Data Preparation

Since we are looking at contributing factors to a car crash, information that occurs after the crash event is not utilizable. So, we drop unncessary columns and transform the 'object' data types in the dataframe into a 'int' data type in order for us to pass it into a model.

For missing values, if there was a way to fill those in with a reasonable explaination, we did so. Luckily, missing values mostly fell under the columns which we determined not relevant to the crash.

First we dropped all unnecessary columns such as injury severity and street direction. After dropping missing values we could not reasonably populate ourselves, we were left with a dataset of ~950,000 entries. Then, we mapped all the object data types into an int or float 64 to pass into our model. Finally, we created a Target column to label the crashes as Preventable or Less Preventable.

Import packages

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.dummy import DummyClassifier
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Read the csv files using pandas

In [2]:
```
df_crash = pd.read_csv('../data/raw_data/Traffic_Crashes_Crashes.csv', low_memory=False
```

In [3]:
```
df_vehicle = pd.read_csv('../data/raw_data/Traffic_Crashes_Vehicle.csv', low_memory=Fal
```

In [4]:
```
df_people = pd.read_csv('../data/raw_data/Traffic_Crashes_People.csv', low_memory=False
```

Cleaning Crashes

Dropping Columns

Within the Crashes dataset, our team determined that there were a lot of columns that did not contribute any value to the stakeholder's business problem, which is identifying causes of a car crash. Information such as injuries may tell us the severity of the crash, but not a clue on to how the crash occured. Same applies to when the police were notified and time it took place.

Columns such as 'INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I' may contain relevant information, however they hold too many null values for us to fill, so we decided to drop them.

Columns such 'STREET_DIRECTION' and 'ALIGNMENT' tells us the road shape, but since it is not feasible to change any of these roads - meaning they will always have turns and a direction - so we also left these columns from our final dataset.

In [5]:
```
# Drop columns that are not relevant to "Car Crash Contributors". For example, informat
# tell us a contributing factor to a crash.

not_relevant_info = ['RD_NO', 'CRASH_DATE', 'CRASH_DATE_EST_I', 'LANE_CNT', 'REPORT_TYP
                     'INTERSECTION_RELATED_I', 'NOT_RIGHT_OF_WAY_I', 'DATE_POLICE_NOTIF
                     'HIT_AND_RUN_I', 'PHOTOS_TAKEN_I', 'STATEMENTS_TAKEN_I', 'FIRST_CR
                     'DOORING_I', 'WORK_ZONE_I', 'WORK_ZONE_TYPE', 'WORKERS_PRESENT_I',
                     'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_FATAL', 'INJURIE
                     'INJURIES_NON_INCAPACITATING', 'INJURIES_REPORTED_NOT_EVIDENT', 'I
                     'INJURIES_REPORTED_NOT_EVIDENT', 'INJURIES_NO_INDICATION', 'INJURI
                     'STREET_NO', 'STREET_DIRECTION', 'STREET_NAME', 'BEAT_OF_OCCURRENC
                     'LOCATION', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH', 'POS

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

Using the helper functions, if the value in the column contains any of these values, return a '1', otherwise, return '0'

In [6]:
```
# Define different functions to categorize columns into numarical columns for our model
def traffic_control_transfomer(val):
    if val in relevant_device_present:
        return 1
    else:
        return 0

def device_working(val):
    if val in working_devices:
        return 1
    else:
        return 0

def device_present(val):
    if val == True:
        return 1
    else:
        return 0

def weather_condition(val):
    if val in bad_weather_cond:
        return 1
    else:
        return 0

def visibility(val):
    if val in bad_visibility:
        return 1
    else:
        return 0

# List would contain just one variable so we just spelled it out here
def trafficway_danger(val):
    if val == 'NOT DIVIDED':
        return 1
    else:
        return 0
```

```
def road_cond(val):
    if val in bad_road_cond:
        return 1
    else:
        return 0

def defect_road(val):
    if val in road_defect:
        return 1
    else:
        return 0

def boolean(val):
    if val == False:
        return 0
    else:
        return 1
```

Helper functions to correctly categorize different primary causes

In [7]:
```
def crash_cause(val):
    if val in driving:
        return 1
    elif val in behavior:
        return 2
    elif val in road:
        return 3
    else:
        return 4

def one(val):
    if val == 5:
        return 1
    else:
        return val

def two(val):
    if val == 5:
        return 2
    else:
        return val
```

In [8]:
```
# List containing whether a traffic control or warning sign was present in the area.
relevant_device_present = ['TRAFFIC SIGNAL', 'STOP SIGN/FLASHER', 'OTHER', 'LANE USE MA
                  'OTHER WARNING SIGN', 'RAILROAD CROSSING GATE', 'PEDESTRIAN CROSSING S
                  'DELINEATORS', 'FLASHING CONTROL SIGNAL', 'BICYCLE CROSSING SIGN', 'NO
# List containing if the Device was functioning
working_devices = ['FUNCTIONING PROPERLY', 'WORN REFLECTIVE MATERIAL']
# List containing bad weather elements
bad_weather_cond = ['RAIN', 'SNOW', 'FOG/SMOKE/HAZE', 'SLEET/HAIL', 'FREEZING RAIN/DRIZ
                    'SEVERE CROSS WIND GATE', 'BLOWING SAND, SOIL, DIRT']
# List containing bad visibility environment
bad_visibility = ['DARKNESS', 'DUSK', 'DAWN']
# Lists containing bad road conditions
bad_road_cond = ['WET', 'SNOW OR SLUSH', 'ICE', 'SAND, MUD, DIRT']
road_defect = ['RUT, HOLES', 'WORN_SURFACE']
```

Map all columns to int64 to model

In [9]:
```
df_crash['BAD_WEATHER'] = df_crash['WEATHER_CONDITION'].map(weather_condition)
df_crash['BAD_VISIBILITY'] = df_crash['LIGHTING_CONDITION'].map(visibility)
df_crash['BAD_TRAFFIC'] = df_crash['TRAFFICWAY_TYPE'].map(trafficway_danger)
df_crash['BAD_ROAD_CONDITION'] = df_crash['ROADWAY_SURFACE_COND'].map(road_cond)
df_crash['DEFECT_ROAD'] = df_crash['ROAD_DEFECT'].map(defect_road)

df_crash['TRAFFIC_DEVICE_PRESENT'] = df_crash['TRAFFIC_CONTROL_DEVICE'].map(traffic_con
df_crash['DEVICE_WORKING'] = df_crash['DEVICE_CONDITION'].map(device_working)
```

Final 'Device' column that returns "1" when the traffice device is both present, AND working.

In [10]:
```
device_list = (df_crash['TRAFFIC_DEVICE_PRESENT'] == 1) & (df_crash['DEVICE_WORKING'] =
device_dataframe = pd.DataFrame(device_list, columns = ['TRAFFIC_DEVICE'], index = df_c
frames = [df_crash, device_dataframe]
df_crash = pd.concat(frames, axis = 1)
df_crash['WORKING_TRAFFIC_DEVICE'] = df_crash['TRAFFIC_DEVICE'].map(device_present)
```

List for mapping "PRIMARY CONTRIBUTORY CAUSE" as recorded by officer at scene.

In [11]:
```
driving = ['FAILING TO YIELD RIGHT-OF-WAY', 'FOLLOWING TOO CLOSELY', 'IMPROPER OVERTAKI
           'IMPROPER BACKING', 'IMPROPER LANE USAGE', 'IMPROPER TURNING/NO SIGNAL', 'DRI
           'DISREGARDING TRAFFIC SIGNALS', 'DISREGARDING STOP SIGN', 'DISTRACTION - FROM
           'DISTRACTION - FROM OUTSIDE VEHICLE', 'EXCEEDING AUTHORIZED SPEED LIMIT', 'EX
           'DISREGARDING OTHER TRAFFIC SIGNS', 'DISREGARDING ROAD MARKINGS', 'TURNING RI
            'PASSING STOPPED SCHOOL BUS']
behavior = ['PHYSICAL CONDITION OF DRIVER', 'OPERATING VEHICLE IN ERRATIC, RECKLESS, CA
            'UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)', 'CELL
            'HAD BEEN DRINKING (USE WHEN ARREST IS NOT MADE)', 'TEXTING', 'DISTRACTION -
road = ['WEATHER', 'VISION OBSCURED (SIGNS, TREE LIMBS, BUILDINGS, ETC.)', 'ROAD ENGINE
       'ROAD CONSTRUCTION/MAINTENANCE']
other = ['EQUIPMENT - VEHICLE CONDITION', 'EVASIVE ACTION DUE TO ANIMAL, OBJECT, NONMOT
         'BICYCLE ADVANCING LEGALLY ON RED LIGHT', 'OBSTRUCTED CROSSWALKS', 'MOTORCYCLE

df_crash['Target1'] = df_crash['PRIM_CONTRIBUTORY_CAUSE'].map(crash_cause)
```

Because our primary target variable is 'contributory causes to a crash' using the informating recorded by the police report would make the most sense. Who else would know better on the primary cause of the accident than the driver or officer that arrived at the scene.

In [12]:
```
# Update any rows where 'PRIM_CONTRIBUTORY_CAUSE' is "not determined" and 'SEC_CONTRIBU
# 'PRIM_CONTRIBUTORY_CAUSE' to the value entered in'SEC_CONTRIBUTORY_CAUSE'
for index, row in df_crash.iterrows():
    if df_crash.loc[index,'PRIM_CONTRIBUTORY_CAUSE'] == 'UNABLE TO DETERMINE':
        if (df_crash.loc[index,'SEC_CONTRIBUTORY_CAUSE'] != 'UNABLE TO DETERMINE') & (d
            df_crash.loc[index,'PRIM_CONTRIBUTORY_CAUSE'] = df_crash.loc[index,'SEC_CON
```

Converting 'UNABLE TO DETERMINE' columns to be recorded as "road" when 'BAD ROAD CONDITION' is also true

In [13]:
```
df_crash['is_two'] = (df_crash['Target1'] == 4) & (df_crash['BAD_ROAD_CONDITION'] == 1)
df_crash['Target'] = df_crash['Target1'] + df_crash['is_two']
df_crash['is_two'] = df_crash['is_two'].map(two)
df_crash['Target'] = df_crash['Target1'] + df_crash['is_two']
```

In [14]:
```
# Had to separate this helper function due to Python errors
```

```
def road(val):
    if val == 5:
        return 3
    else:
        return val
df_crash['Target'] = df_crash['Target'].map(road)
```

Now we can drop the columns we transformed as they are unusable in our model

In [15]:
```
unnecessary_traffic_columns = ['TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'TRAFFIC_D
                               'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'TRAFFICWAY_TY
                               'PRIM_CONTRIBUTORY_CAUSE', 'SEC_CONTRIBUTORY_CAUSE', 'TRA
df_crash.drop(unnecessary_traffic_columns, axis = 1, inplace = True)
```

Save file to a csv folder so we do not have to run all the data cleaning steps again.

In [16]:
```
df_crash.to_csv('../data/cleaned_data/cleaned_crash_data.csv')
```

Cleaning People

Drop columns that do not have any predictive or inferential value in determining the cause of a car crash

In [17]:
```
df_people.drop(['PERSON_ID', 'PERSON_TYPE', 'RD_NO', 'VEHICLE_ID', 'SEAT_NO', 'CITY', '
         'DRIVERS_LICENSE_STATE', 'DRIVERS_LICENSE_CLASS', 'SAFETY_EQUIPMENT', 'AIRBAG_
         'INJURY_CLASSIFICATION', 'HOSPITAL', 'EMS_AGENCY', 'EMS_RUN_NO', 'PEDPEDAL_ACT
         'PEDPEDAL_LOCATION', 'BAC_RESULT', 'BAC_RESULT VALUE', 'CELL_PHONE_USE'], axis
```

Drop missing and null values

In [18]:
```
df_people.dropna(subset=['AGE'], inplace=True)
df_people.dropna(subset=['DRIVER_ACTION'], inplace=True)
df_people.dropna(subset=['DRIVER_VISION'], inplace=True)
df_people.dropna(subset=['PHYSICAL_CONDITION'], inplace=True)
```

Cleaning the 3 columns with many unknown values

In [19]:
```
df_people = df_people[df_people['DRIVER_VISION']!='UNKNOWN']
df_people = df_people[df_people['DRIVER_ACTION']!='UNKNOWN']
df_people = df_people[df_people['PHYSICAL_CONDITION']!='UNKNOWN']
```

Formatting, cleaning, and binning the AGE column. 15 is the youngest age to legally drive in the state of Illinois with a learner's permit.

In [20]:
```
df_people = df_people[df_people['AGE']>=15]
```

Define Lists for helper functions in order to map

In [21]:
```
dangerous_beh = ['IMPAIRED - ALCOHOL', 'FATIGUED/ASLEEP', 'EMOTIONAL', 'ILLNESS/FAINTED
                 'IMPAIRED - DRUGS', 'IMPAIRED - ALCOHOL AND DRUGS', 'MEDICATED']
bad_vision = ['OTHER', 'MOVING VEHICLES', 'PARKED VEHICLES', 'WINDSHIELD (WATER/ICE)',
              'EMBANKMENT', 'BLOWING MATERIALS', 'SIGNBOARD']
bad_action = ['OTHER', 'FAILED TO YIELD', 'FOLLOWED TOO CLOSELY', 'IMPROPER TURN', 'IMP
```

```
            'IMPROPER PASSING', 'TOO FAST FOR CONDITIONS', 'DISREGARDED CONTROL DEVICE
            'WRONG WAY/SIDE', 'CELL PHONE USE OTHER THAN TEXTING', 'OVERCORRECTED', 'E
            'EVADING POLICE VEHICLE', 'TEXTING', 'STOPPED SCHOOL BUS', 'LICENSE RESTRI
```

Transform more categorical columns into numerical format.

In [22]:
```
def physical_condition(val):
    if val in dangerous_beh:
        return 1
    else:
        return 0

def obscured_vision(val):
    if val in bad_vision:
        return 1
    else:
        return 0

def driver_error(val):
    if val in bad_action:
        return 1
    else:
        return 0

def ages(val):
    if val in highrisk_ages:
        return 1
    else:
        return 0
```

Map and drop object columns as they were tranformed into a data format we can pass into a model columns

In [23]:
```
df_people['DANGEROUS_BEH'] = df_people['PHYSICAL_CONDITION'].map(physical_condition)
df_people['OBSCURED_VIZ'] = df_people['DRIVER_VISION'].map(obscured_vision)
df_people['DRIVER_ERROR'] = df_people['DRIVER_ACTION'].map(driver_error)
unnecessary_columns = ['AGE', 'DRIVER_ACTION', 'DRIVER_VISION', 'PHYSICAL_CONDITION']
df_people.drop(unnecessary_columns, axis = 1, inplace=True)
```

Save file to a csv folder so we do not have to run all the data cleaning steps again.

In [24]:
```
df_people.to_csv('../data/cleaned_data/cleaned_people_data.csv')
```

Cleaning Vehicles

Instantiate pandas DataFrame with only relevant columns

In [25]:
```
df_vehicle = df_vehicle[['CRASH_RECORD_ID', 'NUM_PASSENGERS', 'VEHICLE_DEFECT']]
```

Transform the last categorical columns into numerical format and clean up for our models.

In [26]:
```
No_def = ['NONE', 'UNKNOWN']

def veh_defect(val):
    if val in No_def:
        return 0
```

```
    else:
        return 1

def passengers(val):
    if val >= 1:
        return 1
    else:
        pass
```

In [27]:
```
df_vehicle['VEHICLE_DEFECT'] = df_vehicle['VEHICLE_DEFECT'].map(veh_defect)
df_vehicle['NUM_PASSENGERS'] = df_vehicle['NUM_PASSENGERS'].map(passengers)
```

If missing values for passengers, update to just driver in car

In [28]:
```
df_vehicle.NUM_PASSENGERS = df_vehicle.NUM_PASSENGERS.fillna(0)
```

Save file to a csv folder so we do not have to run all the data cleaning steps again.

In [29]:
```
df_vehicle.to_csv('../data/cleaned_data/cleaned_vehicle_data.csv')
```

Creating the Main Dataframe, "main_df"

In [30]:
```
# Merge the the DataFrames into one using 'CRASH_RECORD_ID' column to join
main_df = df_crash.merge(df_vehicle, left_on='CRASH_RECORD_ID', right_on = 'CRASH_RECOR
main_df = main_df.merge(df_people, left_on='CRASH_RECORD_ID', right_on = 'CRASH_RECORD_
```

Once we merged the dataframes, we noticed that the number of rows in our dataset increased drastically, but not exactly 2x.

After examining the data, we found that some rows had split into two entries due to a crash normally involving two parties (but not always, so hence the ALMOST 2x).

We decided to keep the duplicated entries as no matter how many parties were involved, a cause that determined the crash affected all parties.

Then, we fixed the Target columns to accurately reflect the 'DANGEROUS_BEH' column as the Target column was created before the 'DANGEROUS_BEH' merge. Same logic applies to the 'DRIVER_ERROR' column.

In [31]:
```
# Converting 'UNABLE TO DETERMINE' columns to be recorded as "behavior" when 'DANGEROUS_
main_df['is_two'] = (main_df['Target'] == 4) & (main_df['DANGEROUS_BEH'] == 1)
main_df['is_two'] = main_df['is_two'].map(boolean)
main_df['Target'] = main_df['Target'] + main_df['is_two']
main_df['Target'] = main_df['Target'].map(two)
main_df.drop(['is_two'], axis = 1, inplace = True)

# Converting 'UNABLE TO DETERMINE' columns to be recorded as "driver" when 'DRIVER_ERRO
main_df['is_one'] = (main_df['Target'] == 4) & (main_df['DRIVER_ERROR'] == 1)
main_df['is_one'] = main_df['is_one'].map(boolean)
main_df['Target'] = main_df['Target'] + main_df['is_one']
main_df['Target'] = main_df['Target'].map(one)
main_df.drop(['is_one'], axis = 1, inplace = True)
```

Originally we wanted to do a 4 target classifer - driver

error, behavioral error, road conditions, and other factors.

However, due to the class imbalance, we grouped driver action (driver error, behavioral error) and road / external factors in order to fix the class imbalance problem.

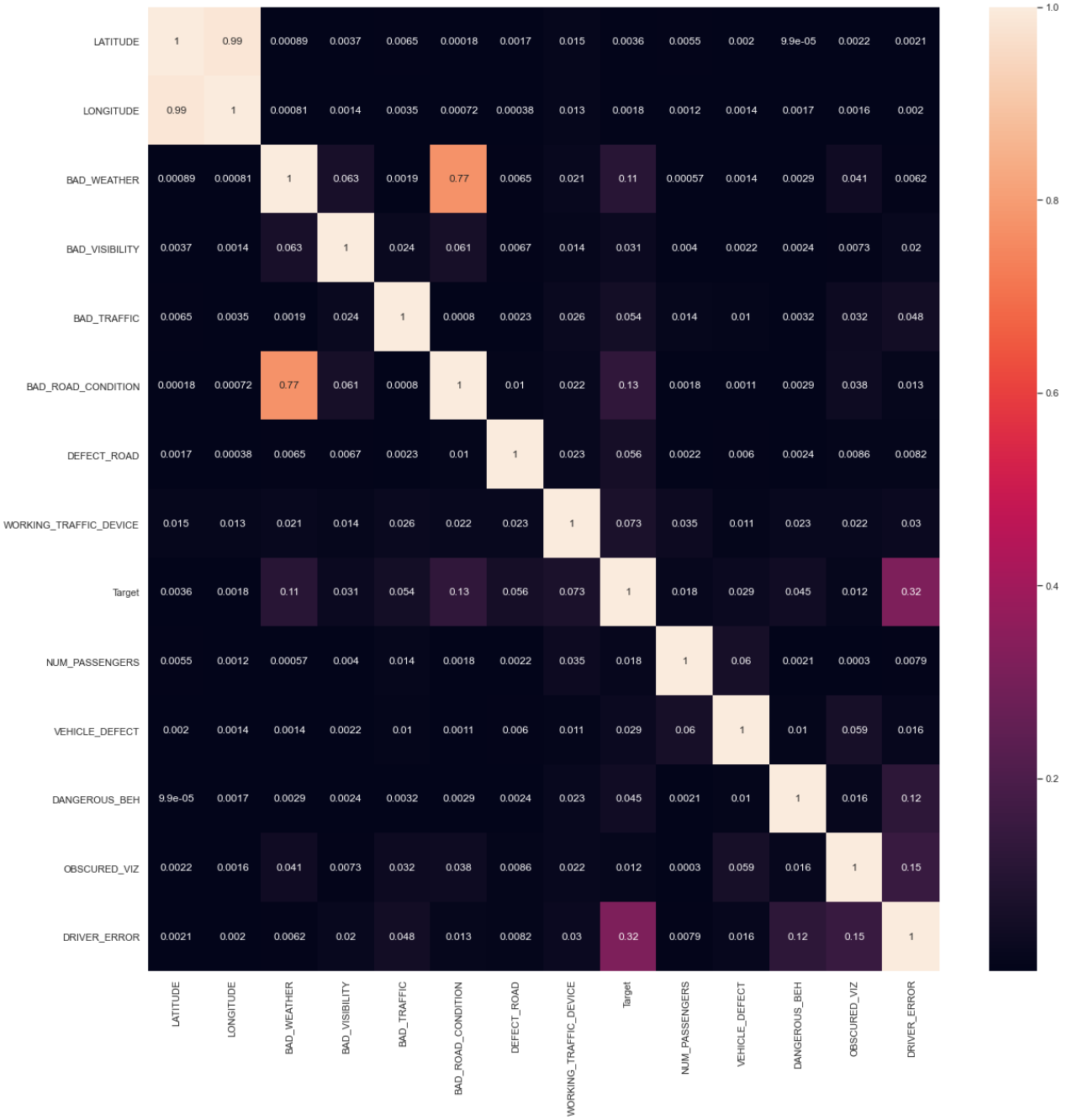
In [32]:
```
def target_map(val):
    if ((val == 1) | (val == 2)):
        return 0
    elif ((val == 3) | (val == 4)):
        return 1

main_df['Target'] = main_df['Target'].map(target_map)
main_df['Target'].value_counts()
```

Out[32]:
```
0    718270
1    233870
Name: Target, dtype: int64
```

Before saving our final modeling data to a csv, we ran a heatmap to see if there are any concerning correlations.

In [33]:
```
sns.set(rc={'figure.figsize':(20, 20)})
sns.heatmap(main_df.corr().abs(),annot=True);
```



Because BAD_WEATHER and BAD_ROAD_CONDITIONS are highly correlated with each other. To not overfit to our data, we dropped BAD_WEATHER from our dataset.

Longitude and Latitude will be dropped in our modeling notebook.

In [34]:
```
main_df.drop('BAD_WEATHER', axis = 1, inplace = True)
```

Save final file to a csv folder for our main cleaned csv file.

In [35]:
```
main_df.to_csv('../data/cleaned_data/cleaned_modeling_data.csv')
```

In [36]:
```
# A preview, if you wanted to load the data to explore
main_df = pd.read_csv('../data/cleaned_data/cleaned_modeling_data.csv')
main_df.drop('Unnamed: 0', axis = 1, inplace = True)
main_df.head()
```

Out[36]:

	CRASH_RECORD_ID	LATITUDE	LONGITUDE	BAD_VISIBILITY	BAD_TI
0	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	41.919664	-87.773288	0	
1	4fd0a3e0897b3335b94cd8d5b2d2b350eb691add56c62d...	41.919664	-87.773288	0	
2	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	41.741804	-87.740954	0	
3	009e9e67203442370272e1a13d6ee51a4155dac65e583d...	41.741804	-87.740954	0	
4	ee9283eff3a55ac50ee58f3d9528ce1d689b1c4180b4c4...	41.773456	-87.585022	0	

In [ ]: