BikeSaferPA: predicting outcomes for cyclists using Pennsylvania crash data, 2002-2021

Introduction to the BikeSaferPA project

Which factors most dramatically impact a cyclist's risk of suffering serious injury or death in a crash?

Can we predict the severity of a cyclist's outcome based on these factors?

To address these questions I've build **BikeSaferPA**, a machine learning model which I designed to predict whether a cyclist in a crash in Pennsylvania will suffer serious injury or fatality as a result. I focused on a publically accessible dataset of crash records in the state during the period of 2002-2021, made available by Pennsylvania Department of Transportation (PENNDOT). This series of Jupyter notebooks will descibe this process.

Project goals

I focused on cyclists involved in crash incidents appearing in this dataset. The goals of this project are:

1. Examine and visualize the prevalence of cyclist-involved crash incidents both across the state and locally, and how the prevalence has changed during 2002-2021.
2. Examine and visualize the prevalence of various crash incident factors among cyclist-involved crash incidents.
3. Determine the factors in a bicycle-involved crash incident which contribute to increased risk of serious cyclist injury or cyclist fatality.
4. Design a machine learning model BikeSaferPA to predict serious cyclist injury or fatality based on various crash features available to use in this dataset.
5. Evaluate the performance of BikeSaferPA and assess the importance of various crash factors on its predictions.

Broad dataset description

PENNDOT publishes a publically accessible dataset containing records regarding all crashes in the state involving motor vehicles, cyclists, pedestrians, etc. The Pennsylvania dataset is somewhat unique among crash datasets in that it includes significantly more data features about bicycles and cyclists involved in these accidents than many other analogous datasets. Datasets for individual years can be downloaded here.

The PENNDOT datasets contain several different .csv files for each year on record. Here is the full list, with the ones I'll use in bold:

- COMMVEH: samples are commercial vehicles involved in crash incidents and features are specific to those types of vehicles
- **CRASH: samples are crash incidents and features include a variety of information about the incident (e.g. when, where, how many and what type of people and vehicles were involved).**
- **CYCLE: samples are pedal cycles and motorcycle vehicles involved in crash incidents and features are specific to those types of vehicles (e.g. helmet and safety device info, accessory info).**
- **FLAG: samples are crash incidents and features are binary fields which flag additional factors in each incident (e.g. factors related to weather, other conditions, participant behavior, presence of fatality). These were intended to help refine particular lookups in the dataset, and many of them can be subsumed by information provided in CRASH.**
- **PERSON: samples are individuals involved in crash incidents, including cyclists, their passengers, and pedestrians and features are characteristics of those individuals (e.g. demographics, drug/alcohol results, position in vehicle)**
- **ROADWAY: samples are roadways involved in crash incidents and features are supplementary roadway info (e.g. route number or name, posted speed limit)**
- TRAILVEH: samples are trailers on vehicles involved in crash incidents and features are specific to trailers
- **VEHICLE: samples are vehicles involved in crash incidents, including bicycles and pedestrians, and features are characteristic of those individuals (e.g. vehicle type and body style, vehicle movement and position)**

Regarding identifiers:

- Each sample in CRASH or FLAG should have a unique CRN (crash record number).
- Each sample in CYCLE, VEHICLE should have a unique (CRN,UNIT_NUM) tuple, as UNIT_NUM is sequential among vehicles in each crash.
- Each sample in PERSON should have a unique (CRN,UNIT_NUM,PERSON_NUM) tuple, as PERSON_NUM is sequential among persons in each vehicle unit.
- Each CRN may correspond to many samples in ROADWAY.

Here are links to two helpful documents provided by PENNDOT:

- Data dictionary
- Additional document with more information

Note: the data dictionary defines "serious injury" and "fatal injury" as:

- serious injury: "incapacitating injury, including bleeding wounds and distorted members (amputations or broken bones), and requires transport of the patient from the scene."
- fatal injury: "the person dies as a result of the injuries sustaines in the crash within 30 days of the crash."

Part I: Obtaining cleaning, and pre-processing the dataset

In this first notebook, I will do the following:

1. Unzip individual year data files,
2. Concatenate across years to build datasets for the entire period of 2002-2021.
3. Perform some initial data cleaning and merging.
4. Export the results to parquet files.

First I import the necessary packages:

In [6]:
```
import numpy as np
import pandas as pd
import openpyxl
import requests
import json
import glob
import zipfile

import warnings
warnings.filterwarnings("ignore")

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth',None)

import sys
np.set_printoptions(threshold=sys.maxsize)
```

Unzip crash data files from individual years

The following block unzips a collection of ZIP files downloaded from this PENNDOT page.

These files are not included in the GitHub repository, due to their file sizes. If you want to try out this process, you'll need to go download them from the PENNDOT page yourself. Your ZIP files should be in the directory **'./data/zip/'** and the extracted CSV filed will end up in **'./data/raw_csv/'**

In [7]:
```
# zip_files = glob.glob('data/zip/*.zip')
# for file in zip_files:
#     print(f'Unzipping {file}...')
#     with zipfile.ZipFile(file, 'r') as zip_ref:
#         zip_ref.extractall('data/raw_csv/')
```

Preparing dataframes for each year

The get_data function will produce for a given year the following dataframes:

- 'bicycles', a dataframe containing samples from the VEHICLE dataset which are bicycles (or other pedal cycles)
- 'persons', a dataframe containing samples from the PERSON dataset whose CRN appears in 'bicycles', and which is supplemented with some relevant features from CYCLE
- 'crashes', a dataframe containing all samples from the CRASH dataset whose CRN appears in 'bicycles', and which is supplemented with some relevant features from FLAG
- 'roadway', a dataframe containing samples from the ROADWAY dataset whose CRN appears in 'bicycles'

Unless you want to download the PENNDOT files yourself and try this out, you can also skip the next block and load in my prepared raw CSV files.

In [3]:
```
# from lib.get_data import extract_data
# bicycle_list, person_list, crash_list, roadway_list= [],[],[],[]

# # Call dataframe collection function for each year
# YEARS = range(2002,2022)
# for year in YEARS:
#     bicycles, persons,crashes,roadway= extract_data(year)
#     bicycle_list.append(bicycles)
#     person_list.append(persons)
#     crash_list.append(crashes)
#     roadway_list.append(roadway)


# # # Concatenate yearly dataframes and reindex
# bicycles = pd.concat(bicycle_list,axis=0).reset_index(drop=True)
# persons = pd.concat(person_list,axis=0).reset_index(drop=True)
# crashes = pd.concat(crash_list,axis=0).reset_index(drop=True)
# roadway = pd.concat(roadway_list,axis=0).reset_index(drop=True)

# del bicycle_list, person_list, crash_list

# # # Write to CSV files
# bicycles.to_csv('./data/raw_csv/bicycles_raw.csv',index=False)
# persons.to_csv('./data/raw_csv/persons_raw.csv',index=False)
# crashes.to_csv('./data/raw_csv/crashes_raw.csv',index=False)
# roadway.to_csv('./data/raw_csv/roadway_raw.csv',index=False)
```

Start here to load in prepared CSV files

In [13]:
```
bicycles = pd.read_csv('./data/raw_csv/bicycles_raw.csv')
persons = pd.read_csv('./data/raw_csv/persons_raw.csv')
crashes = pd.read_csv('./data/raw_csv/crashes_raw.csv')
roadway = pd.read_csv('./data/raw_csv/roadway_raw.csv')
```

Feature recoding and cleanup

A few tasks are needed regarding the dataframes 'bicycles', 'crashes', and 'persons':

1. Address inconsistencies with respect to data types within columns
2. Change 'unknown', 'other' and some non-sensical values to np.nan
3. Recode categorical features to be more descriptive (see the data dictionary)
4. Create from IMPACT_POINT (which uses clock number directions) a simpler feature IMPACT_SIDE

Features in 'bicycles'

In [14]:

```
for feat in ['RDWY_ALIGNMENT','VEH_MOVEMENT','VEH_TYPE','VEH_POSITION','VEH_
    bicycles[feat] = pd.to_numeric(bicycles[feat],errors='coerce')

for feat in ['PC_HDLGHT_IND','PC_HLMT_IND','PC_REAR_RFLTR_IND']:
    bicycles[feat]=bicycles[feat].replace({'Y':1,'N':0,'U':np.nan})

for feat in ['UNIT_NUM','IMPACT_POINT']:
    bicycles[feat]=bicycles[feat].replace({'U':np.nan,99:np.nan})

bicycles['GRADE'] = bicycles['GRADE'].replace({1:'level',2:'uphill',3:'downh
                                               4:'bottom_hill',5:'top_hill',
bicycles['IMPACT_SIDE'] = bicycles['IMPACT_POINT'].replace({0:'non_collision
                                                            11:'front',12:'f
                                                            2:'front_right',
                                                            5:'rear',6:'rear
                                                            8:'rear_left',9:
bicycles['RDWY_ALIGNMENT'] = bicycles['RDWY_ALIGNMENT'].replace({1:'straight
                                                                 3:'curve',4

bicycles['VEH_ROLE'] = bicycles['VEH_ROLE'].replace({0:'non_collision',1:'st
bicycles['VEH_TYPE'] = bicycles['VEH_TYPE'].replace({20:'bicycle',21:'other_
bicycles['VEH_POSITION'] = bicycles['VEH_POSITION'].replace({0:np.nan,1:'rig
                                                            3:'left_lane',4:
                                                            5:'center_turn_l
                                                            7:'oncoming_lane
                                                            9:'right_of_traf
                                                            11:'shoulder_rig
                                                            13:'one_lane_roa
                                                            98:'other',99:np
bicycles['VEH_MOVEMENT'] = bicycles['VEH_MOVEMENT'].replace({1:'straight',2:
                                                            3:'stopped_in_la
                                                            5:'entering_lane
                                                            7:'leaving_lane'
                                                            9:'turning_right
                                                            11:'turning_left
                                                            13:'u_turn',14:'
                                                            16:'curve_right'
                                                            18:'entering_lan
                                                            98:'other',99:np

# One sample has missing UNIT_NUM, and - set it to first avail unit number
bicycles['UNIT_NUM'] = bicycles.UNIT_NUM.fillna(1)
```

```
# Two samples have missing VEH_ROLE, but they have IMPACT_POINT=='non_collis
bicycles['VEH_ROLE'] = bicycles.VEH_ROLE.fillna('non_collision')
```

Features in 'persons'

In [15]:
```
persons['INJ_SEVERITY'] = pd.to_numeric(persons['INJ_SEVERITY'],errors='coer
persons['AGE']=persons['AGE'].replace({99:np.nan})
for feat in ['SEX','TRANSPORTED','UNIT_NUM']:
    persons[feat]=persons[feat].replace({'Y':1,'N':0,'U':np.nan,'0':0,' ':np
persons['INJ_SEVERITY']=persons['INJ_SEVERITY'].replace({0:'no_injury',1:'ki
                                                         3:'susp_minor_injur
                                                         8:'unknown_injury',
persons['PERSON_TYPE']=persons['PERSON_TYPE'].replace({1:'driver',2:'passeng
                                                       8:'other',9:np.nan})
persons['RESTRAINT_HELMET']=persons['RESTRAINT_HELMET'].replace({0:'no_restr
                                                                 3:'lap_shoul
                                                                 5:'motorcycl
                                                                 10:'belt_imp
                                                                 12:'helmet_i
                                                                 23:'booster_
                                                                 90:'unknown_
# Two persons have CRN==2012109815 and UNIT_NUM==1.  Since the bicycle has U
persons.at[35557,'UNIT_NUM']=2
```

Features in 'crashes'

In [16]:
```
for feat in ['WEATHER1','WEATHER2','TIME_OF_DAY']:
    crashes[feat] = pd.to_numeric(crashes[feat],errors='coerce')

for feat in ['DISPATCH_TM','ARRIVAL_TM','TIME_OF_DAY']:
    crashes[feat] = crashes[feat].replace({9999:np.nan})

crashes['HOUR_OF_DAY'] = crashes['HOUR_OF_DAY'].replace({99:np.nan,24:0})

crashes['COLLISION_TYPE'] = crashes['COLLISION_TYPE'].replace({0:'non_collis
                                                               2:'head_on',3
                                                               5:'sideswipe_
                                                               7:'hit_fixed_
                                                               9:'other',98:
crashes['ILLUMINATION'] = crashes['ILLUMINATION'].replace({1:'daylight',2:'d
                                                           5:'dawn',6:'dark_
crashes['ROAD_CONDITION'] = crashes['ROAD_CONDITION'].replace({1:'dry',2:'ic
                                                               4:'oil',5:'san
                                                               8:'water',9:'w
crashes['URBAN_RURAL'] = crashes['URBAN_RURAL'].replace({1:'rural',2:'urbani
crashes['INTERSECT_TYPE'] = crashes['INTERSECT_TYPE'].replace({0:'midblock',
                                                               4:'circle',5:
                                                               7:'ramp_begin
                                                               10:'other',11
crashes['LOCATION_TYPE'] = crashes['LOCATION_TYPE'].replace({0:'not_applicab
                                                             2:'ramp',3:'brid
                                                             5:'toll_booth',6
                                                             7:'driveway_park
crashes['RELATION_TO_ROAD'] = crashes['RELATION_TO_ROAD'].replace({1:'on_roa
```

```
                                                                  4:'roadsi
                                                                  6:'parkin
crashes['TCD_TYPE'] = crashes['TCD_TYPE'].replace({0:'not_applicable',1:'fla
                                                  3:'stop_sign',4:'yield_sign
                                                  7:'officer_or_flagman',8:'o

crashes['TCD_FUNC_CD'] = crashes['TCD_FUNC_CD'].replace({0:'no_controls',1:'
                                                        3:'functioning_proper

for feat in ['WEATHER1','WEATHER2']:
    crashes[feat] = pd.to_numeric(crashes[feat],errors='coerce')

# Replace counts features for buses, trucks, SUVs with binary features indic
for feat in ['BUS','HEAVY_TRUCK','SMALL_TRUCK','SUV','VAN']:
    crashes[feat] = crashes[feat+'_COUNT'].apply(lambda x: 1 if x>0 else x)
    # crashes = crashes.drop(feat+'_COUNT',axis=1)

# Recode 99 as np.nan
for feat in ['WEATHER1','WEATHER2']:
    crashes[feat]=crashes[feat].replace({99:np.nan})

# fill empty WEATHER1 with WEATHER2 when possible
crashes['WEATHER1'] = crashes.WEATHER1.fillna(crashes.WEATHER2)

# if WEATHER1==WEATHER2, set WEATHER2 to be np.nan
crashes.loc[crashes.WEATHER1==crashes.WEATHER2,'WEATHER2']=np.nan

# If WEATHER1==3 (clear) and WEATHER2 is not np.nan, replace WEATHER1 with W
crashes['WEATHER1'] = crashes['WEATHER1'].where((crashes.WEATHER1!=3)|(crash

# Rename and delete WEATHER2
crashes = crashes.rename(columns={'WEATHER1':'WEATHER'})
crashes = crashes.drop(columns='WEATHER2')

crashes['WEATHER'] = crashes['WEATHER'].replace({1:'blowing_sand_soil_dirt',
                                                 4:'cloudy',5:'fog_smog_smoke
                                                 7:'rain',8:'severe_crosswind
                                                 10:'snow',98:'other'})

# Adjust incorrect DEC_LAT, DEC_LONG

# This sample missing fractional part
crashes.at[24770,'DEC_LAT']=np.nan
crashes.at[24770,'DEC_LONG']=np.nan

# These samples locations don't match minicipalities - replace with approx l
crashes.at[3087,'DEC_LAT']=40.0081
crashes.at[3087,'DEC_LONG']=-75.1923
crashes.at[24805,'DEC_LAT']=40.3322
crashes.at[24805,'DEC_LONG']=-75.9278

# Fill NaN lat,lon values with mean by municipality when possible, then fill
for coord in ['DEC_LAT','DEC_LONG']:
    for area in ['MUNICIPALITY','COUNTY']:
        crashes[coord]=crashes.groupby(area)[coord].transform(lambda x:x.fil
```

```
# Fill some missing HOUR_OF_DAY using TIME_OF_DAY, or DISPATCH_TM, or ARRIVA
crashes['HOUR_OF_DAY'] = crashes.HOUR_OF_DAY.fillna(crashes.TIME_OF_DAY.floo
crashes['HOUR_OF_DAY'] = crashes.HOUR_OF_DAY.fillna(crashes.DISPATCH_TM.floo
crashes['HOUR_OF_DAY'] = crashes.HOUR_OF_DAY.fillna(crashes.ARRIVAL_TM.floor

# Fill all five missing ILLUMINATION 'daylight'
crashes['ILLUMINATION'] = crashes.ILLUMINATION.fillna('daylight')

# Don't need ARRIVAL_TM, DISPATCH_TM anymore
crashes = crashes.drop(columns=['DISPATCH_TM','ARRIVAL_TM'])

# Fill missing SPEED_LIMIT with mode by county.
roadway['SPEED_LIMIT'] = roadway.groupby('RDWY_COUNTY',sort=False)['SPEED_LI

# When there are several roadways with the same CRN, don't know which one th
# Use minimum value among all roadway samples with matching CRN
crashes = crashes.merge(pd.DataFrame(roadway.groupby('CRN').min().SPEED_LIMI
del roadway
```

Creating merged 'cyclists' dataframe

It will be useful later to have a dataframe containing only cyclists as samples, and which also have vehicle and crash features. I accomplish this by merging some dataframes.

In [17]:
```
# Merge bicycle vehicle data onto persons and crashes
cols = ['CRN', 'UNIT_NUM','INJ_SEVERITY','PERSON_TYPE',
        'AGE','SEX','RESTRAINT_HELMET']
cyclists = persons[cols].merge(bicycles,on=['CRN','UNIT_NUM'],how='left')

# Isolate cyclists by restricting to samples which inherit VEH_TYPE from bic
cyclists = cyclists[cyclists.VEH_TYPE.notnull()]

# Merge crash data onto cyclists

cols = ['CRN', 'COUNTY', 'MUNICIPALITY', 'BUS_COUNT','COMM_VEH_COUNT',
       'HEAVY_TRUCK_COUNT', 'SMALL_TRUCK_COUNT', 'SUV_COUNT', 'VAN_COUNT',
       'CRASH_MONTH', 'CRASH_YEAR', 'DAY_OF_WEEK', 'HOUR_OF_DAY',
       'COLLISION_TYPE', 'ILLUMINATION', 'INTERSECT_TYPE', 'LOCATION_TYPE',
       'RELATION_TO_ROAD', 'ROAD_CONDITION', 'TCD_TYPE', 'TCD_FUNC_CD',
       'URBAN_RURAL', 'WEATHER', 'AGGRESSIVE_DRIVING', 'ANGLE_CRASH',
       'CELL_PHONE', 'COMM_VEHICLE', 'CROSS_MEDIAN', 'CURVED_ROAD',
        'CURVE_DVR_ERROR','DRUG_RELATED','ALCOHOL_RELATED',
       'DISTRACTED', 'DRINKING_DRIVER', 'DRUGGED_DRIVER', 'FATIGUE_ASLEEP','
       'ICY_ROAD', 'ILLUMINATION_DARK', 'IMPAIRED_DRIVER', 'INTERSECTION',
       'LANE_DEPARTURE', 'NHTSA_AGG_DRIVING','NO_CLEARANCE','NON_INTERSECTIC
       'RUNNING_RED_LT', 'RUNNING_STOP_SIGN', 'RURAL', 'SNOW_SLUSH_ROAD',
       'SPEEDING', 'SPEEDING_RELATED', 'SUDDEN_DEER', 'TAILGATING', 'URBAN',
       'WET_ROAD', 'WORK_ZONE', 'MATURE_DRIVER', 'YOUNG_DRIVER', 'BUS',
       'HEAVY_TRUCK', 'SMALL_TRUCK', 'SUV', 'VAN', 'SPEED_LIMIT']

cyclists = cyclists.merge(crashes[cols],on=['CRN'],how='left').drop('VEH_TYP

del cols, persons
```

Missing data in 'cyclists'

Many fields have missing data which I would like to fill using medians or modes (either groupwise or global). I will later design the BikeSafePA model to predict whether a cyclist suffers serious injury or fatality, so I will wait to implement this median/mode imputation as part of a pipeline to avoid data leakage from test set to training set. However, here's the plan:

- Create a binary SERIOUS_OR_FATALITY which is 1 if INJ_SEVERITY is in ['susp_serious_inj','killed'] and 0 otherwise (including NaN): can do this now
- When only one of 'ROAD_CONDITION' or 'WEATHER' is missing, will sometimes use one to fill the other without relying on the sample distribution: can do this now
- Impute 'AGE' with groupwise median, grouped by 'MUNICIPALITY': implement in pipeline later
- Impute 'HOUR_OF_DAY' with groupwise mode, grouped by ('ILLUMINATION','CRASH_MONTH'): implement in pipeline later
- Impute 'RESTRAINT_HELMET', 'IMPACT_SIDE', 'GRADE', 'VEH_POSITION' by creating a new category 'unknown': can do this now
- Impute all other missing data using dataset mode for feature: implement in pipeline later

In [18]:
```
# When ROAD_CONDITION is dry, it makes sense to fill WEATHER to clear
cyclists.loc[(cyclists.ROAD_CONDITION=='dry')&(cyclists.WEATHER.isna()),'WEA

# When WEATHER is rain, is makes sense to fill ROAD_CONDITION to wet
cyclists.loc[(cyclists.WEATHER=='rain')&(cyclists.ROAD_CONDITION.isna()),'RO

# These have many NaN - create new 'unknown' category
for feature in ['RESTRAINT_HELMET', 'IMPACT_SIDE', 'GRADE', 'VEH_POSITION']:
    cyclists[feature]=cyclists[feature].fillna('unknown')

# Create injury severity flag
cyclists['SERIOUS_OR_FATALITY'] = cyclists['INJ_SEVERITY']\
                                  .apply(lambda x: 1 if x in ['killed','susp_s
# Also bin the age values
cyclists['AGE_BINS'] = pd.cut(cyclists.AGE,bins=range(0,101,10)).astype('str
```

Exporting cleaned dataframes

These will end up in the same directory as this notebook.

In [19]:
```
cyclists.to_csv('cyclists.csv',index=False)
crashes.to_csv('crashes.csv',index=False)
```