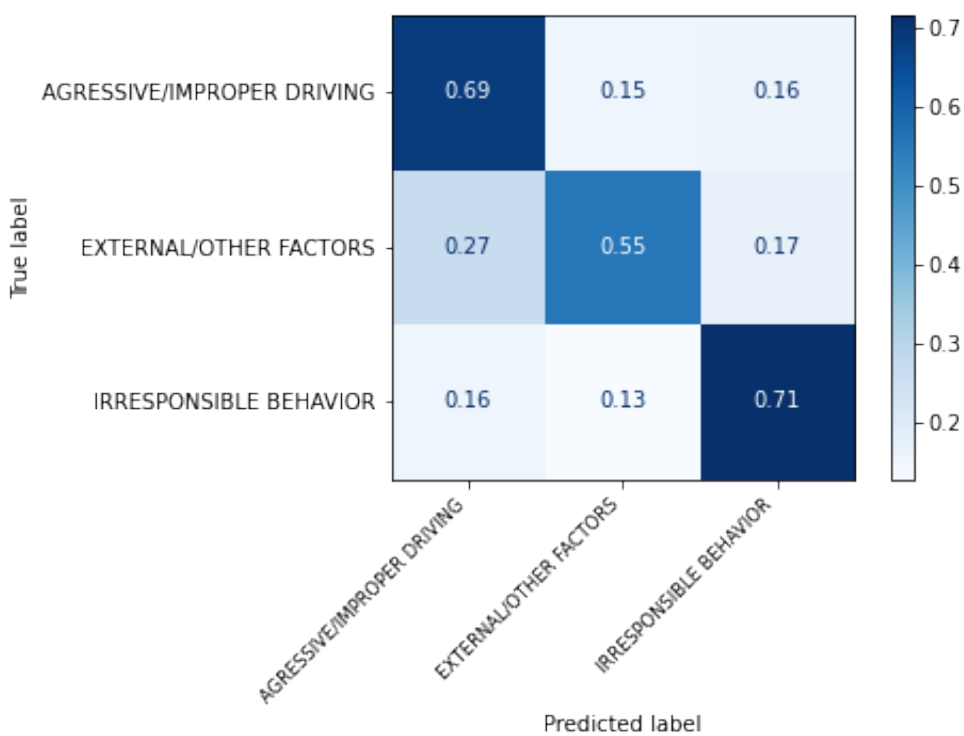
In [111]:
```
mf.print_metrics(y_train,train_preds)
mf.print_metrics(y_test,test_preds)
```

```
Precision Score: 0.8093566432227756
Recall Score: 0.7824833623972384
Accuracy Score: 0.7824833623972384
F1 Score: 0.7890716448831521


Precision Score: 0.7167995257222176
Recall Score: 0.6804569476325202
Accuracy Score: 0.6804569476325202
F1 Score: 0.6925871042884398
```

Confusion Matrix for Model 2:

In [112]:
```
plt.figure(figsize =(10,10))
plot_confusion_matrix(cat_knn_pipeline, X_test, y_test,
                      cmap=plt.cm.Blues,normalize='true')

plt.xticks(rotation=45, horizontalalignment='right', fontsize='small')
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



Model 4 Results:

- The model is **overfitting trainig data.**
- Confusion Matrix shows **worse results than for Logistic Regression.**

Model 3: Decision Tree Classifier

Since the KNN model showed results worse than Logistic Regression, I will build another model with Decision Tree Classifier.

In [113]:
```
tree_pipeline = make_pipeline (transformer, StandardScaler(with_mean = Fa
                              DecisionTreeClassifier(random_state = 202
```

In [114]:
```
history.report(tree_pipeline,X_train, y_train, 'DescisionTree',
               'Tree with Max-Depth = 10')
```

```
Average Score: 0.7139576164753673
```

Out[114]:
```
array([0.72241993, 0.7133452 , 0.73558719, 0.71405694, 0.71743772,
       0.70106762, 0.71014235, 0.70747331, 0.69709913, 0.72094679])
```

In [115]:
```
tree_pipeline.fit(X_train, y_train)
tree_pipeline.score(X_train, y_train)
```

Out[115]:
```
0.719545179543756
```

In [116]:
```
tree_pipeline.score(X_test, y_test)
```

Out[116]:
```
0.704852399508888
```

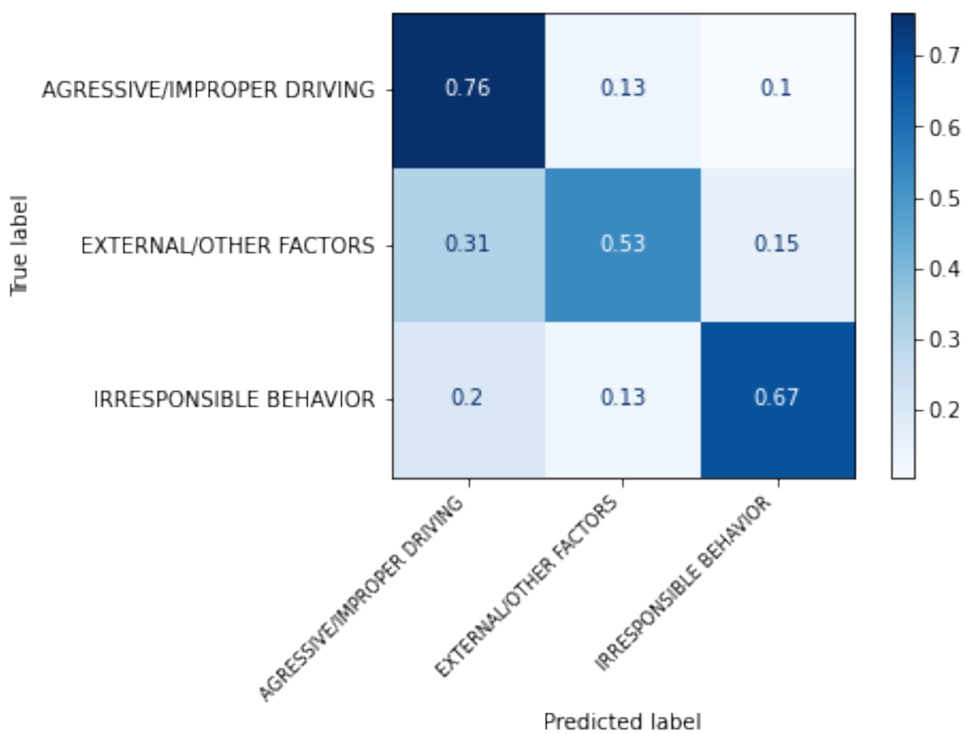
In [117]:
```
train_preds = tree_pipeline.predict(X_train)
test_preds = tree_pipeline.predict(X_test)
```

In [118]:
```
mf.print_metrics(y_train,train_preds)
mf.print_metrics(y_test,test_preds)
```

```
Precision Score: 0.7427757743888319
Recall Score: 0.719545179543756
Accuracy Score: 0.719545179543756
F1 Score: 0.7281092223336197


Precision Score: 0.7294901917236224
Recall Score: 0.704852399508888
Accuracy Score: 0.704852399508888
F1 Score: 0.7141658893962056
```

In [119]:
```
plt.figure(figsize =(10,10))
plot_confusion_matrix(tree_pipeline, X_test, y_test,
                      cmap=plt.cm.Blues,normalize='true')

plt.xticks(rotation=45, horizontalalignment='right', fontsize='small')
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



Model 3 Results:

- The model is **not overfitting.**
- Confusion Matrix shows **better results than KNN**, but still **worse results than for Logistic Regression.**

Modeling Conclusion

After the evaluation of all models performance, I see that the best metrics and coefficients for confusion matrix is achieved by Logistic Regression Classifier. My final model will be Logistic Regression for the purpose of the project.

Evaluation of Final Model

Now I will perform final evaluation of the Logistic Regression Model with the data that wasn't used while building.

In [120]:
```
categ_pipeline.fit(X_all, y_all)

categ_pipeline.score(X_all, y_all)
```

Out[120]:
```
0.7273758524509215
```

In [121]:
```
categ_pipeline.score(X_hold, y_hold)
```

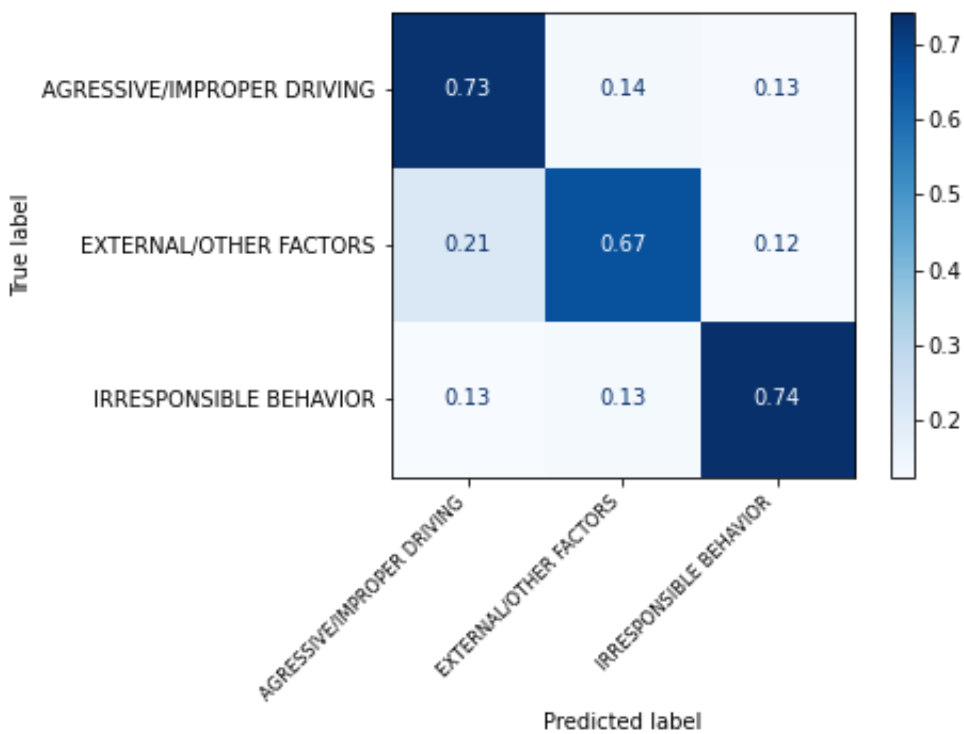
Out[121]:
```
0.7285210985667387
```

In [122]:
```
train_preds = categ_pipeline.predict(X_all)
test_preds = categ_pipeline.predict(X_hold)
mf.print_metrics(y_all,train_preds)
mf.print_metrics(y_hold,test_preds)
```

```
Precision Score: 0.7656607379774959
Recall Score: 0.7273758524509215
Accuracy Score: 0.7273758524509215
F1 Score: 0.7394806784132365


Precision Score: 0.7661430157301438
Recall Score: 0.7285210985667387
Accuracy Score: 0.7285210985667387
F1 Score: 0.7405156892719819
```

In [123]:
```
plt.figure(figsize =(10,10))
plot_confusion_matrix(categ_pipeline, X_hold, y_hold,
                      cmap=plt.cm.Blues,normalize='true')

plt.xticks(rotation=45, horizontalalignment='right', fontsize='small')
plt.show()
```

<Figure size 720x720 with 0 Axes>



Results: The model performance on the unseen data is good. The metrics are high and the confusion matrix has shown that model is able to predict 73% of true values for Agressive/Improper Driving, 67% for External/Other Causes and 74% of Irresponsible Behavior causes.

Conclusion

Based on the analysis and the modeling results the following actions would be appropriate to consider in order to reduce number of traffic accidents:

- Promotion of less aggressive driving
- Promotion of drivers’ safety, especially during traffic time and winter season, such that regulate the roads with hight traffic volume and take actions against the road conditions during winter.
- Assuring enough of driving experience and safety preparations of the drivers with age range of 20 - 30.