Bilgin_Sherifov__telemetric

July 29, 2019

0.0.1 Submition for

0.1 **** - Data Science Challenge

by Bilgin Sherifov Date: ****
Packages to install before running the notebook:

1. Python 3.x
2. Pandas
3. Numpy
4. Matplotlib
5. Seaborn

(python -m pip install --user numpy matplotlib pandas seaborn)

0.1.1 Note: To run the accident detection algorithm:

1. Run all cells in this notebook until section "Run this algorithm for each file and detect head-on-collisions from a moving vehicle"
2. Run the cell below the above-mentined section, reading the instructions under its Notes sub-section

```
In [3]: # import numpy and pandas
        import numpy as np
        import pandas as pd

        # import plotting packages
        import matplotlib.pyplot as plt
        import matplotlib.dates as mdates
        import seaborn as sns; sns.set();
        %matplotlib inline

        # time-related libraries
        import time as tm
        import datetime as dt
        from datetime import timedelta

        # system and OS libraries
        import os
        import os.path


        from IPython.core.display import display, HTML
        display(HTML("<style>.container { width:95% !important; }</style>"))

        sns.set_context("poster") # paper, notebook, talk, poster
```

```
<IPython.core.display.HTML object>
```

1 FUNCTIONS

```
In [4]: def gps_to_km(gps_1, ref_latitude, ref_longitude, scale):
            # source: https://www.movable-type.co.uk/scripts/latlong.html
            rad_per_deg = np.pi / 180
            R = 6378.137 # Radius of earth in KM
            delta_latitude = (gps_1['lat'] - ref_latitude) * rad_per_deg
            delta_longitude = (gps_1['lon'] - ref_longitude) * rad_per_deg

            a = (
                np.sin(delta_latitude/2) * np.sin(delta_latitude/2)
                + np.cos(ref_latitude * rad_per_deg)
                * np.cos(gps_1['lat'] * rad_per_deg)
                * np.sin(delta_longitude/2) * np.sin(delta_longitude/2)
            )
            c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a));
            distances = R * c # km

            return (distances / scale)#.apply(np.round,0).astype(int)


        def gps_to_angle(gps_1, ref_latitude, ref_longitude, scale):
            # source: https://www.movable-type.co.uk/scripts/latlong.html
            rad_per_deg = np.pi / 180
            R = 6378.137 # Radius of earth in KM
            delta_latitude = (gps_1['lat'] - ref_latitude) * rad_per_deg
            delta_longitude = (gps_1['lon'] - ref_longitude) * rad_per_deg

            y = np.sin(delta_longitude) * np.cos(ref_longitude * rad_per_deg);
            x = (
                np.cos(ref_latitude * rad_per_deg) * np.sin(gps_1['lat'] * rad_per_deg)
                - np.sin(ref_latitude * rad_per_deg)
                * np.cos(gps_1['lat'] * rad_per_deg)
                * np.cos(delta_longitude)
            )
            angle_ = np.arctan2(y, x) / rad_per_deg + 180
```

```
    return (angle_ / scale)#.apply(np.round,0).astype(int)


def center_around_ref(data_airports, ref_index, scale):

    temp = (
        data_airports[['lat', 'lon']]
        .subtract(data_airports.loc[ref_index][['lat', 'lon']])
        * scale
    )

    data_airports[['latitude_centered_on_' + code_ref_airport, 'longitude_centered_on_
        temp.subtract(temp.min())
    ).astype(int)

    return data_airports


def from_latlon(latitude, longitude, force_zone_number = None, force_zone_letter=None)

    """This function convert Latitude and Longitude
        to UTM (Universal Transverse Mercator coordinate system) coordinates

        Parameters
        ----------
        latitude: float
            Latitude between 80 deg S and 84 deg N, e.g. (-80.0 to 84.0)
        longitude: float
            Longitude between 180 deg W and 180 deg E, e.g. (-180.0 to 180.0).
            More information see utmzones [1]_
       .. _[1]: http://www.jaworski.ca/utmzones.htm

       source: https://github.com/Turbo87/utm
    """

    def in_bounds(x, lower, upper, upper_strict=False):
        if upper_strict and use_numpy:
            return lower <= mathlib.min(x) and mathlib.max(x) < upper
        elif upper_strict and not use_numpy:
            return lower <= x < upper
        elif use_numpy:
            return lower <= mathlib.min(x) and mathlib.max(x) <= upper
        return lower <= x <= upper

    # ====
    def check_valid_zone(zone_number, zone_letter):
        if not 1 <= zone_number <= 60:
            raise OutOfRangeError('zone number out of range (must be between 1 and 60)

        if zone_letter:
            zone_letter = zone_letter.upper()

            if not 'C' <= zone_letter <= 'X' or zone_letter in ['I', 'O']:
                raise OutOfRangeError('zone letter out of range (must be between C and

# ====
def mixed_signs(x):
    return use_numpy and mathlib.min(x) < 0 and mathlib.max(x) >= 0

# ====
def negative(x):
    if use_numpy:
        return mathlib.max(x) < 0
    return x < 0

# ====
def latlon_to_zone_number(latitude, longitude):
    # If the input is a numpy array, just use the first element
    # User responsibility to make sure that all points are in one zone
    if use_numpy:
        if isinstance(latitude, mathlib.ndarray):
            latitude = latitude.flat[0]
        if isinstance(longitude, mathlib.ndarray):
            longitude = longitude.flat[0]

    if 56 <= latitude < 64 and 3 <= longitude < 12:
        return 32

    if 72 <= latitude <= 84 and longitude >= 0:
        if longitude < 9:
            return 31
        elif longitude < 21:
            return 33
        elif longitude < 33:
            return 35
        elif longitude < 42:
            return 37

    return int((longitude + 180) / 6) + 1

# ====
def latitude_to_zone_letter(latitude):
    # If the input is a numpy array, just use the first element
    # User responsibility to make sure that all points are in one zone
    if use_numpy and isinstance(latitude, mathlib.ndarray):
        latitude = latitude.flat[0]

    if -80 <= latitude <= 84:
        return ZONE_LETTERS[int(latitude + 80) >> 3]
    else:
        return None

# ====
def zone_number_to_central_longitude(zone_number):
    return (zone_number - 1) * 6 - 180 + 3

# ====
try:
    import numpy as mathlib
    use_numpy = True
except ImportError:
    import math as mathlib
    use_numpy = False

__all__ = ['to_latlon', 'from_latlon']

K0 = 0.9996

E = 0.00669438
E2 = E * E
E3 = E2 * E
E_P2 = E / (1.0 - E)

SQRT_E = mathlib.sqrt(1 - E)
_E = (1 - SQRT_E) / (1 + SQRT_E)
_E2 = _E * _E
_E3 = _E2 * _E
_E4 = _E3 * _E
_E5 = _E4 * _E

M1 = (1 - E / 4 - 3 * E2 / 64 - 5 * E3 / 256)
M2 = (3 * E / 8 + 3 * E2 / 32 + 45 * E3 / 1024)
M3 = (15 * E2 / 256 + 45 * E3 / 1024)
M4 = (35 * E3 / 3072)

P2 = (3. / 2 * _E - 27. / 32 * _E3 + 269. / 512 * _E5)
P3 = (21. / 16 * _E2 - 55. / 32 * _E4)
P4 = (151. / 96 * _E3 - 417. / 128 * _E5)
P5 = (1097. / 512 * _E4)

R = 6378137
ZONE_LETTERS = "CDEFGHJKLMNPQRSTUVWXX"
```

```
# -----
if force_zone_number is not None:
    check_valid_zone(force_zone_number, force_zone_letter)

lat_rad = mathlib.radians(latitude)
lat_sin = mathlib.sin(lat_rad)
lat_cos = mathlib.cos(lat_rad)

lat_tan = lat_sin / lat_cos
lat_tan2 = lat_tan * lat_tan
lat_tan4 = lat_tan2 * lat_tan2

if force_zone_number is None:
    zone_number = latlon_to_zone_number(latitude, longitude)
else:
    zone_number = force_zone_number

if force_zone_letter is None:
    zone_letter = latitude_to_zone_letter(latitude)
else:
    zone_letter = force_zone_letter


lon_rad = mathlib.radians(longitude)
central_lon = zone_number_to_central_longitude(zone_number)
central_lon_rad = mathlib.radians(central_lon)

n = R / mathlib.sqrt(1 - E * lat_sin**2)
c = E_P2 * lat_cos**2

a = lat_cos * (lon_rad - central_lon_rad)
a2 = a * a
a3 = a2 * a
a4 = a3 * a
a5 = a4 * a
a6 = a5 * a

m = R * (M1 * lat_rad -
         M2 * mathlib.sin(2 * lat_rad) +
         M3 * mathlib.sin(4 * lat_rad) -
         M4 * mathlib.sin(6 * lat_rad))

easting = K0 * n * (a +
                    a3 / 6 * (1 - lat_tan2 + c) +
                    a5 / 120 * (5 - 18 * lat_tan2 + lat_tan4 + 72 * c - 58 * E_P2)

northing = K0 * (m + n * lat_tan * (a2 / 2 +
                                    a4 / 24 * (5 - lat_tan2 + 9 * c + 4 * c**2) +
                                    a6 / 720 * (61 - 58 * lat_tan2 + lat_tan4 + 60

    if mixed_signs(latitude):
        raise ValueError("latitudes must all have the same sign")
    elif negative(latitude):
        northing += 10000000

    return easting, northing


def load_data(file_path_name):
    df_data = pd.read_csv(file_path_name)
    df_data.rename(columns={' timestamp ':'timestamp'}, inplace=True)
    df_data['timestamp'] = df_data['timestamp'].astype('uint64')

    return df_data

def split_gps_phone(df_data):
    df_gps = dfData[dfData['type'] == 'gps'][['timestamp', 'type', 'lat', 'lon', 'heig
    df_gps.set_index('timestamp', drop=False, inplace=True)
    df_gps['timestamp'] /= 1000

    df_gps.index = pd.to_datetime(df_gps.index, unit='ms')
    x, y = from_latlon(df_gps['lat'].values, df_gps['lon'].values)
    df_gps['lat utm'] = x - x[0]
    df_gps['lon utm'] = y - y[0]
    df_gps['distance'] = (
        (
            df_gps['lat utm'].diff().pow(2)
            + df_gps['lon utm'].diff().pow(2)
        )
        .apply(np.sqrt)
        .cumsum()
        #.rolling(9, win_type='triang', center =True)
        #.mean()
    )
    df_gps['distance'].fillna(method = 'bfill', inplace=True)
    df_gps['derived speed'] = df_gps['distance'].diff().rolling(3, win_type='triang', c
    df_gps['derived speed'].fillna(method = 'bfill', inplace=True)
    df_gps['velocity_x'] = df_gps['lat utm'].diff().rolling(3, win_type='triang', cente
    df_gps['velocity_x'].fillna(method = 'bfill', inplace=True)
    df_gps['velocity_y'] = df_gps['lon utm'].diff().rolling(3, win_type='triang', cente
    df_gps['velocity_y'].fillna(method = 'bfill', inplace=True)
    df_gps['acceleration_x'] = df_gps['velocity_x'].diff()
    df_gps['acceleration_x'].fillna(method = 'bfill', inplace=True)
    df_gps['acceleration_y'] = df_gps['velocity_y'].diff()
    df_gps['acceleration_y'].fillna(method = 'bfill', inplace=True)
    df_gps['derived acc'] = (df_gps['acceleration_x'].pow(2) + df_gps['acceleration_y']
    df_gps['linear acceleration'] = df_gps['speed'].diff().rolling(3, win_type='triang
    df_gps['delta direction'] = (180 - abs(abs(df_gps['bearing'].diff()) - 180)).rollin
    df_gps.fillna(method = 'bfill', inplace=True)

    df_phone = dfData[dfData['type'] == 'accelerometer'][['timestamp', 'type', 'x', 'y
    df_phone['linear acceleration'] = (df_phone['x'].pow(2) + df_phone['y'].pow(2) + di
    m_r = df_phone['linear acceleration'].mean()
    if m_r < 9.0:
        df_phone['x'] *= G / m_r
        df_phone['y'] *= G / m_r
        df_phone['z'] *= G / m_r
        df_phone['linear acceleration'] *= G / m_r
    df_phone['linear acceleration'] -= G
    df_phone['var linear acceleration'] = df_phone['linear acceleration'].diff().abs()
    df_phone.set_index('timestamp', drop=False, inplace=True)
    df_phone.index = pd.to_datetime(df_phone.index, unit='ms')
    df_phone['timestamp'] /= 1000

    return df_gps, df_phone
```

2 WORKSPACE

```
In [5]: DATA_PATH = os.getcwd()
        os.mkdir(os.path.join(DATA_PATH, 'results'))
        RESULTS_PATH = os.path.join(DATA_PATH, 'results')
        print('DATA_PATH:', DATA_PATH)
        print('RESULTS_PATH:', RESULTS_PATH)
```

```
        ---------------------------------------------------------------------------

        FileExistsError                           Traceback (most recent call last)

        <ipython-input-5-a246e717d8b3> in <module>()
          1 DATA_PATH = os.getcwd()
    ----> 2 os.mkdir(os.path.join(DATA_PATH, 'results'))
          3 RESULTS_PATH = os.path.join(DATA_PATH, 'results')
          4 print('DATA_PATH:', DATA_PATH)
          5 print('RESULTS_PATH:', RESULTS_PATH)


        FileExistsError: [WinError 183] Cannot create a file when that file already exists: 'E
```

2.1 VISUAL EXPLORATION OF DATA

2.1.1 First, some data manipulation:

1. Load time series data from a given file into a Pandas data-frame object
2. Split the table into two seperate tables -- one containing only GPS data and the other containing only accelerometer data
3. To make it easier to read for humans, in each table turn unix timestamps into datetime objects with date, hour, minute, second, and milisecond parts
4. In each table, set the timestamp column as a row index.
5. In the table containing the GPS data:

- Add new latitude and longitude columns which are in UTM units and are expressed as meters relative to the initial position of the vehicle
- Derive latitde and longititude velocities from the new UTM coordinates
- Derive latitde and longititude accelerations from the new UTM coordinates (low-pass filter the difference o the UTM coordinates).
- Drive the linear acceleration from the speed signal (low-pass filter the time-difference of the speed data).
- Derive the rate of change of direction from the bearing signal, by taking into account the circular nature (distance between 360 and 10 is the same as between 360 and 350) of the data (low-pass filter the circular time-difference of the bearing signal)

6. In the table containing the accelerometer data:

- Derive linear acceleration from the x, y, and z accelerations
- Check the average value of the linear acceleration. If it is a number close to 1.0, then accelerations are in units of g. Turn those to m/s^2
- Remove 9.80665 m/s^2 from the linear acceleration to get the actual vehicle acceleration
- Derive a measure of the time-varying variation in linear accelerat (low-pass filter the absolute value of the time-difference of the linear acceleration)
-

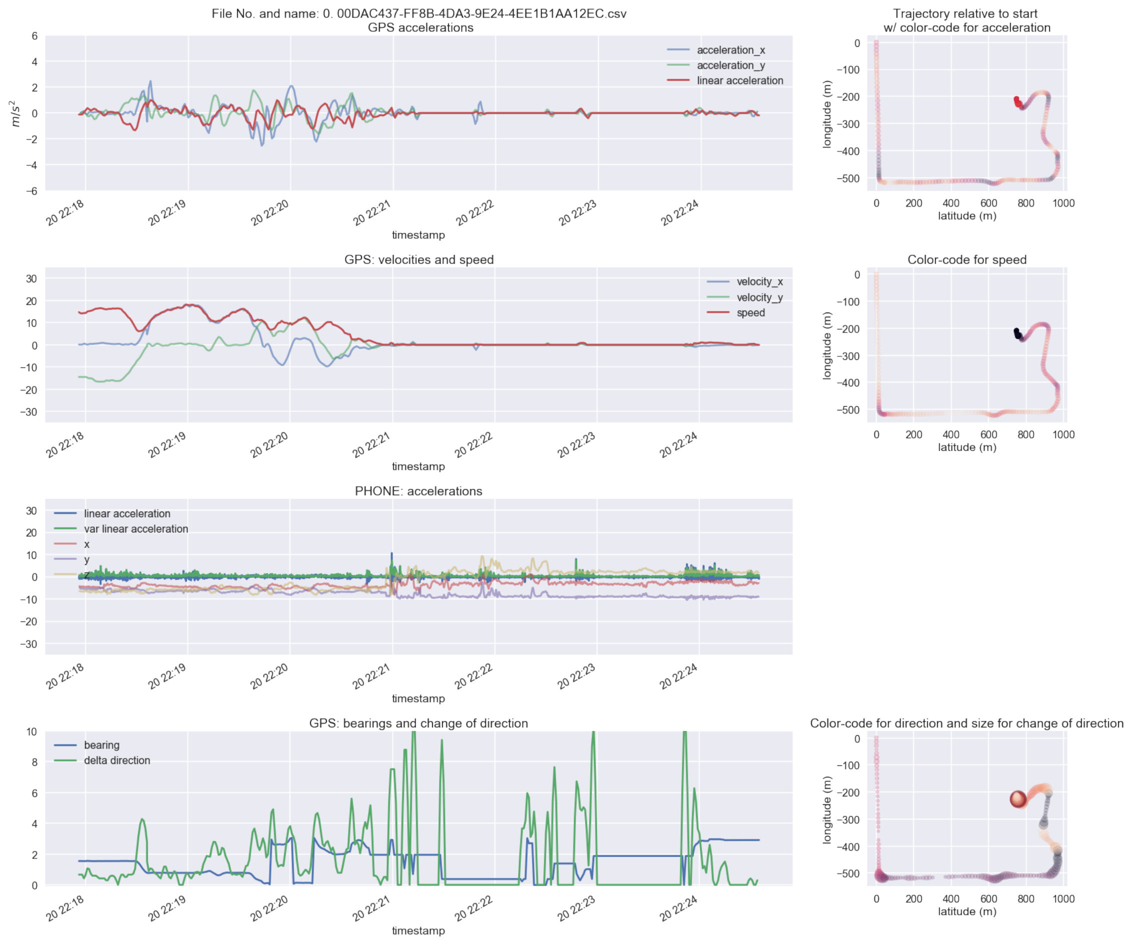
2.1.2 Next, plot the series in each file:

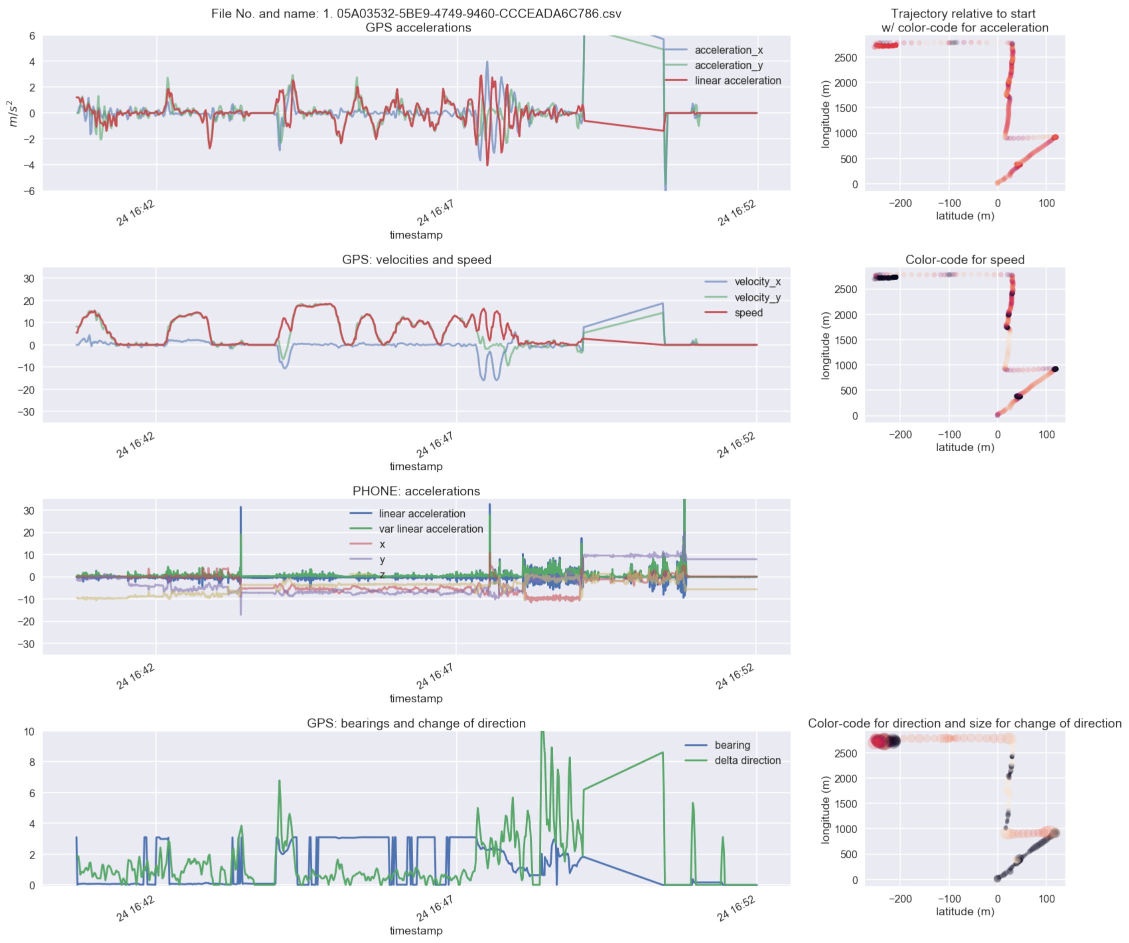
- For each data set (file) there are eight plots, organised in four rows and two columns.
- On each row (except the third), the first (wider) panel plots selected varible as a function of time.
- The second panel in a row is a sctter plot of the latitude vs longitude trajectory, in meters and relative to initil position -- initial position always starts from the point (0,0). Either the color or the size (or both) of the symbols along the trajectory code a given variable, indicated in the sub-title of the panel.

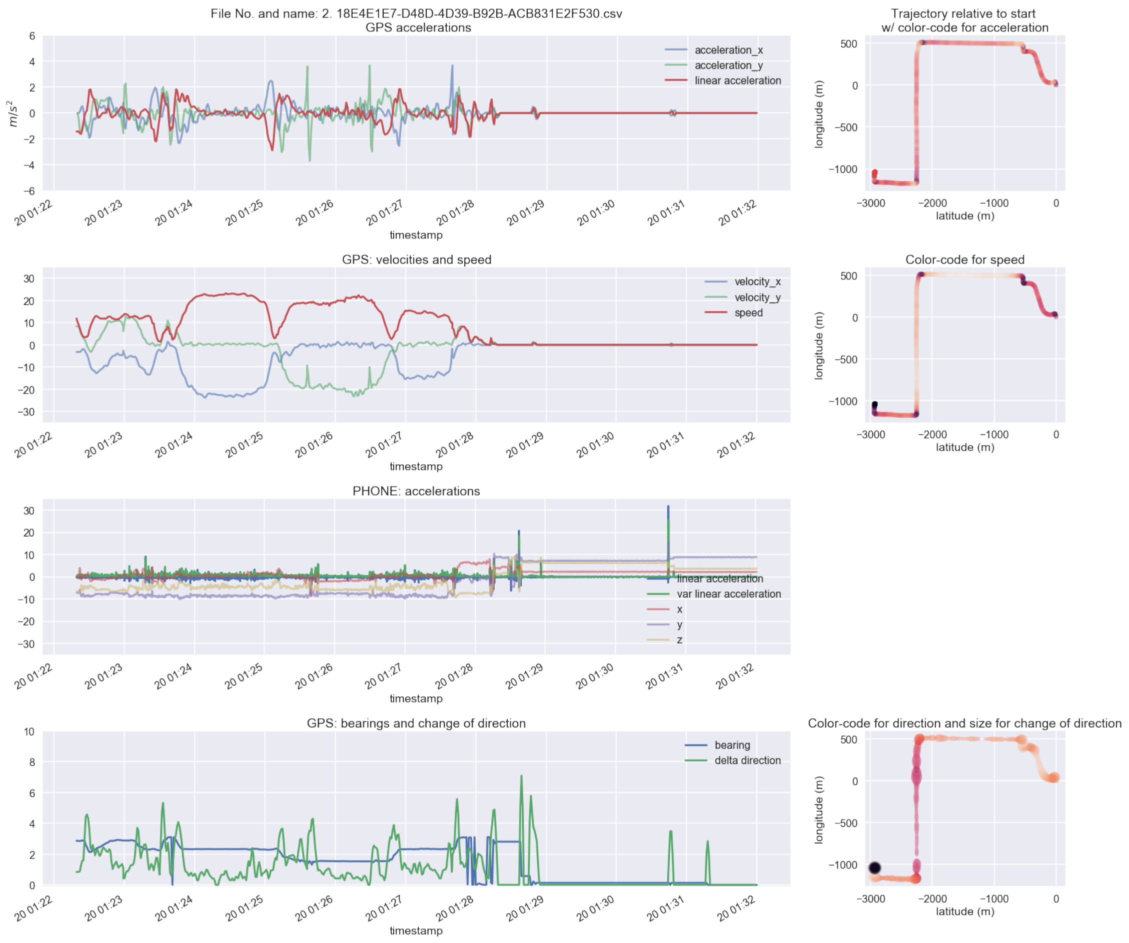
2.1.3 Attention:

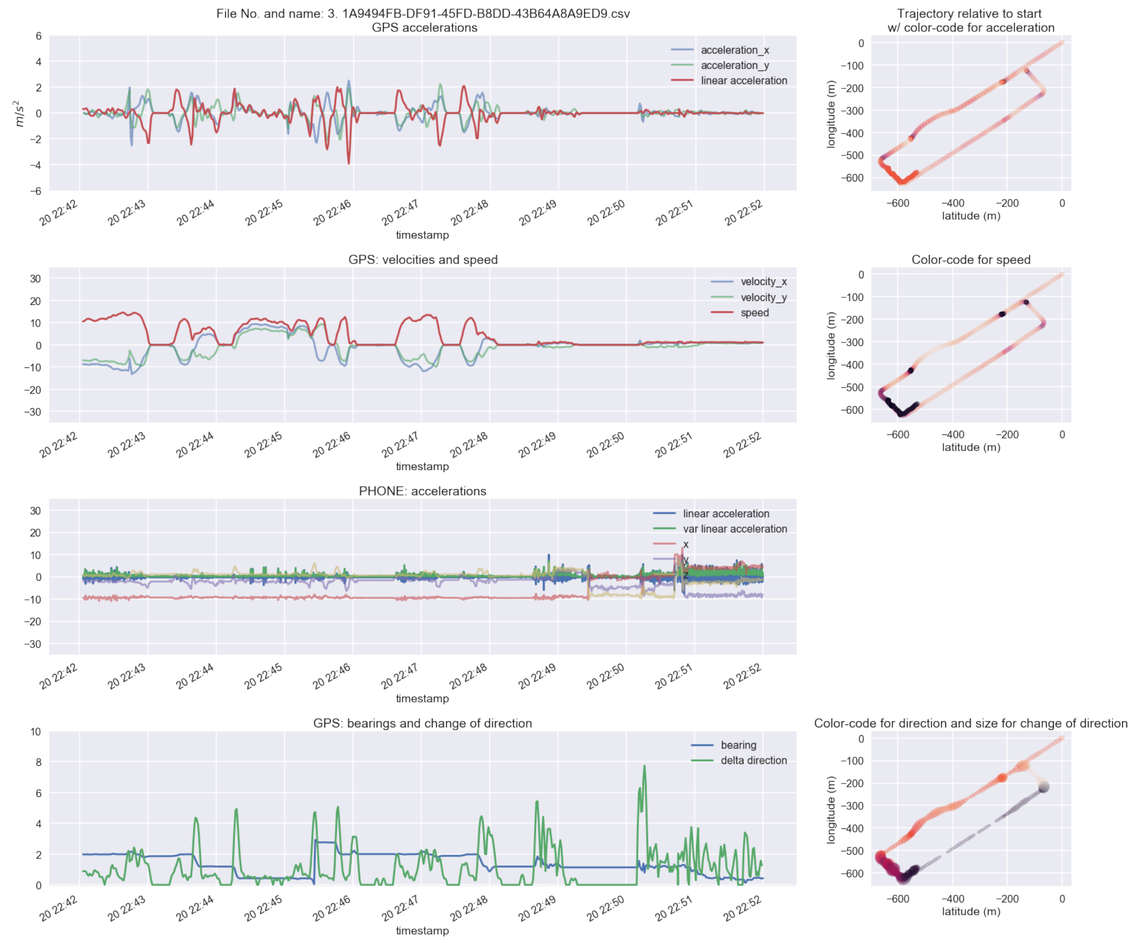
Before running the code below, pease choose ho many files to load and plot by setting the NUMBER_OF_FILES_TO_PLOT flag

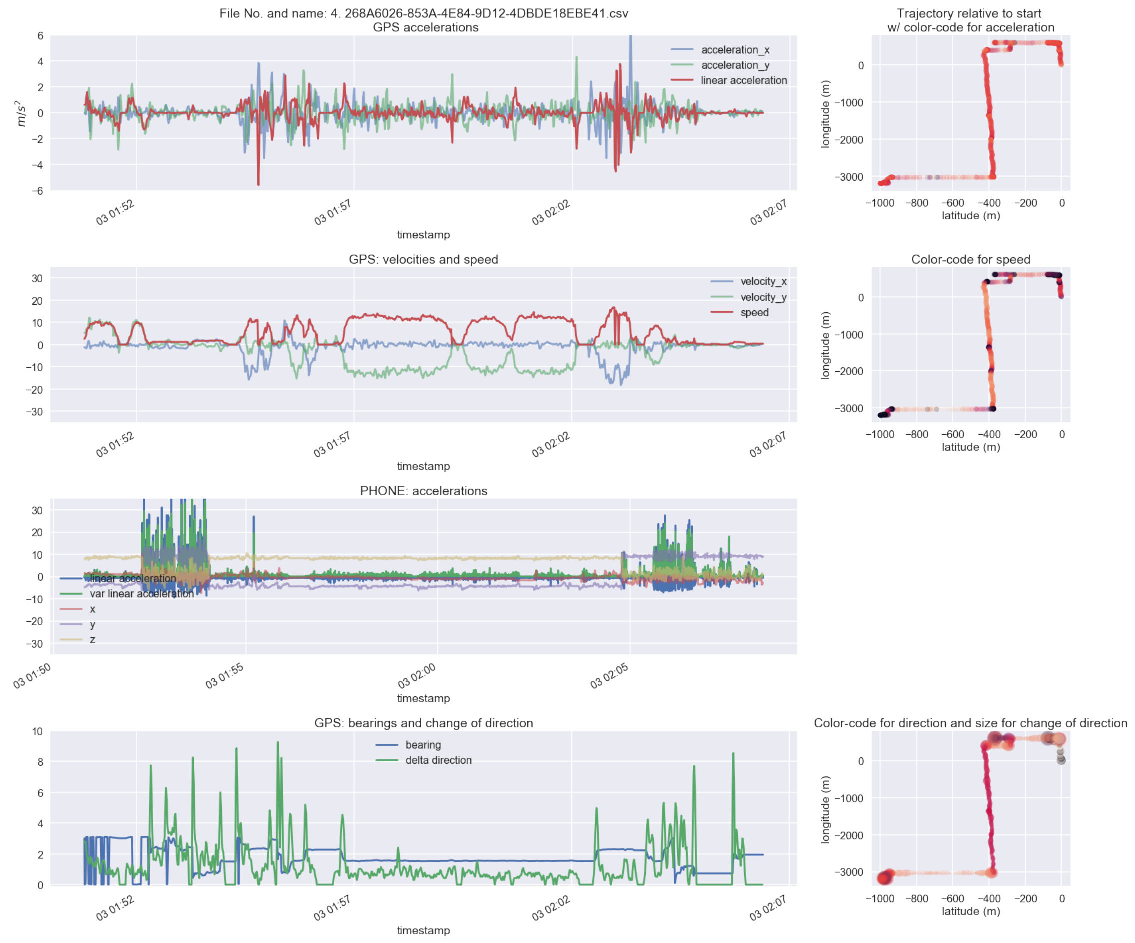
```
In [21]: NUMBER_OF_FILES_TO_PLOT = 5 # max is 25

         G = 9.80665
         alpha_bg = 0.6
         pow_delta_dir = 0.5

         # read the files in the directory
         files = [f for f in os.listdir(DATA_PATH) if f.endswith(".csv")]

         # iterate through the fies and plot
         for idx, file_ in enumerate(files[:NUMBER_OF_FILES_TO_PLOT]):
             dfData = load_data(os.path.join(DATA_PATH, file_))

             dfGPS, dfAccelerometer = split_gps_phone(dfData)

             scl = 10

             plt.close('all')
             plt.figure(figsize=(24,20))

             # ROW 1, accelerations, GPS
             ax = plt.subplot2grid((4,4),(0,0), colspan=3)
             dfGPS['acceleration_x'].rolling(3, win_type='triang', center =True).mean().plot(a
             dfGPS['acceleration_y'].rolling(3, win_type='triang', center =True).mean().plot(a
             dfGPS['linear acceleration'].plot(ax=ax)
             #dfGPS['derived acc'].plot(ax=ax)
             plt.ylim([-6, 6])
             plt.ylabel('$m/s^2$')
             plt.legend()
             plt.title('File No. and name: '+ str(idx) + '. ' +file_ + '\n GPS accelerations')
             plt.subplot2grid((4,4),(0,3), colspan=1)
             plt.scatter(dfGPS['lat utm'].values, dfGPS['lon utm'].values, c=dfGPS['linear acc
             plt.title('Trajectory relative to start \nw/ color-code for acceleration')
             plt.xlabel('latitude (m)')
             plt.ylabel('longitude (m)')


             # ROW 2, speed. GPS
             ax = plt.subplot2grid((4,4),(1,0), colspan=3)
             dfGPS['velocity_x'].plot(ax=ax, alpha = alpha_bg)
             dfGPS['velocity_y'].plot(ax=ax, alpha = alpha_bg)
             dfGPS['speed'].plot(ax=ax)
             #dfGPS['derived speed'].plot(ax=ax)
             plt.ylim([-35, 35])
             plt.legend()
             plt.title('GPS: velocities and speed')
             plt.subplot2grid((4,4),(1,3), colspan=1)
             plt.scatter(dfGPS['lat utm'].values, dfGPS['lon utm'].values, c=dfGPS['speed'].va
             plt.title('Color-code for speed')
             plt.xlabel('latitude (m)')
             plt.ylabel('longitude (m)')

             # ROW 3, accelerations, ACCELEROMETER
             ax = plt.subplot2grid((4,4),(2,0), colspan=3)
             dfAccelerometer['linear acceleration'].plot(ax=ax)
             dfAccelerometer['var linear acceleration'].plot(ax=ax)
             (dfAccelerometer['x']).rolling(7, win_type='triang', center =True).mean().plot(ax
             (dfAccelerometer['y']).rolling(7, win_type='triang', center =True).mean().plot(ax
             (dfAccelerometer['z']).rolling(7, win_type='triang', center =True).mean().plot(ax
             plt.ylim([-35, 35])
             plt.legend()
             plt.title('PHONE: accelerations')

             # ROW 4, position, GPS
             ax = plt.subplot2grid((4,4),(3,0), colspan=3)
             #(.001 * dfGPS['lat utm']).plot(ax=ax)
             #(.001 * dfGPS['lon utm']).plot(ax=ax)
             (np.pi * dfGPS['bearing']/365).plot(ax=ax)
             temp = dfGPS['delta direction'].abs().pow(pow_delta_dir)
             temp.plot(ax=ax, label = 'delta direction')
             plt.ylim([-0.1, 10])
             plt.legend()
             plt.title('GPS: bearings and change of direction')
             plt.subplot2grid((4,4),(3,3), colspan=1)
             plt.scatter(
                 dfGPS['lat utm'].values,
                 dfGPS['lon utm'].values,
                 c =dfGPS['bearing'].values,
                 s = 60 * temp,
                 alpha=0.2)
             plt.title('Color-code for direction and size for change of direction')
             plt.xlabel('latitude (m)')
             plt.ylabel('longitude (m)')

             #print('Duration of series:', (dfAccelerometer.index[-1] - dfAccelerometer.index[
             #print('Distance covered: {:,} meters'.format(int(dfGPS['distance'][-1])))

             plt.tight_layout()
             plt.show()
```


File No. and name: 0. 00DAC437-FF8B-4DA3-9E24-4EE1B1AA12EC.csv
GPS accelerations
acceleration_x
acceleration_y
linear acceleration
m/s^2
timestamp
Trajectory relative to start
w/ color-code for acceleration
longitude (m)
latitude (m)
GPS: velocities and speed
velocity_x
velocity_y
speed
Color-code for speed
PHONE: accelerations
linear acceleration
var linear acceleration
x
y
z
GPS: bearings and change of direction
bearing
delta direction
Color-code for direction and size for change of direction


File No. and name: 1. 05A03532-5BE9-4749-9460-CCCEADA6C786.csv
GPS accelerations
m/s^2
acceleration_x
acceleration_y
linear acceleration
timestamp
24 16:42
24 16:47
24 16:52
Trajectory relative to start w/ color-code for acceleration
longitude (m)
latitude (m)
GPS: velocities and speed
velocity_x
velocity_y
speed
Color-code for speed
PHONE: accelerations
linear acceleration
var linear acceleration
x
y
z
GPS: bearings and change of direction
bearing
delta direction
Color-code for direction and size for change of direction


File No. and name: 2. 18E4E1E7-D48D-4D39-B92B-ACB831E2F530.csv
GPS accelerations
m/s^2
acceleration_x
acceleration_y
linear acceleration
timestamp
Trajectory relative to start w/ color-code for acceleration
longitude (m)
latitude (m)
GPS: velocities and speed
velocity_x
velocity_y
speed
Color-code for speed
PHONE: accelerations
linear acceleration
var linear acceleration
x
y
z
GPS: bearings and change of direction
bearing
delta direction
Color-code for direction and size for change of direction


File No. and name: 3. 1A9494FB-DF91-45FD-B8DD-43B64A8A9ED9.csv
GPS accelerations
acceleration_x
acceleration_y
linear acceleration
m/s^2
timestamp
Trajectory relative to start
w/ color-code for acceleration
longitude (m)
latitude (m)
GPS: velocities and speed
velocity_x
velocity_y
speed
timestamp
Color-code for speed
longitude (m)
latitude (m)
PHONE: accelerations
linear acceleration
var linear acceleration
x
timestamp
GPS: bearings and change of direction
bearing
delta direction
timestamp
Color-code for direction and size for change of direction
longitude (m)
latitude (m)



2.2 Observations

1. It looks like the GPS and the accelerometer data come from different devices. The GPS data seems to come from a device that is mounted to the vehicle while the accelerometer data comes from a mobile phone.
2. Not all files are related to automobiles. For example, the data in the file 844709EE-F9AA-4B66-B682-860339FCBC1C.csv seems to be from a skier or a snowboarder and has a different acceleration profile
3. In many situations the accelerations measured by phone while the vehicle is in motion have much smaller amplitudes than when it is in halt. This probably implies that the phone is subjected to much bigger velocity changes when the owner is walking than when driving.
4. In many files there are periods with missing GPS data.

2.3 Detecting Events

2.3.1 Detecting vehicle-on-vehicle head-on collisions with sensors in the moving vehicle:

- Look for drastic changes in GPS and phone acceleration measurements:
- linear acceleration in the vehicle-mounted GPS device should be below some threshold value

- the phone should register equally sudden changes in acceleration, but it could be in any direction
- The speed of the vehicle right before the incident should be above zero and then come quickly to zero

Severe incedents:

- After the incident, if the driver is severely injured, probably he/she will not be able t move for a while. This indicates that there should be not much changes in phone acceleration data. This can be detected by observig the variations in the linear accelerations in the phone acceleration readings.

Less-sever incident:

- If the accident is mild the driver will probably jump out of the vehicle and get his/her phone with himself/herself. This should be registered as variations in phone acceleration data.

An algorithm:

- The function below implements the above-mentioned heuristics.

- The various parameters used are not derived from any statistical analysis but by eye-balling the data
- It is in no way optimised for speed or accuracy and is not teste for reliability and is just as a proof of concept
- The code is supplied with enough comments to make it easily readble

```
In [18]: def vehicle_on_vehicle_collision_detector(
                df_gps, df_phone,
                thrshld_gps_acc = -3.0, thrshld_phone_acc = 10.0,
                window_after_evet = 30, thrshld_speed=1.0, thrshld_var_acc=0.5
        ):

                # get all instances that pass gps acceleration threshold
                idx_events_gps = df_gps[df_gps['linear acceleration'] <= thrshld_gps_acc].index

                # get all instances that pass phone acceleration threshold
                idx_events_phone = df_phone[df_phone['var linear acceleration'] >= thrshld_phone_a

                # If there are detected instances of GPS acc < thrshld_gps_acc
                # and there are less than 5 such instances
                # and there is also an isntance of Phone acc > thrshld_phone_acc,
                # then ther might be an event
                if (len(idx_events_gps) > 0) and (len(idx_events_gps) < 5) and (len(idx_events_pho
                    # take the last GPS event as possible event time
                    time_of_event_gps = idx_events_gps[-1]

                    # find the phone event nearest to the GPS event
```

```
            temp = list(abs(idx_events_phone - time_of_event_gps))
            idx_nearest_phone = temp.index(min(temp))
            time_of_event_phone = idx_events_phone[idx_nearest_phone]

            # get the indexes of the a points that are 2 seconds before the event
            # and 2 seconds after the event
            delta_time_before_gps = df_gps.loc[:time_of_event_gps].index[-20]
            delta_time_aftere_gps = df_gps.loc[time_of_event_gps:].index[20]

            # get the average speed of the car before and after the event
            mean_spead_before = df_gps.loc[delta_time_before_gps:time_of_event_gps]['speed
            mean_speed_after = df_gps.loc[time_of_event_gps + pd.Timedelta(seconds=2):]['s

            # if the average speed of the car was above zero before th event
            # and then came to halt after the event, then there was a head-on-collision
            # If the speed before the event was > 5 m/s then it might be a sever accident
            # If the speed before the evnt was between 2.5 m/s and 5.0 m/s, then it proba
            if (mean_spead_before >= 5.0) and (mean_speed_after <= thrshld_speed):
                # get the indexes of the time window in which to consider phone movements
                # (from 2 seconds after the incident to window_after_evet seconds after th
                delta_time_start_phone = df_phone.loc[time_of_event_phone:].index[20]
                delta_time_end_phone = min(df_phone.loc[time_of_event_phone:].index[int(wi

                # if the phone doesn't move in that time window, then it might be severe,
                if df_phone.loc[delta_time_start_phone:delta_time_end_phone]['var linear a
                    return time_of_event_gps, time_of_event_phone, 'severe'
                else:
                    return time_of_event_gps, time_of_event_phone, 'mild'
            elif (mean_spead_before >= 2.5) and (mean_speed_after <= thrshld_speed):
                return time_of_event_gps, time_of_event_phone, 'mild'

            else:
                return None, None, None
        else:
            return None, None, None
```

2.3.2 Run this algorithm for each file and detect head-on-collisions from a moving vehicle:

- The code below runs the algorithm for detecting head-on vehicle-to-vehicle collisions.
- If a collision is detected, it indicates in the top-most title figure whether it was mild or severe and saves the figure to the disk
- It also shows on the graphs where the accident occured with a bright green dot
- If no accident was detected, this also is stated in the top-most title of the figure

NOTES:

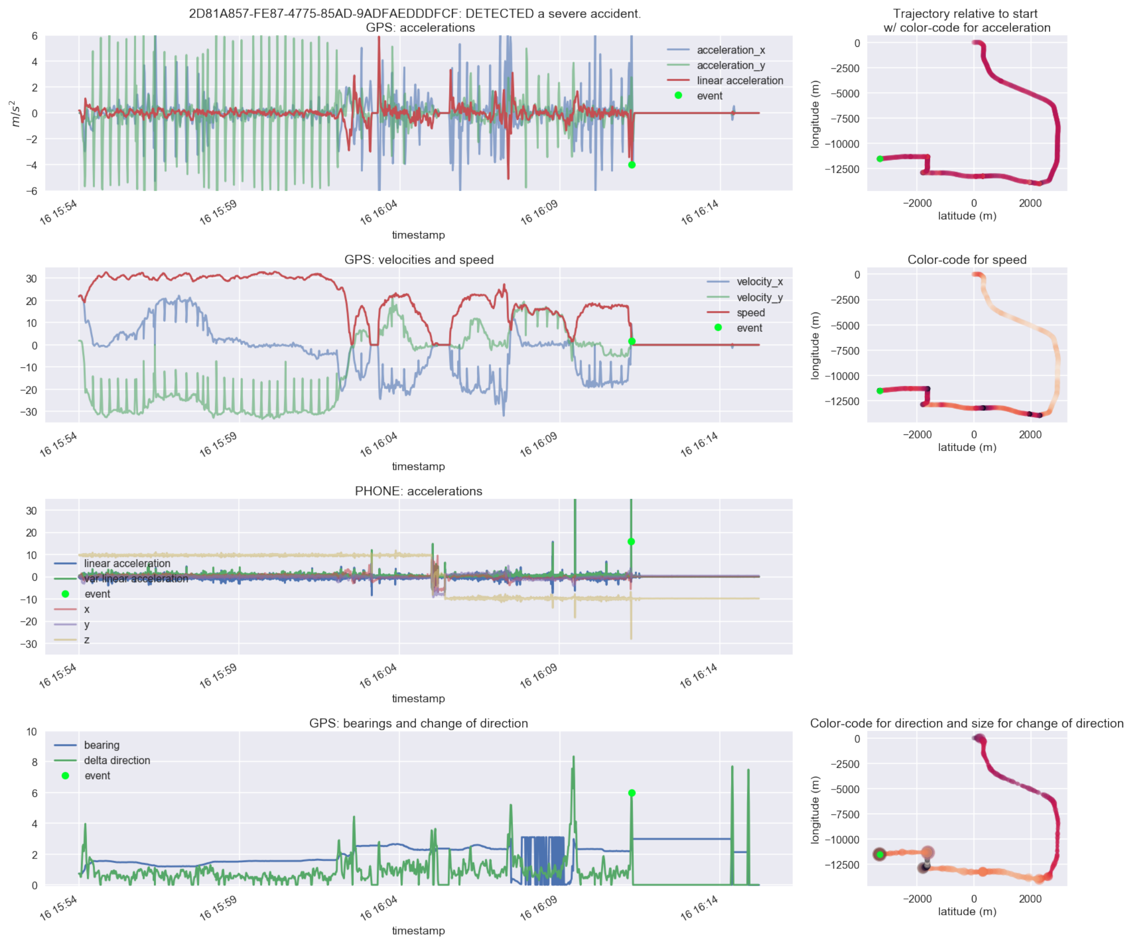
- To try the algorithm on other fies, simply place the files in the current directory and run the

script below
- If you do not want all images to be shown, and just save the ones with detected accidents, then set the flag SHOW_ALL_FIGURES to False

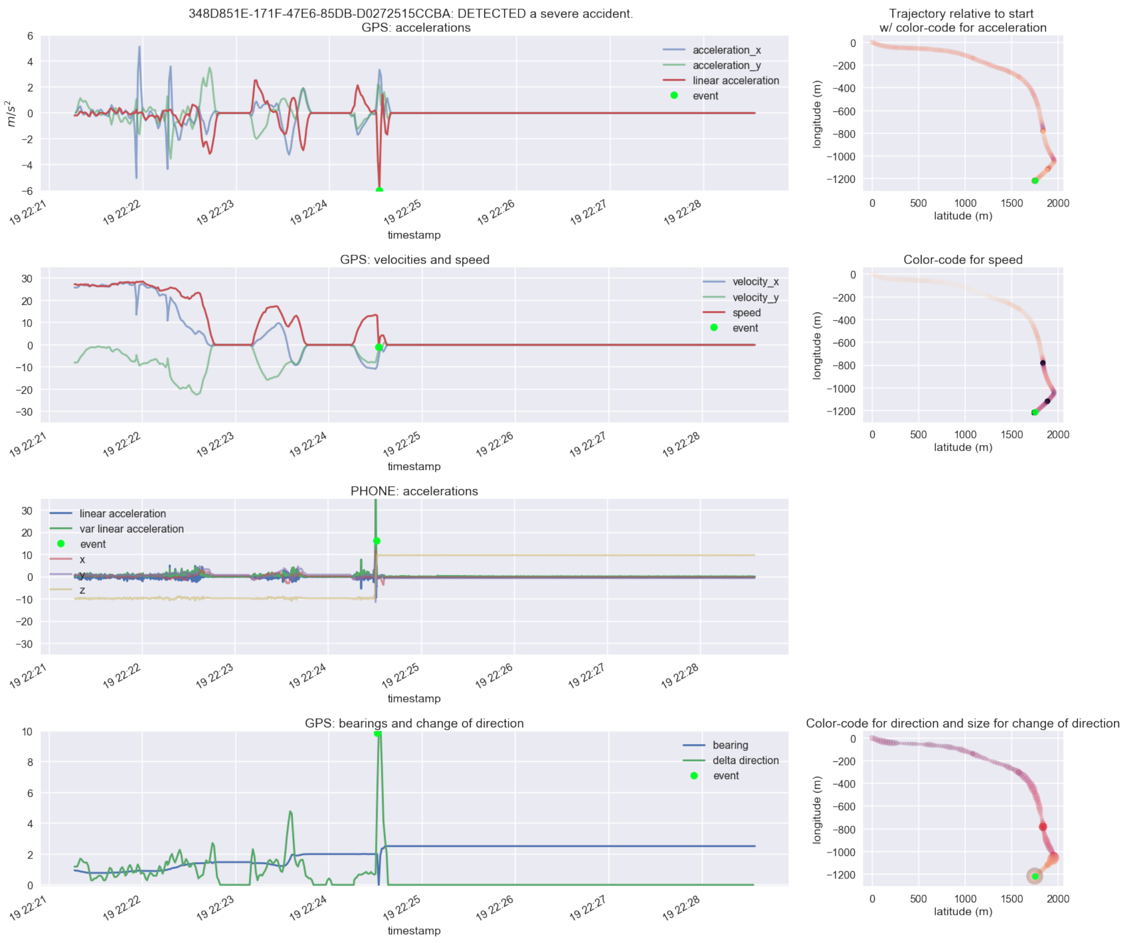
```
In [19]: SHOW_ALL_FIGURES = False
         SHOW_COLLISIONS_ONLY = True

         # read the files in the directory
         files = [f for f in os.listdir(DATA_PATH) if f.endswith(".csv")]

         for idx, file_ in enumerate(files[:]):
             filePathName = os.path.join(DATA_PATH, file_)
             print('File:', file_.split('.')[0])
             dfData = load_data(filePathName)
             dfGPS, dfAccelerometer = split_gps_phone(dfData)
             idxGPS, idxPhone, severity = vehicle_on_vehicle_collision_detector(
                 dfGPS,
                 dfAccelerometer,
                 thrshld_gps_acc = -3.5,
                 thrshld_phone_acc = 10.0,
                 window_after_evet = 20,
                 thrshld_speed=1.0,
                 thrshld_var_acc=0.5)
             if idxGPS is not None:
                 print('  A '+ severity +' head-on vehicle-on-vehicle accident detected')
             else:
                 print('  No head-on vehicle-on-vehicle accident detected')

             scl = 10

             plt.close('all')
             plt.figure(figsize=(24,20))

             # ROW 1, accelerations, GPS
             ax = plt.subplot2grid((4,4),(0,0), colspan=3)
             dfGPS['acceleration_x'].rolling(3, win_type='triang', center =True).mean().plot(a
             dfGPS['acceleration_y'].rolling(3, win_type='triang', center =True).mean().plot(a
             dfGPS['linear acceleration'].plot(ax=ax)
             if idxGPS is not None:
                 ax.plot(idxGPS, dfGPS.loc[idxGPS]['linear acceleration'], 'o', c='#00FF29', l
             #dfGPS['derived acc'].plot(ax=ax)
             plt.ylim([-6, 6])
             plt.ylabel('$m/s^2$')
             plt.legend()
             if idxGPS is not None:
                 plt.title(file_.split('.')[0] +': DETECTED a ' + severity + ' accident.  \n G
             else:
                 plt.title(file_.split('.')[0] +': NO accident.  \n GPS: accelerations')
             plt.subplot2grid((4,4),(0,3), colspan=1)
             plt.scatter(dfGPS['lat utm'].values, dfGPS['lon utm'].values, c=dfGPS['linear acce
             if idxGPS is not None:
                 plt.scatter(dfGPS.loc[idxGPS]['lat utm'], dfGPS.loc[idxGPS]['lon utm'], c='#00
             plt.title('Trajectory relative to start \nw/ color-code for acceleration')
             plt.xlabel('latitude (m)')
             plt.ylabel('longitude (m)')


             # ROW 2, speed. GPS
             ax = plt.subplot2grid((4,4),(1,0), colspan=3)
             dfGPS['velocity_x'].plot(ax=ax, alpha = alpha_bg)
             dfGPS['velocity_y'].plot(ax=ax, alpha = alpha_bg)
             dfGPS['speed'].plot(ax=ax)
             if idxGPS is not None:
                 ax.plot(idxGPS, dfGPS.loc[idxGPS]['speed'], 'o', c='#00FF29', label='event')
             #dfGPS['derived speed'].plot(ax=ax)
             plt.ylim([-35, 35])
             plt.legend()
             plt.title('GPS: velocities and speed')
             plt.subplot2grid((4,4),(1,3), colspan=1)
             plt.scatter(dfGPS['lat utm'].values, dfGPS['lon utm'].values, c=dfGPS['speed'].va
             if idxGPS is not None:
                 plt.scatter(dfGPS.loc[idxGPS]['lat utm'], dfGPS.loc[idxGPS]['lon utm'], c='#00
             plt.title('Color-code for speed')
             plt.xlabel('latitude (m)')
             plt.ylabel('longitude (m)')

             # ROW 3, accelerations, ACCELEROMETER
             ax = plt.subplot2grid((4,4),(2,0), colspan=3)
             dfAccelerometer['linear acceleration'].plot(ax=ax)
             dfAccelerometer['var linear acceleration'].plot(ax=ax)
             if idxGPS is not None:
                 ax.plot(idxPhone, dfAccelerometer.loc[idxPhone]['var linear acceleration'], '
             (dfAccelerometer['x']).rolling(7, win_type='triang', center =True).mean().plot(ax
             (dfAccelerometer['y']).rolling(7, win_type='triang', center =True).mean().plot(ax
             (dfAccelerometer['z']).rolling(7, win_type='triang', center =True).mean().plot(ax
             plt.ylim([-35, 35])
             plt.legend()
             plt.title('PHONE: accelerations')

             # ROW 4, position, GPS
             ax = plt.subplot2grid((4,4),(3,0), colspan=3)
             #(.001 * dfGPS['lat utm']).plot(ax=ax)
             #(.001 * dfGPS['lon utm']).plot(ax=ax)
             (np.pi * dfGPS['bearing']/365).plot(ax=ax)
             temp = dfGPS['delta direction'].abs().pow(pow_delta_dir)
             temp.plot(ax=ax, label = 'delta direction')
```

```
            if idxGPS is not None:
                ax.plot(idxPhone, np.power(np.abs(dfGPS.loc[idxGPS]['delta direction']), pow_c
            plt.ylim([-0.1, 10])
            plt.legend()
            plt.title('GPS: bearings and change of direction')
            plt.subplot2grid((4,4),(3,3), colspan=1)
            plt.scatter(
                dfGPS['lat utm'].values,
                dfGPS['lon utm'].values,
                c =dfGPS['bearing'].values,
                s = 60 * temp,
                alpha=0.2)
            if idxGPS is not None:
                plt.scatter(dfGPS.loc[idxGPS]['lat utm'], dfGPS.loc[idxGPS]['lon utm'], c='#00
            plt.title('Color-code for direction and size for change of direction')
            plt.xlabel('latitude (m)')
            plt.ylabel('longitude (m)')

            plt.tight_layout()
            if idxGPS is not None:
                plt.savefig(os.path.join(RESULTS_PATH,file_.split('.')[0]+'.png'))

            if SHOW_COLLISIONS_ONLY and idxGPS is not None:
                plt.show()

            if SHOW_ALL_FIGURES:
                plt.show()
```

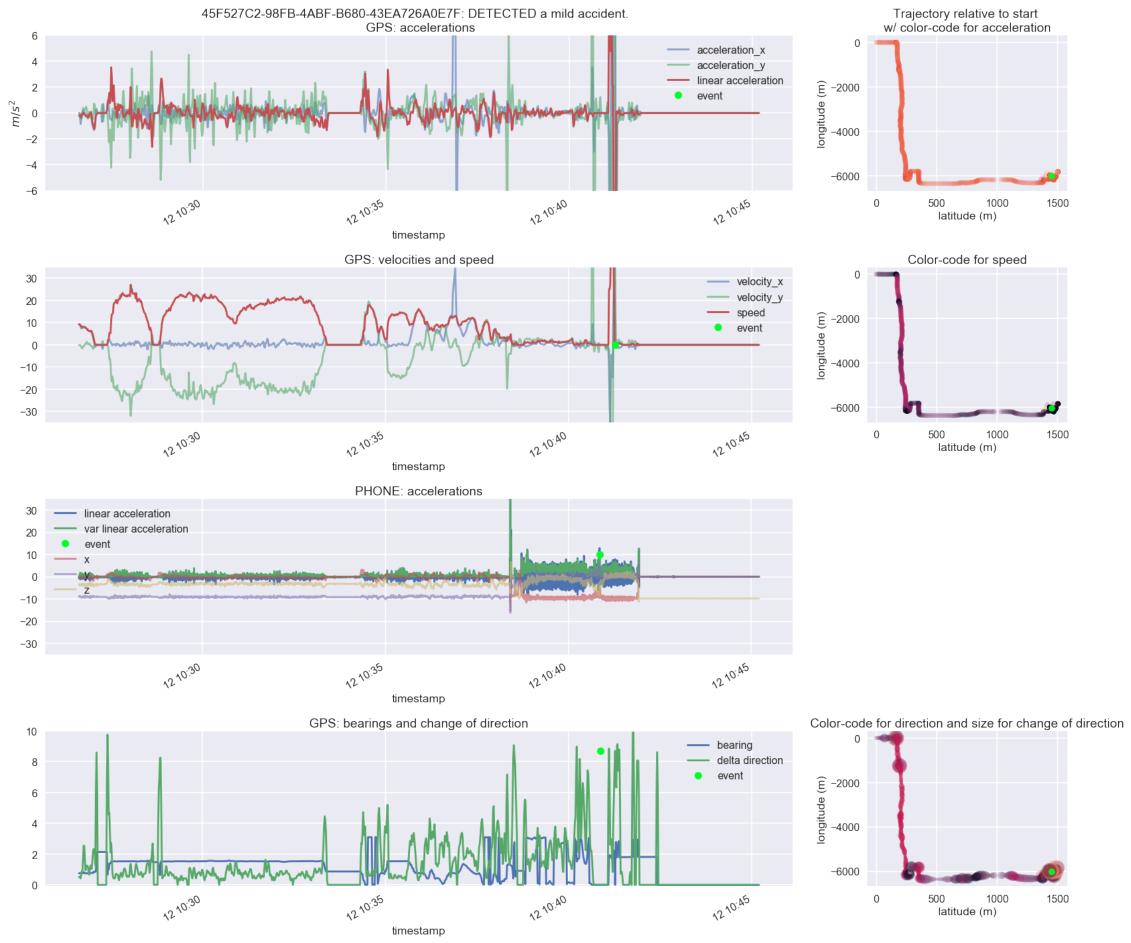
```
File: 00DAC437-FF8B-4DA3-9E24-4EE1B1AA12EC
  No head-on vehicle-on-vehicle accident detected
File: 05A03532-5BE9-4749-9460-CCCEADA6C786
  No head-on vehicle-on-vehicle accident detected
File: 18E4E1E7-D48D-4D39-B92B-ACB831E2F530
  No head-on vehicle-on-vehicle accident detected
File: 1A9494FB-DF91-45FD-B8DD-43B64A8A9ED9
  No head-on vehicle-on-vehicle accident detected
File: 268A6026-853A-4E84-9D12-4DBDE18EBE41
  No head-on vehicle-on-vehicle accident detected
File: 2D81A857-FE87-4775-85AD-9ADFAEDDDFCF
  A severe head-on vehicle-on-vehicle accident detected
```



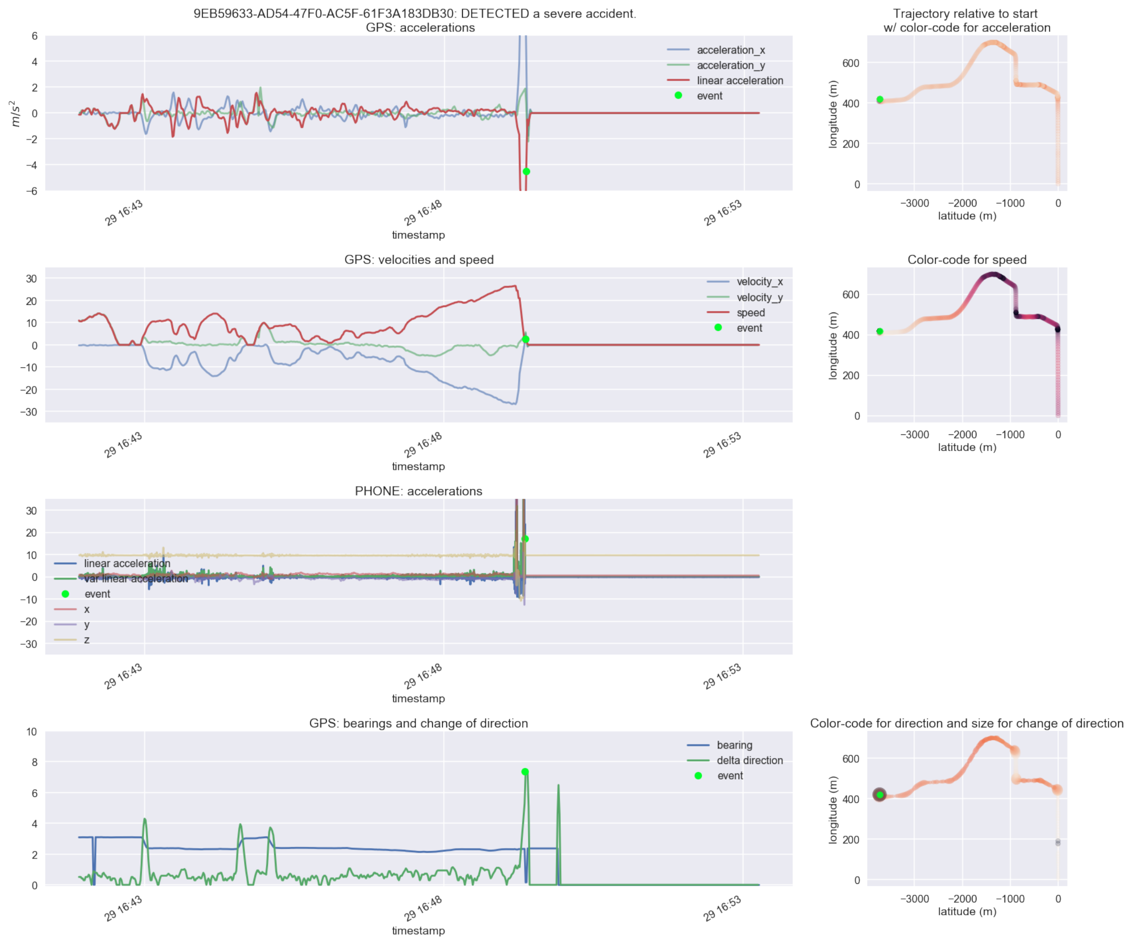
```
File: 348D851E-171F-47E6-85DB-D0272515CCBA
  A severe head-on vehicle-on-vehicle accident detected
```



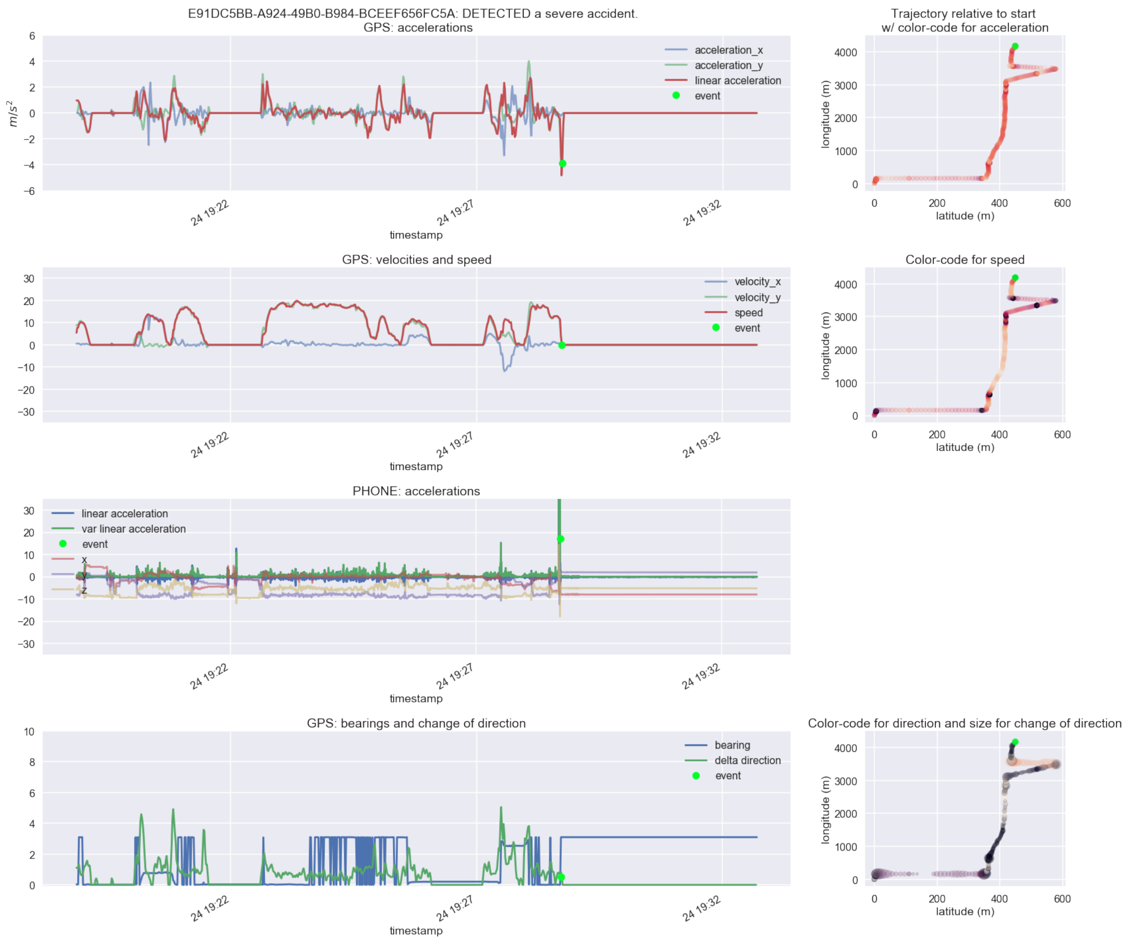
```
File: 4475987E-0048-4D49-84A9-70E439C66B5B
  No head-on vehicle-on-vehicle accident detected
File: 45F527C2-98FB-4ABF-B680-43EA726A0E7F
  A mild head-on vehicle-on-vehicle accident detected
```



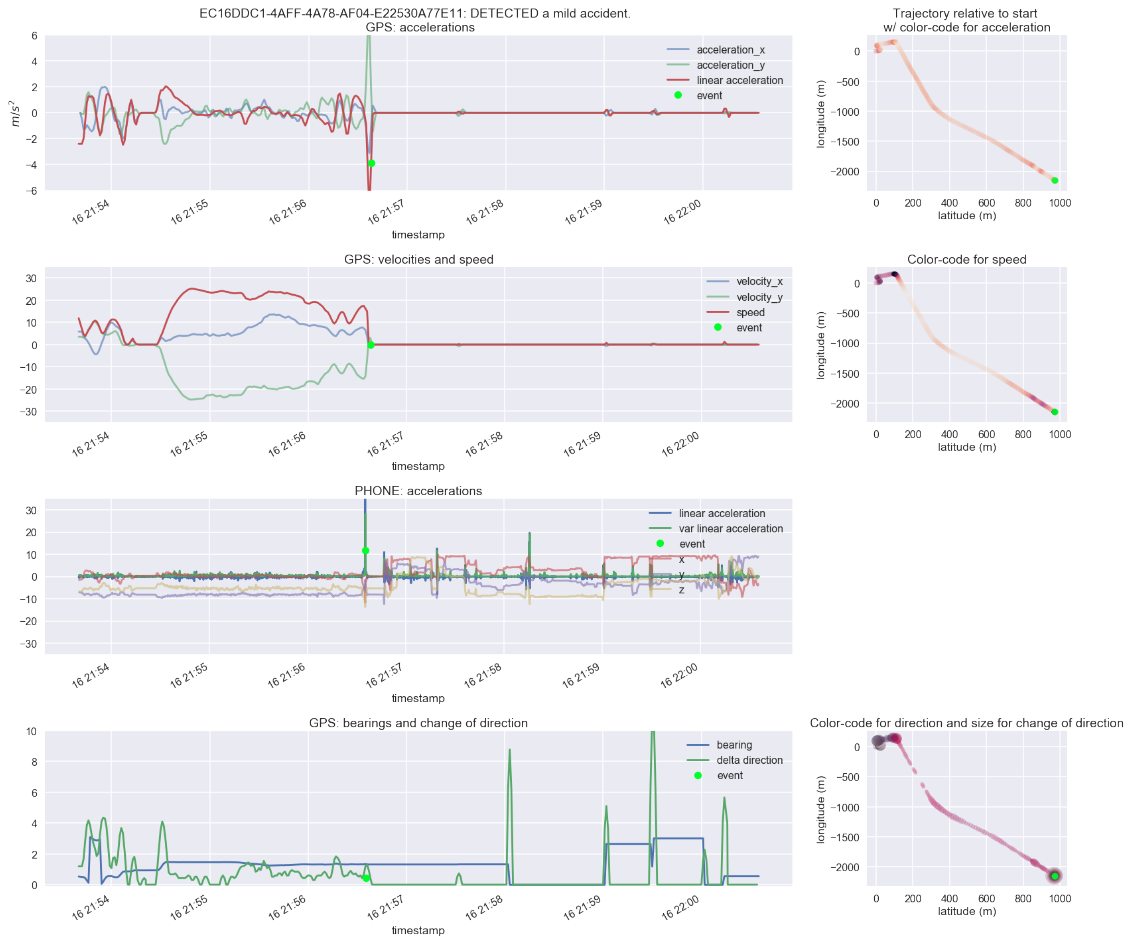
```
File: 4D735D9B-15C6-48F2-A78E-BDFCBC523115
  No head-on vehicle-on-vehicle accident detected
File: 5F209F01-4235-47BE-ADEB-2A7877C95271
  No head-on vehicle-on-vehicle accident detected
File: 636C5BAE-D065-4B45-B985-8830B5D8973A
  No head-on vehicle-on-vehicle accident detected
File: 68E6E44C-2A24-42B8-9314-C6A9FFC5EC8D
  No head-on vehicle-on-vehicle accident detected
File: 78C58FEE-A616-436F-8709-6D755FACF525
  No head-on vehicle-on-vehicle accident detected
File: 7D4D575C-26BD-4CD3-90FD-D97FBD433947
  No head-on vehicle-on-vehicle accident detected
File: 844709EE-F9AA-4B66-B682-860339FCBC1C
  No head-on vehicle-on-vehicle accident detected
File: 9EB59633-AD54-47F0-AC5F-61F3A183DB30
  A severe head-on vehicle-on-vehicle accident detected
```



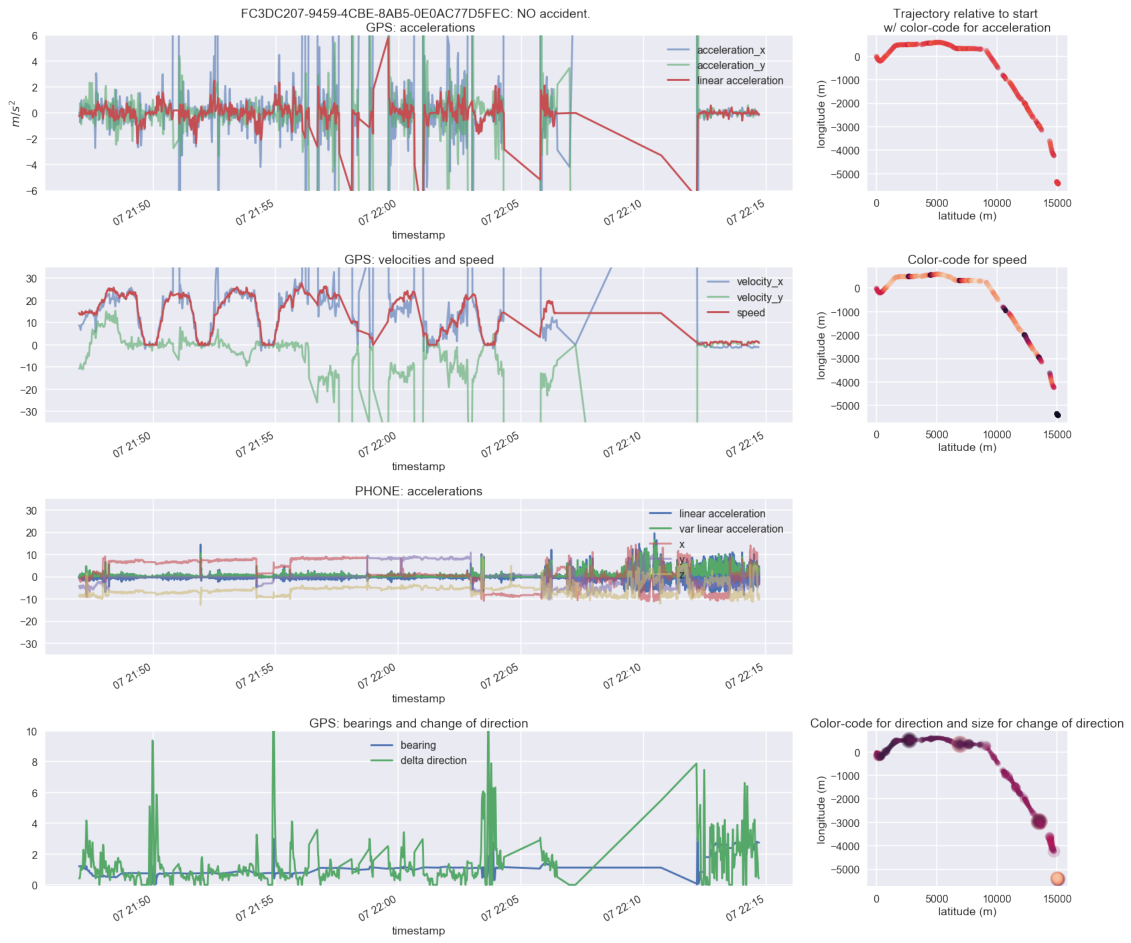
```
File: b0b25b30-c3b3-48a4-9e20-810363501c64
  No head-on vehicle-on-vehicle accident detected
File: B194DCB3-8906-47E7-963D-32985B5ABD51
  No head-on vehicle-on-vehicle accident detected
File: BBDE8492-57F2-4697-87B7-BEB6100EBAE1
  No head-on vehicle-on-vehicle accident detected
File: E66D6050-11EA-4816-A096-0B0E88234BFD
  No head-on vehicle-on-vehicle accident detected
File: E91DC5BB-A924-49B0-B984-BCEEF656FC5A
  A severe head-on vehicle-on-vehicle accident detected
```



```
File: EC16DDC1-4AFF-4A78-AF04-E22530A77E11
  A mild head-on vehicle-on-vehicle accident detected
```



```
File: EF0AFFAA-FC7C-4EF9-B2C4-BD437CD78E01
  No head-on vehicle-on-vehicle accident detected
File: FC3DC207-9459-4CBE-8AB5-0E0AC77D5FEC
  No head-on vehicle-on-vehicle accident detected
```



2.3.3 Detecting vehicle-on-vehicle head-on collision with sensors in the parked vehicle:

- Look for drastic changes in GPS and phone acceleration values:
- linear acceleration in the vehicle-mounted GPS device should be above some threshold value
- the phone should register equally sudden changes in acceleration, but it could be in any direction
- The speed of the vehicle right before the incident should be zero and then detect slight increase due to it being pushed a few meters

Severe incedents:

- After the incident, if the driver is severely injured, probably he/she will not be able to move for a while. This indicates that ther should be not much changes in phone acceleration data. This can be detected by observig the variations in the linear accelerations in the phone acceleration readings.

Less-sever incident:

- If the accident is mild the driver will probably jump out of the vehicle and get his/her phone with himself/herself. This should be registered as variations in phone acceleration data.

An algorithm: No algorithm is provided for this event type.

2.3.4 Detecting vehicle-on-vehicle collision from the side, with sensors in the parked vehicle:

- Look for drastic changes in GPS and phone acceloremeter acceleration values:
- linear acceleration in the vehicle-mounted GPS device should be below some threshold value
- the phone should register equally sudden changes in acceleration, but it could be in any direction
- The speed of the vehicle right before the incident should be zero and then detect slight increase due to it being pushed a few meters
- There should be sudden change in the bearing data.

Severe incedents:

- After the incident, if the driver is severely injured, probably he/she will not be able to move for a while. This indicates that ther should be not much changes in phone acceleration data. This can be detected by observig the variations in the linear accelerations in the phone acceleration readings.

Less-sever incident:

- If the accident is mild the driver will probably jump out of the vehicle and get his/her phone with himself/herself. This should be registered as variations in phone acceleration data.

An algorithm: No algorithm is provided for this event type.

2.3.5 Detecting vehicle-on-pedestrian or vehicle-on-cyclist collision:

- In these situations there is an initial drastic deceleration, followed by further attempts by the driver to slow down the vehicle (source https://iopscience.iop.org/article/10.1088/1757-899X/252/1/012007/pdf).

2.4 HOW TO DO THIS PROPERLY FOR PRODUCTION:

2.4.1 Collect more data about:

1. Individual drivers and build a driver profile
2. Individual routes or sections of roads and build route profiles
3. Vehicle types and build a profie for each type
4. Accident types and try to identify unique signatures

2.4.2 Improve the data

Model the data generating and collecting processes and improve the accuracy of the data by removing possible drifts, noise sources, and missing data. Kalman filters might be particularly useful in this task.

2.4.3 Come up with test statistics

Using the large collection of data and taking into account the various profiles, try to come up with statistics that could be used to detect various events (along with confidence intervals) and use these to build rules for identifyning different accident types and their severity

2.4.4 Or try Machine Learning

Try building ML algorithms that can be trained on the data corpus. Probably one can have several classes for different accident types (and no accidents) and use recurrent networks to train classifiers that can extract temporal features per driver type, road type, vehicle type, day of the week, hour of the day etc and learn to map them to one of the accident types (or no accident)

2.4.5 Optimise the code for speed

Do not use Pandas. Use Numpy arrays and digital filters to detect peaks etc.