PREDICTIVE ANALYTICS FOR VEHICLE COLLISIONS THROUGH THE USE OF NEURAL NETWORKS

Anthony Asilo

OVERVIEW

- Vehicles, whether speaking of cars, trucks, airplanes, drones, or even ships, serve a single purpose: to provide a means of transportation from one point to another.
- Humans are usually the ones driving these vehicles, and humans are prone to error, so this provides a solution to combat vehicle collisions.

GOAL

- Combat collisions by using a neural network to potentially detect and predict collisions before they occur, to assist in object avoidance.

CONTENTS IMPLEMENTED

- Python script to generate data set based on a spherical model of earth.
- Python script to process data, train a neural network and predict whether a vehicle will collide into another vehicle
- README.md for Instructions (will also be mentioned in the latter part of this presentation)

SOME NEEDED INFORMATION

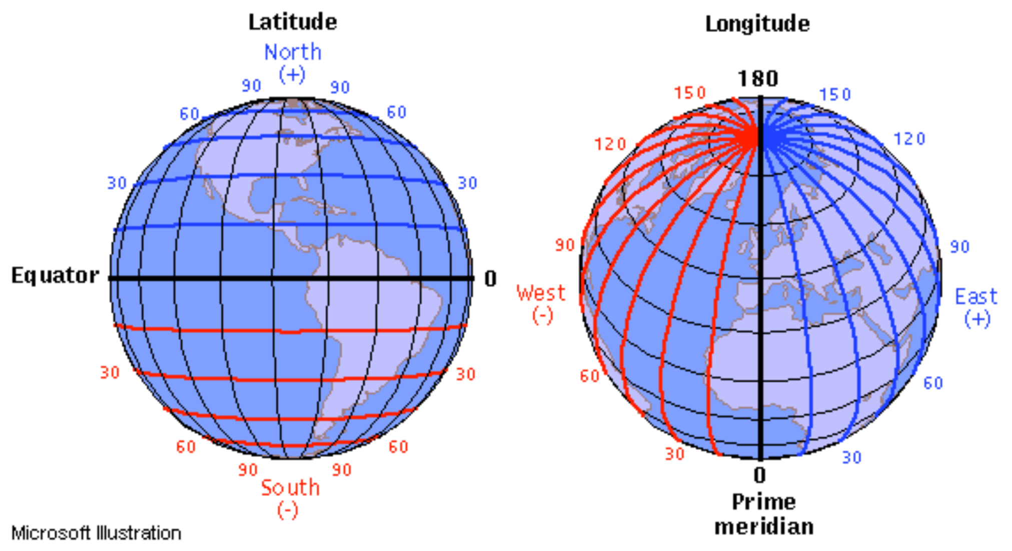
This section will provides context for what our data will represent

EARTH AND ITS COORDINATE SYSTEM

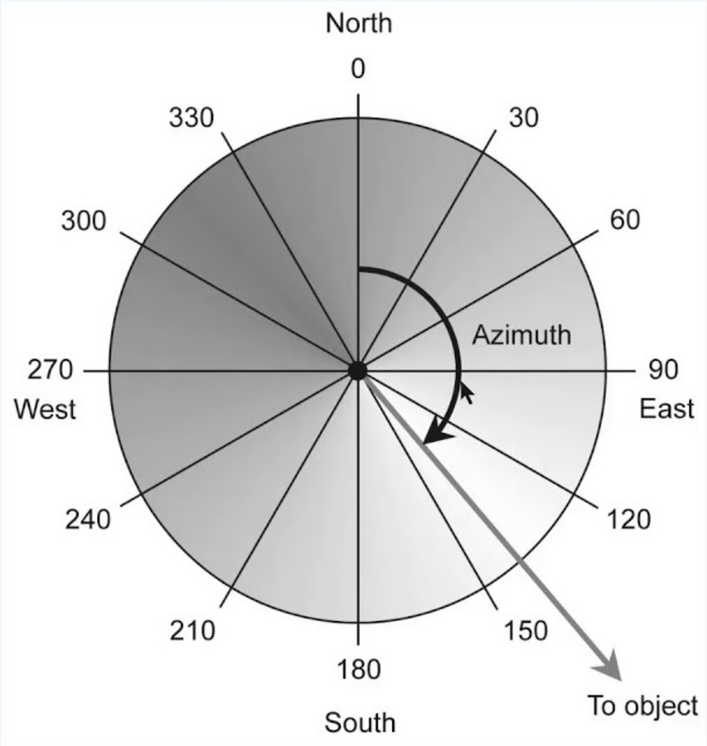
- Every coordinate on earth is measured as a point, with latitude and longitude.
- For our purposes, we will use Earth as a spherical model where the calculations give about a 0.3% error, instead of as an ellipsoid.

EARTH AND ITS COORDINATE SYSTEM (CONT.)

- Latitudinal lines show how much north or south of the equator a place is located, and the equator is our starting point for measuring latitude, which is 0 degrees latitude. Latitude increases to 90 degrees as you move towards the north or south pole.
- Longitudinal lines show how east or west of the Prime Meridian a place is located, which is a universal line that is marked as 0 degrees longitude. Longitude increases by 180 degrees as you move east or west

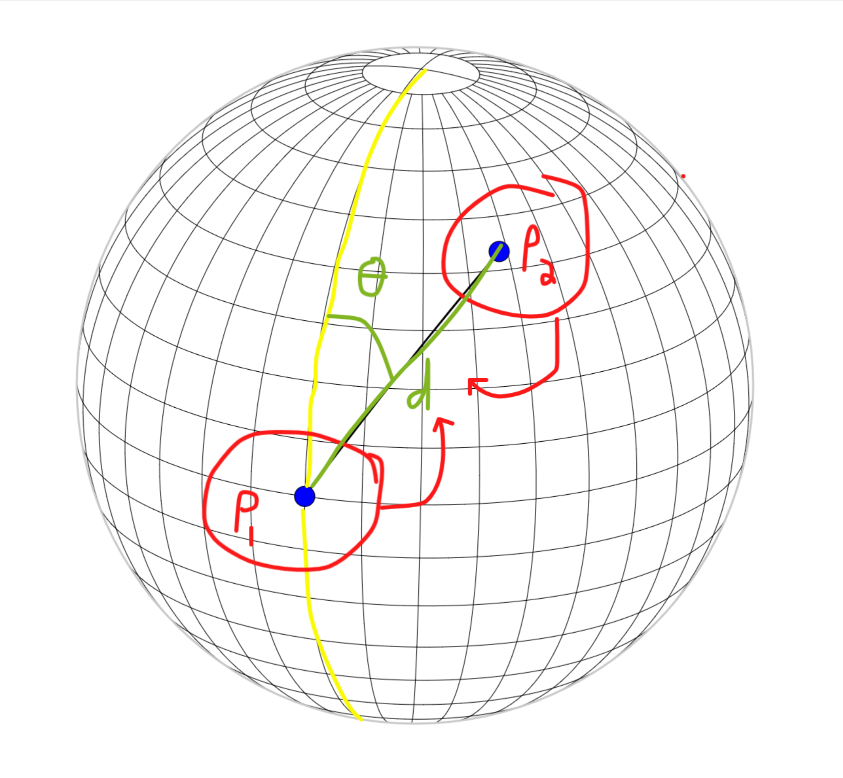


EARTH AND ITS COORDINATE SYSTEM (CONT.)



- An Azimuth is a measurement of direction, and we will use azimuths to describe the direction of vehicles when they are driving to keep consistency. An Azimuth points 0° North and increases to 360° clockwise.

EARTH AND ITS COORDINATE SYSTEM (CONT.)



Inverse Coordinates – given two points on a sphere, a distance between the points and the relative azimuth is returned

Haversine formula:

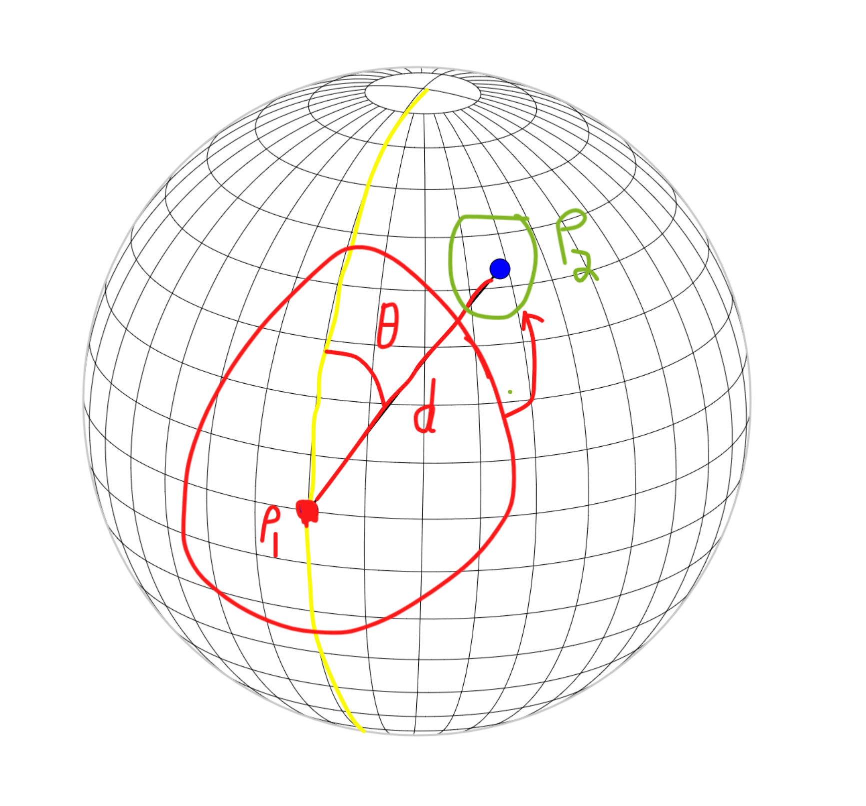
$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \mathrm{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!*

EARTH AND ITS COORDINATE SYSTEM (CONT.)



Terminal Coordinates – given a point on a sphere, a distance between the points, and the relative azimuth, a point made up of the latitude and longitude of the second coordinate is returned

Formula: $\varphi_2 = \text{asin}(\sin\varphi_1 \cdot \cos\delta + \cos\varphi_1 \cdot \sin\delta \cdot \cos\theta)$

$\lambda_2 = \lambda_1 + \text{atan2}(\sin\theta \cdot \sin\delta \cdot \cos\varphi_1, \cos\delta - \sin\varphi_1 \cdot \sin\varphi_2)$

where φ *is latitude,* λ *is longitude,* θ *is the bearing (clockwise from north),* δ *is the angular distance d/R; d being the distance travelled, R the earth's radius*

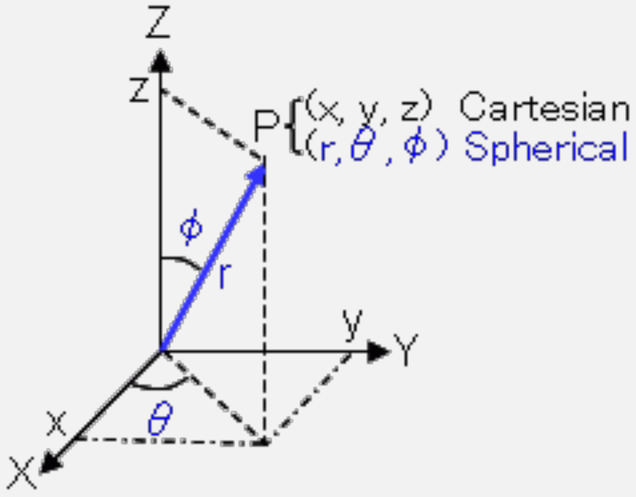
CARTESIAN AND SPHERICAL COORDINATES

- Cartesian coordinates represent a 3 dimensional coordinate system with x, y, z

- To convert Cartesian to Spherical, use:

$$\theta \quad = arctan2(y, x)$$

$$\phi \quad = arctan2(z, \sqrt{x^2 + y^2})$$



- Spherical coordinates represent a 3 dimensional coordinate system with angles between the x, y and z.

- To convert Spherical to Cartesian, use:

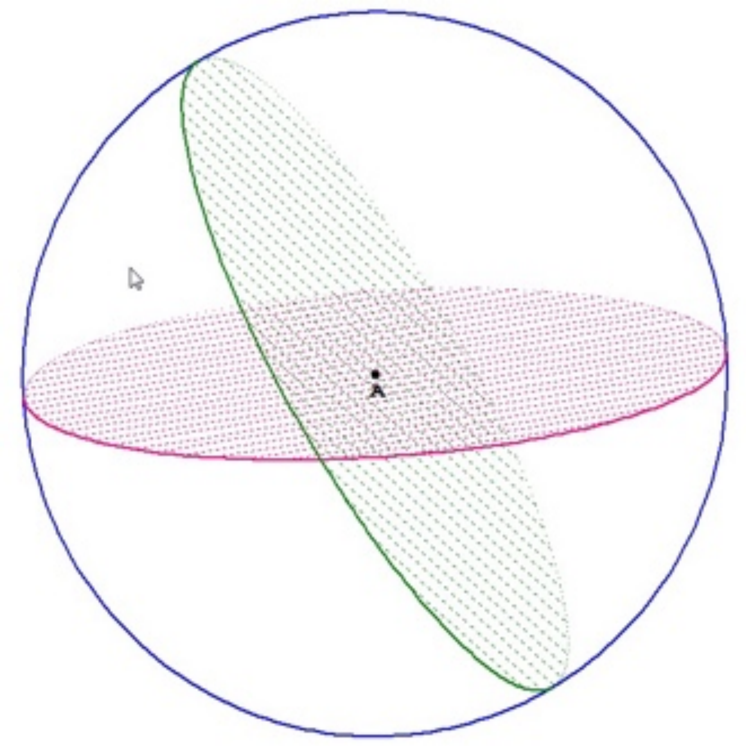
$$x \quad = cos(\theta)cos(\phi)$$

$$y \quad = sin(\theta)cos(\phi)$$

$$z \quad = sin(\phi)$$

GREAT CIRCLE INTERSECTIONS (GCI)

- A GCI is where two circles intersect on a sphere
- The function takes 8 parameters: a start and an end latitude/longitude for the Host vehicle, and a start and end latitude/longitude for the Guest vehicle.
- Each coordinate is converted to a cartesian point, and then each vehicle's arc paths has their start and end coordinate multiplied as a cross product.
- Because two circles intersect on the sphere, there are two possible intersections.



GREAT CIRCLE INTERSECTIONS (CONT.)

- The formula is as follows:

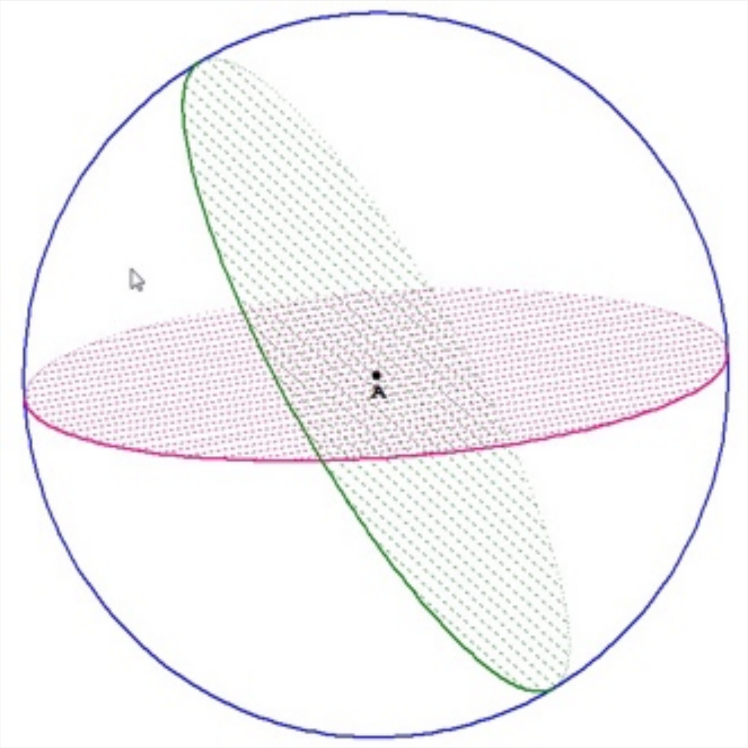
$$h = \frac{dc - fa}{ea - db}, g = \frac{-bh - c}{a}, k = \sqrt{\frac{r^2}{g^2 + h^2 + 1}}, where$$

$$< a, b, c > is\ Host_{start} \times Host_{End},$$

$$< d, e, f > is\ Guest_{start} \times Guest_{End},$$
$$and\ r = Radius\ of\ Earth \approx 6371.137m$$

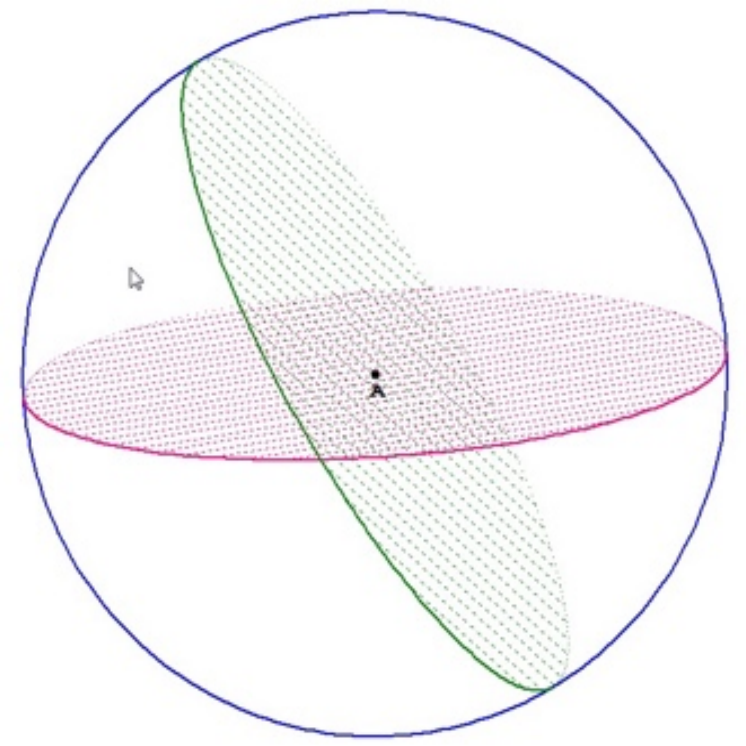
The two great circles intersect at the following points:

$$(gk, hk, k), (-gk, -hk, -k)$$



GREAT CIRCLE INTERSECTIONS (CONT.)

- Now that we have both points as Cartesian points, we convert them back into Latitude and Longitude.
- It needs to be determined next which point is the correct POI, better known as a point of intersection.
- The POI needed is found in our last method, checkIfLies(), which takes two points on the sphere and determines if these intersection points lie within arc segments by doing an angle test, by normalizing the dot product.
- $\theta = \arccos\left(\frac{a \cdot b}{||a||||b||}\right)$

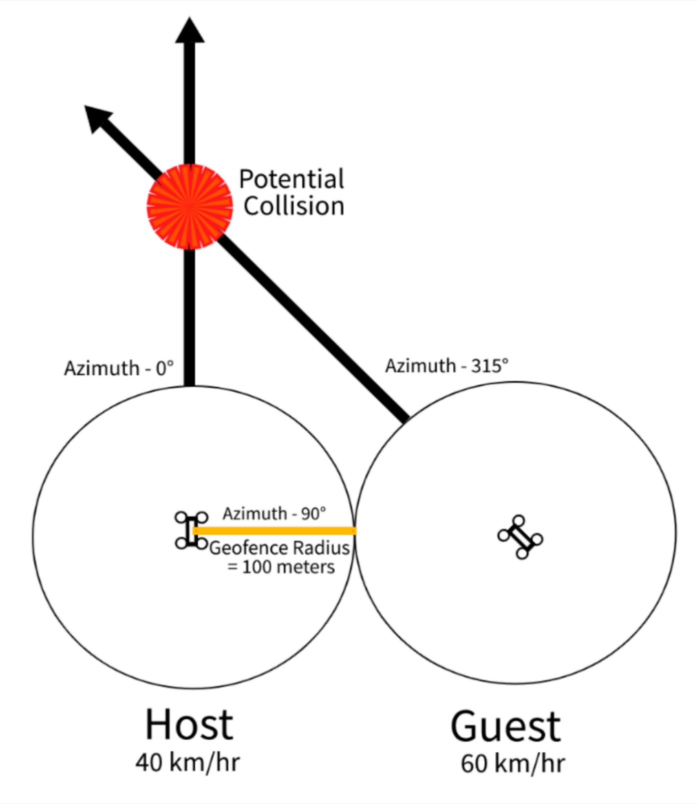


IOT (INTERNET OF THINGS)

- In a perfect world, all vehicles, whether they be road, sea, or air, could essentially have an IoT device that utilizes its geographical coordinates for location-based services.
- A Location-Based-Service (LBS) is a service that considers one's geographic location to provide a user with accurate and factual information.
- Data would be streamed and sent in real-time when a vehicle is detected in another vehicles geofence, which can be thought of a perimeter that defines the area or bounds of a location based service.

VEHICLES

- The vehicles implemented will each have their own unique hash code to ensure individuality.
- Vehicles Have Latitude, Longitude, and Velocity
- Velocity is Speed and direction (Azimuth)
- All geofences defined for the simulation are 100 meters radius



DATA GENERATION

STEP ONE : IMPORTS

```
from math import acos,sin,cos,radians,degrees,atan2,asin,acos,pi,sqrt,isclose
import random
import numpy as np
import pandas as pd
import csv
```

STEP TWO : DEFINE GLOBAL VARS

```
#Global variables

radar_radius = .1    # radius of radar detection is 100 meters (.1 km) #.85 also works nice!
radar_update = .5     # time delay between each location update for vehicle
r = 6371.137          # radius of earth at equator in kilometers
azimuth = 0           # global var for azimuth
data = pd.DataFrame()

#init columns to append data to to add to dataframe and then export as csv
host_latitude = []
host_longitude = []
host_speed = []
host_azimuth = []
host_distance = []
host_time = []

host_azi_to_guest = []

guest_latitude = []
guest_longitude = []
guest_speed = []
guest_azimuth = []
guest_distance = []
guest_time = []

intersection_latitude = []
intersection_longitude = []
collision = []
```

STEP 3 : DEFINE VEHICLE CLASS

```python
# AZIMUTH RULE --> NORTH = 0°, SOUTH = 180°, EAST = 90°, WEST = 270°
class Vehicle:
    #vehicle contstructor initializes based on name, hashid, x and y coords, and velocity (speed and direction)
    #initialize with speed 0, and motion false. Starting vehicles will allow them to move
    def __init__(self, name, ID, x, y, s, d):
        self.id = ID
        self.name = name
        self.coords = [x, y]
        self.velocity = [s, d]
        self.motion = True
        self.reward = None
    #function prints a description of the vehicle
    def get_velocity(self):
        print("getter method called")
        return self.velocity[0]

     # function to set value of _age
    def set_velocity(self, a):
        print("setter method called")
        self.velocity[0] = a

    # function to delete _age attribute
    def del_velocity(self):
        del self.velocity[0]

    speed = property(get_velocity, set_velocity, del_velocity)

    def get_direction(self):
        print("getter method called")
        return self.velocity[1]

     # function to set value of _age
    def set_direction(self, a):
        print("setter method called")
        self.velocity[1] = a

    # function to delete _age attribute
    def del_direction(self):
        del self.velocity[1]

    direction = property(get_direction, set_direction, del_direction)

    def describe(self):
        print("Vehicle [%s] at (%f , %f) moving %s° at %d KPH \n" % (self.id, self.coords[0], self.coords[1], self.velocity[1], self.velocity[0]))
```

STEP 5A : INVERSE COORD FUNCTION

```python
#with two coordinated, fine the distance between them on the sphere and the azimuth (angle of first coord relative to north!)
def inverseCoords(lat1, lon1, lat2, lon2):

    # The math module contains a function named
    # radians which converts from degrees to radians.
    lon1 = radians(lon1)
    lon2 = radians(lon2)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    global azimuth
    azimuth = degrees(atan2(sin(dlon) * cos(lat2),cos(lat1) * sin(lat2) - sin(lat1) * cos(lat2) * cos(dlon)))
    if(azimuth < 0):
        azimuth += 360
    elif(azimuth > 360):
        azimuth -= 360
    c = 2 * asin(sqrt(a))

    # Radius of earth in kilometers. Use 3956 for miles
    #r = 6371
    # calculate the result
    #print(str(azimuth) + "°")
    return (c * r , azimuth)
```

STEP 5B : TERMINAL COORD FUNCTION

```python
# Given a start coordinate with a direction and distance, find the end coordinate!
def terminalCoords(lat1, lon1, azimuth, dis):
    #r = 6371 #Radius of the Earth
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    azimuth = radians(azimuth)

    lat2 = asin( sin(lat1) * cos(dis/r) + cos(lat1) * sin(dis/r) * cos(azimuth))
    lon2 = lon1 + atan2( sin(azimuth) * sin(dis/r) * cos(lat1), cos(dis/r) - sin(lat1) * sin(lat2))

    lat2 = degrees(lat2)
    if(lat2 < -180):
        lat2 += 360
    elif(lat2 > 180):
        lat2-= 360

    lon2 = degrees(lon2)
    if(lon2 < -180):
        lon2 += 360
    elif(lon2 > 180):
        lon2-= 360

    tCoord = ( lat2, lon2 )

    return tCoord
```

STEP 5C : CONVERSION FUNCTION

```python
# Takes a Latitude and longitude and converts into a cartesian x y z
def cartesian(lat,lon):
    return [cos(lat)*cos(lon), sin(lat)*cos(lon), sin(lon)]

# Takes Cartesiian x y z in radians and returns a latlon in degrees
def LatLon(point):
    return (degrees(atan2(point[1],point[0])) , degrees(atan2(point[2], sqrt(point[0]**2 + point[1]**2))) )
```

STEP 5D : GCI FUNCTION

```python
# Takes two arcs of great circles and finds the point of intersection
# Say that 1 and 2 are the start and end points for the first arc
# and that 3 and 4 and the start and end points for the second arc
def GCI(lat1, lon1, lat2, lon2, lat3, lon3, lat4, lon4):

    # Convert all points into radians
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)
    lat3 = radians(lat3)
    lon3 = radians(lon3)
    lat4 = radians(lat4)
    lon4 = radians(lon4)

    # Convert Lat Lon points into Cartesian X Y Z points
    a0 = np.array(cartesian(lat1,lon1))
    a1 = np.array(cartesian(lat2,lon2))
    b0 = np.array(cartesian(lat3,lon3))
    b1 = np.array(cartesian(lat4,lon4))

    # Cross Product of arcs
    p = np.cross(a0,a1)
    q = np.cross(b0,b1)

    # Evaluating the bounds of the intersection
    h = (((q[0] * p[2] ) - (q[2] * p[0])) / ((q[1] * p[0]) - (q[0] * p[1])))
    g = ( ((-1) * (p[1] * h) - p[2] )/ p[0] )
    k = sqrt((r**2)/(g**2 + h**2 + 1))

    # Two possible intersections in the form of cartesian
    xo1 = [ g * k , h * k , k ]
    xo2 = [ -g * k , -h * k , -k ]
    #print(xo1)
    #print(xo2)

    # POI is Point Of Intersection
    POI = None

    # booleans checking which of the two points on sphere is the actual intersection
    bool_axo1 = isclose(degrees(checkIfLies(a0,xo1)) + degrees(checkIfLies(a1,xo1)), degrees(checkIfLies(a0,a1)), abs_tol=1e-8)
    bool_bxo1 = isclose(degrees(checkIfLies(b0,xo1)) + degrees(checkIfLies(b1,xo1)), degrees(checkIfLies(b0,b1)), abs_tol=1e-8)
    bool_axo2 = isclose(degrees(checkIfLies(a0,xo2)) + degrees(checkIfLies(a1,xo2)), degrees(checkIfLies(a0,a1)), abs_tol=1e-8)
    bool_bxo2 = isclose(degrees(checkIfLies(b0,xo2)) + degrees(checkIfLies(b1,xo2)), degrees(checkIfLies(b0,b1)), abs_tol=1e-8)

    # boolean logic check, sets POI to point if and only of both points are valid along arc
    # POI is the cartesian converted back to a coordinate (Lat Lon)
    if( bool_axo1 and bool_bxo1 ):
        POI = LatLon(xo1)
    elif( bool_axo2 and bool_bxo2 ):
        POI = LatLon(xo2)

    return(POI)
```

Formatter autopep

STEP 5E : FIND ACTUAL POINT OF INTERSECTION FROM GCI

```
# Whichever point if closer to the point
def checkIfLies(pI, pF):
    pI = np.array(pI)
    pF = np.array(pF)
    theta_pIpF = acos(np.dot(pI, pF)/(sqrt(pI[0]**2 + pI[1]**2 + pI[2]**2) * sqrt(pF[0]**2 + pF[1]**2 + pF[2]**2)))
    return theta_pIpF
```

STEP 5F : TRIGONOMETRIC FUNCTIONS

- These help assist with generating the Guest vehicles coordinates by making the guest vehicles azimuth point towards the Host vehicles azimuth to inflict a point of intersection

```
#Method to calculate relative angle of random point near host vehicle to possible collision
# inverts the current angle
def oppose(x):
    return x+180 if x < 180 else x-180

#Method to calculate relative angle of random point near host vehicle to possible collision
# gets median (half angle) of two angles
def half(x, y):

    if(abs(x-y) > 180):

        if(x >= y):
            print('x', x)
            x -= 360

        elif(y >= x):
            print('y', y)
            x += 360

    z = ((x + y) / 2)
    return z+360 if z < 0 else z
```

STEP 6 : MAIN METHOD

- First creates tests to ensure accuracy with each method

```python
# Main driver method that runs all code.
def main():
    #define Host Vehicle
    HOST = Vehicle("Subaru Imprezza", "19i28b", 33.779398, -84.413279, 30, 270)
    HOST.describe()

    #define Guest Vehicle
    GUEST = Vehicle("Honda Accord", "1bofq9", 33.779322, -84.413278, 50, 90)
    GUEST.describe()

    # INVERSE COORDS - Retrieves distance and azimuth given two points
    lat1 = 33.779398
    lat2 = 33.993333
    lon1 = -84.413279
    lon2 = -84.173888
    #r = 6371
    dis = inverseCoords(lat1, lon1, lat2, lon2)
    print("----------INVERSECOORD:----------")
    print("From (%f , %f) to (%f , %f)," % (lat1,lon1,lat2,lon2))
    print("The distance is %f K.M and the azimuth is %f°.\n" % (dis[0],dis[1]))

    # TERMINAL COORDS - Retrieves end coord given start coord, azimuth, and distance
    brng = 42.823054 #Bearing is 90 degrees converted to radians.
    d = 32.469116 #Distance in km
    lat1 = 33.779398
    lon1 = -84.413279
    tcoord = terminalCoords(lat1,lon1,brng,d)
    print("----------TERMINALCOORD:----------")
    print("From (%f , %f)" % (lat1,lon1))
    print("with a distance of %f K.M and an azimuth of %f°," % (d, brng))
    print("the terminal coords is (%f , %f)\n" % (tcoord[0], tcoord[1]))

    #GREAT CIRCLE INTERSECTION - finds point of intersection given two arcs, each having a start and end coord
    lat1 = 33.779398
    lon1 = -84.413279
    lat2 = 33.993333
    lon2 = -84.173888
    lat3 = 33.880452
    lon3 = -84.1588087
    lat4 = 33.890248
    lon4 = -84.4401807
    T = GCI(lat1, lon1, lat2, lon2, lat3, lon3, lat4, lon4)
    print("----------INTERSECTION:----------")
    print("With ArcPath1 with a startpoint of (%f , %f) and an endpoint of (%f , %f)" % (lat1,lon1,lat2,lon2))
    print("and ArcPath2 with a startpoint of (%f , %f) and an endpoint of (%f , %f)" % (lat3,lon3,lat4,lon4))
    print("The intersection is (%f , %f)\n" % (T[0],T[1]))
```

STEP 6A : MAIN METHOD (CONT.)

- Declare HOST vehicle with coordinate, speed, and azimuth.
- Define local vars, HOST and GUEST times to intersection, and BOOL TIME which is True if the time it takes for each vehicle to get to the intersection is within .035 second, if not False.
- Every time it is True, increment BOOL counter.
- BOOL COUNT is out loop invariant while it is less than 1000. In other words out dataset will have 1000 rows of information where the vehicles collide.

```
# Based on HOST coordinates, generate GUEST vehicle a constant radius away from HOST but random angle,
# then find approximate azimuth for GUEST needed to potentially collide into HOST vehicle.
print("----------NEW:----------")
HOST_A = Vehicle("Nissan Sentra", "914h80", 33.779398, -84.413279, 50, 0)
HOST_A.describe()

HOST_TIME_TO_INTERSECTION = 0
GUEST_TIME_TO_INTERSECTION = 1
BOOL_TIME = False
BOOL_COUNT = 0
while(BOOL_COUNT < 1000):
```

STEP 6A : MAIN METHOD (CONT.)

- Generate GUEST vehicle at coordinates using Terminal Coordinate function given HOST vehicle coordinate, random azimuth and 2* distance of radar radius, because that is the furthest the vehicles can be away from each other without being undetected from radar.
- Generate GUEST Azimuth using the Trigonometric functions with HOST azimuth
- Generate GUEST Speed with random() relative weights to the HOST speed

```
while(BOOL_COUNT < 1000):

    #generate vehicle at a constant distance but at a random angle from the HOST vehicle, then use half() and oppose() to calculate possible angle for guest vehicle to drive in to collide with HOST
    direwayfrom = random.randint(0,360)
    host_azi_to_guest.append(dirawayfrom)
    print(dirawayfrom)
    tcoord = terminalCoords(HOST_A.coords[0], HOST_A.coords[1], dirawayfrom, radar_radius)
    print(tcoord)
    GUEST_A = Vehicle("Nissan Altima", "rh0319", tcoord[0], tcoord[1], 50, half(HOST_A.velocity[1], oppose(dirawayfrom)))
    GUEST_A.describe()

    randy1 = [0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]
    k = random.randint(0, 1)
    x = random.choices(randy1, weights=(10,10,10,10,10,10,10,10,10,10,10,5,5,5,2,2,2,1,1,1,1), k=1)
    if(k == 1):
        t = GUEST_A.velocity[0] + x[0]
        GUEST_A.set_velocity(t)
    else:
        t = GUEST_A.velocity[0] - x[0]
        GUEST_A.set_velocity(t)
    print("RANDY")
    print(GUEST_A.velocity[0])
    print(GUEST_A.velocity[1])
    if(GUEST_A.velocity[0] < 0 ):
        print("INRANDYINADAD")
        GUEST_A.set_velocity(abs(t))
        GUEST_A.set_direction(oppose(GUEST_A.velocity[1]))
    print(GUEST_A.velocity[0])
    print(GUEST_A.velocity[1])
    GUEST_A.describe()
    #find their intersection and then determine time for each one to get to intersection
    #distance half the earth arc length and then find intersection only for intersection, then scratch that and only care about points before intersection
    #distance of half earth
    #dhe = (180/360) * pi * r / 4
    dhe = 10
    print("DHE\t\t\t\t",dhe)
    print("HOST\t\t\t\t",HOST_A.coords[0], HOST_A.coords[1], HOST_A.velocity[1])
    print("GUEST\t\t\t\t",GUEST_A.coords[0], GUEST_A.coords[1], GUEST_A.velocity[1])
    HOST_A_TERMINAL = terminalCoords(HOST_A.coords[0], HOST_A.coords[1], HOST_A.velocity[1], dhe)
    print("HOSTTERM\t\t\t",HOST_A_TERMINAL,)
    GUEST_A_TERMINAL = terminalCoords(GUEST_A.coords[0], GUEST_A.coords[1], GUEST_A.velocity[1], dhe)
    print("GUESTTERM\t\t\t",GUEST_A_TERMINAL)
```

STEP 6A : MAIN METHOD (CONT.)

- Calculate intersection point.
- Calculate time it takes for each vehicle to get to intersection. Mark Boolean true if it takes approximately the same amount of time to reach intersection.
- Append each set of information to row of each column.

```
try:
    print("BEFORE T")
    T = GCI(HOST_A.coords[0],HOST_A.coords[1], HOST_A_TERMINAL[0],HOST_A_TERMINAL[1], GUEST_A.coords[0],GUEST_A.coords[1], GUEST_A_TERMINAL[0],GUEST_A_TERMINAL[1])

    #SET THEM NOT EQUAL SO LOOP INVARIANT IS TRUE
    print("IM INSIDE THE TRY")

    print("VELOCITY\t\t\t",GUEST_A.velocity[0])
    HOST_A_TO_POTENTIAL_COLLISION = inverseCoords(HOST_A.coords[0], HOST_A.coords[1], T[0], T[1])
    GUEST_A_TO_POTENTIAL_COLLISION = inverseCoords(GUEST_A.coords[0], GUEST_A.coords[1], T[0], T[1])
    print("HOSTPOTENTIAL\t\t\t",HOST_A_TO_POTENTIAL_COLLISION)
    print("GUESTPOTENTIAL\t\t\t",GUEST_A_TO_POTENTIAL_COLLISION)
    #TIME IN SECONDS FOR EACH VEHICLE TO GET TO INTERSECTION
    HOST_TIME_TO_INTERSECTION = (HOST_A_TO_POTENTIAL_COLLISION[0] / HOST_A.velocity[0]) * 60 * 60
    GUEST_TIME_TO_INTERSECTION = (GUEST_A_TO_POTENTIAL_COLLISION[0] / GUEST_A.velocity[0]) * 60 * 60
    print("HOST_TIME_TO_INTERSECTION\t", HOST_TIME_TO_INTERSECTION)
    print("GUEST_TIME_TO_INTERSECTION\t", GUEST_TIME_TO_INTERSECTION)
    print("BOOLTIME")
    BOOL_TIME = isclose(HOST_TIME_TO_INTERSECTION, GUEST_TIME_TO_INTERSECTION, abs_tol=.035)
    if(BOOL_TIME == True):
        BOOL_COUNT += 1
    print("BOOL_TIME\t\t\t", BOOL_TIME)
    print("BOOL_COUNT\t\t\t", BOOL_COUNT)
except:
    print("COORDINATES NEVER INTERSECTED")
print("T\t\t\t\t",T)
print()
host_latitude.append(HOST_A.coords[0])
host_longitude.append(HOST_A.coords[1])
host_speed.append(HOST_A.velocity[0])
host_azimuth.append(HOST_A.velocity[1])
host_distance.append(HOST_A_TO_POTENTIAL_COLLISION[0])
host_time.append(HOST_TIME_TO_INTERSECTION)

guest_latitude.append(GUEST_A.coords[0])
guest_longitude.append(GUEST_A.coords[1])
guest_speed.append(GUEST_A.velocity[0])
guest_azimuth.append(GUEST_A.velocity[1])
guest_distance.append(GUEST_A_TO_POTENTIAL_COLLISION[0])
guest_time.append(GUEST_TIME_TO_INTERSECTION)
if(T == None):
    intersection_latitude.append(None)
    intersection_longitude.append(None)
else:
    intersection_latitude.append(T[0])
    intersection_longitude.append(T[1])
collision.append(int(BOOL_TIME))
print("----------------------------------------------------------------------")
```

STEP 6A : MAIN METHOD (CONT.)

- When loop completes, add all columns of data to a pandas dataframe, and export the data as a csv file

- Sample output of the Loop:

```
60
79.0
Vehicle [rh0319] at (33.780232 , -84.413684) moving 79.0° at 60 KPH

DHE                              10
HOST                             33.779398 -84.413279 0
GUEST                            33.78023181788039 -84.41368431056708 79.0
HOSTTERM                         (33.86932822676029, -84.413279)
GUESTTERM                        (33.79734580564846, -84.30745460911976)
BEFORE T
IM INSIDE THE TRY
VELOCITY                         60
HOSTPOTENTIAL                    (0.10011503451648597, 359.9964151024692)
GUESTPOTENTIAL                   (0.038177770272440736, 78.82851326459465)
HOST_TIME_TO_INTERSECTION        7.20828248518699
GUEST_TIME_TO_INTERSECTION       2.290666216346444
BOOLTIME
BOOL_TIME                        False
BOOL_COUNT                       72
T                                (33.78029833677386, -84.4132790677745)

--------------------------------------------------------------------
203
(33.778570187177216, -84.41370174852032)
Vehicle [rh0319] at (33.778570 , -84.413702) moving 11.5° at 50 KPH

setter method called
RANDY
10
11.5
10
11.5
Vehicle [rh0319] at (33.778570 , -84.413702) moving 11.5° at 10 KPH

DHE                              10
HOST                             33.779398 -84.413279 0
GUEST                            33.778570187177216 -84.41370174852032 11.5
HOSTTERM                         (33.86932822676029, -84.413279)
GUESTTERM                        (33.86669315827553, -84.3921090575779)
BEFORE T
IM INSIDE THE TRY
VELOCITY                         10
HOSTPOTENTIAL                    (0.10054931256327228, 359.9964152597624)
GUESTPOTENTIAL                   (0.19652200552003216, 11.466038602046806)
HOST_TIME_TO_INTERSECTION        7.239550504555605
GUEST_TIME_TO_INTERSECTION       70.74792198721157
BOOLTIME
BOOL_TIME                        False
BOOL_COUNT                       72
T                                (33.78030224224617, -84.4132790680655)

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

```python
data["HOST_LAT"] = host_latitude
data["HOST_LON"] = host_longitude
data["HOST_DIRECTION"] = host_azimuth
data["HOST_SPEED"] = host_speed
data["HOST_DIS"] = host_distance
data["HOST_TIME"] = host_time

data["HOST_AZI_TO_GUEST"] = host_azi_to_guest

data["GUEST_LAT"] = guest_latitude
data["GUEST_LON"] = guest_longitude
data["GUEST_DIRECTION"] = guest_azimuth
data["GUEST_SPEED"] = guest_speed
data["GUEST_DIS"] = guest_distance
data["GUEST_TIME"] = guest_time

data["INTERSECTION_LAT"] = intersection_latitude
data["INTERSECTION_LON"] = intersection_longitude
data["COLLISION"] = collision
print(data)
data.to_csv('./vehicle_collision_data.csv')
```

DATASET FILE CONTENTS

A1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1		HOST_LAT	HOST_LON	HOST_DIREC	HOST_SPEED	HOST_DIS	HOST_TIME	HOST_AZI_T	GUEST_LAT	GUEST_LON	GUEST_DIRE	GUEST_SPEE	GUEST_DIS	GUEST_TIME	INTERSECTIO	INTERSECTIO	COLLISION
2	0	33.779398	-84.413279	0	50	0.10027791	7.22000942	326	33.7801436	-84.413884	73	15	0.05855021	14.0520492	33.7802998	-84.413279	0
3	1	33.779398	-84.413279	0	50	0.09972437	7.18015493	30	33.7801768	-84.412738	285	130	0.05169919	1.43166995	33.7802948	-84.413279	0
4	2	33.779398	-84.413279	0	50	0.09841458	7.08584967	120	33.7789483	-84.412342	330	95	0.171837	6.51171782	33.780283	-84.413279	0
5	3	33.779398	-84.413279	0	50	0.09846455	7.08944779	97	33.7792884	-84.412205	318.5	15	0.14864876	35.675702	33.7802835	-84.413279	0
6	4	33.779398	-84.413279	0	50	0.10017576	7.21265441	333	33.7801993	-84.41377	76.5	30	0.04672433	5.60691904	33.7802989	-84.413279	0
7	5	33.779398	-84.413279	0	50	0.09977225	7.183602	27	33.7801993	-84.412788	283.5	50	0.04664251	3.35826061	33.7802953	-84.413279	0
8	6	33.779398	-84.413279	0	50	0.10124117	7.2893642	269	33.7793823	-84.414361	44.5	90	0.14353358	5.74134315	33.7803085	-84.413279	0
9	7	33.779398	-84.413279	0	50	0.09843879	7.08759313	100	33.7792418	-84.412213	320	85	0.15202024	6.4385042	33.7802833	-84.413279	0
10	8	33.779398	-84.413279	0	50	0.0997076	7.17894723	31	33.7801689	-84.412722	285.5	45	0.0533763	4.27010394	33.7802947	-84.413279	0
11	9	33.779398	-84.413279	0	50	0.10102894	7.27408353	223	33.7787403	-84.414017	21.5	15	0.18703893	44.8893434	33.7803066	-84.413279	0
12	10	33.779398	-84.413279	0	50	0.10088167	7.26348041	216	33.7786704	-84.413915	18	10	0.19104808	68.7773073	33.7803052	-84.413279	0
13	11	33.779398	-84.413279	0	50	0.09847005	7.08984344	127	33.7788568	-84.412415	333.5	75	0.17762175	8.5258439	33.7802835	-84.413279	0
14	12	33.779398	-84.413279	0	50	0.09902183	7.12957207	64	33.7797922	-84.412307	302	90	0.10547405	4.21896187	33.7802885	-84.413279	0
15	13	33.779398	-84.413279	0	50	0.10069989	7.25039224	303	33.7798878	-84.414186	61.5	110	0.09576217	3.13403464	33.7803036	-84.413279	0
16	14	33.779398	-84.413279	0	50	0.09871856	7.10773599	79	33.7795696	-84.412217	309.5	50	0.12640921	9.1014631	33.7802858	-84.413279	0
17	15	33.779398	-84.413279	0	50	0.10032312	7.2232646	195	33.7785293	-84.413559	7.5	30	0.19860851	23.8330217	33.7803002	-84.413279	0
18	16	33.779398	-84.413279	0	50	0.10065814	7.24738623	207	33.7785967	-84.41377	13.5	5	0.19511254	140.481026	33.7803032	-84.413279	0
19	17	33.779398	-84.413279	0	50	0.0988132	7.11455031	74	33.7796459	-84.412239	307	80	0.11965749	5.38458712	33.7802866	-84.413279	0
20	18	33.779398	-84.413279	0	50	0.10001076	7.2007749	352	33.7802886	-84.41343	86	15	0.01394581	3.34699383	33.7802974	-84.413279	0
21	19	33.779398	-84.413279	0	50	0.09974075	7.18133386	29	33.7801845	-84.412754	284.5	5	0.05001777	36.0127936	33.780295	-84.413279	0
22	20	33.779398	-84.413279	0	50	0.0986775	7.10477997	141	33.7786991	-84.412598	340.5	95	0.18728425	7.09708736	33.7802854	-84.413279	1
23	21	33.779398	-84.413279	0	50	0.10054931	7.2395505	203	33.7785702	-84.413702	11.5	35	0.19652201	20.213692	33.7803022	-84.413279	0
24	22	33.779398	-84.413279	0	50	0.09904364	7.13114235	63	33.7798063	-84.412315	301.5	0	0.10400852	20.213692	33.7802887	-84.413279	0
25	23	33.779398	-84.413279	0	50	0.09904364	7.13114235	91	33.7793823	-84.412197	135.5	35	0.10400852	20.213692			0
26	24	33.779398	-84.413279	0	50	0.10023189	7.21669634	329	33.7801689	-84.413836	74.5	0	0.05350408	20.213692	33.7802994	-84.413279	0
27	25	33.779398	-84.413279	0	50	0.09883304	7.11597915	73	33.7796609	-84.412244	306.5	105	0.11827915	4.055285	33.7802868	-84.413279	0
28	26	33.779398	-84.413279	0	50	0.09841849	7.08613159	103	33.7791957	-84.412225	321.5	30	0.15529092	18.6349101	33.7802831	-84.413279	0
29	27	33.779398	-84.413279	0	50	0.10006604	7.20475491	343	33.780258	-84.413595	81.5	140	0.02956553	0.76025637	33.7802979	-84.413279	0

NEURAL NETWORK

Implementation on separate python file, using Keras, a machine learning API by Google on TensorFlow

GOAL OF NEURAL NETWORK

- Multivariate Linear Regression with Neural Networks
- Use MvLRNN to predict if vehicles will collide given input
- Input nodes were experimented with, with all of them to only a few, decided to keep 7 of them (because keeping the others did not help or made it worse):
- HOST distance, azimuth, and time to intersection, GUEST distance, azimuth, and time to intersection, and GUEST speed
- Output node was whether the Vehicles collided (1) or did not collide (0)

STEP 1 : IMPORT

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.wrappers.scikit_learn import KerasRegressor
```

STEP 2 : DATA PREPROCESSING AND ASSIGNING I/O

```
#Get dataset, print shape and view
data = pd.read_csv("./vehicle_collision_data.csv")
print(data.shape)
data.head()

# preprocess by removing any NaN values
data = data.dropna()
print(data.shape)
data.head()

#Drop non needed columns and Y value for INPUT
x = data.drop(['Unnamed: 0','COLLISION',"HOST_LAT","HOST_LON","GUEST_LAT","GUEST_LON","HOST_DIRECTION","HOST_SPEED","INTERSECTION_LAT","INTERSECTION_LON"], axis = 1).to_numpy()
print(x.shape)
print(x)

# keep Collision true/false as output
y = data['COLLISION'].to_numpy()
y = np.reshape(y,(-1,1))
print(y.shape)
print(y)
```

STEP 3 : NORMALIZE, TRANSFORM, AND TRAIN THE DATA

```
#normalize the input and output!
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

# transform the data
print(scaler_x.fit(x))
xscale=scaler_x.transform(x)
print(scaler_y.fit(y))
yscale=scaler_y.transform(y)

#train the data,
X_train, X_test, y_train, y_test = train_test_split(xscale, yscale)
```

STEP 4 : CREATE NN, COMPILE, AND TEST THE TRAINING TO FIT MODEL

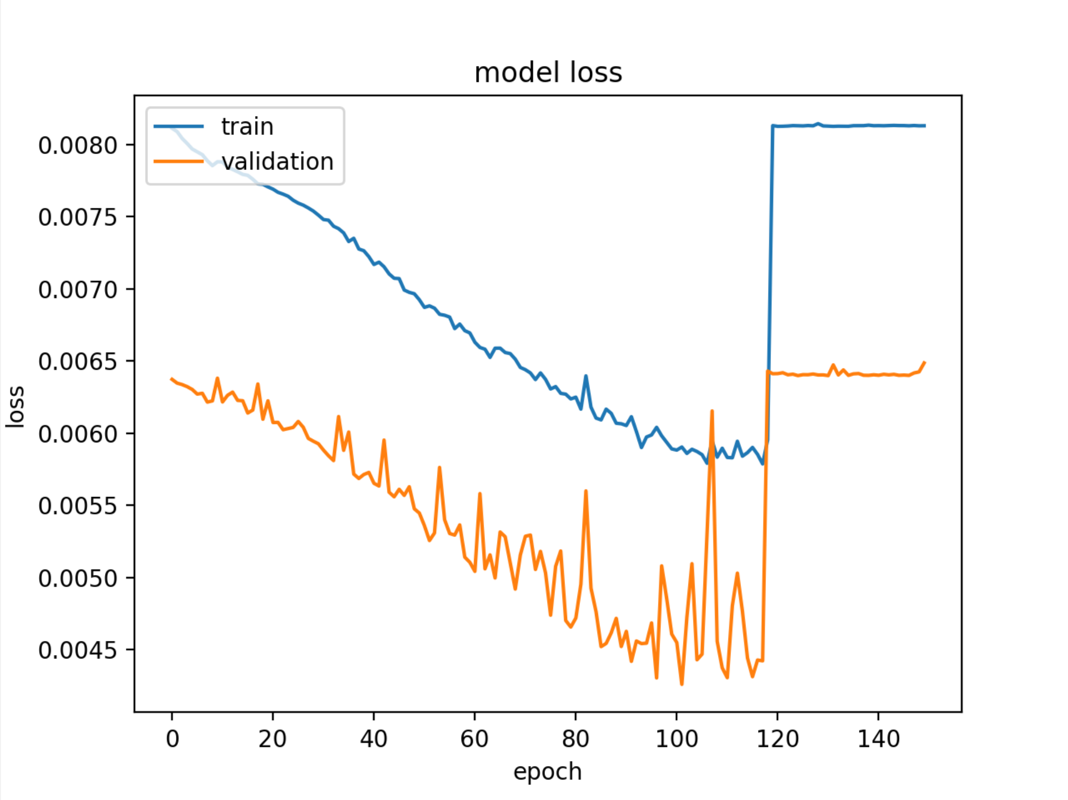
```python
#create NN with 2 hidden layers, one with 6 nodes and one with 4 nodes
model = Sequential()
model.add(Dense(6, input_dim=7, kernel_initializer='normal', activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()

#compiles NN
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])

#test the training and fit the model
history = model.fit(X_train, y_train, epochs=150, batch_size=50,  verbose=1, validation_split=0.2)
```

```
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0149
Epoch 128/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0160
Epoch 129/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0165 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0149
Epoch 130/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0151
Epoch 131/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0137
Epoch 132/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0065 - val_mse: 0.0065 - val_mae: 0.0213
Epoch 133/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0164 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0150
Epoch 134/150
1578/1578 [==============================] - 3s 2ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0189
Epoch 135/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0143
Epoch 136/150
1578/1578 [==============================] - 2s 2ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0161
Epoch 137/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0164
Epoch 138/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0140
Epoch 139/150
1578/1578 [==============================] - 3s 2ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0123
Epoch 140/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0149
Epoch 141/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0138
Epoch 142/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0156
Epoch 143/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0147
Epoch 144/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0155
Epoch 145/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0135
Epoch 146/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0143
Epoch 147/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0126
Epoch 148/150
1578/1578 [==============================] - 2s 2ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0163 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0169
Epoch 149/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0064 - val_mse: 0.0064 - val_mae: 0.0177
Epoch 150/150
1578/1578 [==============================] - 2s 1ms/step - loss: 0.0081 - mse: 0.0081 - mae: 0.0162 - val_loss: 0.0065 - val_mse: 0.0065 - val_mae: 0.0220
dict_keys(['loss', 'mse', 'mae', 'val_loss', 'val_mse', 'val_mae'])
```

STEP 5 : GRAPH LOSS AND LEARNING

```
#format and show key values
print(history.history.keys())
# "Loss"
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Concerns: Although the predictions itself worked, the output for the training and validation loss was not as expected possibly for two reasons: Not optimizing the neural network in terms of batch size, epochs, input and hidden layer number, etc., but is likely due to overfitting the data.



STEP 6 : PREDICTIONS

```
# random point for prediction
Xnew = np.array([[0.100278,7.220009,326,80,15,0.058550,7.22]])
Xnew= scaler_x.transform(Xnew)
#predict!
ynew= model.predict(Xnew)
#invert normalize
ynew = scaler_y.inverse_transform(ynew)
Xnew = scaler_x.inverse_transform(Xnew)
print("X=%s, Predicted=%s" % (Xnew[0], ynew[0]))
```

```
dict_keys(['loss', 'mse', 'mae', 'val_loss', 'val_mse', 'val_mae'])
X=[1.002780e-01 7.220009e+00 3.260000e+02 8.000000e+01 1.500000e+01
 5.855000e-02 7.220000e+00], Predicted=[0.01572663]
(base) Anthonys-MBP-2:Vehicle-Avoidance-AI anthonyasilo$
```

This is an example of a prediction, which is so far accurate and haven't had problems with it

CONCERNS

- It is crucial that the correct independent variables are used as input for the neural network: this would have been possible if not as much time was spent getting the coordinate system working or if more time was permitted.
- While the implementation of regression based neural networks have potential to provide accuracy, this is only possible when being fine-tuned to fit one's data. Although the predictions itself worked, the output for the training and validation loss was not as expected
- It would probably make more sense to also give more randomization in terms of the host vehicles location, speed, and azimuth, that way the neural network has more to data to normalize and derive.
- When generating the randomness, it was kept simple: The host vehicle started in the same location every time. This was on purpose because when looking on a global scale, it won't matter what the coordinates are but rather the speeds and directions of each vehicle relative to each other towards an intersection. However, if one point is defined for the vehicles whether it is the host coordinates, guest coordinates, or the intersection coordinates, all other points can be found.

FUTURE

- Generate more data for consumption, and then optimize the learning algorithm to minimize loss without overfitting.
- This model fits a spherical earth, where the calculations give about a 0.3% error, so it would make sense to update the coordinate system to fit an ellipsoidal earth.
- Establish a form of communication between vehicles and allow the simulation for vehicles to actually move.

REFERENCES

- **References**
- "New WHO report highlights insufficient progress to tackle lack of safety on the world's roads." Koninklijke Brill NV, doi: 10.1163/2210-7975_HRD-9841-20180026.
- "National Motor Vehicle Crash Causation Survey: Report to Congress," p. 47.
- "Understanding Latitude and Longitude." https://journeynorth.org/tm/LongitudeIntro.html (accessed Dec. 14, 2020).
- "Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript." https://www.movable-type.co.uk/scripts/latlong.html (accessed Dec. 14, 2020).
- "python - Get lat/long given current point, distance and bearing," *Stack Overflow*. https://stackoverflow.com/questions/7222382/get-lat-long-given-current-point-distance-and-bearing (accessed Dec. 14, 2020).
- E. Spinielli, "Understanding Great Circle Arcs Intersection Algorithm," *Enrico's blog*, Oct. 19, 2014. https://enrico.spinielli.net/2014/10/19/understanding-great-circle-arcs_57/ (accessed Dec. 14, 2020).
- D. Bertels, "Intersection of Great Circles," p. 37.
- "Finding The Intersection Of Two Arcs That Lie On A Sphere." https://blog.mbedded.ninja/mathematics/geometry/spherical-geometry/finding-the-intersection-of-two-arcs-that-lie-on-a-sphere/ (accessed Dec. 14, 2020).
- "Keras: Regression-based neural networks," *DataScience+*. https://datascienceplus.com/keras-regression-based-neural-networks/ (accessed Dec. 14, 2020).
- "Debugging a Machine Learning model written in TensorFlow and Keras | by Lak Lakshmanan | Towards Data Science." https://towardsdatascience.com/debugging-a-machine-learning-model-written-in-tensorflow-and-keras-f514008ce736 (accessed Dec. 14, 2020).
- K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of Real-time Processing Technologies of IoT Data Streams," Journal of Information Processing, vol. 24, no. 2, pp. 195–202, 2016, doi: 10.2197/ipsjjip.24.195.
- G. Brambilla, M. Amoretti, F. Medioli, and F. Zanichelli, "Blockchain-based Proof of Location," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp. 146–153, Jul. 2018, doi: 10.1109/QRS-C.2018.00038.
- C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," Robotics and Autonomous Systems, vol. 114, pp. 1–18, Apr. 2019, doi: 10.1016/j.robot.2019.01.003.