Phone Sensor Challenge

This was a 7 day challenge to detect car crashes using only the x, y and z accelerometers from mobile phones during a journey. The data is unlabelled so this is an unsupervised problem. Contrary to normal scientific reports, I have written the report in the first person to help convey my thought process during the challenge.

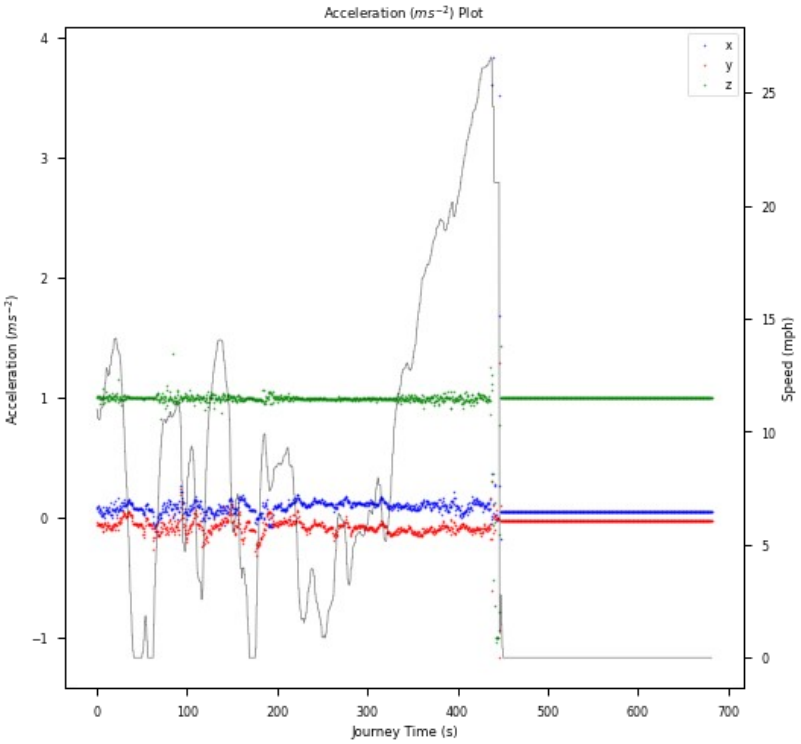
In this report I have examined the journey data manually to try to identify posiible crashes then trained an autoencoder as a method of anomaly detection based on the input features acceleration magnitude, jerk magnitude and change in velocity.

Data Exploration

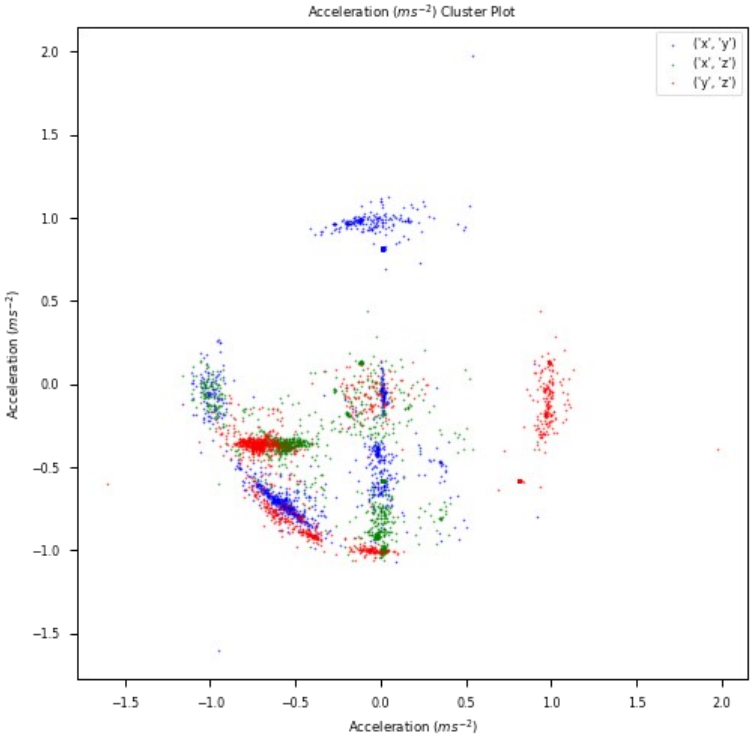
The initial step was to explore the data and see what I was working with. I have provided a separate script (exploration.py) which was the script I used to develop ideas and do preliminary data exploration. There is also a Jupyter notebook (DataExploration.ipynb) which can be used to examine the data.

The data is a mixture of gps and accelerometer data from a mobile phone. The two are separated and contained different information in their rows. GPS data is sampled at around 1Hz and accelerometer data is sampled at around 10Hz, however, there is variation in the timestep size. The GPS part of the data contains information about latitude, longitude, speed, bearing, height and accuracy. The accelerometer data contains accelerometer readings in x, y and z. I chose to concentrate on the accelerometer data to make predictions and used the GPS data to examine the journey and give extra information about last known speed. I used fill forward to fill the NaN values in the accelerometer rows and dropped the GPS rows.

Looking at the accelerometer data (see the plots on the next page), it was quite messy in places. I did not know the orientation of the phone in the car or how the axes aligned with the cars reference frame. Sometimes one axis measures 1G suggesting it is aligned with the vertical direction (1G is gravity acting on the accelerometer). However, it appears the phone orientation can change during the journey as the axis measuring 1G could change. As the phone could move and I didn’t have enough information to track the phone’s orientation I chose to try to abstract the accelerometer data from the orientation using the magnitude and other derived features. Another observation was that the data often becomes very messy when the vehicle is travelling at low speeds or is stationary and can contain large acceleration values. I would propose that the phone is being handled and this is causing the messy data. The large spikes could be caused by a drop or other sudden impact. In the final criteria for the model I decided to filter any prediction where the speed was less than 5mph to eliminate the large spikes caused at low speed or while stationary.



Acceleration plot showing noise at low speed
File: 05A03532-5BE9-4749-9460-CCCEADA6C786.csv



Acceleration cluster plot
File: 05A03532-5BE9-4749-9460-CCCEADA6C786.csv

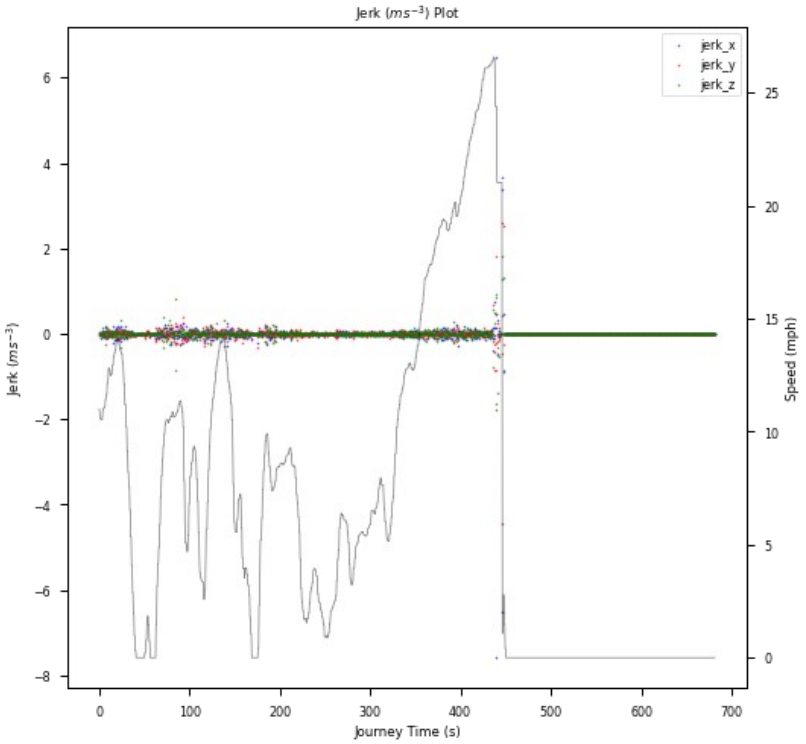
Preprocessing

The data preprocessing step cleans the data and prepares features which can be used to aid prediciton.
I found the following items which needed to be cleaned:

- Acceleration can be in ms^{-2} of G.
  - Fixed by looking at the peak acceleration magnitude. If it was greater than 15 then the data was likely in ms^{-2}. Converted to G by dividing by $9.81ms^{-2.}$
- One of the files had a space in the timestamp column name. ◦ Fixed by stripping whitespaces from the columns.
- Convert the unix epoch timestamps to datetime.
  - Converted in pandas using pd.to_datetime().
- The time periods were uneven.
  - resampled into 0.5s time windows using the mean values of the points within the window.

Resampling the data also had the effect of noise reduction. As this is signal data I had looked at applying a Fast Fourier Transform to identify noisy frequencies with the intention of applying a low pass filter. However, this did not work very well and the resampling had already provided a good level of noise reduction.

I spent a lot of time thinking about how to try to predict accidents. The most similar technology I could think of was airbag deployment so I looked into what criteria are used in airbag systems. The most popular criteria seemed to be jerk, change in velocity and acceleration. Jerk is defined as the derivative of the acceleration with respect to time. In other words, how quickly the acceleration is changing. This makes sense as a good criteria as there is a limit to how quickly a car can accelerate or break compared to how quickly it can accelerate or break if it hits a solid, stationary object. Change in velocity obeys similar constraints as jerk and is defined as the integral of acceleration with respect to time. I've calculated these numerical in the code. A further advantage to using these features is that it normalises the data around zero. The raw accelerations has a variable mean based on how the the axis is aligned to gravity and can change if the phone is moved within the vehicle. Using the derivative data removes these features.



Jerk plot showing the data normalised around zero
File: 05A03532-5BE9-4749-9460-CCCEADA6C786.csv

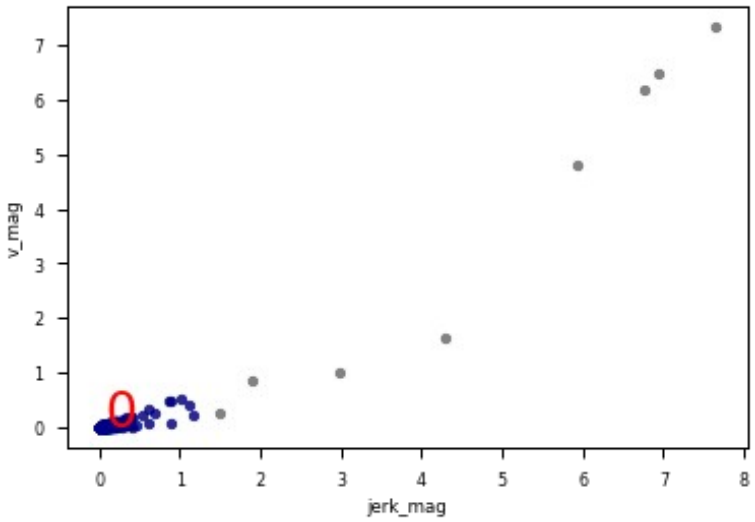
Methodology

The main objectives of the challenge are to identify journeys containing possible accident events and identify the time of the event, the severity of the accident and the confidence of the prediction. The data provided is unlabelled so my first task was to look through the data manually and see if I could identify potential accidents. My methodology here was to plot the journey on a map using the mplleaflet() function in the Jupyter notebook using the GPS locations and speed. I made a quick note for each journey and a human based prediciton on whether there was an accident. The notes are available in the results section.

The journey above has come to an abrupt stop at high speed on the Eisenhower Expressway. This was identified as a potential accident. The journey begins on the right of the image. Speed is indicated by colour.
File: 9EB59633-AD54-47F0-AC5F-61F3A183DB30.csv

As the data is unlabelled I knew I would need to use some form of unsupervised learning model. Initial attempts were made by looking at cluster plots of the aceleration data (as shown earlier). Examining the cluster plots, the majority of the data fell into nice clusters, but with some large outliers. This lead me to look at anomaly detection methods. A crash is an anomalous event so this made sense. I tried two models initially: DBSCAN and autoencoders.

DBSCAN (Density-based spatial clustering of applications with noise) is a clustering algorithm which which looks for groups of points which are closely packed together and labels the remaining points as outliers or noise. The user can set the search distance and the number of neighbour required to define a core point. Clusters are determined automatically and in unsupervised learning the clusters could be used for potential label creation. This is why DBSCAN is a more useful algorithm for cluster detection than the more popular k-means clustering, which requires the user to guess the potential number of clusters before modelling and assigns all points to a cluster regardless of whether or not they are outliers.



Initial attempt at using DBSCAN. The majority of the data is clustered between 0 and 1 with some outliers defined as anomalies.
file: 9EB59633-AD54-47F0-AC5F-61F3A183DB30.csv

An autoencoder is an artificial neural network which learns the important features of a dataset by reducing and compressing the dataset into a simpler representation, then attempts to recreate the original dataset from the reduced representation. This is useful in anomaly detection because the anomalous points will not be an important part of the reduction and will not be recreated by the autoencoder. The anomalous points are found by looking at the loss between the original dataset and the reconstruction. A large loss suggests an anomalous point.

Due to the time constraint I had to choose a model to go with and chose to use the autoencoder as I was more familiar with using this type of model compared to DBSCAN.

The advantages of using an autoencoder are:

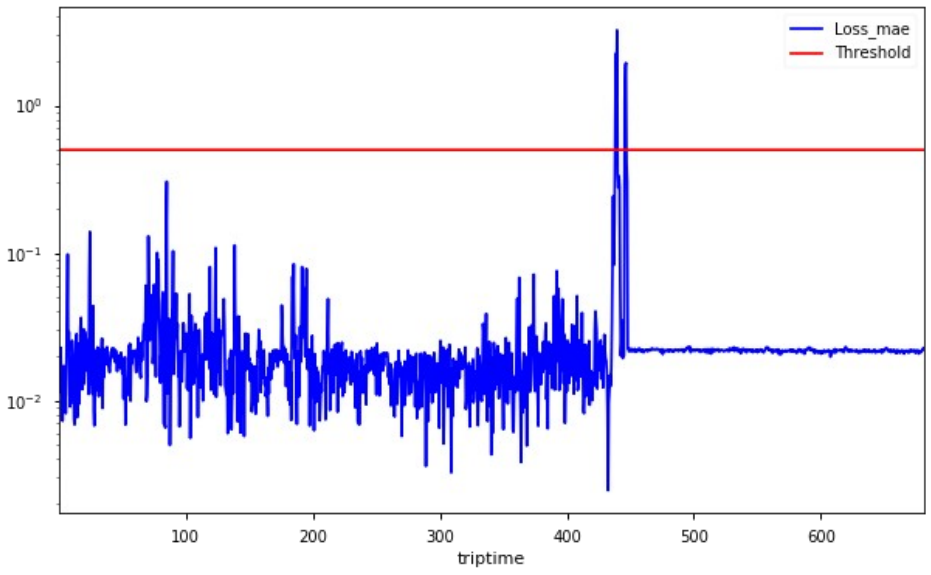
- Accuracy
- high-dimensionality – compared to DBSCAN which can only compare two columns at a time.
- Personally more familiar with the architecture and metrics (speed of execution)

The disadvantages of using an autoencoder are:

- Slow – the model took around 8 mins to run on my machine for the 25 test cases. The DBSCAN method was very fast to run.

Model

I used Keras to create the autoencoder model. Structurally autoencoders are quite simple and are a mirror image of node layers, having the same number of input layers as output layers. I chose to use 3 hiddens layers. Layer 1 and 3 contained 10 nodes and the central layer contained 2 nodes. The central layer is a compressed representation of the input and the autoencoder will try to recreate the input from this representation. This works as a form of dimensionality reduction or noise reduction and when the output is reconstructed the loss between and anomaly and the reconstruction will be large as the anomalous points are lost in the compressed representation. By setting a threshold value for loss, peaks in the loss above the threshold can be used to detect anomalies. I chose the intial threshold manually and refined it by checking against my notes on the dataset. The final threshold value was chosen as 0.5.
As inputs to the autoencoder I chose acceleration magnitude, jerk magnitude and the minimum change in velocity. Using the magnitudes removed direction dependence and could identify large spikes in both acceleration and jerk. The minimum velocity represented a large reduction of velocity as would be experienced in a crash.



Example of the autoencoder reconstruction loss exceeding the threshold. file: 9EB59633-AD54-47F0-AC5F-61F3A183DB30.csv

Evaluation Metrics

The autoencoder is trained on each dataset separately so there is not one general model which is trained then used for prediciton on new datasets like in other machine learning problems. For this reason I have not mentioned model generalisation. Generalisation comes from choosing an appropriate threshold value for the reconstruction loss. This is dependent on the training variables and their physical limits – for the case of a car crash the limits are based on normal acceleration and velocity limits. Given more time I would perform a more thorough study of these limits as they should be reasonably constant.

I initially used 20 training epochs and reduced the number of training epochs down to 10 as the model loss was already converging at around 10 epochs for the majority of the datasets. The initial difference between training and validation loss was quite large, however, they tended to converge by the 10th iteration and as the model is trained on each dataset, overfitting is not as important as it is for general models.

Severity and Confidence

I have used fairly simple criteria for defining the severity and confidence. Severity is based on the speed at the time of the potential accident. Most journeys appear to take place at maximum speeds of less than 30mph. The only one with a higher speed (50mph) appears to be due to inaccuracy of the GPS signal as the car appears to be in an underground car park and the signal jumps around. Due to the very noisy data and sometimes erroneous signals cause when the vehicle is either stationary or travelling at very low speeds, I have filtered any potential crashes that occur at less than 5mph. This is a trade off which eliminates the autoencoder triggering for journeys which I do not think include a potential crash due to the vehicle being stationary or containing very messy data at low speed. The advantage is a clearer peak in reconstruction loss for the events which I do believe have a potential crash. Given more time I would develop the criteria further to include the jerk and/or change in velocity as these give a better indication of the impulse and deceleration felt during the collision. The severity criteria are as follows:

Condition	Severity
Speed > 20 mph	0
Speed = 14mph to 20 mph	0
Speed < 14mph	0

For the confidence I have used the autoencoder reconstruction loss. When combined with the filtering of autoencoder events triggered at less than 5mph, the threshold of 0.5 provided a good indicator of the accidents which I thought could be a potential crash.

Condition	Confidence
Reconstruction loss > 1.5	High
Reconstruction loss = 0.7 to 1.5	Medium
Reconstruciton loss < 0.7	Low

Results

The model can be run using the predict.py file which uses the crash_detection.py module. Details are provided in README.txt.

The results of the model are provided in the table on the next page with my predictions and notes from examining the data. The details of the predicted crashes are given below. The full table with filenames is provided in results.csv.

As the data was unlabelled it was difficult to verify if the model performed well at identifying accidents. Based on my notes about which journeys may have contained accidents, the model predicted these as well as some of the journeys marked as 'maybe'. Two journeys which I identified as normal the model predicts as accidents and both are low confidence, minor predictions.

Table of Predicted Crashes

Index	Time	Severity	Confidence	Collision Speed	Max Loss
0	2015-05-14 00:42:27.500	Minor	Low	10.96	0.61
5	2015-03-29 16:49:13.500	Serious	High	24.39	3.24
6	2015-04-24 19:28:41.000	Minor	High	11.94	2.00
7	2015-04-19 22:24:30.000	Minor	Medium	13.51	1.31
11	2015-06-18 12:35:41.500	Minor	Low	5.80	0.53
12	2015-03-19 21:10:47.000	Moderate	Low	18.24	0.57
22	2015-04-13 21:21:08.500	Minor	Medium	5.45	0.72
23	2015-03-16 21:56:35.000	Moderate	Low	16.21	0.58

Conclusions

- An autoencoder model was used as a form of anomaly detection.
- Acceleration magnitude, jerk magnitude and largest reduciton in velocity were chosen as the input features.
- Without labelled data it was hard to determine which journeys included accidents.
- The data is noisy and it is difficult to determine what is happening at low speeds based only on the x,y and z accelerometer values.

Table of Observations with Model Predictions

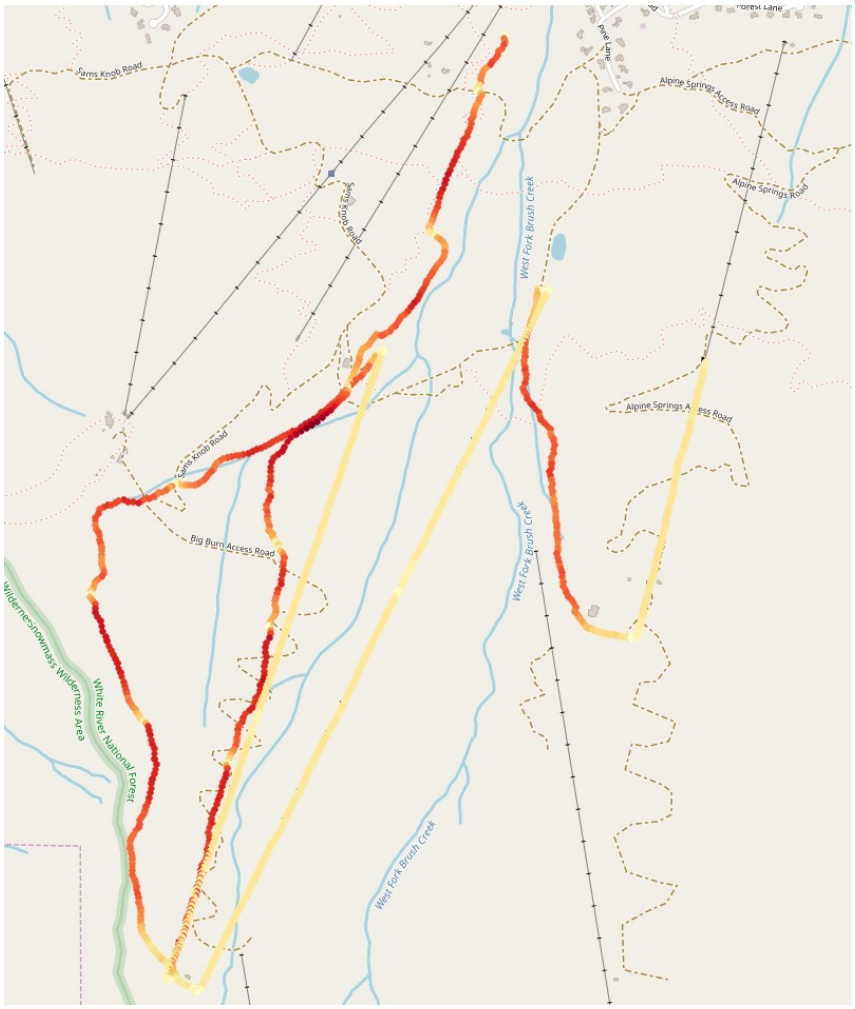
File	Human Crash	Jerk	AE trigger	Loss MAE peak	Note	Model Crash
0	No	-3	Yes	0.7	Normal trip, ends in tunnel. End occurs in a dead zone	1
1	No		No	0.4	Parks at airport	0
2	No		No	0.4	Parks at uni Messy at low speed	0
3	No		No	0.3	Parks at church V. messy < 5mph	0
4	No		Yes	0.7	Parks near park, doesn't look like a crash. V. messy < 5 mph and hits 2G	0
5	Yes		Yes	2.0	Looks like a crash on the highway.	1
6	Maybe		Yes	0.7	Could be a crash, decelerates quickly. Acceleration in a normal range	1
7	Maybe	1.5	Yes	>1	Looks like it could be a bump at traffic lights	1
8	No		Yes	>1	Goes into car park, very messy here. Gps speed spike from 0 to 50 (artefact)	0
9	Maybe		Yes	<1	No large accels	0
10	No		No	<0.4	Looks standard normal accel range	0
11	No		No	<0.4	Normal and parks normal accel	1
12	No		Yes	0.6	Skiing in Aspen the skiing parts are very messy	1
13	Maybe	3	No	<0.4	Ends in an odd place on the highway	0

14	Maybe	1.0	Yes	0.5	Accel and jerk are low	0
15	Maybe	3	Yes	0.8	Accel normal, high jerk Occurs while v low speed	0
16	No	2	Yes	0.8	Doesn't look like a crash occurs at v. low speed	0
17	No	1.5	No	0.3	Trip to the shops	0

18	No	2	No	0.1	Seems ot drive normally and park	0
19	No	1.5	No	<0.4	Spin around the block	0
20	No	-3	Yes	0.5	Only just triggers AE Journey looks normal	0
21	No	1	No	0.3	Looks normal	0
22	Yes	5	Yes	1.1	High AE trigger and jerk. Cant tell from map	1
23	Yes	3	Yes	0.9	Loses speed very quickly then stops Journey ends abruptly	1
24	No	4	Yes	0.9	Peak occurs while stationary	0

Finally…

The journey in file 844709EE-F9AA-4B66-B682-860339FCBC1C.csv appears to be a day out on the ski slopes in Aspen. The skiier must have left their phone in their pocket. It also registers as a crash! Confidence: Low, Severity: Moderate, Speed at time of collision:
18.24mph. Hopefully they did not get hurt.



File: 844709EE-F9AA-4B66-B682-860339FCBC1C.csv