Chicago Car Crashes Exploration

Authors: Lenore Perconti, Jakub Rybicki, Noble Tang

Flatiron Data Science School, Phase Three Project

10/28/21

Overview

In this project we looked at traffic incident data from the City of Chicago. We joined two data sets, processed them, and used modeling to infer information on traffic incidents at night.

Business Understanding

Our stakeholder is City of Chicago Department of Transportation. They are interested in learning more about what factors contribute to severe traffic incidents for drivers at night.

- **Severe** traffic incidents we defined as `FATAL` or `INCAPACITATING` from the `INJURY_TYPE` column.
- **Night** we defined as the hours between 10pm to 5 am, or hours 22 through 5 in the `CRASH_HOUR` column.

Data Understanding and Cleaning

The data used was sourced from the following websites:

https://data.cityofchicago.org/Transportation/Traffic-Crashes-Crashes/85ca-t3if

https://data.cityofchicago.org/Transportation/Traffic-Crashes-People/u6pd-qa9d

The data was downloaded on Friday, October 22. Attempts to reproduce this notebook may result in inconcistancies since the online source is updated weekly.

These datasets are include very recent data and information from 2013 to present. The number of variables and columns in this dataset made it challenging to clean and use, however the broad scope of the data makes it a good candidate for exploring the business problem at hand.

The large size of the original datasets were too large to upload to github, we create a cleaned and merged dataset below that is included in the dataset.

In [1]:
```
#Importing the neccessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, f
from sklearn.metrics import roc_curve, classification_report, plot_roc_curve
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as imb_Pipeline
```

In [2]:
```
crash_df = pd.read_csv('data/Traffic_Crashes_-_crashes.csv')
people_df = pd.read_csv('data/Traffic_Crashes_-_people.csv', low_memory=False)
```

Dropping Unneccessary Columns

`Crash_df` dropping Justification:

- `RD_NO` - Police Dep. Report number, another identifying number associated with each record, we kept CRASH_RECORD_ID as the joining record number for each dataframe.
- `CRASH_DATE_EST_I` - used when crash is reported to police days after the crash, this dataframe inclues crash day of week, hour and month so we can drop the specific date.
- `CRASH_DATE` - this dataframe inclues crash day of week, hour and month so we can drop the specific date.
- `REPORT_TYPE` - administrative report type, not a factor relevant to causing a crash.
- `HIT_AND_RUN_I` - not a factor relevant to causing a crash.
- `DATE_POLICE_NOTIFIED` - not a factor relevant to causing a crash.
- `BEAT_OF_OCCURENCE` - not a factor relevant to causing a crash.
- `PHOTOS_TAKEN_I` - not a factor relevant to causing a crash.
- `STATEMENTS_TAKEN` - not a factor relevant to causing a crash.
- `LANE_COUNT` - Dropping lane count because we found too many null values that we don't want to skew data with mean/median, and don't want to assume a distribution for synthetic data

Basing our severity of injury off of information from the people_df dataframe, including this and other injury related columns would cause multicolliniarity in our modeling.

- `MOST_SEVERE_INJURY` -
- `INJURIES_FATAL`
- `INJURIES_NON_INCAPACITATING`
- `INJURIES_REPORTED_NOT_EVIDENT`
- `INJURIES_NO_INDICATION`
- `INJURIES_UNKNOWN`

Location related info we dropped - not enough time for the scope of this project

- `LONGITUDE`

- LATITUDE
- STREET_NO

In [3]:
```
crash_df_cleaned = crash_df[['CRASH_RECORD_ID', 'TRAFFIC_CONTROL_DEVICE', 'DEVIC
                             'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'ROADWAY
                             'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH']]
```

people_df dropping dustification:

- PERSON_ID - unique ID for each person record, will use CRASH_RECORD_ID
- RD_NO - Police Dep. Report number, another identifying number associated with each record, we kept CRASH_RECORD_ID as the joining record number for each dataframe.
- VEHICLE_ID - another indentifying factor we don't need
- CRASH_DATE - time records coming from crash.csv dataset
- SEAT_NO - too small representation .09% of dataset

Location info not looked into:

- CITY
- STATE
- ZIPCODE

- SEX - not relevant to causing a crash
- DRIVER_LICENCE_STATE - not a factor relevant
- DRIVER_LICENCE_CLASS - not a factor relevant
- SAFETY_EQUIPMENT - included safety equipment worn by pedestrians, cyclists, etc. would have been too time consuming to wade through.
- AIRBAG_DEPLOYED - not a factor relevant
- EJECTION not a factor relevant

Hospital and EMS info not relevant to learning about causes of crash:

- HOSPITAL
- EMS_AGENCY
- EMS_RUN_NO

- DRIVER_VISION - 40% of data is unknown vision. Hard to make assumptions fairly
- PHYSICAL_CONDITION - condition of the driver after the accident does not play a role in causation
- PEDPEDAL_ACTION - action of a pedestrian varies between instances. Holds no info that city could change
- PEDPEDAL_VISIBILITY - clothing of the pedestrian holds no info that city could enforce
- PEDPEDAL_LOCATION - location of the pedestrian at time of crash holds no info that city could enforce
- CELL_PHONE_USE - not enough data to utlize

In [4]:
```
people_df_cleaned = people_df[['CRASH_RECORD_ID', 'AGE',
                               'BAC_RESULT VALUE', 'INJURY_CLASSIFICATION', 'PER
```

Getting only incidents that occured at night:

Night = between 10 pm and 6 am, these are the nighttime hours defined by licencing.

In [5]:
```
night_time_df = crash_df_cleaned.copy()
night_time_df = night_time_df[(night_time_df['CRASH_HOUR'] >= 22) | (night_time_
night_time_df.columns
```

Out[5]:
```
Index(['CRASH_RECORD_ID', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'ROADWAY_SURFACE_COND',
       'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH'],
      dtype='object')
```

Joining the two data sets

In [6]:
```
#checking the shape
night_time_df.shape, people_df_cleaned.shape
```

Out[6]:
```
((93448, 9), (1224613, 5))
```

In [7]:
```
merge = pd.merge(night_time_df, people_df_cleaned, how='left', on='CRASH_RECORD_
merge.shape
```

Out[7]:
```
(188733, 13)
```

In [8]:
```
#no longer need Crash ID - not useful for modeling
merge = merge.drop(columns=['CRASH_RECORD_ID'])
```

Exploring Columns Further

Target Variable: `INJURY_CLASSIFICATION`

- This includes all people involved in incident, cyclists, passengers, drivers, etc.

In [9]:
```
#taking a look at Injury Classification
merge['INJURY_CLASSIFICATION'].value_counts()
```

Out[9]:
```
NO INDICATION OF INJURY     166735
NONINCAPACITATING INJURY     12730
REPORTED, NOT EVIDENT         5501
INCAPACITATING INJURY         2834
FATAL                          310
Name: INJURY_CLASSIFICATION, dtype: int64
```

In [10]:
```
#Making Injury Classification into a binary to indicate "serious" incidents
# fatal and incapacitate = 1
merge.loc[(merge['INJURY_CLASSIFICATION'] == 'FATAL') |
            (merge['INJURY_CLASSIFICATION'] == 'INCAPACITATING INJURY') |
            (merge['INJURY_CLASSIFICATION'] == 'NONINCAPACITATING INJURY') |
            (merge['INJURY_CLASSIFICATION'] == 'REPORTED, NOT EVIDENT'), 'INJURY_

# else = 0
merge.loc[(merge['INJURY_CLASSIFICATION'] == 'NO INDICATION OF INJURY'), 'INJURY

merge['INJURY_CLASSIFICATION'].fillna(0, inplace=True)
```

In [11]:
```
#normalizing Injury Classification
```

```
merge["INJURY_CLASSIFICATION"].value_counts(normalize=True)
```

Out[11]:
```
0    0.886745
1    0.113255
Name: INJURY_CLASSIFICATION, dtype: float64
```

Traffic control device

Transforming traffic control device into a new column, 0 for no control device or a malfunctioning device, 1 for control device functioning properly

In [12]:
```
merge.loc[merge['TRAFFIC_CONTROL_DEVICE'] == 'NO CONTROLS', 'TRAFFIC_CONTROL_DEV
merge.loc[merge['TRAFFIC_CONTROL_DEVICE'] != 0, 'TRAFFIC_CONTROL_DEVICE'] = 1

merge.loc[merge.DEVICE_CONDITION == 'FUNCTIONING PROPERLY', 'DEVICE_CONDITION']
merge.loc[merge.DEVICE_CONDITION != 1, 'DEVICE_CONDITION'] = 0

merge['DEVICE_CONDITION'] = merge['DEVICE_CONDITION'].astype(float)
merge['TRAFFIC_CONTROL_DEVICE'] = merge['TRAFFIC_CONTROL_DEVICE'].astype(float)
```

Weather

- categorized 1 as clear weather, rest as 0 for unfavorable weather

In [13]:
```
# 1 is clear
merge.loc[merge['WEATHER_CONDITION'] == 'CLEAR', 'WEATHER_CONDITION'] = 1

# 0 is not clear
merge.loc[merge['WEATHER_CONDITION'] != 1, 'WEATHER_CONDITION'] = 0

merge['WEATHER_CONDITION'] = merge['WEATHER_CONDITION'].astype(float)
```

Roadway Surface Condition

- Binned `OTHER` with `UNKNOWN`

In [14]:
```
merge.loc[merge['ROADWAY_SURFACE_COND'] == 'OTHER', 'ROADWAY_SURFACE_COND'] = 'U
```

In [15]:
```
merge['ROADWAY_SURFACE_COND'].value_counts()
```

Out[15]:
```
DRY                  135461
WET                   31526
UNKNOWN               13430
SNOW OR SLUSH          6527
ICE                    1712
SAND, MUD, DIRT          77
Name: ROADWAY_SURFACE_COND, dtype: int64
```

Age

`AGE` had a lot of outliers including some negative ages and zeros which likely were typos. To limit these outliers we dropped them.

In [16]:
```
merge.loc[merge['AGE'] <= 0, 'AGE'] = None
```

In [17]:
```
merge.dropna(subset=['AGE'], inplace=True)
```

Driver

Limited dataset to just show `PERSON_TYPE = DRIVER`. Pedestrians, cyclists and other people types would not have influence over a car crash as much as a driver would.

In [18]:
```
merge = merge.loc[merge['PERSON_TYPE'] == 'DRIVER']
```

In [19]:
```
merge.drop(columns=['PERSON_TYPE'], inplace=True)
```

Blood Alcohol Content

For this column we binned the data into 1 = over the legal limit (.08) and 0 = under the limit.

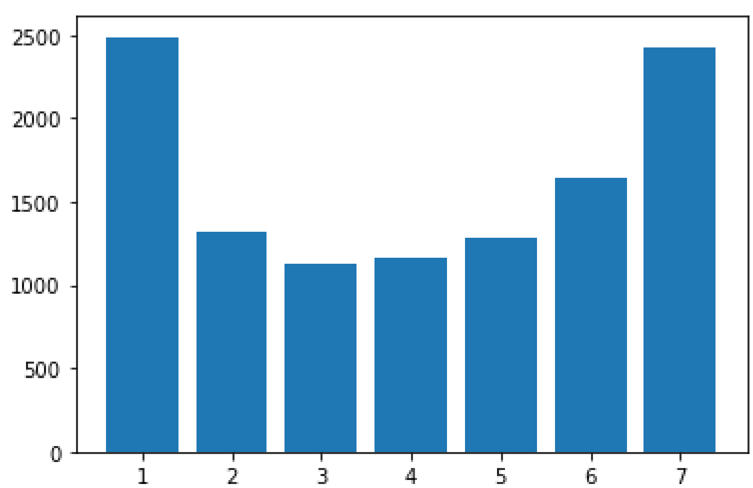
In [20]:
```
merge['BAC_RESULT VALUE'].fillna(0, inplace=True)

# 1 is drunk
merge.loc[merge['BAC_RESULT VALUE'] >= 0.08, 'BAC_RESULT VALUE'] = 1

# 0 is not drunk
merge.loc[merge['BAC_RESULT VALUE'] < 0.08, 'BAC_RESULT VALUE'] = 0
```

Day of week

Binned weekends and weekdays, we saw in the histogram crashes occured on weekends more than weekdays.

In [21]:
```
#Plotting for the presentation to show Day of Week crashes
injury_df = merge.copy()

injury_df = injury_df.loc[injury_df['INJURY_CLASSIFICATION'] == 1]

fig, ax = plt.subplots()

ax.bar(list(injury_df['CRASH_DAY_OF_WEEK'].value_counts().index),
       injury_df['CRASH_DAY_OF_WEEK'].value_counts().values)
```

Out[21]: `<BarContainer object of 7 artists>`



In [22]:
```
# binning weekends and weekday nights

# 1 value is a weekend night
merge.loc[merge['CRASH_DAY_OF_WEEK'] >= 6, 'CRASH_DAY_OF_WEEK'] = 1
```

```
# 0 value is a weekday night
merge.loc[merge['CRASH_DAY_OF_WEEK'] != 1, 'CRASH_DAY_OF_WEEK'] = 0
```

Compile Final DF

Exporting the `final_df` into csv file

In [23]:
```
final_df = merge.copy()
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 95484 entries, 3 to 188727
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   TRAFFIC_CONTROL_DEVICE   95484 non-null  float64
 1   DEVICE_CONDITION         95484 non-null  float64
 2   WEATHER_CONDITION        95484 non-null  float64
 3   LIGHTING_CONDITION       95484 non-null  object 
 4   ROADWAY_SURFACE_COND     95484 non-null  object 
 5   CRASH_HOUR               95484 non-null  int64  
 6   CRASH_DAY_OF_WEEK        95484 non-null  int64  
 7   CRASH_MONTH              95484 non-null  int64  
 8   AGE                      95484 non-null  float64
 9   BAC_RESULT VALUE         95484 non-null  float64
 10  INJURY_CLASSIFICATION    95484 non-null  int64  
dtypes: float64(5), int64(4), object(2)
memory usage: 8.7+ MB
```

In [24]:
```
#This is where the clean_data.csv is created and can be found in the github.
#clean_data = final_df.to_csv('clean_data.csv', index = False)
```

Modeling

For all our models we built a pipeline that accomplished the following:

- Column Transformers One Hot Encoded categorical columns
- Used SMOTE strategy to synthesize new examples for the minority class and address class imbalance
- Perform a grid search to find optimal

First Model

Smote Oversampling + Decision Tree

This model shows a tree that helps us find the features that we are interested in

In [25]:
```
X = final_df.drop(columns=['INJURY_CLASSIFICATION'])
y = final_df['INJURY_CLASSIFICATION']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=11)
```

In [54]:
```
# Create a column transformer
col_transformer = ColumnTransformer(transformers=[
    ('ohe', OneHotEncoder(categories='auto', handle_unknown='ignore'), ['LIGHTIN
], remainder='passthrough')

over = SMOTE(sampling_strategy='minority')
#under = RandomUnderSampler(sampling_strategy=0.5)

# Create a pipeline containing the column transformer and model
pipeline = imb_Pipeline(steps=[
    ('col_transformer', col_transformer),
    ('o', over),
    ('classifier', DecisionTreeClassifier(random_state=11))
])
```

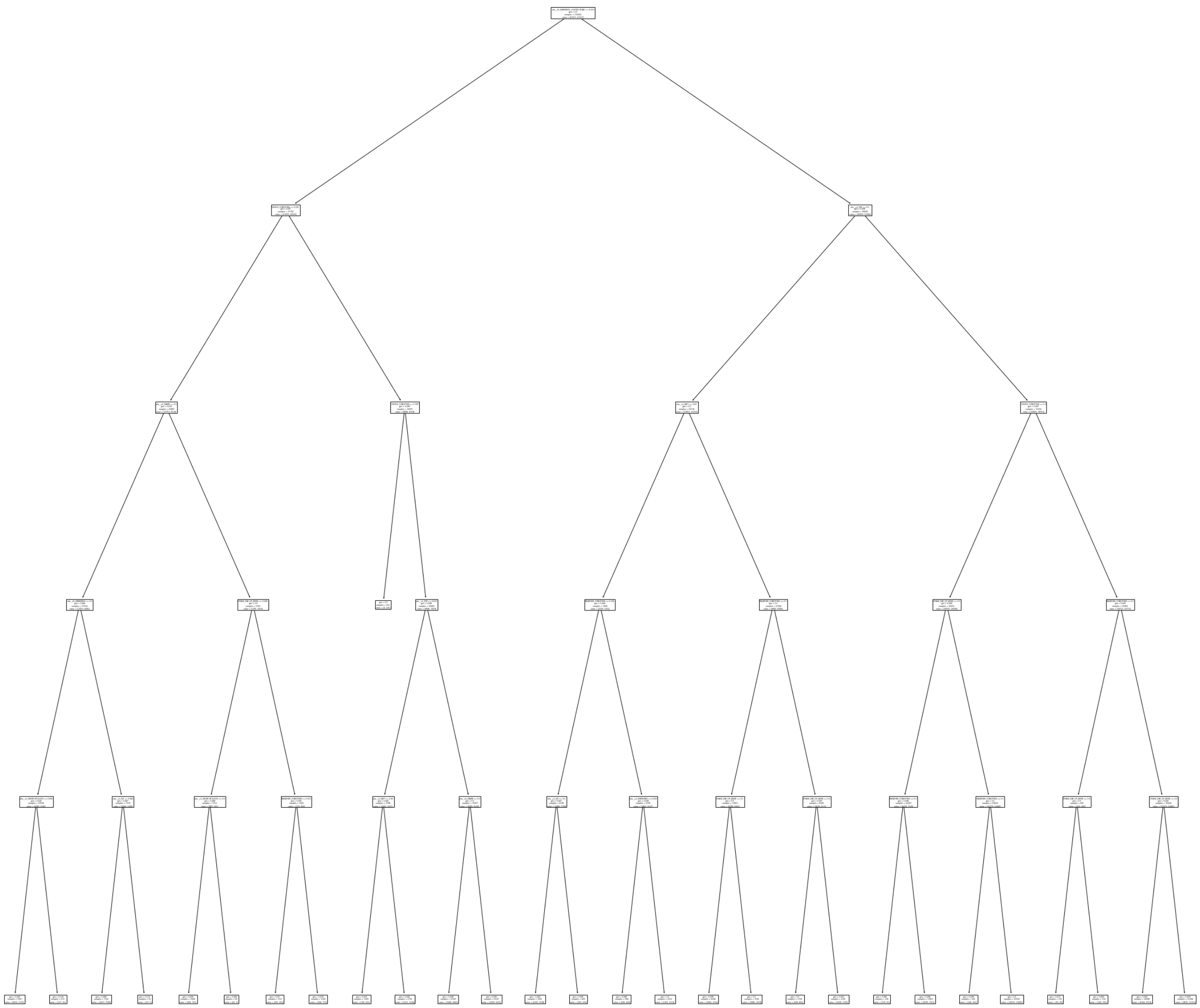
In [55]:
```
param_grid = [{'classifier__max_depth':[1, 3, 5]}]

grid_search = GridSearchCV(estimator=pipeline,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5
                           )

grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)
cv_score_smoted = grid_search.best_score_
test_score_smoted = grid_search.score(X_test, y_test)
```

In [56]:
```
pipeline = imb_Pipeline(steps=[
    ('col_transformer', col_transformer),
    ('o', over),
    ('u', under),
    ('classifier', DecisionTreeClassifier(random_state=11, max_depth=5))
])

fig, ax = plt.subplots(figsize=(40, 40))

pipeline.fit(X_train, y_train)
feature_list = pipeline['col_transformer'].get_feature_names()
plot_tree(pipeline['classifier'], ax=ax, feature_names=feature_list)
```

Out[56]:
```
[Text(1066.1785714285713, 1993.2, 'ohe__x0_DARKNESS, LIGHTED ROAD <= 0.259\ngini
= 0.5\nsamples = 134454\nvalue = [67227, 67227]'),
 Text(538.0714285714286, 1630.8000000000002, 'DEVICE_CONDITION <= 0.001\ngini =
0.49\nsamples = 37762\nvalue = [21599, 16163]'),
 Text(318.85714285714283, 1268.4, 'ohe__x0_DAWN <= 0.5\ngini = 0.476\nsamples =
20887\nvalue = [12753, 8134]'),
 Text(159.42857142857142, 906.0, 'ohe__x0_DARKNESS <= 0.5\ngini = 0.466\nsamples
= 17534\nvalue = [11053, 6481]'),
 Text(79.71428571428571, 543.5999999999999, 'ohe__x1_SNOW OR SLUSH <= 0.935\ngin
i = 0.445\nsamples = 10160\nvalue = [6769, 3391]'),
 Text(39.857142857142854, 181.19999999999982, 'gini = 0.449\nsamples = 9927\nval
ue = [6552, 3375]'),
 Text(119.57142857142856, 181.19999999999982, 'gini = 0.128\nsamples = 233\nvalu
e = [217, 16]'),
 Text(239.1428571428571, 543.5999999999999, 'ohe__x1_ICE <= 0.962\ngini = 0.487
\nsamples = 7374\nvalue = [4284, 3090]'),
 Text(199.28571428571428, 181.19999999999982, 'gini = 0.488\nsamples = 7322\nval
ue = [4237, 3085]'),
```

```
 Text(279.0, 181.19999999999982, 'gini = 0.174\nsamples = 52\nvalue = [47, 5]'),
 Text(478.2857142857142, 906.0, 'CRASH_DAY_OF_WEEK <= 0.048\ngini = 0.5\nsamples
= 3353\nvalue = [1700, 1653]'),
 Text(398.57142857142856, 543.5999999999999, 'ohe__x1_SNOW OR SLUSH <= 0.5\ngini
= 0.488\nsamples = 1712\nvalue = [987, 725]'),
 Text(358.71428571428567, 181.19999999999982, 'gini = 0.492\nsamples = 1648\nval
ue = [926, 722]'),
 Text(438.4285714285714, 181.19999999999982, 'gini = 0.089\nsamples = 64\nvalue
= [61, 3]'),
 Text(558.0, 543.5999999999999, 'WEATHER_CONDITION <= 0.015\ngini = 0.491\nsampl
es = 1641\nvalue = [713, 928]'),
 Text(518.1428571428571, 181.19999999999982, 'gini = 0.49\nsamples = 359\nvalue
= [205, 154]'),
 Text(597.8571428571428, 181.19999999999982, 'gini = 0.478\nsamples = 1282\nvalu
e = [508, 774]'),
 Text(757.2857142857142, 1268.4, 'DEVICE_CONDITION <= 0.997\ngini = 0.499\nsampl
es = 16875\nvalue = [8846, 8029]'),
 Text(717.4285714285713, 906.0, 'gini = 0.0\nsamples = 210\nvalue = [0, 210]'),
 Text(797.1428571428571, 906.0, 'ohe__x1_DRY <= 0.033\ngini = 0.498\nsamples = 1
6665\nvalue = [8846, 7819]'),
 Text(717.4285714285713, 543.5999999999999, 'ohe__x1_WET <= 0.05\ngini = 0.481\n
samples = 3788\nvalue = [2261, 1527]'),
 Text(677.5714285714286, 181.19999999999982, 'gini = 0.433\nsamples = 1083\nvalu
e = [740, 343]'),
 Text(757.2857142857142, 181.19999999999982, 'gini = 0.492\nsamples = 2705\nvalu
e = [1521, 1184]'),
 Text(876.8571428571428, 543.5999999999999, 'ohe__x0_DAWN <= 0.5\ngini = 0.5\nsa
mples = 12877\nvalue = [6585, 6292]'),
 Text(836.9999999999999, 181.19999999999982, 'gini = 0.498\nsamples = 10347\nval
ue = [5486, 4861]'),
 Text(916.7142857142857, 181.19999999999982, 'gini = 0.491\nsamples = 2530\nvalu
e = [1099, 1431]'),
 Text(1594.2857142857142, 1630.8000000000002, 'ohe__x1_DRY <= 0.5\ngini = 0.498
\nsamples = 96692\nvalue = [45628, 51064]'),
 Text(1275.4285714285713, 1268.4, 'ohe__x1_WET <= 0.03\ngini = 0.5\nsamples = 25
136\nvalue = [12823, 12313]'),
 Text(1116.0, 906.0, 'WEATHER_CONDITION <= 0.301\ngini = 0.494\nsamples = 7480\n
value = [4159, 3321]'),
 Text(1036.2857142857142, 543.5999999999999, 'ohe__x1_ICE <= 0.5\ngini = 0.487\n
samples = 4290\nvalue = [2496, 1794]'),
 Text(996.4285714285713, 181.19999999999982, 'gini = 0.481\nsamples = 3841\nvalu
e = [2293, 1548]'),
 Text(1076.142857142857, 181.19999999999982, 'gini = 0.495\nsamples = 449\nvalue
= [203, 246]'),
 Text(1195.7142857142856, 543.5999999999999, 'ohe__x1_UNKNOWN <= 0.283\ngini =
0.499\nsamples = 3190\nvalue = [1663, 1527]'),
 Text(1155.8571428571427, 181.19999999999982, 'gini = 0.488\nsamples = 968\nvalu
e = [560, 408]'),
 Text(1235.5714285714284, 181.19999999999982, 'gini = 0.5\nsamples = 2222\nvalue
= [1103, 1119]'),
 Text(1434.8571428571427, 906.0, 'WEATHER_CONDITION <= 0.5\ngini = 0.5\nsamples
= 17656\nvalue = [8664, 8992]'),
 Text(1355.142857142857, 543.5999999999999, 'CRASH_DAY_OF_WEEK <= 0.5\ngini = 0.
5\nsamples = 13611\nvalue = [6790, 6821]'),
 Text(1315.2857142857142, 181.19999999999982, 'gini = 0.499\nsamples = 6046\nval
ue = [2904, 3142]'),
 Text(1395.0, 181.19999999999982, 'gini = 0.5\nsamples = 7565\nvalue = [3886, 36
79]'),
 Text(1514.5714285714284, 543.5999999999999, 'CRASH_DAY_OF_WEEK <= 0.5\ngini =
0.497\nsamples = 4045\nvalue = [1874, 2171]'),
 Text(1474.7142857142856, 181.19999999999982, 'gini = 0.5\nsamples = 1704\nvalue
= [838, 866]'),
 Text(1554.4285714285713, 181.19999999999982, 'gini = 0.493\nsamples = 2341\nval
ue = [1036, 1305]'),
 Text(1913.1428571428569, 1268.4, 'DEVICE_CONDITION <= 0.5\ngini = 0.497\nsample
```

```
s = 71556\nvalue = [32805, 38751]'),
 Text(1753.7142857142856, 906.0, 'CRASH_DAY_OF_WEEK <= 0.5\ngini = 0.499\nsample
s = 34631\nvalue = [16593, 18038]'),
 Text(1673.9999999999998, 543.5999999999999, 'WEATHER_CONDITION <= 0.5\ngini =
0.496\nsamples = 14007\nvalue = [6414, 7593]'),
 Text(1634.142857142857, 181.19999999999982, 'gini = 0.485\nsamples = 196\nvalue
= [115, 81]'),
 Text(1713.8571428571427, 181.19999999999982, 'gini = 0.496\nsamples = 13811\nva
lue = [6299, 7512]'),
 Text(1833.4285714285713, 543.5999999999999, 'WEATHER_CONDITION <= 0.5\ngini =
0.5\nsamples = 20624\nvalue = [10179, 10445]'),
 Text(1793.5714285714284, 181.19999999999982, 'gini = 0.499\nsamples = 306\nvalu
e = [146, 160]'),
 Text(1873.2857142857142, 181.19999999999982, 'gini = 0.5\nsamples = 20318\nvalu
e = [10033, 10285]'),
 Text(2072.5714285714284, 906.0, 'WEATHER_CONDITION <= 0.5\ngini = 0.493\nsample
s = 36925\nvalue = [16212, 20713]'),
 Text(1992.8571428571427, 543.5999999999999, 'CRASH_DAY_OF_WEEK <= 0.5\ngini =
0.5\nsamples = 481\nvalue = [233, 248]'),
 Text(1952.9999999999998, 181.19999999999982, 'gini = 0.473\nsamples = 151\nvalu
e = [93, 58]'),
 Text(2032.7142857142856, 181.19999999999982, 'gini = 0.489\nsamples = 330\nvalu
e = [140, 190]'),
 Text(2152.285714285714, 543.5999999999999, 'CRASH_DAY_OF_WEEK <= 0.5\ngini = 0.
492\nsamples = 36444\nvalue = [15979, 20465]'),
 Text(2112.428571428571, 181.19999999999982, 'gini = 0.492\nsamples = 14498\nval
ue = [6344, 8154]'),
 Text(2192.142857142857, 181.19999999999982, 'gini = 0.493\nsamples = 21946\nval
ue = [9635, 12311]')]
```



In [57]:
```
cv_score_smoted, test_score_smoted
```

Out[57]:
```
(0.45654389962698094, 0.4555165732837618)
```

In [58]:
```
confusion_matrix(y_pred, y_test)
```

Out[58]:
```
array([[7204,  795],
       [9603, 1495]])
```

Baseline Model

A baseline model helps us find the features we are most interested in and where we can focus modeling iterations on to get better models.

SMOTE logistic regression with just traffic control device (picked from our tree above). This is the simplest model to give us a place to start.

In [59]:
```
X = final_df[['TRAFFIC_CONTROL_DEVICE']]
```

```
y = final_df['INJURY_CLASSIFICATION']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=11)
```

In [60]:
```
# Create a column transformer
col_transformer = ColumnTransformer(transformers=[
    ('ohe', OneHotEncoder(categories='auto', handle_unknown='ignore'), ['LIGHTIN
], remainder='passthrough')

over = SMOTE(sampling_strategy='minority')
under = RandomUnderSampler(sampling_strategy='not minority')

# Create a pipeline containing the column transformer and model
pipeline = imb_Pipeline(steps=[
    ('col_transformer', col_transformer),
    ('o', over),
    ('u', under),
    ('classifier', DecisionTreeClassifier(random_state=11))
])
```

In [50]:
```
param_grid = [{'classifier__max_depth':[1, 3, 5]}]

grid_search = GridSearchCV(estimator=pipeline,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5
                           )

grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)
print(grid_search.best_params_)
cv_score_smoted = grid_search.best_score_
test_score_smoted = grid_search.score(X_test, y_test)
```

```
{'classifier__max_depth': 5}
```

In [51]:
```
cv_score_smoted, test_score_smoted
```

Out[51]:
```
(0.4576437333619146, 0.45844897104257215)
```

In [52]:
```
confusion_matrix(y_pred, y_test)
```

Out[52]:
```
array([[7268,  803],
       [9539, 1487]])
```

In [53]:
```
roc_auc_score(y_pred, y_test)
```

Out[53]:
```
0.5176855212726038
```

Logsitic Regression Baseline Discussion:

Our ROC score is barely better than random chance (0.508). That's ok though, this gives us a baseline!

Model iteration 1

SMOTE logistic regression with all features.

This is our attempt to improve on the baseline model

In [36]:
```
X = final_df.drop(columns=['INJURY_CLASSIFICATION'])
y = final_df['INJURY_CLASSIFICATION']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=11)
```

In [37]:
```
# Create a column transformer
col_transformer = ColumnTransformer(transformers=[
    ('ohe', OneHotEncoder(categories='auto', handle_unknown='ignore'), ['LIGHTIN
], remainder='passthrough')

over = SMOTE(sampling_strategy='minority')
under = RandomUnderSampler(sampling_strategy='not minority')

# Create a pipeline containing the column transformer and model
pipeline = imb_Pipeline(steps=[
    ('col_transformer', col_transformer),
    ('o', over),
    ('u', under),
    ('scaler', StandardScaler()),
    ('logistic_regressor', LogisticRegression(random_state=42))
])
```

In [38]:
```
param_grid = [{'logistic_regressor__max_iter': [50, 100, 250, 500],
              # 'logistic_regressor__C': [1e-10, 1e-100],
               'logistic_regressor__penalty': ['none', 'l2']
              }]

grid_search = GridSearchCV(estimator=pipeline,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5
                           )

grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)
cv_score_smoted_log = grid_search.best_score_
test_score_smoted_log = grid_search.score(X_test, y_test)
```

In [39]:
```
print('Logistic regression, all features:')
print(f'cv score: {cv_score_smoted_log} and test score {test_score_smoted_log}')
print(f'confusion matrix: {confusion_matrix(y_pred, y_test)}')
print(f'ROC Accuracy Score: {roc_auc_score(y_pred, y_test)}')
print(f'Classification Report:')
print(classification_report(y_pred, y_test))
```

```
Logistic regression, all features:
cv score: 0.5259795633461695 and test score 0.5252133843011991
confusion matrix: [[8681  941]
```

```
 [8126 1349]]
ROC Accuracy Score: 0.5222889771626039
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.90      0.66      9622
           1       0.59      0.14      0.23      9475

    accuracy                           0.53     19097
   macro avg       0.55      0.52      0.44     19097
weighted avg       0.55      0.53      0.44     19097
```

SMOTE logistic regression: all features Discussion:

Our accuracy is slightly better than the baseline with just one feature. We'd still like to see something better.

Model iteration 2: smote knn

This model uses all the features in the final_df. would produce favorable results but needs tuning

In [40]:
```
X = final_df.drop(columns=['INJURY_CLASSIFICATION'])
y = final_df['INJURY_CLASSIFICATION']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=11)
```

In [41]:
```
X_t, X_val, y_t, y_val = train_test_split(X_train, y_train,
                                          random_state=42,
                                          test_size=0.2)
```

In [42]:
```
# Create a column transformer
col_transformer = ColumnTransformer(transformers=[
    ('ohe', OneHotEncoder(categories='auto', handle_unknown='ignore'), ['LIGHTIN
], remainder='passthrough')

over = SMOTE(sampling_strategy='minority')
under = RandomUnderSampler(sampling_strategy='not minority')

# Create a pipeline containing the column transformer and model
pipeline = imb_Pipeline(steps=[
    ('col_transformer', col_transformer),
    ('o', over),
    ('u', under),
    ('knn_classifier', KNeighborsClassifier())
])
```

In [43]:
```
param_grid = [{'knn_classifier__n_neighbors': [3,5,9,12,15],
               'knn_classifier__metric': ['minkowski','manhattan']}]

grid_search = GridSearchCV(estimator=pipeline,
                           param_grid=param_grid,
                           scoring='accuracy',
```

```
                                cv=5
                                )

grid_search.fit(X_t, y_t)

y_hat = grid_search.predict(X_val)
print(grid_search.best_params_)
cv_score_smoted_knn = grid_search.best_score_
test_score_smoted_knn = grid_search.score(X_test, y_test)
```

```
{'knn_classifier__metric': 'manhattan', 'knn_classifier__n_neighbors': 3}
```

In [44]:

```
print('Logistic regression, all features:')
print(f'cv score: {cv_score_smoted_knn} and test score {test_score_smoted_knn}')
print(f'confusion matrix: {confusion_matrix(y_pred, y_test)}')
print(f'ROC Accuracy Score: {roc_auc_score(y_pred, y_test)}')
print(f'Classification Report:')
print(classification_report(y_pred, y_test))
```

```
Logistic regression, all features:
cv score: 0.769542929657807 and test score 0.7775043200502697
confusion matrix: [[8681  941]
 [8126 1349]]
ROC Accuracy Score: 0.5222889771626039
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.90      0.66      9622
           1       0.59      0.14      0.23      9475

    accuracy                           0.53     19097
   macro avg       0.55      0.52      0.44     19097
weighted avg       0.55      0.53      0.44     19097
```

Model 3 Discussion:

This model could produce favorable results but needs tuning

Final Model: smote knn selective

This model uses the features determined by the decision tree from ‘smote oversampling’ to produce the best outcome

In [45]:

```
X = final_df.drop(columns=['INJURY_CLASSIFICATION', 'BAC_RESULT VALUE', 'AGE', '
                          'TRAFFIC_CONTROL_DEVICE'])
y = final_df['INJURY_CLASSIFICATION']

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    stratify=y,
                                                    random_state=11)
```

In [46]:

```
X_t, X_val, y_t, y_val = train_test_split(X_train, y_train,
                                          random_state=42,
                                          test_size=0.2)
```

In [61]:

```
col_transformer = ColumnTransformer(transformers=[
    ('ohe', OneHotEncoder(categories='auto', handle_unknown='ignore'), ['LIGHTIN
```

```
], remainder='passthrough')

over = SMOTE(sampling_strategy='minority')
under = RandomUnderSampler(sampling_strategy='not minority')

# Create a pipeline containing the column transformer and model
pipeline = imb_Pipeline(steps=[
    ('col_transformer', col_transformer),
    ('o', over),
    ('u', under),
    ('knn_classifier', KNeighborsClassifier(metric='minkowski'))
])
```

In [ ]:
```
# this took too long to run at the time of project submission, added markdown fo

param_grid = [{'knn_classifier__n_neighbors': [9,12]}]

grid_search = GridSearchCV(estimator=pipeline,
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5
                           )

grid_search.fit(X_t, y_t)

y_hat = grid_search.predict(X_val)
print(grid_search.best_params_)
cv_score_smoted_knn = grid_search.best_score_
test_score_smoted_knn = grid_search.score(X_test, y_test)
```

Output: {'knn_classifier__n_neighbors': 12}

In [ ]:
```
print('Logistic regression, all features:')
print(f'cv score: {cv_score_smoted_knn} and test score {test_score_smoted_knn}')
print(f'confusion matrix: {confusion_matrix(y_pred, y_test)}')
print(f'ROC Accuracy Score: {roc_auc_score(y_pred, y_test)}')
print(f'Classification Report:')
print(classification_report(y_pred, y_test))
```

output:

cv Score: (0.8569278871922539, 0.8507618997748337) confusion matrix: ([12759, 681], [1761, 77])

ROC accuracy score: 0.8401623249116377

classification report: 0.10158311345646438

F1 score: 0.059322033898305086

Discussion and Conclusion

In conclusion, our best classification model was done with K_Nearest_Neighbors run with selected key features resulting in an accuracy score of 87%. These features were

`DEVICE_CONDITION` , `WEATHER_CONDITION` , `LIGHTING_CONDITION` , `ROADWAY_SURFACE_COND` , `CRASH_DAY_OF_WEEK` .

Due to the imbalanced classification found within our target variable as well as in our slected features we had implemented the SMOTE technique of oversampling and undersampling within our test splits. We also used a pipeline and GridSearchCV to select our optimal hyperparameters within this model which resulted in this model outperforming the decision tree and logistic regression models.

The complexity of this dataset did make implimenting a succesful model a challenge. Further research would involve more time and care spent on EDA and transofrming the data.

Future Research

Further research we suggest to include budgetary and resource restrictions that the DOT must consider when implementing solutions.

In [ ]: