1 Predicting Contributory Causes of Chicago Auto Accidents

Author: Dom Garcia (mailto:dlgarcia.017@gmail.com)

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

USAA wants to better understand the liability associated with their customers' accidents in order to determine the premiums they should be charging. As one of the country's biggest metropolitan areas with a variety of weather conditions, Chicago is a prime candidate for studying auto accidents.

The company hired a team of data scientists to study the primary cause of these accidents, which will provide insight on whether their customers premiums should be adjusted. Furthermore, the team's research should make the company more aware of what conditions contribute most to predicting accident causes.

As the head of this team, I am in charge of building a classifier that will let the company know, given the facts about a customer's accident, whether that customer is due for an increased premium.

4 Importing Data, Necessary Libraries

The data in this project is provided by the City of Chicago (https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if (https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if)) & sourced from various 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 [2]:
```
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
from sklearn.dummy import DummyClassifier
from xgboost import XGBClassifier
```

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*. I'm not interesting in building a model using features made up of mostly 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 target values provide useful insight, 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 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)
```

Since I've imputed N for the null values in the *WORK_ZONE_I* column, I impute a 'NONE' string for the corresponding *WORK_ZONE_TYPE* column.

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 things 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_CONTRO
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	35	N
1	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	35	STOP SI
2	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	30	TRA
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.

- **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_CONTRC
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	35	N(
1	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	35	STOP SI
2	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	30	TR/
3	f636d4a51a88015ac89031159b1f1952b8d92e49d11aeb...	30	N(
4	9c974548026c1b962569040bd8fa08ae643ffc28c15ebd...	10	

After creating the new binned target, 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 any 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 [22]:
```
df_crash['Primary Cause'].value_counts()
```

Out[22]:
```
Reckless Driving                       232557
Ignoring Traffic Signs & Warnings       20106
Outside Hazard                          19210
Impairment/Distraction                  10768
Name: Primary Cause, dtype: int64
```

In [23]:
```
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 [24]:
```
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 [25]:
```
downsampled['Primary Cause'].value_counts()
```

Out[25]:
```
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 entries. In this section, I look at a few distinct subgroup of features and manipulate each one individually before bringing everything together again at the end.

In [26]:
```
downsampled.reset_index(drop=True, inplace=True)
downsampled.head()
```

Out[26]:

	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

Recall that the *LANE_CNT* column had lots of strange values when they were previously listed. There's a decent chance that other numeric columns have similarly nonsensical values that need to be cleaned up. I make sure to do just that in the following section.

Additionally, this is a good place to determine on a case-by-case basis whether a numeric column is a continuous measure or a set of categorical labels.

In [27]:
```
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 [28]:
```
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 [29]:
```
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)
downsampled['LANE_CNT'].value_counts(dropna=False, normalize=True)
```

Out[29]:
```
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 [30]:
```
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[30]:
```
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 [31]:
```
downsampled['CRASH_HOUR'] -= 1
downsampled.loc[downsampled['CRASH_HOUR'] == -1, 'CRASH_HOUR'] = 23
```

In [32]:
```
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[32]:
```
Afternoon        0.359111
Morning          0.342912
Evening/Night    0.174284
Late Night       0.123693
Name: Time of Day, dtype: float64
```

In [33]:
```
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 opt to assign each day a number indicating 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 [34]:
```
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[34]:
```
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

Converting all columns with binary Y/N entries to 1/0 entries.

In [35]:
```
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 [36]:
```
downsampled[binaries].head()
```

Out[36]:

	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 [37]:
```
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
```

Before creating dummy columns, I convert all numeric (or originally numeric) categoricals to object types.

In [38]:
```
downsampled['CRASH_MONTH'] = downsampled['CRASH_MONTH'].apply(str)
downsampled['Time of Day'] = downsampled['Time of Day'].apply(str)
downsampled['LANE_CNT'] = downsampled['LANE_CNT'].apply(str)
```

In [39]:
```
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 [40]:
```python
for c in categorical:
    print(f'Value counts for column {c}' + '\n')
    print(downsampled[c].value_counts(normalize=True))
    print('\n')
```

```
Value counts for column TRAFFIC_CONTROL_DEVICE

NO CONTROLS                 0.459795
TRAFFIC SIGNAL              0.351004
STOP SIGN/FLASHER           0.155207
UNKNOWN                     0.015059
OTHER                       0.006482
LANE USE MARKING            0.003049
YIELD                       0.002066
OTHER WARNING SIGN          0.001866
OTHER REG. SIGN             0.001752
RAILROAD CROSSING GATE      0.000812
POLICE/FLAGMAN              0.000613
PEDESTRIAN CROSSING SIGN    0.000556
OTHER RAILROAD CROSSING     0.000399
FLASHING CONTROL SIGNAL     0.000399
SCHOOL ZONE                 0.000385
DELINEATORS                 0.000328
NO PASSING                  0.000114
```

In [41]:
```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 are combined before being split into training & test sets for future models.

In [42]:
```
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 [43]:
```
X = pd.concat([downsampled_num, dummies], axis=1)
y = downsampled['Primary Cause']
```

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=26)
```

7 Baseline Dummy Classifiers

Before proceeding with constructing and interpreting a classifier, it would certainly help to have a naive strategy to compare the classifier to.

"Is the model a better predictor than randomly guessing?"

This section, in which a couple of different stabs at 'randomly guessing' are made, allows me to confidently answer that question after future model construction.

7.1 Strategy: Stratified

Here, I run a dummy classifier that 'randomly guesses' based on the relative count of each target outcome. So 'Reckless Driving' and 'Ignoring Traffic Signs & Warnings' are equally likely to be chosen, 'Outside Hazard' is slightly less likely, and 'Impairment/Distraction' is significantly less likely.

In [45]:
```
dummy_clf = DummyClassifier(strategy='stratified', random_state=26)
dummy_clf.fit(X_train, y_train)
```

Out[45]:
```
DummyClassifier(random_state=26, strategy='stratified')
```

In [46]:
```
y_hat_train = dummy_clf.predict(X_train)
y_hat_test = dummy_clf.predict(X_test)

print(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
print(f'Training Accuracy for Stratified Dummy Classifier: {(accuracy_score
print(f'Testing Accuracy for Stratified Dummy Classifier: {(accuracy_score(
```

```
[[1448  801 1315 1452]
 [ 735  397  772  782]
 [1351  731 1307 1373]
 [1457  789 1426 1412]]
                                   precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.29      0.29      0.29      501
6
           Impairment/Distraction       0.15      0.15      0.15      268
6
                   Outside Hazard       0.27      0.27      0.27      476
2
                 Reckless Driving       0.28      0.28      0.28      508
4

                         accuracy                           0.26     1754
8
                        macro avg       0.25      0.25      0.25     1754
8
                     weighted avg       0.26      0.26      0.26     1754
8

Training Accuracy for Stratified Dummy Classifier: 26.03434519965047%
Testing Accuracy for Stratified Dummy Classifier: 26.008661955778432%
```

7.2 Strategy: Uniform

Just to ensure that the results & scores aren't limited to the dummy classifier above, I run a new one that 'randomly guesses' in a way that everyone's familiar with: uniformly. Essentially, in this naive model each cause is equally likely to be predicted.

In [47]:
```
dummy_uni_clf = DummyClassifier(strategy='uniform', random_state=26)
dummy_uni_clf.fit(X_train, y_train)
```

Out[47]:
```
DummyClassifier(random_state=26, strategy='uniform')
```

In [48]:
```
y_hat_train = dummy_uni_clf.predict(X_train)
y_hat_test = dummy_uni_clf.predict(X_test)

print(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
print(f'Training Accuracy for Uniform Dummy Classifier: {(accuracy_score(y_
print(f'Testing Accuracy for Uniform Dummy Classifier: {(accuracy_score(y_t
```

```
[[1233 1270 1305 1208]
 [ 714  637  662  673]
 [1203 1168 1209 1182]
 [1269 1283 1318 1214]]
                                  precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.28      0.25      0.26      501
6
           Impairment/Distraction       0.15      0.24      0.18      268
6
                   Outside Hazard       0.27      0.25      0.26      476
2
                 Reckless Driving       0.28      0.24      0.26      508
4

                         accuracy                           0.24     1754
8
                        macro avg       0.24      0.24      0.24     1754
8
                     weighted avg       0.26      0.24      0.25     1754
8

Training Accuracy for Uniform Dummy Classifier: 24.8983701227157%
Testing Accuracy for Uniform Dummy Classifier: 24.464326418965125%
```

At least in these naive baseline models, there seems to be no issue with over/underfitting. Additionally, both models end up with a test accuracy of about 25%, which is now the 'random guess' success rate to compare to going forward.

8 Multinomial Logistic Regression & Lasso for Feature Selection

In this section, I use the only multiclass classifier at my disposal with Lasso (L1) regularization in order to find out which features are not particularly useful for modeling.

To start, I iterate through a few different values of the multinomial logistic regression's C parameter (inverse regularization strength). I would like to make sure that the model has a C-value that allows for the most accurate modeling before I use the classifier for feature selection.

In [49]:
```
mm = MinMaxScaler()

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

8.1 Optimizing 'C' Parameter for Model Accuracy

In [50]:
```
c_vals = [0.01, 0.1, 0.5, 1, 5]
acc = []

for c in c_vals:
    mlr = LogisticRegression(multi_class='multinomial', solver='saga', max_
                            C=c, penalty='l1', random_state=26)
    mlr.fit(X_train_scaled, y_train)
    y_hat_test = mlr.predict(X_test_scaled)
    score = accuracy_score(y_test, y_hat_test)
    acc.append(score)

print(dict(zip(c_vals, acc)))
```

```
{0.01: 0.5987576931844085, 0.1: 0.6114656940961933, 0.5: 0.61397310234784
59, 1: 0.6139731023478459, 5: 0.6132892637337588}
```

8.2 Running Model with C = 1 & Lasso Penalty Observation

In [51]:
```
mlr = LogisticRegression(multi_class='multinomial', solver='saga', max_iter
                         C=1, penalty='l1', random_state=26)
mlr.fit(X_train_scaled, y_train)
```

Out[51]:
```
LogisticRegression(C=1, max_iter=1000, multi_class='multinomial', penalty
='l1',
                   random_state=26, solver='saga')
```

In [52]:
```
y_hat_train = mlr.predict(X_train_scaled)
y_hat_test = mlr.predict(X_test_scaled)

print('Confusion matrix for MLR:', '\n', confusion_matrix(y_test, y_hat_tes
print(classification_report(y_test, y_hat_test))
print(f'Training Accuracy for Multinomial Logistic Regression Classifier: {
print(f'Testing Accuracy for Multinomial Logistic Regression Classifier: {(
```

```
Confusion matrix for MLR:
 [[3616  227  408  765]
 [ 258 1107  488  833]
 [ 576  493 2820  873]
 [ 840  376  637 3231]]
                                  precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.68      0.72      0.70      501
6
           Impairment/Distraction       0.50      0.41      0.45      268
6
                   Outside Hazard       0.65      0.59      0.62      476
2
                  Reckless Driving       0.57      0.64      0.60      508
4

                         accuracy                           0.61     1754
8
                        macro avg       0.60      0.59      0.59     1754
8
                     weighted avg       0.61      0.61      0.61     1754
8

Training Accuracy for Multinomial Logistic Regression Classifier: 61.3996
428707116%
Testing Accuracy for Multinomial Logistic Regression Classifier: 61.39731
0234784584%
```

Here is where the multinomial logistic regression classifier distinguishes itself. Instead of listing one set of features & feature importances for all predictions, it lists the 'feature un-importances' (in this case a list of features with penalized coefficients) **for each target class.**

In [53]:
```
class_dict = {0: 'Ignoring Traffic Signs & Warnings',
              1: 'Impairment/Distraction',
              2: 'Outside Hazard',
              3: 'Reckless Driving'}
zero_coef = []

for i in range(4):
    print(f'Penalized features for output {class_dict[i]}:' + '\n')
    print(list(X_train.columns[(mlr.coef_[i] == 0)]))
    print('\n')
    zero_coef.append(list(X_train.columns[(mlr.coef_[i] == 0)]))
```

```
Penalized features for output Ignoring Traffic Signs & Warnings:

['INJURIES_INCAPACITATING', 'INJURIES_NON_INCAPACITATING', 'INJURIES_REPO
RTED_NOT_EVIDENT', 'INJURIES_UNKNOWN', 'TRAFFIC_CONTROL_DEVICE_BICYCLE CR
OSSING SIGN', 'TRAFFIC_CONTROL_DEVICE_DELINEATORS', 'TRAFFIC_CONTROL_DEVI
CE_NO PASSING', 'TRAFFIC_CONTROL_DEVICE_OTHER RAILROAD CROSSING', 'TRAFFI
C_CONTROL_DEVICE_PEDESTRIAN CROSSING SIGN', 'TRAFFIC_CONTROL_DEVICE_POLIC
E/FLAGMAN', 'TRAFFIC_CONTROL_DEVICE_RAILROAD CROSSING GATE', 'TRAFFIC_CON
TROL_DEVICE_RR CROSSING SIGN', 'TRAFFIC_CONTROL_DEVICE_SCHOOL ZONE', 'TRA
FFIC_CONTROL_DEVICE_TRAFFIC SIGNAL', 'DEVICE_CONDITION_FUNCTIONING IMPROP
ERLY', 'DEVICE_CONDITION_FUNCTIONING PROPERLY', 'DEVICE_CONDITION_MISSIN
G', 'WEATHER_CONDITION_CLOUDY/OVERCAST', 'WEATHER_CONDITION_FOG/SMOKE/HAZ
E', 'WEATHER_CONDITION_FREEZING RAIN/DRIZZLE', 'WEATHER_CONDITION_SEVERE 
CROSS WIND GATE', 'WEATHER_CONDITION_SLEET/HAIL', 'FIRST_CRASH_TYPE_OTHER
OBJECT', 'FIRST_CRASH_TYPE_OVERTURNED', 'FIRST_CRASH_TYPE_PEDESTRIAN', 'F
IRST_CRASH_TYPE_REAR TO REAR', 'TRAFFICWAY_TYPE_CENTER TURN LANE', 'TRAFF
ICWAY_TYPE_L-INTERSECTION', 'TRAFFICWAY_TYPE_NOT REPORTED', 'TRAFFICWAY_T
YPE_ROUNDABOUT', 'TRAFFICWAY_TYPE_TRAFFIC ROUTE', 'ALIGNMENT_CURVE ON GRA
DE', 'ALIGNMENT_CURVE ON HILLCREST', 'ALIGNMENT_CURVE, LEVEL', 'ALIGNMENT
_STRAIGHT ON HILLCREST', 'ROADWAY_SURFACE_COND_ICE', 'ROAD_DEFECT_SHOULDE
R DEFECT', 'DAMAGE_$500 OR LESS', 'WORK_ZONE_TYPE_UTILITY', 'MOST_SEVERE_
INJURY_INCAPACITATING INJURY', 'MOST_SEVERE_INJURY_NONINCAPACITATING INJU
RY', 'CRASH_MONTH_1', 'CRASH_MONTH_7', 'LANE_CNT_3']


Penalized features for output Impairment/Distraction:

['WORK_ZONE_I', 'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITAT
ING', 'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT', 'IN
JURIES_UNKNOWN', 'TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN', 'TRAFFIC
_CONTROL_DEVICE_DELINEATORS', 'TRAFFIC_CONTROL_DEVICE_NO PASSING', 'TRAFF
IC_CONTROL_DEVICE_OTHER', 'TRAFFIC_CONTROL_DEVICE_OTHER RAILROAD CROSSIN
G', 'TRAFFIC_CONTROL_DEVICE_PEDESTRIAN CROSSING SIGN', 'TRAFFIC_CONTROL_D
EVICE_POLICE/FLAGMAN', 'TRAFFIC_CONTROL_DEVICE_RAILROAD CROSSING GATE',
'TRAFFIC_CONTROL_DEVICE_RR CROSSING SIGN', 'TRAFFIC_CONTROL_DEVICE_YIEL
D', 'DEVICE_CONDITION_MISSING', 'DEVICE_CONDITION_OTHER', 'DEVICE_CONDITI
ON_WORN REFLECTIVE MATERIAL', 'WEATHER_CONDITION_FOG/SMOKE/HAZE', 'FIRST_
CRASH_TYPE_OTHER OBJECT', 'FIRST_CRASH_TYPE_OVERTURNED', 'FIRST_CRASH_TYP
E_REAR TO REAR', 'FIRST_CRASH_TYPE_SIDESWIPE SAME DIRECTION', 'FIRST_CRAS
H_TYPE_TRAIN', 'TRAFFICWAY_TYPE_DIVIDED - W/MEDIAN BARRIER', 'TRAFFICWAY_
TYPE_FIVE POINT, OR MORE', 'TRAFFICWAY_TYPE_L-INTERSECTION', 'TRAFFICWAY_
TYPE_NOT REPORTED', 'TRAFFICWAY_TYPE_ROUNDABOUT', 'ALIGNMENT_CURVE ON HIL
LCREST', 'ALIGNMENT_CURVE, LEVEL', 'ALIGNMENT_STRAIGHT ON GRADE', 'ALIGNM
ENT_STRAIGHT ON HILLCREST', 'ROADWAY_SURFACE_COND_DRY', 'ROAD_DEFECT_WORN
SURFACE', 'DAMAGE_OVER $1,500', 'WORK_ZONE_TYPE_NONE', 'WORK_ZONE_TYPE_UN
```

```
KNOWN', 'WORK_ZONE_TYPE_UTILITY', 'MOST_SEVERE_INJURY_NONINCAPACITATING I
NJURY', 'CRASH_MONTH_1', 'CRASH_MONTH_7', 'LANE_CNT_4']


Penalized features for output Outside Hazard:

['INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_REPORTED_NOT_EVIDENT', 'IN
JURIES_UNKNOWN', 'TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN', 'TRAFFIC
_CONTROL_DEVICE_FLASHING CONTROL SIGNAL', 'TRAFFIC_CONTROL_DEVICE_NO PASS
ING', 'TRAFFIC_CONTROL_DEVICE_OTHER', 'TRAFFIC_CONTROL_DEVICE_OTHER RAILR
OAD 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 PR
OPERLY', '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', 'T
RAFFICWAY_TYPE_CENTER TURN LANE', 'TRAFFICWAY_TYPE_L-INTERSECTION', 'TRAF
FICWAY_TYPE_NOT DIVIDED', 'TRAFFICWAY_TYPE_NOT REPORTED', 'TRAFFICWAY_TYP
E_ROUNDABOUT', 'TRAFFICWAY_TYPE_TRAFFIC ROUTE', 'ALIGNMENT_CURVE ON HILLC
REST', 'ALIGNMENT_STRAIGHT ON GRADE', 'ROADWAY_SURFACE_COND_DRY', 'ROAD_D
EFECT_SHOULDER DEFECT', 'ROAD_DEFECT_WORN SURFACE', 'DAMAGE_OVER $1,500',
'WORK_ZONE_TYPE_MAINTENANCE', 'WORK_ZONE_TYPE_UNKNOWN', 'WORK_ZONE_TYPE_U
TILITY', 'MOST_SEVERE_INJURY_INCAPACITATING INJURY', 'LANE_CNT_3', 'LANE_
CNT_4', 'LANE_CNT_Missing']


Penalized features for output Reckless Driving:

['WORK_ZONE_I', 'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIES_INCAPACITAT
ING', 'INJURIES_UNKNOWN', 'TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN',
'TRAFFIC_CONTROL_DEVICE_DELINEATORS', 'TRAFFIC_CONTROL_DEVICE_OTHER RAILR
OAD CROSSING', 'TRAFFIC_CONTROL_DEVICE_RAILROAD CROSSING GATE', 'TRAFFIC_
CONTROL_DEVICE_RR CROSSING SIGN', 'TRAFFIC_CONTROL_DEVICE_SCHOOL ZONE',
'TRAFFIC_CONTROL_DEVICE_YIELD', 'DEVICE_CONDITION_FUNCTIONING IMPROPERL
Y', 'DEVICE_CONDITION_FUNCTIONING PROPERLY', 'DEVICE_CONDITION_MISSING',
'DEVICE_CONDITION_OTHER', 'DEVICE_CONDITION_WORN REFLECTIVE MATERIAL', 'W
EATHER_CONDITION_CLOUDY/OVERCAST', 'WEATHER_CONDITION_FOG/SMOKE/HAZE', 'W
EATHER_CONDITION_FREEZING RAIN/DRIZZLE', 'WEATHER_CONDITION_RAIN', 'FIRST
_CRASH_TYPE_PEDESTRIAN', 'TRAFFICWAY_TYPE_DIVIDED - W/MEDIAN BARRIER', 'T
RAFFICWAY_TYPE_FIVE POINT, OR MORE', 'TRAFFICWAY_TYPE_L-INTERSECTION', 'T
RAFFICWAY_TYPE_NOT DIVIDED', 'TRAFFICWAY_TYPE_ROUNDABOUT', 'TRAFFICWAY_TY
PE_TRAFFIC ROUTE', 'ALIGNMENT_CURVE ON GRADE', 'ALIGNMENT_CURVE ON HILLCR
EST', 'ALIGNMENT_STRAIGHT ON HILLCREST', 'ROADWAY_SURFACE_COND_ICE', 'ROA
D_DEFECT_SHOULDER DEFECT', 'DAMAGE_$500 OR LESS', 'WORK_ZONE_TYPE_MAINTEN
ANCE', 'WORK_ZONE_TYPE_NONE', 'WORK_ZONE_TYPE_UTILITY', 'MOST_SEVERE_INJU
RY_INCAPACITATING INJURY', 'LANE_CNT_Missing']
```

8.3 Making Feature Selection Decisions Based on Penalization

I considered taking the information on penalized coefficients into a few different directions. Should I only drop the features that are penalized for all 4 target classes (see list below)? Should I drop the features from the smallest penalization list in order to preserve the dataset as much as possible?

Ultimately I decide to drop all of the penalized features for the 'Impairment/Distraction' class, since it's the class that the logistic regression classifier (and the naive classifiers) struggle to predict the most, based on per-class F1 metric.

In [54]:
```
common_penalized_cols = list(set.intersection(*map(set,
                                            [zero_coef[i] for i in range(4)])))
common_penalized_cols
```

Out[54]:
```
['TRAFFIC_CONTROL_DEVICE_OTHER RAILROAD CROSSING',
 'TRAFFICWAY_TYPE_ROUNDABOUT',
 'TRAFFIC_CONTROL_DEVICE_RR CROSSING SIGN',
 'ALIGNMENT_CURVE ON HILLCREST',
 'TRAFFIC_CONTROL_DEVICE_RAILROAD CROSSING GATE',
 'TRAFFIC_CONTROL_DEVICE_BICYCLE CROSSING SIGN',
 'TRAFFICWAY_TYPE_L-INTERSECTION',
 'WORK_ZONE_TYPE_UTILITY',
 'INJURIES_UNKNOWN']
```

In [55]:
```
to_remove = list(X_train.columns[(mlr.coef_[1] == 0)])

len(to_remove)
```

Out[55]:
```
45
```

In [56]:
```
X_train.drop(columns=to_remove, axis=1, inplace=True)
X_test.drop(columns=to_remove, axis=1, inplace=True)
```

In [57]:
```
X_train.head()
```

Out[57]:

	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	

9 Decision Tree

Eventually, 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 fine-tuned decision tree to get initial impressions.

9.1 Cross-Validation & Hyperparameter Tuning

In [58]:
```
tree_clf = DecisionTreeClassifier()
mean_dt_cv = np.mean(cross_val_score(tree_clf, X_train, y_train, cv=3))

print(f'Mean Cross-Validation Score: {mean_dt_cv :.2%}')
```

```
Mean Cross-Validation Score: 47.95%
```

The cross-validation score of about 47.93% isn't as high as I'd hope, especially considering how significant of a drop it is from the Multinomial Logistic Regression's accuracy of about 61%.

Next, I attempt to improve the decision tree performance by tuning the classifier's hyperparamters with *GridSearchCV()*

In [59]:
```
dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 10],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [5, 10, 15],
    'max_features': [None, 3, 5, 10],
}
```

In [60]:
```
dt_grid_search = GridSearchCV(tree_clf, dt_param_grid, cv=3, return_train_s
dt_grid_search.fit(X_train, y_train)
```

Out[60]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 10],
                         'max_features': [None, 3, 5, 10],
                         'min_samples_leaf': [5, 10, 15],
                         'min_samples_split': [5, 10]},
             return_train_score=True)
```

In [61]:
```
dt_gs_training_score = np.mean(dt_grid_search.cv_results_['mean_train_score
dt_gs_testing_score = dt_grid_search.score(X_test, y_test)

print(f'Mean Training Score: {dt_gs_training_score :.2%}')
print(f'Mean Test Score: {dt_gs_testing_score :.2%}')
print('Best Parameter Combination Found During Grid Search:')
dt_grid_search.best_params_
```

```
Mean Training Score: 48.39%
Mean Test Score: 59.18%
Best Parameter Combination Found During Grid Search:
```

Out[61]:
```
{'criterion': 'gini',
 'max_depth': 10,
 'max_features': None,
 'min_samples_leaf': 15,
 'min_samples_split': 5}
```

In the dictionary above, you can see that the exhaustive grid search cross-validation has determined what the decision tree's criterion, max_depth, max_features, min_samples_leaf &

min_samples_split should be, based on the options (*dt_param_grid*) it was fed. I use this information to construct a decision tree in the hopes that it at least outperforms the mean cross-validation score.

9.2 Exploring Decision Tree with Optimized Hyperparameters

In [62]:
```
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=10,
                                  min_samples_leaf = 15, min_samples_split
                                  random_state=26)
tree_clf.fit(X_train, y_train)
```

Out[62]:
```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=15, min_samples_spl
it=5,
                       random_state=26)
```

In [63]:
```
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: 0.29
Variable: TRAFFIC_CONTROL_DEVICE_NO CONTROLS Importance: 0.11
Variable: WEATHER_CONDITION_CLEAR Importance: 0.11
Variable: INJURIES_NO_INDICATION Importance: 0.07
Variable: HIT_AND_RUN_I        Importance: 0.06
Variable: FIRST_CRASH_TYPE_TURNING Importance: 0.05
Variable: CRASH_TYPE_NO INJURY / DRIVE AWAY Importance: 0.04
Variable: NUM_UNITS            Importance: 0.03
Variable: WEATHER_CONDITION_SNOW Importance: 0.03
Variable: FIRST_CRASH_TYPE_PARKED MOTOR VEHICLE Importance: 0.03
Variable: FIRST_CRASH_TYPE_REAR END Importance: 0.02
Variable: ROADWAY_SURFACE_COND_ICE Importance: 0.02
Variable: CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH Importance: 0.02
Variable: MOST_SEVERE_INJURY_NO INDICATION OF INJURY Importance: 0.02
Variable: INTERSECTION_RELATED_I Importance: 0.01
Variable: TRAFFIC_CONTROL_DEVICE_TRAFFIC SIGNAL Importance: 0.01
Variable: ROADWAY_SURFACE_COND_WET Importance: 0.01
Variable: ROAD_DEFECT_NO DEFECTS Importance: 0.01
Variable: POSTED_SPEED_LIMIT   Importance: 0.0
```

Above is an ordered list of feature importances for the decision tree. Only 18 features ended up having a noticeable (non-zero) importance to the classifier, with the 5 most important being:

- *FIRST_CRASH_TYPE_ANGLE*: Whether the first crash occured at an angle
- *TRAFFIC_CONTROL_DEVICE_NO CONTROLS*: Whether there was a traffic control device at the scene
- *WEATHER_CONDITION_CLEAR*: Whether it was clear/dry outside
- *INJURIES_NO_INDICATION*: Whether there was no indication of injuries at the accident
- *HIT_AND_RUN_I*: Whether the accident was a hit-and-run collision

I next use the decision tree to make predictions using both the training & test sets.

In [64]:
```
y_hat_train = tree_clf.predict(X_train)
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'Training Accuracy for Decision Tree Classifier: {(accuracy_score(y_
print(f'Testing Accuracy for Decision Tree Classifier: {(accuracy_score(y_t
```

```
[[3521  155  447  893]
 [ 276  905  540  965]
 [ 539  492 2703 1028]
 [ 840  275  713 3256]]
                                   precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.68      0.70      0.69       501
6
           Impairment/Distraction       0.50      0.34      0.40       268
6
                   Outside Hazard       0.61      0.57      0.59       476
2
                 Reckless Driving       0.53      0.64      0.58       508
4

                         accuracy                           0.59      1754
8
                        macro avg       0.58      0.56      0.57      1754
8
                     weighted avg       0.59      0.59      0.59      1754
8

Training Accuracy for Decision Tree Classifier: 60.7347745146461%
Testing Accuracy for Decision Tree Classifier: 59.18053339411898%
```

The training & test accuracies are within ~1.5% of each other, which is a great sign that there is no over/underfitting with this classifier. The accuracy score of about 60%, however, isn't optimal and is even a marginal decrease from the multinomial logistic regression classifier. Though it is worth acknowledging that an accuracy of 60% is still much more reliable than the naive guessing methods. For that reason, the classifier is still useful.

Based on per-class F1 scores, which take into account the opposing precision & recall scores, it seems the classifier is best at predicting (in descending order):

1. Ignoring Traffic Signs & Warnings
2. Outside Hazard
3. Reckless Driving
4. Impairment/Distraction

Let's see if expanding my scope to a random forest will verify the findings about feature importance and prediction metrics!

10 Random Forest

The process for my random forest classifier follows the same steps as the decision tree. Only slight cuts have been made (I've cut an option from this *param_grid* & don't ask to *return_train_score*) for the sake of improving processing speed.

10.1 Cross-Validation & Hyperparameter Tuning

In [65]:
```
rf_clf = RandomForestClassifier()
mean_rf_cv = np.mean(cross_val_score(rf_clf, X_train, y_train, cv=3))

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

```
Mean Cross Validation Score for Random Forest Classifier: 58.69%
```

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, 15],
}
```

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

Out[67]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 10],
                         'max_features': [3, 5, 10, 15],
                         'min_samples_leaf': [5, 10, 15],
                         'min_samples_split': [5, 10]})
```

In [68]:
```
rf_gs_testing_score = rf_grid_search.score(X_test, y_test)

print(f'Mean Test Score: {rf_gs_testing_score :.2%}')
print('Best Parameter Combination Found During Grid Search:')
rf_grid_search.best_params_
```

```
Mean Test Score: 60.64%
Best Parameter Combination Found During Grid Search:
```

Out[68]:
```
{'criterion': 'gini',
 'max_depth': 10,
 'max_features': 15,
 'min_samples_leaf': 5,
 'min_samples_split': 10}
```

10.2 Exploring Random Forest with Optimized Hyperparameters

Now that the best parameter combination has been found above, I create the final, optimized random forest.

In [69]:
```
forest = RandomForestClassifier(criterion='gini', n_estimators=100,
                                max_depth=10, max_features=15,
                                min_samples_leaf=5, min_samples_split=10,
                                random_state=26)
forest.fit(X_train, y_train)
```

Out[69]:
```
RandomForestClassifier(max_depth=10, max_features=15, min_samples_leaf=5,
                       min_samples_split=10, random_state=26)
```

In [70]:
```
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: 0.06
Variable: TRAFFIC_CONTROL_DEVICE_NO CONTROLS Importance: 0.06
Variable: WEATHER_CONDITION_CLEAR Importance: 0.06
Variable: DEVICE_CONDITION_FUNCTIONING PROPERLY Importance: 0.05
Variable: FIRST_CRASH_TYPE_REAR END Importance: 0.05
Variable: HIT_AND_RUN_I        Importance: 0.04
Variable: NUM_UNITS            Importance: 0.04
Variable: INJURIES_NO_INDICATION Importance: 0.04
Variable: DEVICE_CONDITION_NO CONTROLS Importance: 0.04
Variable: WEATHER_CONDITION_SNOW Importance: 0.04
Variable: ROADWAY_SURFACE_COND_SNOW OR SLUSH Importance: 0.04
Variable: CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH Importance: 0.04
Variable: FIRST_CRASH_TYPE_PARKED MOTOR VEHICLE Importance: 0.03
Variable: FIRST_CRASH_TYPE_TURNING Importance: 0.03
Variable: CRASH_TYPE_NO INJURY / DRIVE AWAY Importance: 0.03
Variable: FIRST_CRASH_TYPE_FIXED OBJECT Importance: 0.02
Variable: ROADWAY_SURFACE_COND_ICE Importance: 0.02
Variable: TRAFFIC_CONTROL_DEVICE_STOP SIGN/FLASHER Importance: 0.01
```

With the random forest, I now have 25 features with noticeable (non-zero) importances. Let's take a look at the five most important features for prediction and see how they compare to those of the decision tree:

- *FIRST_CRASH_TYPE_ANGLE*: Whether the first crash occured at an angle
- *INTERSECTION_RELATED_I*: Whether an intersection was related to the accident
- *TRAFFIC_CONTROL_DEVICE_NO CONTROLS*: Whether there was a traffic control device at the scene
- *WEATHER_CONDITION_CLEAR*: Whether it was clear/dry outside
- *DEVICE_CONDITION_FUNCTIONING PROPERLY*: Whether the traffic control device at the scene was functioning properly

The 5 most important for both classifiers are similar, and *FIRST_CRASH_TYPE_ANGLE* is still the most important feature for prediction. But the *INTERSECTION_RELATED_I* & *DEVICE_CONDITION_FUNCTIONING PROPERLY* columns have both risen into this top tier since the decision tree was run.

Next, I look at the predictive performance of the decision tree classifier on both training and test sets.

In [71]:
```
y_hat_train = forest.predict(X_train)
y_hat_test = forest.predict(X_test)

print('Confusion matrix for RF:', '\n', confusion_matrix(y_test, y_hat_test
print(classification_report(y_test, y_hat_test))
print(f'Training Accuracy for Random Forest Classifier: {(accuracy_score(y_
print(f'Testing Accuracy for Random Forest Classifier: {(accuracy_score(y_t
```

```
Confusion matrix for RF:
 [[3458  163  442  953]
 [ 242  903  499 1042]
 [ 511  390 2808 1053]
 [ 736  231  663 3454]]
                                  precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.70      0.69      0.69       501
6
           Impairment/Distraction       0.54      0.34      0.41       268
6
                   Outside Hazard       0.64      0.59      0.61       476
2
                 Reckless Driving       0.53      0.68      0.60       508
4

                         accuracy                           0.61      1754
8
                        macro avg       0.60      0.57      0.58      1754
8
                     weighted avg       0.61      0.61      0.60      1754
8

Training Accuracy for Random Forest Classifier: 61.6161999924015%
Testing Accuracy for Random Forest Classifier: 60.53681331205836%
```

Expanding into a random forest of 100 trees gives a marginal boost to most of the metrics I'm looking closely at, from training/test accuracies to per-class/avg F1 score. Still, this classifier's performance also falls just short of the multinomial logistic regression.

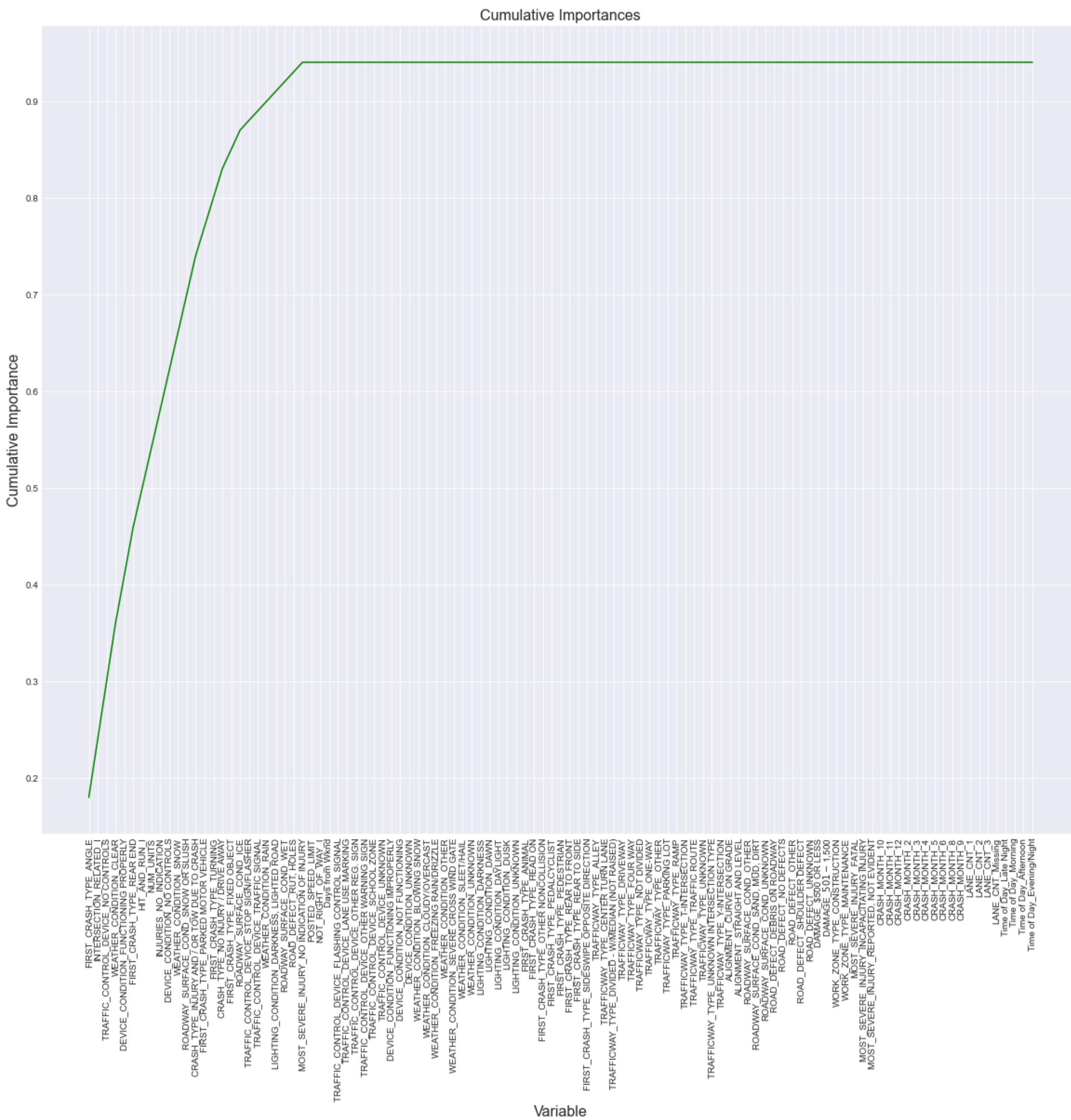
At the very least, this classifier has given an expanded list of import features to keep, which should improve the next model performance compared to the smaller list of features to keep based on the decision tree.

In the plot below, you should be able to tell that the cumulative feature importance plateaus after the list of non-zero features above. Based on this, I decide to drop all features that have minimal (zero) importance to the random forest.

In [73]:

```
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.style.use('seaborn-darkgrid')
plt.plot(x_values, cumulative_importances, 'g-')
plt.xticks(x_values, sorted_features, rotation = 'vertical')
plt.xlabel('Variable', fontsize=16)
plt.ylabel('Cumulative Importance', fontsize=16)
plt.title('Cumulative Importances', fontsize=16)
plt.savefig('CumulativeImportance.png')
```



In [74]:
```
significant = sorted_features[:25]

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

11 XGBoost

Now that I've tried a variety of classification algorithms, I opt for the gradient-boosting XGBoost library. My hope is that XGBoost's ability to combine predictive results and make adjustments along the way (as opposed to, say, random forest's tallying of all results at the end of running) will produce higher scores than the other classifiers.

In this section, I start with a baseline model before exhaustively optimizing hyperparameters using *GridSearchCV*. Then, I run a final XGBoost model with the optimal inputs before making observations on its performance

11.1 Initial XGBoost

In [75]:
```
xg = XGBClassifier()

xg.fit(X_train_red, y_train)
```

Out[75]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [77]:
```
y_hat_train = xg.predict(X_train_red)
y_hat_test = xg.predict(X_test_red)

print('Confusion matrix for XG:', '\n', confusion_matrix(y_test, y_hat_test
print(classification_report(y_test, y_hat_test))
print(f'Training Accuracy for XGBoost: {(accuracy_score(y_train, y_hat_trai
print(f'Testing Accuracy for XGBoost: {(accuracy_score(y_test, y_hat_test))
```

```
Confusion matrix for XG:
 [[3435  217  442  922]
 [ 231 1048  465  942]
 [ 485  491 2779 1007]
 [ 747  338  635 3364]]
                                  precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.70      0.68      0.69      501
6
           Impairment/Distraction       0.50      0.39      0.44      268
6
                   Outside Hazard       0.64      0.58      0.61      476
2
                  Reckless Driving       0.54      0.66      0.59      508
4

                         accuracy                           0.61     1754
8
                        macro avg       0.60      0.58      0.58     1754
8
                     weighted avg       0.61      0.61      0.60     1754
8

Training Accuracy for XGBoost: 62.524220204399526%
Testing Accuracy for XGBoost: 60.55390927741053%
```

11.2 Cross-Validation & Hyperparameter Tuning

In [78]:
```
param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [5, 10],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100],
}
```

In [79]:
```
xg_grid_search = GridSearchCV(xg, param_grid, scoring='accuracy', cv=None,
xg_grid_search.fit(X_train_red, y_train)
```

Out[79]:
```
GridSearchCV(estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynod
e=1,
                                     colsample_bytree=1, gamma=0, gpu_id=
-1,
                                     importance_type='gain',
                                     interaction_constraints='',
                                     learning_rate=0.300000012,
                                     max_delta_step=0, max_depth=6,
                                     min_child_weight=1, missing=nan,
                                     monotone_constraints='()',
                                     n_estimators=100, n_jobs=0,
                                     num_parallel_tree=1,
                                     objective='multi:softprob', random_s
tate=0,
                                     reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=None, subsample=1,
                                     tree_method='exact', validate_parame
ters=1,
                                     verbosity=None),
             n_jobs=1,
             param_grid={'learning_rate': [0.1, 0.2], 'max_depth': [5, 1
0],
                         'min_child_weight': [1, 2], 'n_estimators': [10
0],
                         'subsample': [0.5, 0.7]},
             scoring='accuracy')
```

In [80]:
```
best_parameters = xg_grid_search.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))
```

```
Grid Search found the following optimal parameters:
learning_rate: 0.1
max_depth: 5
min_child_weight: 2
n_estimators: 100
subsample: 0.7
```

11.3 XGBoost with Optimized Hyperparameters

Now that the optimal parameters have been given, I construct one last XGBoost classifier and see if performance has improved over the initial model.

In [82]:
```
xg = XGBClassifier(learning_rate=0.1, max_depth=5,
                       min_child_weight=2, n_estimators=100,
                       subsample=0.7)
xg.fit(X_train_red, y_train)
```

Out[82]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=2, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=0.7,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [83]:
```
y_hat_train = xg.predict(X_train_red)
y_hat_test = xg.predict(X_test_red)

print('Confusion matrix for XG:', '\n', confusion_matrix(y_test, y_hat_test
print(classification_report(y_test, y_hat_test))
print(f'Training Accuracy for XGBoost: {(accuracy_score(y_train, y_hat_trai
print(f'Testing Accuracy for XGBoost: {(accuracy_score(y_test, y_hat_test))
```

```
Confusion matrix for XG:
 [[3435  215  429  937]
 [ 231 1036  461  958]
 [ 474  501 2780 1007]
 [ 750  322  619 3393]]
                                   precision    recall  f1-score   suppor
t

Ignoring Traffic Signs & Warnings       0.70      0.68      0.69       501
6
          Impairment/Distraction       0.50      0.39      0.44       268
6
                  Outside Hazard       0.65      0.58      0.61       476
2
                Reckless Driving       0.54      0.67      0.60       508
4

                        accuracy                           0.61      1754
8
                       macro avg       0.60      0.58      0.58      1754
8
                    weighted avg       0.61      0.61      0.60      1754
8

Training Accuracy for XGBoost: 61.12229778503856%
Testing Accuracy for XGBoost: 60.65648506952359%
```

The performance of the optimized XGBoost is quite similar to the optimized random forest, with an accuracy of 60.65% on the test data (60.5% for random forest) and no sign of over/underfitting. Additionally, the the 'Impairment/Distraction' F1 score is improved over the decision tree & random forest models.

Once more, however, the model's overall accuracy and per-class F1 scores are slightly inferior to those of the multinomial logistic regression classifier, which seems to be the best model for this dataset.

In [84]:
```
feature_list = list(X_train_red.columns)
importances = list(xg.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: 0.23999999463558197
Variable: TRAFFIC_CONTROL_DEVICE_NO CONTROLS Importance: 0.11999999731779
099
Variable: WEATHER_CONDITION_CLEAR Importance: 0.10000000149011612
Variable: FIRST_CRASH_TYPE_TURNING Importance: 0.07999999821186066
Variable: FIRST_CRASH_TYPE_PARKED MOTOR VEHICLE Importance: 0.05999999865
889549
Variable: FIRST_CRASH_TYPE_REAR END Importance: 0.05000000074505806
Variable: CRASH_TYPE_INJURY AND / OR TOW DUE TO CRASH Importance: 0.05000
000074505806
Variable: ROADWAY_SURFACE_COND_ICE Importance: 0.03999999910593033
Variable: HIT_AND_RUN_I        Importance: 0.029999999329447746
Variable: NUM_UNITS            Importance: 0.029999999329447746
Variable: WEATHER_CONDITION_SNOW Importance: 0.029999999329447746
Variable: ROADWAY_SURFACE_COND_SNOW OR SLUSH Importance: 0.02999999932944
7746
Variable: FIRST_CRASH_TYPE_FIXED OBJECT Importance: 0.029999999329447746
Variable: INTERSECTION_RELATED_I Importance: 0.019999999552965164
Variable: ROADWAY_SURFACE_COND_WET Importance: 0.019999999552965164
Variable: ROAD_DEFECT_RUT, HOLES Importance: 0.019999999552965164
Variable: INJURIES_NO_INDICATION Importance: 0.009999999776482582
Variable: DEVICE_CONDITION_NO CONTROLS Importance: 0.009999999776482582
Variable: TRAFFIC_CONTROL_DEVICE_STOP SIGN/FLASHER Importance: 0.00999999
9776482582
Variable: TRAFFIC_CONTROL_DEVICE_TRAFFIC SIGNAL Importance: 0.00999999977
6482582
Variable: WEATHER_CONDITION_RAIN Importance: 0.009999999776482582
Variable: LIGHTING_CONDITION_DARKNESS, LIGHTED ROAD Importance: 0.0099999
99776482582
Variable: MOST_SEVERE_INJURY_NO INDICATION OF INJURY Importance: 0.009999
999776482582
Variable: DEVICE_CONDITION_FUNCTIONING PROPERLY Importance: 0.0
Variable: CRASH_TYPE_NO INJURY / DRIVE AWAY Importance: 0.0
```

Out[84]:
```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
```

```
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None]
```

Before I conclude, let's take a look at the 5 most important features for XGBoost and see how they compare to those of the decision tree & random forest.

- *FIRST_CRASH_TYPE_ANGLE*: Whether the first crash occured at an angle
- *TRAFFIC_CONTROL_DEVICE_NO CONTROLS*: Whether there was a traffic control device at the scene
- *WEATHER_CONDITION_CLEAR*: Whether it was clear/dry outside
- *FIRST_CRASH_TYPE_TURNING*: Whether the first crash involved a turning vehicle
- *FIRST_CRASH_TYPE_PARKED MOTOR VEHICLE*: Whether the first crash involved a parked vehicle

The top 3 most important features should be familiar by this point, but it is interesting that the XGBoost classifier has given more importance to a few of the *FIRST_CRASH_TYPE* dummy columns.

12 Results

The Multinomial Logistic Regression classifier performs best out of all of my classifiers (though not by much), with an accuracy score of about 61.4% on test data & a weighted average F1 score of 0.61. Though these figures aren't necessarily optimal, the model is still about 2.5 times more accurate than the project's naive "random guesses" (whether uniform or based on class counts), and there was no evidence of over/underfitting throughout the project.

The classifier's cause predictions are, in order from most to least dependable (based on per-class F1): 'Ignoring Traffic Signs & Warnings', 'Outside Hazard', 'Reckless Driving', & 'Impairment/Distraction.' In other words, it seems best at predicting whether a crash is primarily caused by ignoring traffic signs & warnings, and worst at predicting whether a crash is primarily caused by impairment/distraction.

Finally, the following features had the biggest predictive importance across multiple classifiers, and should be considered important aspects in determining the cause of an auto accident:

- First crash type: at an angle, during a turn, collision with a parked vehicle, rear end collision
- Outdoor conditions: clear, snowy
- Whether there was a traffic control device at the scene
- Whether the accident involved a personal injury or a towed vehicle
- Whether the accident was a hit-and-run
- Road surface conditions: ice, snow/slush, wet
- Number of parties involved
- Whether the accident occured at an intersection

13 Conclusions

Based on my findings, I would recommend the following to USAA:

- **If a client's accident cause is determined to be 'Ignoring Traffic Signs & Warnings': Consider increasing their premium.**
- **If a client's accident cause is determined to be 'Outside Hazard': Consider minimal or no increase to the premium, as the accident has been caused by something out of the client's control.**
- **If a client's accident cause is determined to be 'Reckless Driving': Consider increasing their premium.** Additionally, flag the client as a candidate to drop if the reckless driving continues. It is harder to write off such causes as accidental or momentary, like one conceivably could with 'Ignoring Traffic Signs & Warnings.'
- **If a client's accident cause is determined to be 'Impairment/Distraction': Do not make any conclusions about the case, and use the prediction as a prompt to investigate more closely.** This is due to both the class' poor performance metrics & its relative ambiguity ('Impairment/Distraction' covers everything from drunk driving to medical emergencies while operating a vehicle)

14 Future Work

Given more time, I would do the following:

- **Look closely at coefficient penalties for my logistic regression model & determine which features are most impactful for each specific class**. Additionally, this exploration could determine which 'impact features' all of the classes share.
- **Inspect the city's dataset on people involved in these same accidents.** Using common indices across the two datasets, I could create a feature that measures how many people were in the vehicles in each wreck and explore the impact that this feature has on the target.
- **Look into whether I could break the 'Impairment/Distraction' class down any more**. Perhaps, as this class undergoes specification, the performance for both the class(es) and overall model would improve significantly.