1 Predicting Contributory Causes to Chicago Traffic Accidents

2 Overview

In this project, I inspect a dataset covering traffic accidents in the city of Chicago, IL (http://www.chicago.gov (http://www.chicago.gov)) and construct a classifier that predicts the primary cause of the accident.

3 Business Problem

The City of Chicago is always looking to improve the safety of their roads for both drivers and pedestrians. Over the last couple of years, the number of on-street traffic deaths in Chicago has jumped from 96 in 2019 to 136 in 2020.

Concerned by this increase, the city has hired a team of data scientists to inspect data sourced from police reports. Because accidents are attributable to many causes, one of the goals of the team is to find patterns between the underlying conditions of a particular crash and its primary cause.

With a clearer insight on what factors are most relevant to predicting a crash's primary cause, the City of Chicago hopes to use this project's findings to make their driver safety initiatives more effective.

4 Importing Data, Necessary Libraries

The data in this project is provided by the city of Chicago, IL (https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if (https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if)) & sourced from reports by the city's police department.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [22]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score, Repe
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import precision_score, recall_score, accuracy_score,
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.utils import resample
```

In [3]:
```
pd.set_option('display.max_columns', None)
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
df_crash = pd.read_csv('Data/Traffic_Crashes_-_Crashes.csv')
```

5 Initial Inspection & Cleanup of Crash Data

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 491197 entries, 0 to 491196
Data columns (total 49 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   CRASH_RECORD_ID                491197 non-null  object 
 1   RD_NO                          487508 non-null  object 
 2   CRASH_DATE_EST_I               36924 non-null   object 
 3   CRASH_DATE                     491197 non-null  object 
 4   POSTED_SPEED_LIMIT             491197 non-null  int64  
 5   TRAFFIC_CONTROL_DEVICE         491197 non-null  object 
 6   DEVICE_CONDITION               491197 non-null  object 
 7   WEATHER_CONDITION              491197 non-null  object 
 8   LIGHTING_CONDITION             491197 non-null  object 
 9   FIRST_CRASH_TYPE               491197 non-null  object 
 10  TRAFFICWAY_TYPE                491197 non-null  object 
 11  LANE_CNT                       198965 non-null  float64
 12  ALIGNMENT                      491197 non-null  object 
 13  ROADWAY_SURFACE_COND           491197 non-null  object 
 14  ROAD_DEFECT                    491197 non-null  object 
 15  REPORT_TYPE                    479198 non-null  object 
 16  CRASH_TYPE                     491197 non-null  object 
 17  INTERSECTION_RELATED_I         110843 non-null  object 
 18  NOT_RIGHT_OF_WAY_I             23159 non-null   object 
 19  HIT_AND_RUN_I                  145010 non-null  object 
 20  DAMAGE                         491197 non-null  object 
 21  DATE_POLICE_NOTIFIED           491197 non-null  object 
 22  PRIM_CONTRIBUTORY_CAUSE        491197 non-null  object 
 23  SEC_CONTRIBUTORY_CAUSE         491197 non-null  object 
 24  STREET_NO                      491197 non-null  int64  
 25  STREET_DIRECTION               491194 non-null  object 
 26  STREET_NAME                    491196 non-null  object 
 27  BEAT_OF_OCCURRENCE             491192 non-null  float64
 28  PHOTOS_TAKEN_I                 6170 non-null    object 
 29  STATEMENTS_TAKEN_I             9917 non-null    object 
 30  DOORING_I                      1563 non-null    object 
 31  WORK_ZONE_I                    3155 non-null    object 
 32  WORK_ZONE_TYPE                 2487 non-null    object 
 33  WORKERS_PRESENT_I              758 non-null     object 
 34  NUM_UNITS                      491197 non-null  int64  
 35  MOST_SEVERE_INJURY             490200 non-null  object 
 36  INJURIES_TOTAL                 490211 non-null  float64
 37  INJURIES_FATAL                 490211 non-null  float64
 38  INJURIES_INCAPACITATING        490211 non-null  float64
 39  INJURIES_NON_INCAPACITATING    490211 non-null  float64
 40  INJURIES_REPORTED_NOT_EVIDENT  490211 non-null  float64
 41  INJURIES_NO_INDICATION         490211 non-null  float64
 42  INJURIES_UNKNOWN               490211 non-null  float64
 43  CRASH_HOUR                     491197 non-null  int64  
 44  CRASH_DAY_OF_WEEK              491197 non-null  int64  
 45  CRASH_MONTH                    491197 non-null  int64  
 46  LATITUDE                       488458 non-null  float64
 47  LONGITUDE                      488458 non-null  float64
 48  LOCATION                       488458 non-null  object 
```

```
dtypes: float64(11), int64(6), object(32)
memory usage: 183.6+ MB
```

5.1 Dropping Columns From df_crash

Because this project is concerned with the conditions immediately surrounding an auto accident, I drop all columns that pertain to the police reports generated after the crash.

I also drop some columns that are almost entirely null values, like *DOORING_I* & *WORKERS_PRESENT_I*, since I'm not interesting in building a model using features made mostly of imputed values.

Finally, I drop the *LATITUTDE* & *LONGITUDE* columns due to time constraints, and the *CRASH_DATE* column because I'd prefer to focus more closely on the existing *CRASH_HOUR*, *CRASH_DAY_OF_WEEK* & *CRASH_MONTH* columns.

In [6]:
```
to_drop = ['RD_NO', 'CRASH_DATE_EST_I', 'REPORT_TYPE', 'DATE_POLICE_NOTIFIE
           'STREET_NO', 'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRE
          'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'SEC_CONTRIBUTORY_CAUSE',
          'DOORING_I', 'WORKERS_PRESENT_I', 'LATITUDE', 'LONGITUDE', 'CRASH

df_crash.drop(columns=to_drop, axis=1, inplace=True)
```

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

Out[7]:

	CRASH_RECORD_ID	POSTED_SPEED_LIMIT	TRAFFIC_CONTRO
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	35	N
1	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	35	STOP SI
2	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	30	TRA
3	f8960f698e870ebdc60b521b2a141a5395556bc3704191...	30	N
4	8eaa2678d1a127804ee9b8c35ddf7d63d913c14eda61d6...	20	N

5.2 Filtering Out Crashes With 'Unable to Determine' & 'Not Applicable' Primary Causes

Neither of these values for our target are useful, so I filter any corresponding entries out.

In [8]:
```
df_crash['PRIM_CONTRIBUTORY_CAUSE'].value_counts(normalize=True)
```

Out[8]:
```
UNABLE TO DETERMINE
0.370397
FAILING TO YIELD RIGHT-OF-WAY
0.109856
FOLLOWING TOO CLOSELY
0.105823
NOT APPLICABLE
0.053665
IMPROPER OVERTAKING/PASSING
0.047482
IMPROPER BACKING
0.043773
FAILING TO REDUCE SPEED TO AVOID CRASH
0.043127
IMPROPER LANE USAGE
0.038573
IMPROPER TURNING/NO SIGNAL
0.033139
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
0.031275
DISREGARDING TRAFFIC SIGNALS
0.018178
WEATHER
0.017317
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE
MANNER    0.012486
DISREGARDING STOP SIGN
0.011046
DISTRACTION - FROM INSIDE VEHICLE
0.007317
EQUIPMENT - VEHICLE CONDITION
0.006272
PHYSICAL CONDITION OF DRIVER
0.005875
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
0.005839
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
0.005332
DRIVING ON WRONG SIDE/WRONG WAY
0.004713
DISTRACTION - FROM OUTSIDE VEHICLE
0.004410
EXCEEDING AUTHORIZED SPEED LIMIT
0.004035
EXCEEDING SAFE SPEED FOR CONDITIONS
0.003428
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
0.002816
ROAD CONSTRUCTION/MAINTENANCE
0.002415
DISREGARDING OTHER TRAFFIC SIGNS
0.002134
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
0.001861
CELL PHONE USE OTHER THAN TEXTING
```

```
0.001399
DISREGARDING ROAD MARKINGS
0.001376
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
0.001107
ANIMAL
0.000841
TURNING RIGHT ON RED
0.000700
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER, ET
C.)           0.000472
TEXTING
0.000438
DISREGARDING YIELD SIGN
0.000381
RELATED TO BUS STOP
0.000334
BICYCLE ADVANCING LEGALLY ON RED LIGHT
0.000134
PASSING STOPPED SCHOOL BUS
0.000130
OBSTRUCTED CROSSWALKS
0.000065
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
0.000039
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: float64
```

In [9]:
```
df_crash = df_crash[df_crash['PRIM_CONTRIBUTORY_CAUSE'] != 'UNABLE TO DETER
```

In [10]:
```
df_crash = df_crash[df_crash['PRIM_CONTRIBUTORY_CAUSE'] != 'NOT APPLICABLE'
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 282899 entries, 0 to 491195
Data columns (total 32 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   CRASH_RECORD_ID                282899 non-null  object 
 1   POSTED_SPEED_LIMIT             282899 non-null  int64  
 2   TRAFFIC_CONTROL_DEVICE         282899 non-null  object 
 3   DEVICE_CONDITION               282899 non-null  object 
 4   WEATHER_CONDITION              282899 non-null  object 
 5   LIGHTING_CONDITION             282899 non-null  object 
 6   FIRST_CRASH_TYPE               282899 non-null  object 
 7   TRAFFICWAY_TYPE                282899 non-null  object 
 8   LANE_CNT                       122317 non-null  float64
 9   ALIGNMENT                      282899 non-null  object 
 10  ROADWAY_SURFACE_COND           282899 non-null  object 
 11  ROAD_DEFECT                    282899 non-null  object 
 12  CRASH_TYPE                     282899 non-null  object 
 13  INTERSECTION_RELATED_I         79490 non-null   object 
 14  NOT_RIGHT_OF_WAY_I             11985 non-null   object 
 15  HIT_AND_RUN_I                  62666 non-null   object 
 16  DAMAGE                         282899 non-null  object 
 17  PRIM_CONTRIBUTORY_CAUSE        282899 non-null  object 
 18  WORK_ZONE_I                    2194 non-null    object 
 19  WORK_ZONE_TYPE                 1779 non-null    object 
 20  NUM_UNITS                      282899 non-null  int64  
 21  MOST_SEVERE_INJURY             282641 non-null  object 
 22  INJURIES_TOTAL                 282644 non-null  float64
 23  INJURIES_FATAL                 282644 non-null  float64
 24  INJURIES_INCAPACITATING        282644 non-null  float64
 25  INJURIES_NON_INCAPACITATING    282644 non-null  float64
 26  INJURIES_REPORTED_NOT_EVIDENT  282644 non-null  float64
 27  INJURIES_NO_INDICATION         282644 non-null  float64
 28  INJURIES_UNKNOWN               282644 non-null  float64
 29  CRASH_HOUR                     282899 non-null  int64  
 30  CRASH_DAY_OF_WEEK              282899 non-null  int64  
 31  CRASH_MONTH                    282899 non-null  int64  
dtypes: float64(8), int64(5), object(19)
memory usage: 71.2+ MB
```

5.3 Dealing With Null Values

From the results below, it is clear that significant imputation will have to be done for a handful of columns in our dataset. Since there are currently hundreds of thousands of entries, I am comfortable dropping the relatively small amount (~255) of entries that have nulls in all of the *INJURIES* columns.

As for the remaining columns, I am going to have to make judgments on a case-by-case basis.

In [12]:
```
df_crash.isnull().sum()
```

Out[12]:
```
CRASH_RECORD_ID                         0
POSTED_SPEED_LIMIT                      0
TRAFFIC_CONTROL_DEVICE                  0
DEVICE_CONDITION                        0
WEATHER_CONDITION                       0
LIGHTING_CONDITION                      0
FIRST_CRASH_TYPE                        0
TRAFFICWAY_TYPE                         0
LANE_CNT                           160582
ALIGNMENT                               0
ROADWAY_SURFACE_COND                    0
ROAD_DEFECT                             0
CRASH_TYPE                              0
INTERSECTION_RELATED_I             203409
NOT_RIGHT_OF_WAY_I                 270914
HIT_AND_RUN_I                      220233
DAMAGE                                  0
PRIM_CONTRIBUTORY_CAUSE                 0
WORK_ZONE_I                        280705
WORK_ZONE_TYPE                     281120
NUM_UNITS                               0
MOST_SEVERE_INJURY                    258
INJURIES_TOTAL                        255
INJURIES_FATAL                        255
INJURIES_INCAPACITATING               255
INJURIES_NON_INCAPACITATING           255
INJURIES_REPORTED_NOT_EVIDENT         255
INJURIES_NO_INDICATION                255
INJURIES_UNKNOWN                      255
CRASH_HOUR                              0
CRASH_DAY_OF_WEEK                       0
CRASH_MONTH                             0
dtype: int64
```

First, an inspection of the values for each column containing nulls:

In [13]:
```
has_nulls = ['LANE_CNT', 'INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I',
             'HIT_AND_RUN_I', 'WORK_ZONE_I', 'WORK_ZONE_TYPE', 'MOST_SEVERE_
             'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITATING',
             'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT',
             'INJURIES_NO_INDICATION', 'INJURIES_UNKNOWN']

for col in has_nulls:
    print(f'Value Counts for {col}' + '\n')
    print(df_crash[col].value_counts(dropna=False, normalize=True))
    print('-----------------------' + '\n')
```

```
Value Counts for LANE_CNT

NaN         0.567630
2.0         0.197749
4.0         0.119926
1.0         0.053821
3.0         0.021106
0.0         0.016999
6.0         0.011414
5.0         0.005058
8.0         0.004850
7.0         0.000534
10.0        0.000322
99.0        0.000184
9.0         0.000148
11.0        0.000064
12.0        0.000057
22.0        0.000032
20.0        0.000028
```

All of the columns listed in *fill_null_n* consist of binary results Y & N. I elect to impute the missing values with the N or 'no' result. I am working under the belief that, if an officer is unable to write down an answer to a binary question at the scene, the real result is much more likely to be 'no' than 'yes.'

In [14]:
```
fill_null_n = ['INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN

for col in fill_null_n:
    df_crash[col].fillna('N', inplace=True)
```

In [15]:
```
df_crash['WORK_ZONE_TYPE'].fillna('NONE', inplace=True)
```

For the *LANE_CNT* column, I impute the missing values with the placeholder 'missing' value of 0 lanes. Though the values are still confusing for this feature, I am temporarily leaving it that way before tweaking it later on.

In [16]:
```
df_crash['LANE_CNT'].fillna(0.0, inplace=True)

df_crash['LANE_CNT'].value_counts(dropna=False, normalize=True)
```

Out[16]:
```
0.0         0.584629
2.0         0.197749
4.0         0.119926
1.0         0.053821
3.0         0.021106
6.0         0.011414
5.0         0.005058
8.0         0.004850
7.0         0.000534
10.0        0.000322
99.0        0.000184
9.0         0.000148
11.0        0.000064
12.0        0.000057
22.0        0.000032
20.0        0.000028
16.0        0.000018
14.0        0.000011
15.0        0.000011
30.0        0.000011
21.0        0.000007
44.0        0.000004
28.0        0.000004
41.0        0.000004
433634.0    0.000004
40.0        0.000004
60.0        0.000004
Name: LANE_CNT, dtype: float64
```

Now that I have dealt with all 6 non-*INJURY* columns, I proceed to drop all remaining nulls from the DataFrame.

In [17]:
```
df_crash.dropna(inplace=True)

df_crash.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 282641 entries, 0 to 491195
Data columns (total 32 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   CRASH_RECORD_ID                282641 non-null  object 
 1   POSTED_SPEED_LIMIT             282641 non-null  int64  
 2   TRAFFIC_CONTROL_DEVICE         282641 non-null  object 
 3   DEVICE_CONDITION               282641 non-null  object 
 4   WEATHER_CONDITION              282641 non-null  object 
 5   LIGHTING_CONDITION             282641 non-null  object 
 6   FIRST_CRASH_TYPE               282641 non-null  object 
 7   TRAFFICWAY_TYPE                282641 non-null  object 
 8   LANE_CNT                       282641 non-null  float64
 9   ALIGNMENT                      282641 non-null  object 
 10  ROADWAY_SURFACE_COND           282641 non-null  object 
 11  ROAD_DEFECT                    282641 non-null  object 
 12  CRASH_TYPE                     282641 non-null  object 
 13  INTERSECTION_RELATED_I         282641 non-null  object 
 14  NOT_RIGHT_OF_WAY_I             282641 non-null  object 
 15  HIT_AND_RUN_I                  282641 non-null  object 
 16  DAMAGE                         282641 non-null  object 
 17  PRIM_CONTRIBUTORY_CAUSE        282641 non-null  object 
 18  WORK_ZONE_I                    282641 non-null  object 
 19  WORK_ZONE_TYPE                 282641 non-null  object 
 20  NUM_UNITS                      282641 non-null  int64  
 21  MOST_SEVERE_INJURY             282641 non-null  object 
 22  INJURIES_TOTAL                 282641 non-null  float64
 23  INJURIES_FATAL                 282641 non-null  float64
 24  INJURIES_INCAPACITATING        282641 non-null  float64
 25  INJURIES_NON_INCAPACITATING    282641 non-null  float64
 26  INJURIES_REPORTED_NOT_EVIDENT  282641 non-null  float64
 27  INJURIES_NO_INDICATION         282641 non-null  float64
 28  INJURIES_UNKNOWN               282641 non-null  float64
 29  CRASH_HOUR                     282641 non-null  int64  
 30  CRASH_DAY_OF_WEEK              282641 non-null  int64  
 31  CRASH_MONTH                    282641 non-null  int64  
dtypes: float64(8), int64(5), object(19)
memory usage: 71.2+ MB
```

In [18]:
```
df_crash.reset_index(drop=True, inplace=True)
df_crash.head()
```

Out[18]:

	CRASH_RECORD_ID	POSTED_SPEED_LIMIT	TRAFFIC_CONTRC
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	35	N
1	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	35	STOP SI
2	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	30	TR/
3	f636d4a51a88015ac89031159b1f1952b8d92e49d11aeb...	30	N
4	9c974548026c1b962569040bd8fa08ae643ffc28c15ebd...	10	

6 Data Manipulation for Modeling

6.1 Creating Bins for Target Column, *PRIM_CONTRIBUTORY_CAUSE*

As seen immediately below, there are too many target results to make an effective classifier for. Therefore, I decide to bin the current values into the following categories:

- Outside Hazard: Accidents primarily caused by hazards or distractions that the driver or passenger(s) cannot control while in the vehicle.
- Impairment/Distraction: Accidents primarily caused by a driver's impairment or by a distraction within in the car (usually cell phones).
- Reckless Driving: Accidents primarily caused by a driver failing to follow commonly understood safe driving procedure.
- Ignoring Traffic Signs & Warnings: Accidents primarily caused by a driver failing to follow legal warnings, signs or signals posted on the road.

In [19]:
```
df_crash['PRIM_CONTRIBUTORY_CAUSE'].value_counts(normalize=True)
```

Out[19]:
```
FAILING TO YIELD RIGHT-OF-WAY
0.190914
FOLLOWING TOO CLOSELY
0.183880
IMPROPER OVERTAKING/PASSING
0.082518
IMPROPER BACKING
0.075983
FAILING TO REDUCE SPEED TO AVOID CRASH
0.074876
IMPROPER LANE USAGE
0.067021
IMPROPER TURNING/NO SIGNAL
0.057582
DRIVING SKILLS/KNOWLEDGE/EXPERIENCE
0.054277
DISREGARDING TRAFFIC SIGNALS
0.031588
WEATHER
0.029840
OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR AGGRESSIVE
MANNER    0.021628
DISREGARDING STOP SIGN
0.019194
DISTRACTION - FROM INSIDE VEHICLE
0.012702
EQUIPMENT - VEHICLE CONDITION
0.010752
PHYSICAL CONDITION OF DRIVER
0.010207
VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)
0.010144
UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)
0.009266
DRIVING ON WRONG SIDE/WRONG WAY
0.008173
DISTRACTION - FROM OUTSIDE VEHICLE
0.007646
EXCEEDING AUTHORIZED SPEED LIMIT
0.006974
EXCEEDING SAFE SPEED FOR CONDITIONS
0.005947
ROAD ENGINEERING/SURFACE/MARKING DEFECTS
0.004893
ROAD CONSTRUCTION/MAINTENANCE
0.004186
DISREGARDING OTHER TRAFFIC SIGNS
0.003708
EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST
0.003230
CELL PHONE USE OTHER THAN TEXTING
0.002431
DISREGARDING ROAD MARKINGS
0.002392
HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)
```

```
0.001911
ANIMAL
0.001461
TURNING RIGHT ON RED
0.001217
DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD PLAYER, ET
C.)          0.000821
TEXTING
0.000761
DISREGARDING YIELD SIGN
0.000658
RELATED TO BUS STOP
0.000580
BICYCLE ADVANCING LEGALLY ON RED LIGHT
0.000234
PASSING STOPPED SCHOOL BUS
0.000226
OBSTRUCTED CROSSWALKS
0.000113
MOTORCYCLE ADVANCING LEGALLY ON RED LIGHT
0.000067
Name: PRIM_CONTRIBUTORY_CAUSE, dtype: float64
```

In [20]:
```
def label_cause(row):
    out_hzd = ['WEATHER', 'EQUIPMENT - VEHICLE CONDITION',
           'VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)',
          'DISTRACTION - FROM OUTSIDE VEHICLE', 'ROAD ENGINEERING/SURFACE/M
          'EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOTORIST', 'ANIMAL']
    imp_dist = ['DISTRACTION - FROM INSIDE VEHICLE', 'PHYSICAL CONDITION OF
           'UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECT
           'CELL PHONE USE OTHER THAN TEXTING', 'HAD BEEN DRINKING (USE WHE
           'DISTRACTION - OTHER ELECTRONIC DEVICE (NAVIGATION DEVICE, DVD P
    reckless = ['FAILING TO YIELD RIGHT-OF-WAY', 'FOLLOWING TOO CLOSELY', '
           'IMPROPER BACKING', 'FAILING TO REDUCE SPEED TO AVOID CRASH', 'I
           'IMPROPER TURNING/NO SIGNAL', 'DRIVING SKILLS/KNOWLEDGE/EXPERIEN
           'OPERATING VEHICLE IN ERRATIC, RECKLESS, CARELESS, NEGLIGENT OR
           'DRIVING ON WRONG SIDE/WRONG WAY', 'EXCEEDING SAFE SPEED FOR CON

    if row in out_hzd:
        return 'Outside Hazard'
    if row in imp_dist:
        return 'Impairment/Distraction'
    if row in reckless:
        return 'Reckless Driving'
    else:
        return 'Ignoring Traffic Signs & Warnings'

df_crash['Primary Cause'] = df_crash['PRIM_CONTRIBUTORY_CAUSE'].apply(label

df_crash.head()
```

Out[20]:

	CRASH_RECORD_ID	POSTED_SPEED_LIMIT	TRAFFIC_CONTRO
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	35	N
1	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	35	STOP SI
2	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	30	TRA
3	f636d4a51a88015ac89031159b1f1952b8d92e49d11aeb...	30	N
4	9c974548026c1b962569040bd8fa08ae643ffc28c15ebd...	10	

After creating the new bin column, I drop the old column.

In [21]:
```
df_crash.drop('PRIM_CONTRIBUTORY_CAUSE', axis=1, inplace=True)
```

6.2 Dealing With Class Imbalance

Looking at the distribution of primary causes across the current dataset, there is a significant class imbalance issue. Namely, the fact that there are at least 10 times as many 'Reckless Driving' instances as there are of any other cause is likely to throw my classifier off. Therefore, I choose to undersample from the 'Reckless Driving' entries and proceed with the resulting dataset of about 70,000 entries for the remainder of the project.

In [23]:
```
df_crash['Primary Cause'].value_counts()
```

Out[23]:
```
Reckless Driving                       232557
Ignoring Traffic Signs & Warnings       20106
Outside Hazard                          19210
Impairment/Distraction                  10768
Name: Primary Cause, dtype: int64
```

In [24]:
```
outside = df_crash[df_crash['Primary Cause'] == 'Outside Hazard']
impair = df_crash[df_crash['Primary Cause'] == 'Impairment/Distraction']
reck = df_crash[df_crash['Primary Cause'] == 'Reckless Driving']
ignored = df_crash[df_crash['Primary Cause'] == 'Ignoring Traffic Signs & W
```

In [25]:
```
reck_downsampled = resample(reck, replace=False,
                             n_samples=len(ignored),
                            random_state = 26)

to_join = [reck_downsampled, impair, outside, ignored]
downsampled = pd.concat(to_join)
```

In [26]:
```
downsampled['Primary Cause'].value_counts()
```

Out[26]:
```
Ignoring Traffic Signs & Warnings      20106
Reckless Driving                       20106
Outside Hazard                         19210
Impairment/Distraction                 10768
Name: Primary Cause, dtype: int64
```

6.3 Feature Manipulation for Initial Model

When inspecting the head of the DataFrame, it becomes apparent that most of the features will need manipulation, especially since most of the columns consist of text. In this section, I look at each subgroup of features and manipulate each one individually before bringing everything together again at the end.

In [28]:
```
downsampled.reset_index(drop=True, inplace=True)
downsampled.head()
```

Out[28]:

	CRASH_RECORD_ID	POSTED_SPEED_LIMIT	TRAFFIC_CONTROI
0	27f3aa4bb36ec9e8f3149347071c0ea1cc1ed701b40ccf...	30	NC
1	d494fa51e643b56aea140c44dc223b614f35cf871acd60...	30	STOP SIC
2	15d994fad715893aa2a5f0ad1cf85104ce070b8d6d30a9...	30	TRAI
3	17d9cb117ec3e666e2b3f75d1182e286d999cf77914539...	30	TRAI
4	c968924a8f0c29f87186bb863a06c5847b8d848c5273b6...	0	STOP SIC

6.3.1 A Closer Look At Some Numeric Features

In [29]:
```
downsampled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70190 entries, 0 to 70189
Data columns (total 32 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   CRASH_RECORD_ID                70190 non-null  object 
 1   POSTED_SPEED_LIMIT             70190 non-null  int64  
 2   TRAFFIC_CONTROL_DEVICE         70190 non-null  object 
 3   DEVICE_CONDITION               70190 non-null  object 
 4   WEATHER_CONDITION              70190 non-null  object 
 5   LIGHTING_CONDITION             70190 non-null  object 
 6   FIRST_CRASH_TYPE               70190 non-null  object 
 7   TRAFFICWAY_TYPE                70190 non-null  object 
 8   LANE_CNT                       70190 non-null  float64
 9   ALIGNMENT                      70190 non-null  object 
 10  ROADWAY_SURFACE_COND           70190 non-null  object 
 11  ROAD_DEFECT                    70190 non-null  object 
 12  CRASH_TYPE                     70190 non-null  object 
 13  INTERSECTION_RELATED_I         70190 non-null  object 
 14  NOT_RIGHT_OF_WAY_I             70190 non-null  object 
 15  HIT_AND_RUN_I                  70190 non-null  object 
 16  DAMAGE                         70190 non-null  object 
 17  WORK_ZONE_I                    70190 non-null  object 
 18  WORK_ZONE_TYPE                 70190 non-null  object 
 19  NUM_UNITS                      70190 non-null  int64  
 20  MOST_SEVERE_INJURY             70190 non-null  object 
 21  INJURIES_TOTAL                 70190 non-null  float64
 22  INJURIES_FATAL                 70190 non-null  float64
 23  INJURIES_INCAPACITATING        70190 non-null  float64
 24  INJURIES_NON_INCAPACITATING    70190 non-null  float64
 25  INJURIES_REPORTED_NOT_EVIDENT  70190 non-null  float64
 26  INJURIES_NO_INDICATION         70190 non-null  float64
 27  INJURIES_UNKNOWN               70190 non-null  float64
 28  CRASH_HOUR                     70190 non-null  int64  
 29  CRASH_DAY_OF_WEEK              70190 non-null  int64  
 30  CRASH_MONTH                    70190 non-null  int64  
 31  Primary Cause                  70190 non-null  object 
dtypes: float64(8), int64(5), object(19)
memory usage: 17.1+ MB
```

In [30]:
```
num = ['POSTED_SPEED_LIMIT', 'LANE_CNT', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK',

for n in num:
    print(f'Unique values for column {n}' + '\n')
    print(downsampled[n].unique())
    print('\n')
```

```
Unique values for column POSTED_SPEED_LIMIT

[30  0 25 35 40 20 15 10 45  5  9  3 50 33 63 55 60 31 24 99 39 32  2 12
 34  1 70]


Unique values for column LANE_CNT

[ 2.  0.  4.  6.  1.  3.  8.  5.  7. 99. 10. 11.  9. 21. 16. 12. 22.]


Unique values for column CRASH_HOUR

[15 18 17  9 14  1 12 16 10  0  7 23 20  2  5 11  8 19 13  4 21 22  3  6]


Unique values for column CRASH_DAY_OF_WEEK

[5 2 3 7 1 4 6]


Unique values for column CRASH_MONTH

[ 9  4  2  8  1 12 10 11  7  6  5  3]
```

First, I address the many nonsensical values for the *LANE_CNT* column. My choice is to treat the column as a categorical, with a bin for each sensible one-way lane count (1,2,3 or 4) and a bin for all remaining values which will take the label "Missing."

In [31]:
```
lane_dict = {1.0: '1', 2.0: '2', 3.0: '3', 4.0: '4',
             0.0: 'Missing', 5.0: 'Missing', 6.0: 'Missing',
             7.0: 'Missing', 8.0: 'Missing', 9.0: 'Missing',
             10.0: 'Missing', 11.0: 'Missing', 12.0: 'Missing',
             16.0: 'Missing', 21.0: 'Missing', 22.0: 'Missing',
             99.0: 'Missing'}

downsampled['LANE_CNT'] = downsampled['LANE_CNT'].map(lane_dict).astype('ob
downsampled['LANE_CNT'].value_counts(normalize=True)
```

Out[31]:
```
Missing    0.612822
2          0.200940
4          0.109930
1          0.057487
3          0.018820
Name: LANE_CNT, dtype: float64
```

Now, looking at the *POSTED_SPEED_LIMIT* column, I elect to keep the column numerical. Instead of binning, I remap the column so that every data point is assigned a value between 15 & 70 in increments of 5 (i.e. 15mph, 20mph, 25mph,..., 65mph, 70mph), based on personal interpretations of the unconventional values.

The new column is much more representative of common US speed limits. Additionally, over 95% of the data already has a value of 15, 20, 25, 30, 35 or 40, so this edit isn't a significant change for the vast majority of the entries.

In [32]:
```
speed_dict = {0: 15, 1: 15, 2: 20, 3: 30,
              5: 50, 9: 15, 10: 15, 12: 15,
              24: 25, 31: 30, 32: 30, 33: 35,
              34: 35, 39: 40, 63: 65, 99: 30,
              15: 15, 20: 20, 25: 25, 30: 30,
              35: 35, 40: 40, 45: 45, 50: 50,
              55: 55, 60: 60, 65: 65, 70: 70}

downsampled['POSTED_SPEED_LIMIT'] = downsampled['POSTED_SPEED_LIMIT'].map(s
downsampled['POSTED_SPEED_LIMIT'].value_counts(dropna=False, normalize=True
```

Out[32]:
```
30    0.748981
35    0.081009
25    0.057985
15    0.048397
20    0.035789
40    0.013563
45    0.007665
50    0.005257
55    0.001239
60    0.000085
70    0.000014
65    0.000014
Name: POSTED_SPEED_LIMIT, dtype: float64
```

Next, I attempt to bin the *CRASH_HOUR* column into distinct 'time of day' categories: Late Night (11PM-4AM), Morning (5AM-11AM), Afternoon (12PM-5PM) & Evening/Night (6PM - 10PM). Since I've determined my first category (Late Night) to start at 11PM, I will have the hour count start at 0 = 11 PM instead of 0 = midnight.

In [33]:
```
downsampled['CRASH_HOUR'] -= 1
downsampled.loc[downsampled['CRASH_HOUR'] == -1, 'CRASH_HOUR'] = 23
```

In [34]:
```
bins = [0, 5, 12, 18, 23]
label = ['Late Night', 'Morning', 'Afternoon', 'Evening/Night']

downsampled['Time of Day'] = pd.cut(downsampled['CRASH_HOUR'], bins=bins, l
                                    include_lowest=True, ordered=False)
downsampled['Time of Day'].value_counts(dropna=False, normalize=True)
```

Out[34]:
```
Afternoon        0.359111
Morning          0.342912
Evening/Night    0.174284
Late Night       0.123693
Name: Time of Day, dtype: float64
```

In [35]:
```
downsampled.drop('CRASH_HOUR', axis=1, inplace=True)
```

I am now left to deal with the *CRASH_DAY_OF_WEEK* & *CRASH_MONTH* columns. Since the months do not have any comparable numeric value to me, I decide to ignore them in this section & treat them as a categorical variable later on.

However, when it comes to the day of week feature, I believe it could be adjusted in a way that gives meaning to the numerical values. Instead of assigning a separate number to each day of the week, I prefer to assign each day a number indicated how 'far' from the weekend the day is. This is because of my initial feeling that reckless driving & crashes in general are more likely to happen on weekends.

The labels will be assigned as follows: Wed - 3 days away, Tues & Thurs - 2 days away, Mon & Fri - 1 day away, Sat & Sun - 0 days away. My idea, then, is that as the new *Days from Wknd* feature increases numerically, crashes (specifically fatal crashes from drunk driving) are less likely to occur.

In [36]:
```
day_dict = {4: 3, 3: 2, 5: 2, 2: 1, 6: 1, 1: 0, 7: 0}

downsampled['Days from Wknd'] = downsampled['CRASH_DAY_OF_WEEK'].map(day_di
downsampled.drop('CRASH_DAY_OF_WEEK', axis=1, inplace=True)
downsampled['Days from Wknd'].value_counts(dropna=False, normalize=True)
```

Out[36]:
```
1    0.295854
0    0.288716
2    0.277561
3    0.137869
Name: Days from Wknd, dtype: float64
```

6.3.2 Converting All Binary Columns to 0's & 1's

In [37]:
```
binaries = ['INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I', 'HIT_AND_RUN_I'

for b in binaries:
    downsampled[b] = downsampled[b].map({'Y': 1, 'N': 0})
```

In [38]:
```
downsampled[binaries].head()
```

Out[38]:

	INTERSECTION_RELATED_I	NOT_RIGHT_OF_WAY_I	HIT_AND_RUN_I	WORK_ZONE_I
0	0	0	1	0
1	1	0	0	0
2	0	0	0	0
3	0	0	1	0
4	1	0	0	0

6.3.3 Dealing With Categorical Features

Next, I create a DataFrame of dummy columns for all of the categorical features, which make up most of the columns here.

In [39]:
```
downsampled.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70190 entries, 0 to 70189
Data columns (total 32 columns):
 #   Column                         Non-Null Count  Dtype   
---  ------                         --------------  -----   
 0   CRASH_RECORD_ID                70190 non-null  object  
 1   POSTED_SPEED_LIMIT             70190 non-null  int64   
 2   TRAFFIC_CONTROL_DEVICE         70190 non-null  object  
 3   DEVICE_CONDITION               70190 non-null  object  
 4   WEATHER_CONDITION              70190 non-null  object  
 5   LIGHTING_CONDITION             70190 non-null  object  
 6   FIRST_CRASH_TYPE               70190 non-null  object  
 7   TRAFFICWAY_TYPE                70190 non-null  object  
 8   LANE_CNT                       70190 non-null  object  
 9   ALIGNMENT                      70190 non-null  object  
 10  ROADWAY_SURFACE_COND           70190 non-null  object  
 11  ROAD_DEFECT                    70190 non-null  object  
 12  CRASH_TYPE                     70190 non-null  object  
 13  INTERSECTION_RELATED_I         70190 non-null  int64   
 14  NOT_RIGHT_OF_WAY_I             70190 non-null  int64   
 15  HIT_AND_RUN_I                  70190 non-null  int64   
 16  DAMAGE                         70190 non-null  object  
 17  WORK_ZONE_I                    70190 non-null  int64   
 18  WORK_ZONE_TYPE                 70190 non-null  object  
 19  NUM_UNITS                      70190 non-null  int64   
 20  MOST_SEVERE_INJURY             70190 non-null  object  
 21  INJURIES_TOTAL                 70190 non-null  float64 
 22  INJURIES_FATAL                 70190 non-null  float64 
 23  INJURIES_INCAPACITATING        70190 non-null  float64 
 24  INJURIES_NON_INCAPACITATING    70190 non-null  float64 
 25  INJURIES_REPORTED_NOT_EVIDENT  70190 non-null  float64 
 26  INJURIES_NO_INDICATION         70190 non-null  float64 
 27  INJURIES_UNKNOWN               70190 non-null  float64 
 28  CRASH_MONTH                    70190 non-null  int64   
 29  Primary Cause                  70190 non-null  object  
 30  Time of Day                    70190 non-null  category
 31  Days from Wknd                 70190 non-null  int64   
dtypes: category(1), float64(7), int64(8), object(16)
memory usage: 16.7+ MB
```

In [40]:
```
downsampled['CRASH_MONTH'] = downsampled['CRASH_MONTH'].apply(str)
```

In [41]:
```
categorical = ['TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'WEATHER_CONDI
                'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE', 'TRAFFICWAY_TYPE',
               'ALIGNMENT', 'ROADWAY_SURFACE_COND', 'ROAD_DEFECT', 'CRASH_TY
               'DAMAGE', 'WORK_ZONE_TYPE', 'MOST_SEVERE_INJURY', 'CRASH_MONT
                'LANE_CNT', 'Time of Day']
```

In [42]:
```python
for c in categorical:
    print(f'Value counts for column {c}' + '\n')
    print(downsampled[c].value_counts(normalize=True))
    print('\n')
```

```
OTHER                        0.009275
FUNCTIONING IMPROPERLY       0.007822
NOT FUNCTIONING              0.003690
WORN REFLECTIVE MATERIAL     0.000627
MISSING                      0.000185
Name: DEVICE_CONDITION, dtype: float64


Value counts for column WEATHER_CONDITION

CLEAR                       0.736430
RAIN                        0.119732
SNOW                        0.082148
CLOUDY/OVERCAST             0.032668
UNKNOWN                     0.014204
OTHER                       0.005770
SLEET/HAIL                  0.003320
FOG/SMOKE/HAZE              0.002394
FREEZING RAIN/DRIZZLE       0.001923
```

In [43]:
```python
dummies = pd.get_dummies(downsampled[categorical])

dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70190 entries, 0 to 70189
Columns: 138 entries, TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN to Tim
e of Day_Evening/Night
dtypes: uint8(138)
memory usage: 9.2 MB
```

Now, I create a subset of the main **downsampled** DataFrame, **downsampled_num**, containing only the features with numeric values. This and the **dummies** DataFrame can be manipulated separately and/or combined at any point before being fed into a chosen classifier.

In [44]:
```
downsampled_num = downsampled.drop(columns=categorical, axis=1)
downsampled_num = downsampled_num.drop(columns=['CRASH_RECORD_ID', 'Primary

downsampled_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70190 entries, 0 to 70189
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   POSTED_SPEED_LIMIT              70190 non-null  int64  
 1   INTERSECTION_RELATED_I          70190 non-null  int64  
 2   NOT_RIGHT_OF_WAY_I              70190 non-null  int64  
 3   HIT_AND_RUN_I                   70190 non-null  int64  
 4   WORK_ZONE_I                     70190 non-null  int64  
 5   NUM_UNITS                       70190 non-null  int64  
 6   INJURIES_TOTAL                  70190 non-null  float64
 7   INJURIES_FATAL                  70190 non-null  float64
 8   INJURIES_INCAPACITATING         70190 non-null  float64
 9   INJURIES_NON_INCAPACITATING     70190 non-null  float64
 10  INJURIES_REPORTED_NOT_EVIDENT   70190 non-null  float64
 11  INJURIES_NO_INDICATION          70190 non-null  float64
 12  INJURIES_UNKNOWN                70190 non-null  float64
 13  Days from Wknd                  70190 non-null  int64  
dtypes: float64(7), int64(7)
memory usage: 7.5 MB
```

In [45]:
```
X = pd.concat([downsampled_num, dummies], axis=1)
y = downsampled['Primary Cause']
```

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=26)
```

7 Multinomial Logistic Regression & Lasso For Feature Selection

Using Multinomial Logistic Regression & Lasso to narrow feature selection for future models.

In [47]:
```
ss = StandardScaler()
mm = MinMaxScaler()

X_train_scaled = mm.fit_transform(X_train)
X_test_scaled = mm.transform(X_test)
```

In [48]:
```
mlr = LogisticRegression(multi_class='multinomial', solver='saga', max_iter
                          C=1, penalty='l1', random_state=26)
sel_ = SelectFromModel(estimator=mlr)

sel_.fit(X_train_scaled, y_train)
```

Out[48]:
```
SelectFromModel(estimator=LogisticRegression(C=1, max_iter=1000,
                                              multi_class='multinomial',
                                              penalty='l1', random_state=2
6,
                                              solver='saga'))
```

In [49]:
```
sel_.get_support()
```

Out[49]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True, False,  True, False,  True,  True,  True,
        True,  True,  True, False,  True,  True,  True,  True, False,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
       False,  True,  True,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True])
```

In [50]:
```
selections = X_train.columns[(sel_.get_support())]

print(f'Total Features: {X_train.shape[1]}')
print(f'Selected Features: {len(selections)}')
print(f'Features with Coefficients Shrunk to Zero: {np.sum(sel_.estimator_.
```

```
Total Features: 152
Selected Features: 143
Features with Coefficients Shrunk to Zero: 167
```

In [51]:
```
for i in list(range(4)):
    print(f'Feature List {i+1} has penalized this many columns: {np.sum((se
```

```
Feature List 1 has penalized this many columns: 44
Feature List 2 has penalized this many columns: 45
Feature List 3 has penalized this many columns: 39
Feature List 4 has penalized this many columns: 39
```

In [52]:
```
to_remove = list(X_train.columns[(sel_.estimator_.coef_ == 0)[2].ravel().to

to_remove
```

Out[52]:
```
['INJURIES_TOTAL',
 'INJURIES_FATAL',
 'INJURIES_REPORTED_NOT_EVIDENT',
 'INJURIES_UNKNOWN',
 'TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN',
 'TRAFFIC_CONTROL_DEVICE_FLASHING CONTROL SIGNAL',
 'TRAFFIC_CONTROL_DEVICE_NO PASSING',
 'TRAFFIC_CONTROL_DEVICE_OTHER',
 'TRAFFIC_CONTROL_DEVICE_OTHER RAILROAD CROSSING',
 'TRAFFIC_CONTROL_DEVICE_RAILROAD CROSSING GATE',
 'TRAFFIC_CONTROL_DEVICE_RR CROSSING SIGN',
 'TRAFFIC_CONTROL_DEVICE_SCHOOL ZONE',
 'TRAFFIC_CONTROL_DEVICE_TRAFFIC SIGNAL',
 'DEVICE_CONDITION_FUNCTIONING PROPERLY',
 'DEVICE_CONDITION_WORN REFLECTIVE MATERIAL',
 'WEATHER_CONDITION_RAIN',
 'WEATHER_CONDITION_SLEET/HAIL',
 'FIRST_CRASH_TYPE_REAR TO REAR',
 'FIRST_CRASH_TYPE_SIDESWIPE SAME DIRECTION',
 'FIRST_CRASH_TYPE_TRAIN',
 'TRAFFICWAY_TYPE_CENTER TURN LANE',
 'TRAFFICWAY_TYPE_L-INTERSECTION',
 'TRAFFICWAY_TYPE_NOT DIVIDED',
 'TRAFFICWAY_TYPE_NOT REPORTED',
 'TRAFFICWAY_TYPE_ROUNDABOUT',
 'TRAFFICWAY_TYPE_TRAFFIC ROUTE',
 'ALIGNMENT_CURVE ON HILLCREST',
 'ALIGNMENT_STRAIGHT ON GRADE',
 'ROADWAY_SURFACE_COND_DRY',
 'ROAD_DEFECT_SHOULDER DEFECT',
 'ROAD_DEFECT_WORN SURFACE',
 'DAMAGE_OVER $1,500',
 'WORK_ZONE_TYPE_MAINTENANCE',
 'WORK_ZONE_TYPE_UNKNOWN',
 'WORK_ZONE_TYPE_UTILITY',
 'MOST_SEVERE_INJURY_INCAPACITATING INJURY',
 'LANE_CNT_3',
 'LANE_CNT_4',
 'LANE_CNT_Missing']
```

In [53]:
```
X_train.drop(columns=to_remove, axis=1, inplace=True)
X_test.drop(columns=to_remove, axis=1, inplace=True)
```

In [54]:
```
X_train.head()
```

Out[54]:

	POSTED_SPEED_LIMIT	INTERSECTION_RELATED_I	NOT_RIGHT_OF_WAY_I	HIT_AND_RU
58010	30	1	0	
57601	30	1	0	
54993	30	0	0	
36318	30	0	0	
29660	30	0	0	

8 Baseline Decision Tree

Ultimately, I'm going to be running a random forest classifier on this dataset. But since a random forest is just an aggregate of many decision trees, let's take a look at a basic decision tree in order to get a simplified idea of what features the algorithm might deem important.

In [55]:
```
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=5, random_sta
tree_clf.fit(X_train, y_train)
```

Out[55]:
```
DecisionTreeClassifier(max_depth=5, random_state=26)
```

In [56]:
```
feature_list = list(X_train.columns)
importances = list(tree_clf.feature_importances_)

feature_importances = [(feature, round(importance, 2)) for feature, importa
feature_importances = sorted(feature_importances, key = lambda x: x[1], rev
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_
```

```
Variable: FIRST_CRASH_TYPE_ANGLE Importance: 0.4
Variable: WEATHER_CONDITION_CLEAR Importance: 0.15
Variable: TRAFFIC_CONTROL_DEVICE_NO CONTROLS Importance: 0.13
Variable: INJURIES_NO_INDICATION Importance: 0.08
Variable: FIRST_CRASH_TYPE_TURNING Importance: 0.07
Variable: HIT_AND_RUN_I        Importance: 0.05
Variable: WEATHER_CONDITION_SNOW Importance: 0.05
Variable: CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH Importance: 0.03
Variable: FIRST_CRASH_TYPE_PARKED MOTOR VEHICLE Importance: 0.02
Variable: TRAFFIC_CONTROL_DEVICE_STOP SIGN/FLASHER Importance: 0.01
Variable: ROADWAY_SURFACE_COND_WET Importance: 0.01
Variable: POSTED_SPEED_LIMIT   Importance: 0.0
Variable: INTERSECTION_RELATED_I Importance: 0.0
Variable: NOT_RIGHT_OF_WAY_I   Importance: 0.0
Variable: WORK_ZONE_I          Importance: 0.0
Variable: NUM_UNITS            Importance: 0.0
Variable: INJURIES_INCAPACITATING Importance: 0.0
Variable: INJURIES_NON_INCAPACITATING Importance: 0.0
Variable: Days from Wknd       Importance: 0.0
Variable: TRAFFIC_CONTROL_DEVICE_DELINEATORS Importance: 0.0
```

In [57]:
```
y_hat_test = tree_clf.predict(X_test)

print(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
print(f'Testing Accuracy for Decision Tree Classifier: {(accuracy_score(y_t
```

```
[[3287  277  720  732]
 [ 185  935  755  811]
 [ 412  710 2737  903]
 [ 800  589  866 2829]]
                                    precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.70      0.66      0.68      501
6
           Impairment/Distraction       0.37      0.35      0.36      268
6
                   Outside Hazard       0.54      0.57      0.56      476
2
                 Reckless Driving       0.54      0.56      0.55      508
4

                         accuracy                           0.56     1754
8
                        macro avg       0.54      0.53      0.54     1754
8
                     weighted avg       0.56      0.56      0.56     1754
8

Testing Accuracy for Decision Tree Classifier: 55.77843628903578%
```

Based on the above scores, the decision tree seems to be performing poorly, but still doing significantly better than a random guess! Furthermore, this tree seems best at predicting whether the primary crash cause is 'Ignoring Traffic Signs & Warnings.'

Lastly, looking at the feature importances, it appears there only a subset of 11 features had any noticeable impact on the classifier:

- If the first crash occurred at an angle
- If the weather was clear
- If there were no traffic control devices at the scene
- If there was no indication of injury
- If the first crash occurred during a turn
- If the crash occurred as a hit and run incident
- If there was snowy weather
- If there was an injury and/or a towed vehicle due to the crash
- If the first crash involved a parked vehicle
- If there was a traffic signal at the crash
- If the roads were wet

Let's expand to a random forest and see if its results verify what we've found here.

9 Random Forest

In [58]:
```
forest = RandomForestClassifier(n_estimators=100, max_depth=5, random_state
forest.fit(X_train, y_train)
```

Out[58]:
```
RandomForestClassifier(max_depth=5, random_state=26)
```

In [59]:
```
feature_list = list(X_train.columns)
importances = list(forest.feature_importances_)

feature_importances = [(feature, round(importance, 2)) for feature, importa
feature_importances = sorted(feature_importances, key = lambda x: x[1], rev
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_
```

```
Variable: FIRST_CRASH_TYPE_ANGLE Importance: 0.18
Variable: INTERSECTION_RELATED_I Importance: 0.09
Variable: TRAFFIC_CONTROL_DEVICE_NO CONTROLS Importance: 0.09
Variable: DEVICE_CONDITION_NO CONTROLS Importance: 0.09
Variable: WEATHER_CONDITION_CLEAR Importance: 0.06
Variable: NUM_UNITS            Importance: 0.05
Variable: WEATHER_CONDITION_SNOW Importance: 0.05
Variable: FIRST_CRASH_TYPE_REAR END Importance: 0.05
Variable: INJURIES_NO_INDICATION Importance: 0.04
Variable: CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH Importance: 0.04
Variable: CRASH_TYPE_NO INJURY / DRIVE AWAY Importance: 0.04
Variable: HIT_AND_RUN_I        Importance: 0.03
Variable: ROADWAY_SURFACE_COND_SNOW OR SLUSH Importance: 0.03
Variable: FIRST_CRASH_TYPE_PARKED MOTOR VEHICLE Importance: 0.02
Variable: FIRST_CRASH_TYPE_TURNING Importance: 0.02
Variable: ROADWAY_SURFACE_COND_ICE Importance: 0.02
Variable: INJURIES_NON_INCAPACITATING Importance: 0.01
Variable: TRAFFIC_CONTROL_DEVICE_STOP SIGN/FLASHER Importance: 0.01
Variable: FIRST_CRASH_TYPE_FIXED OBJECT Importance: 0.01
Variable: TRAFFICWAY_TYPE_FOUR WAY Importance: 0.01
```

In [60]:
```
y_hat_test = forest.predict(X_test)
print(confusion_matrix(y_test, y_hat_test))
```

```
[[3685   57  456  818]
 [ 413  358  765 1150]
 [ 687  147 2695 1233]
 [1059   58  623 3344]]
```

In [61]:
```
print(f'Training Accuracy: {(forest.score(X_train, y_train))*100}%')
print(f'Test Accuracy: {(forest.score(X_test, y_test))*100}%')
```

```
Training Accuracy: 57.585198130770l1%
Test Accuracy: 57.45384089354913%
```

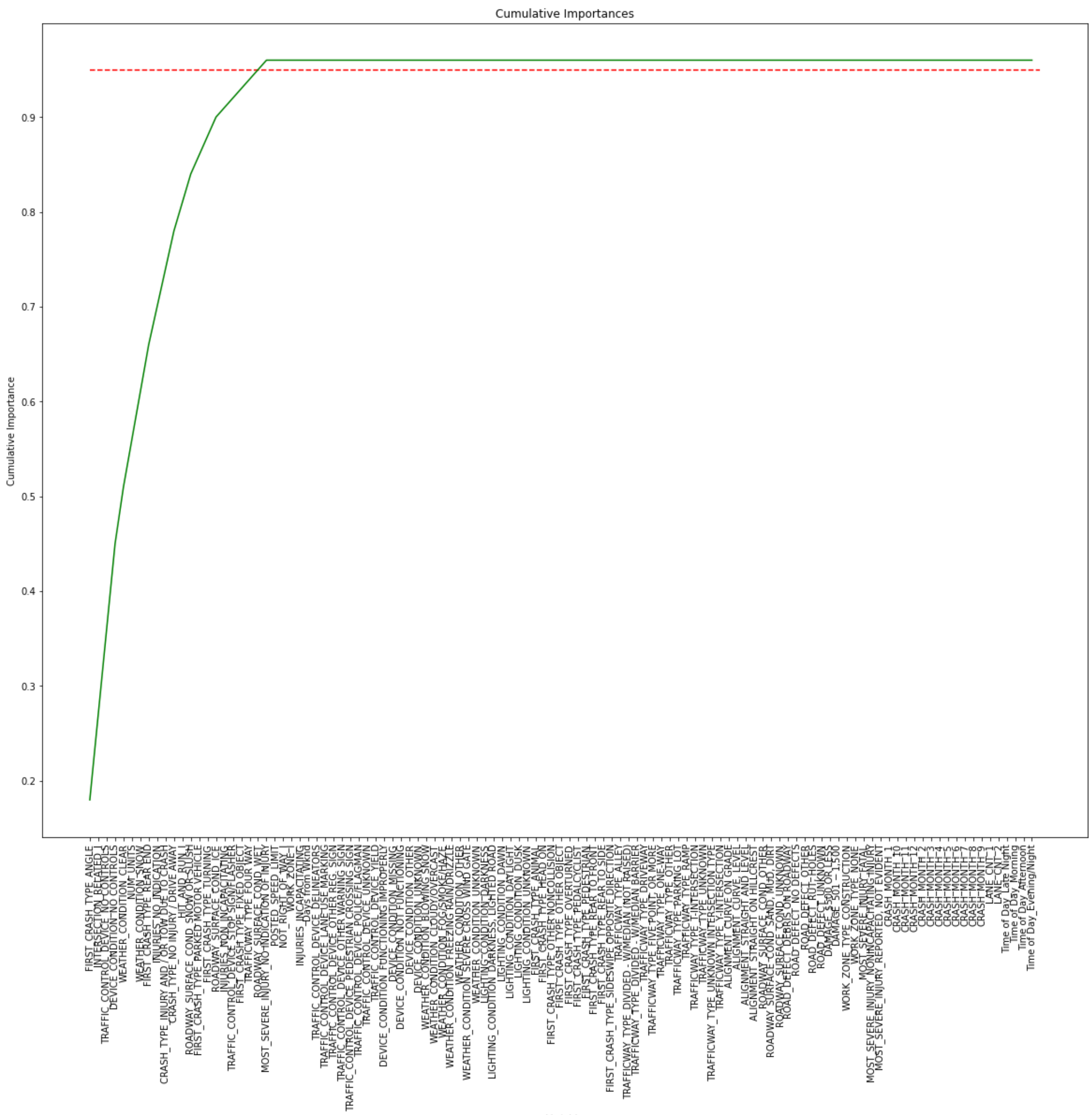
Expanding into a Random Forest of 100 trees only gave a marginal boost of about 2% to the accuracy score, unfortunately. Looking at the feature importances, though, it seems as though I now have 22 features making noticeable contribution to the classification.

After these 22 columns, the cumulative feature importance has surpassed 95%, as seen in the plot below. Thus, I am going to proceed under the assumption that the remaining features won't be impactful enough to be needed anymore.

In [62]:

```
sorted_importances = [importance[1] for importance in feature_importances]
sorted_features = [importance[0] for importance in feature_importances]
cumulative_importances = np.cumsum(sorted_importances)

fig = plt.figure(figsize=(20,16))
x_values = list(range(len(importances)))
plt.plot(x_values, cumulative_importances, 'g-')
plt.hlines(y = 0.95, xmin=0, xmax=len(sorted_importances), color = 'r', lin
plt.xticks(x_values, sorted_features, rotation = 'vertical')
plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.title('Cum
```



In [63]:
```
significant = sorted_features[:22]

X_train_red = X_train[significant]
X_test_red = X_test[significant]
```

10 Tuning Random Forest Hyperparameters

In [64]:
```
rf_clf = RandomForestClassifier()
mean_rf_cv_score = np.mean(cross_val_score(rf_clf, X_train_red, y_train, cv
```

In [65]:
```
print(f"Mean Cross Validation Score for Random Forest Classifier: {mean_rf_
```

```
Mean Cross Validation Score for Random Forest Classifier: 57.35%
```

In [66]:
```
rf_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [5, 10, 15],
    'max_features': [3, 5, 10],
}
```

In [67]:
```
rf_grid_search = GridSearchCV(rf_clf, rf_param_grid, cv=3)
rf_grid_search.fit(X_train_red, y_train)

print(f"Training Accuracy: {rf_grid_search.best_score_ :.2%}")
print("")
print(f"Optimal Parameters: {rf_grid_search.best_params_}")
```

```
Training Accuracy: 59.86%

Optimal Parameters: {'criterion': 'gini', 'max_depth': 10, 'max_feature
s': 5, 'min_samples_leaf': 10, 'min_samples_split': 10}
```

In [68]:
```
rf_clf = RandomForestClassifier(criterion='entropy', max_depth=10, max_feat
                                min_samples_leaf=5, min_samples_split=5, n_e
                                random_state=26)
rf_clf.fit(X_train_red, y_train)
```

Out[68]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features=3,
                       min_samples_leaf=5, min_samples_split=5,
                       random_state=26)
```

In [69]:
```
y_hat_test = rf_clf.predict(X_test_red)

print(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
print(f'Testing Accuracy for Decision Tree Classifier: {(rf_clf.score(X_tra
print(f'Testing Accuracy for Decision Tree Classifier: {(rf_clf.score(X_tes
```

```
[[3468  156  447  945]
 [ 250  830  621  985]
 [ 544  390 2811 1017]
 [ 761  241  670 3412]]
                                   precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.69      0.69      0.69      501
6
           Impairment/Distraction       0.51      0.31      0.39      268
6
                   Outside Hazard       0.62      0.59      0.60      476
2
                 Reckless Driving       0.54      0.67      0.60      508
4

                         accuracy                           0.60     1754
8
                        macro avg       0.59      0.57      0.57     1754
8
                     weighted avg       0.60      0.60      0.59     1754
8

Testing Accuracy for Decision Tree Classifier: 60.48782341096462%
Testing Accuracy for Decision Tree Classifier: 59.95550490084 34%
```