ENTS 749C: ADVANCED TOPICS IN NETWORKING - VEHICULAR NETWORKS

**MINI PROJECT #1**

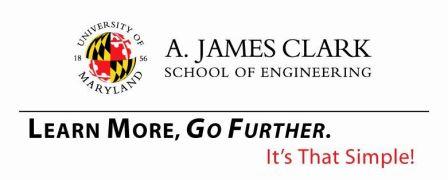
Submitted by

PRATHAPANI, NIKHIL

UID: 113-66-0344

EMAIL: [nprathap@umd.edu](mailto:nprathap@umd.edu)

Section: 0101



**TABLE OF CONTENTS**

Title Page…………………………………………………………………………………………………………0

Problem Statement………………………………………………………………………………………………..2

1. Assumptions…………………………………………………………………………………………..2
2. Python based Data Analysis…………………………………………………………………………..2
3. Data Plots…………………………………………………………………………………………….14
4. Observations………………………………………………………………………………………....21

**PROBLEM STATEMENT:**

Alice drives her Controller Area Network (CAN) enabled car. She has volunteered to make her vehicle’s CAN bus data available to an insurance company XYZ for the purpose of earning the “safe driver” price reduction. Her vehicle data can be used to create position traces and monitor some of vehicular system parameters when moving on these traces.

XYZ obtains the data file called alicedata.json. Now XYZ has to mine this data file to assess the scenario that Alice’s vehicle went through and learn her driving behavior. Python data must be processed and the desired plots must be generated.

1. **Assumptions:**

There are 12 different signal names in the given data.

The following parameters are assumed to have following units:

1. Vehicle Speed (KM/HR) –kilometer/hour
2. Accelerator Pedal Position (Radians)
3. Engine Speed (RPM-rotations per minute)
4. Torque at transmission (lb-feet)
5. Latitude (degrees)
6. Longitude (degrees)
7. Steering Wheel Angle (degrees)
8. Fuel consumed since restart (gallon)
9. Odometer (KM)
10. Fuel Level (Gallon)
11. Brake Pedal Status (True/False)
12. Transmission gear position (neutral/first/second/third/fourth)

The given alicedata.json file is loaded as a list of dictionaries and suitable operations have been performed on it.

Each dictionary has the keys: ‘Name’, ‘Time Stamp’, ‘Value’ and each of them have corresponding values.

1. **Python-based Data Analysis:**

* func1()

**Code:**

import json

data = []

def func1(alicedata):

with open(alicedata) as json\_file:

for line in json\_file:

global data

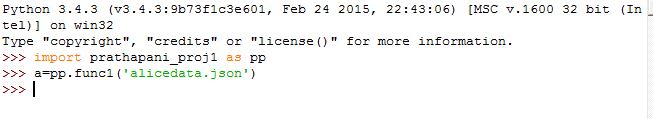
data.append(json.loads(line)) #reading data from the json file

#print(line)

return data #returning the function for calling it later

#e1=func1('alicedata.json')

**Output:**



* func2()

**Code:**

def func2(alicedata):

e1=func1(alicedata) #retrieving the output list of function 1 , by calling it along with the parameter.

from pprint import pprint

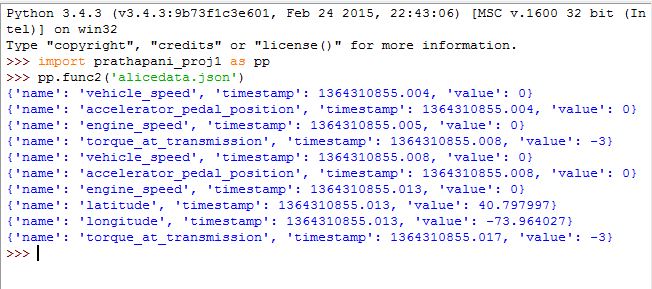
#pprint(e1[:10])

for ten\_entries\_data\_file in e1[:10]:

pprint(ten\_entries\_data\_file)

#func2('alicedata.json')

**Output:**



* func3()

**Code:**

def func3(alicedata):

e1=func1(alicedata)

lst\_signal\_names\_unique=[]

lst1\_signal\_names\_all=[]

lst\_1=[]

lst\_a=[]

count\_dict\_number\_of\_occurences = {}

for signal\_name\_unique in e1:

if not signal\_name\_unique['name'] in lst\_signal\_names\_unique: #ignoring repititions

lst\_signal\_names\_unique.append(signal\_name\_unique['name'])

print(lst\_signal\_names\_unique)

for signal\_name\_all in e1:

lst1\_signal\_names\_all.append(signal\_name\_all['name'])

for word in lst1\_signal\_names\_all:

if word in count\_dict\_number\_of\_occurences: #finding number of occurences

count\_dict\_number\_of\_occurences[word]+= 1

else:

count\_dict\_number\_of\_occurences[word] = 1

print(count\_dict\_number\_of\_occurences)

'''

#This is another way of finding the number of occurences of a list element

from collections import Counter

c=Counter(lst1)

print(c)

'''

inp=input("Enter a SIGNAL NAME from the following: --> 'vehicle\_speed','accelerator\_pedal\_position', 'engine\_speed', 'torque\_at\_transmission', 'latitude', 'longitude', 'steering\_wheel\_angle', 'fuel\_consumed\_since\_restart', 'odometer','fuel\_level', 'brake\_pedal\_status', 'transmission\_gear\_position'\n")

if(inp!='transmission\_gear\_position'):

for i in e1:

lst1\_signal\_names\_all.append(i['name'])

if(i['name']== inp):

lst\_1.append(i['value'])

print("The minimum value of",inp, "is" ,min(lst\_1),"\n","The maximum value of",inp,"is",max(lst\_1))

print("number of occurrences and value range of ",inp," is:",count\_dict\_number\_of\_occurences[inp])

elif(inp=='transmission\_gear\_position'): #transmission\_gear\_position has string values. Hence mapping it to integers for future use

for i in e1:

if(i['name']== 'transmission\_gear\_position'):

lst\_a.append(i['value'])

#print(lst\_a)

for k in range(len(lst\_a)):

if(lst\_a[k]=='first'):

lst\_a[k]=1

elif(lst\_a[k]=='second'):

lst\_a[k]=2

elif(lst\_a[k]=='three'):

lst\_a[k]=3

elif(lst\_a[k]=='fourth'):

lst\_a[k]=4

else:

lst\_a[k]=0 #for 'neutral' gear

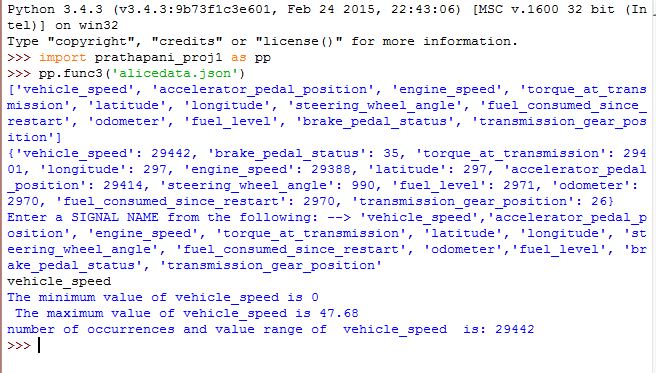
#print(lst\_a)

print("The minimum value of",inp, "is" ,min(lst\_a),"\n","The maximum value of",inp,"is",max(lst\_a))

print("number of occurrences and value range of ",inp," is:",len(lst\_a))

#func3('alicedata.json')

**Output:**



* func4()

**Code:**

def func4(alicedata):

e1=func1(alicedata)

lst\_odometer\_reading=[]

lst1\_trip\_time=[]

for odometer\_reading in e1:

if(odometer\_reading['name']== 'odometer'):

lst\_odometer\_reading.append(odometer\_reading['value'])

print("The vehicle trip distance over recorded data:", lst\_odometer\_reading[-1]-lst\_odometer\_reading[0],"kilometer or",0.621371\*(lst\_odometer\_reading[-1]-lst\_odometer\_reading[0]),"miles")

#print("The vehicle trip distance over recorded data:", max(lst)-min(lst),"kilometer or",0.621371\*(max(lst)-min(lst)),"miles")

for trip\_time in e1:

if(trip\_time['name']== 'vehicle\_speed'):

lst1\_trip\_time.append(trip\_time['timestamp'])

print("total trip time period:",lst1\_trip\_time[-1]-lst1\_trip\_time[0],"seconds")

'''

#This is another way of finding distance between two points using spherical coordinates. Latitude and Longitude in this case

inp='latitude'

inp1='longitude'

for i in e1:

if(i['name']== 'latitude'):

lst\_1.append(i['value'])

print("1st and last latitude of recorded data:",lst\_1[-1],lst\_1[0])

lat1=lst\_1[0]

lat2=lst\_1[-1]

for i in e1:

if(i['name']== 'longitude'):

lst\_2.append(i['value'])

print("1st and last longitude of recorded data:",lst\_2[-1],lst\_2[0])

lon1=lst\_2[0]

lon2=lst\_2[-1]

from math import radians, cos, sin, asin, sqrt

#def distancelatlon(lon1, lat1, lon2, lat2):

"""

Calculate the great circle distance between two points

on the earth (specified in decimal degrees)

"""

# convert decimal degrees to radians

lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

#latitude-longitude formula

dlon = lon2 - lon1

dlat = lat2 - lat1

a = sin(dlat/2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(dlon/2)\*\*2

c = 2 \* asin(sqrt(a))

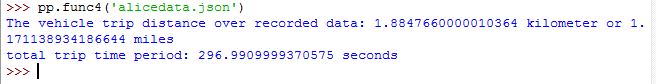
miles = 3956.2691\* c

#print("The vehicle trip distance over recorded data:", miles," miles")

'''

#func4('alicedata.json')

**Output:**



* func5()

**Code:**

def func5(alicedata):

e1=func1(alicedata)

lst1=[]

lst2=[]

lst3=[]

lst4=[]

lst5=[]

lst6=[]

lst7=[]

lst8=[]

lst9=[]

lst10=[]

lst11=[]

lst12=[]

lst\_1=[]

lst\_a=[]

from matplotlib import pylab as pl

from matplotlib import pyplot as pl

fig = pl.figure()

import numpy as np

for time\_stamp in e1:

lst\_1.append(time\_stamp['timestamp'])

for i in e1:

if(i['name']== 'vehicle\_speed'):

lst1.append(i['value'])

pl.plot(lst\_1[0:len(lst1)],lst1)

fig.text(0.8,0.8,'vehicle speed vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('vehicle\_Speed')

pl.show()

#fig.suptitle('vehicle speed vs trip time', fontsize=20)

#plt.xlabel('trip time', fontsize=18)

#plt.ylabel('vehicle speed', fontsize=18)

fig = pl.figure()

for i in e1:

if(i['name']== 'accelerator\_pedal\_position'):

lst2.append(i['value'])

pl.plot(lst\_1[0:len(lst2)],lst2)

fig.text(0.8,0.8,'accelerator pedal position vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('accelerator\_pedal\_position')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'engine\_speed'):

lst3.append(i['value'])

pl.plot(lst\_1[0:len(lst3)],lst3)

fig.text(0.8,0.8,'engine speed vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('engine\_speed')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'torque\_at\_transmission'):

lst4.append(i['value'])

pl.plot(lst\_1[0:len(lst4)],lst4)

fig.text(0.8,0.8,'torque\_at\_transmission vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('torque\_at\_transmission')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'latitude'):

lst5.append(i['value'])

pl.plot(lst\_1[0:len(lst5)],lst5)

fig.text(0.8,0.8,'latitude vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('latitude')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'longitude'):

lst6.append(i['value'])

pl.plot(lst\_1[0:len(lst6)],lst6)

fig.text(0.8,0.8,'longitude vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('longitude')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'steering\_wheel\_angle'):

lst7.append(i['value'])

pl.plot(lst\_1[0:len(lst7)],lst7)

fig.text(0.8,0.8,'steering\_wheel\_angle vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('steering\_wheel\_angle')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'fuel\_consumed\_since\_restart'):

lst8.append(i['value'])

pl.plot(lst\_1[0:len(lst8)],lst8)

fig.text(0.8,0.8,'fuel\_consumed\_since\_restart vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('fuel\_consumed\_since\_restart')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'odometer'):

lst9.append(i['value'])

pl.plot(lst\_1[0:len(lst9)],lst9)

fig.text(0.8,0.8,'odometer vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('odometer')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'fuel\_level'):

lst10.append(i['value'])

pl.plot(lst\_1[0:len(lst10)],lst10)

fig.text(0.8,0.8,'fuel\_level vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('fuel\_level')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'brake\_pedal\_status'):

lst11.append(i['value'])

pl.plot(lst\_1[0:len(lst11)],lst11)

fig.text(0.8,0.8,'brake\_pedal\_status vs time stamp',ha='center')

pl.xlabel('timestamp')

pl.ylabel('brake\_pedal\_status')

pl.show()

fig = pl.figure()

for i in e1:

if(i['name']== 'transmission\_gear\_position'):

lst\_a.append(i['value'])

#print(lst\_a)

for k in range(len(lst\_a)):

if(lst\_a[k]=='first'):

lst\_a[k]=1

elif(lst\_a[k]=='second'):

lst\_a[k]=2

elif(lst\_a[k]=='three'):

lst\_a[k]=3

elif(lst\_a[k]=='fourth'):

lst\_a[k]=4

else:

lst\_a[k]=0 #for 'neutral' gear

pl.plot(lst\_1[0:len(lst\_a)],lst\_a)

fig.text(0.8,0.8,'transmission\_gear\_position',ha='center')

pl.xlabel('timestamp')

pl.ylabel('transmission\_gear\_position')

pl.show()

#func5('alicedata.json')

**Output: \*The outputs are plotted in dataplots section of report**

C:\Users\Nikhil\Desktop\func5.JPG

* func6()

**Code:**

def func6(alicedata):

e1=func1(alicedata)

inp='vehicle\_speed'

lst\_signal\_name=[]

lst1\_vehicle\_speed\_values=[]

lst\_1=[]

count = {}

for i in e1:

if(i['name']== 'vehicle\_speed'):

lst1\_vehicle\_speed\_values.append(i['value'])

for i in e1:

lst\_signal\_name.append(i['name'])

for word in lst\_signal\_name:

if word in count:

count[word]+= 1

else:

count[word] = 1

print("The minimum value of vehicle\_speed is" ,min(lst1\_vehicle\_speed\_values),"\n","The maximum value of vehicle\_speed is",max(lst1\_vehicle\_speed\_values))

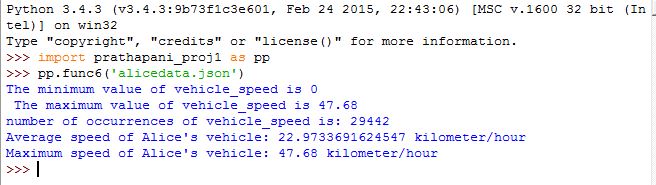
print("number of occurrences of vehicle\_speed is:",count[inp])

print("Average speed of Alice's vehicle:",sum(lst1\_vehicle\_speed\_values)/count[inp],"kilometer/hour")

print("Maximum speed of Alice's vehicle:",max(lst1\_vehicle\_speed\_values),"kilometer/hour")

#func6('alicedata.json')

**Output:**



* func7()

**Code:**

def func7(alicedata):

e1=func1(alicedata)

lst1\_latitude=[]

lst2\_longitude=[]

import pygmaps

#import webbrowser

for latitude\_values in e1:

if(latitude\_values['name']== 'latitude'):

lst1\_latitude.append(latitude\_values['value']) #appending all latitude values to a list

for longitude\_values in e1:

if(longitude\_values['name']== 'longitude'):

lst2\_longitude.append(longitude\_values['value']) #appending all longitude values to a list

mymap = pygmaps.maps(lst1\_latitude[int((len(lst1\_latitude))/2)], lst2\_longitude[int((len(lst2\_longitude))/2)], 16)

path = list(zip(lst1\_latitude,lst2\_longitude)) #creating a new list with

mymap.addpath(path,"#00FF00")

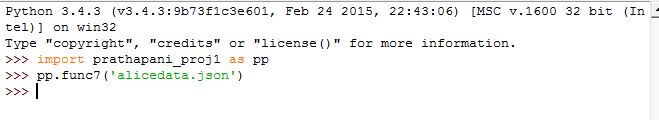
mymap.addpoint(lst1\_latitude[0], lst2\_longitude[0], "#0000FF") #starting point - Blue

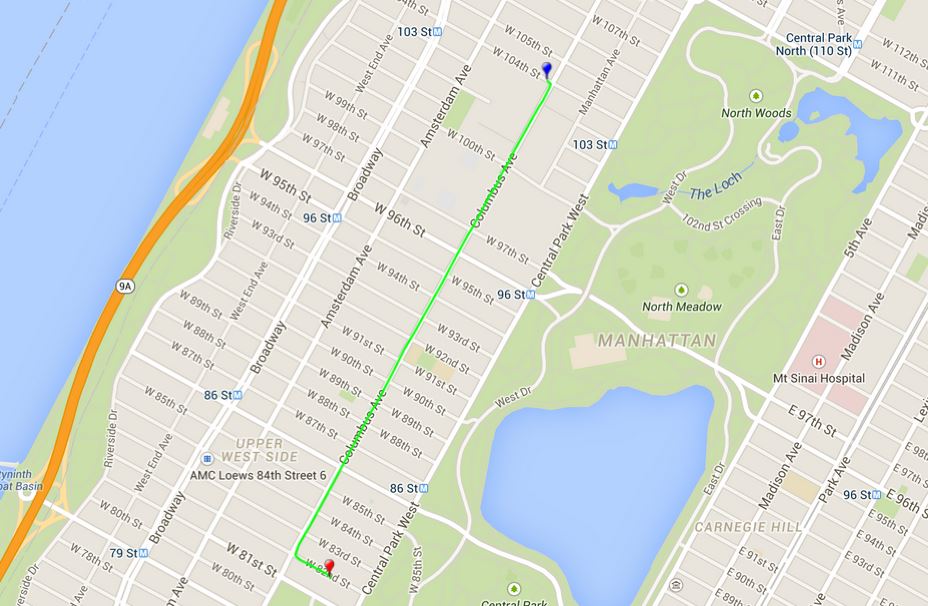
mymap.addpoint(lst1\_latitude[-1], lst2\_longitude[-1], "#FF0000") #ending point - Red

mymap.draw('./mymap.html')

#func7('alicedata.json')

**Output:**





* func8()

**Code:**

def func8(alicedata):

e1=func1(alicedata)

lst\_odometer\_reading=[]

lst1\_fuel\_consumed=[]

for odometer\_reading in e1:

if(odometer\_reading['name']== 'odometer'):

lst\_odometer\_reading.append(odometer\_reading['value'])

print("The vehicle trip distance over recorded data:", lst\_odometer\_reading[-1]-lst\_odometer\_reading[0],"kilometer or",0.621371\*(lst\_odometer\_reading[-1]-lst\_odometer\_reading[0]),"miles")

for fuel\_level in e1:

if(fuel\_level['name']== 'fuel\_consumed\_since\_restart'):

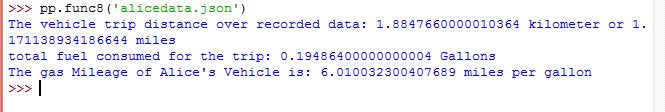
lst1\_fuel\_consumed.append(fuel\_level['value'])

print("total fuel consumed for the trip:",lst1\_fuel\_consumed[-1]-lst1\_fuel\_consumed[0],"Gallons")

print("The gas Mileage of Alice's Vehicle is:", (0.621371\*(lst\_odometer\_reading[-1]-lst\_odometer\_reading[0]))/(lst1\_fuel\_consumed[-1]-lst1\_fuel\_consumed[0]),"miles per gallon")

#func8('alicedata.json')

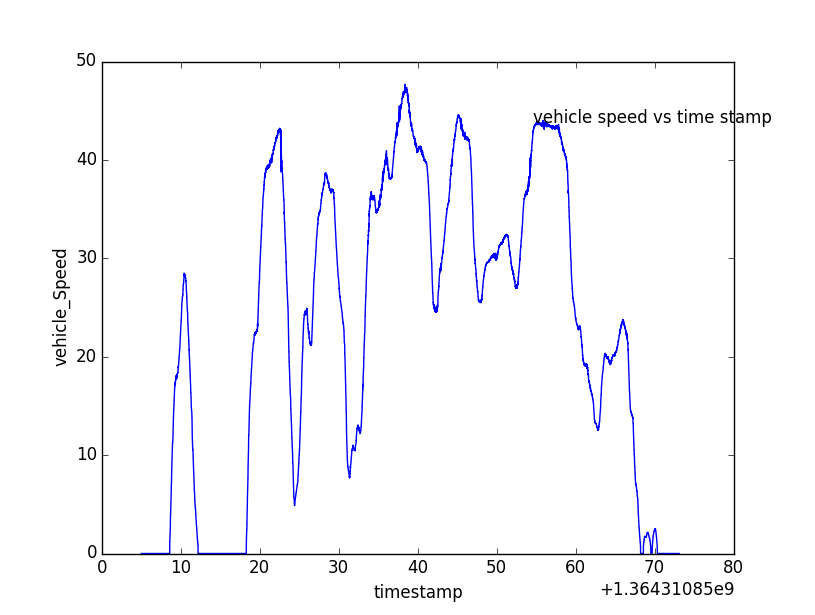
**Output:**



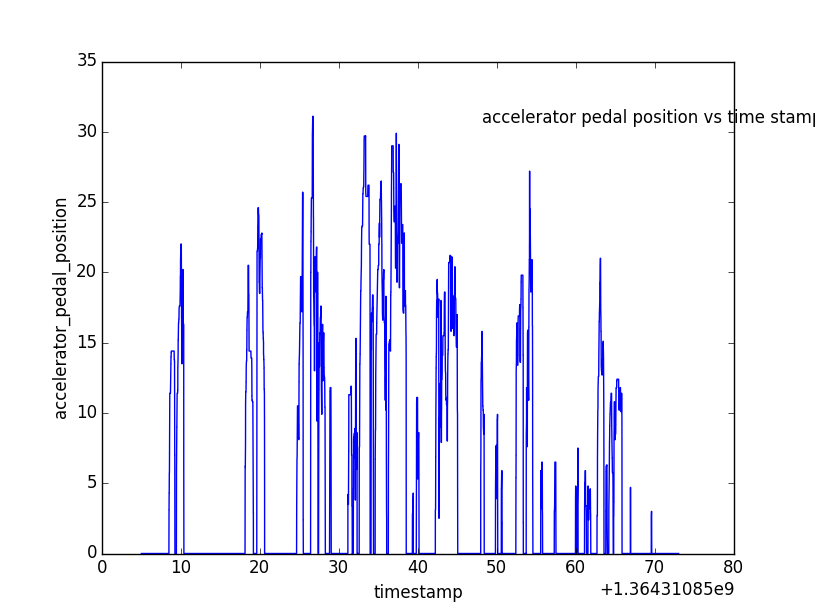
1. **Data Plots:**

func5():

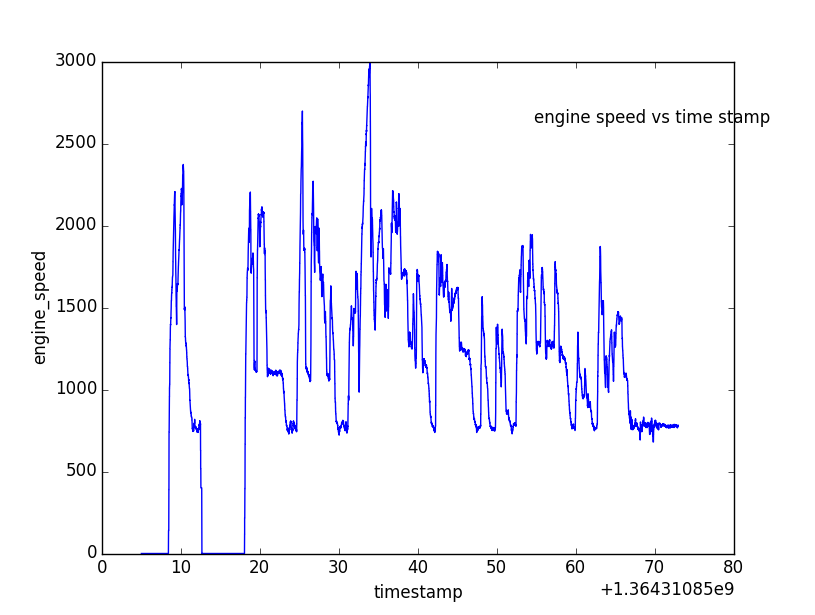
III-a) vehicle speed vs time stamp



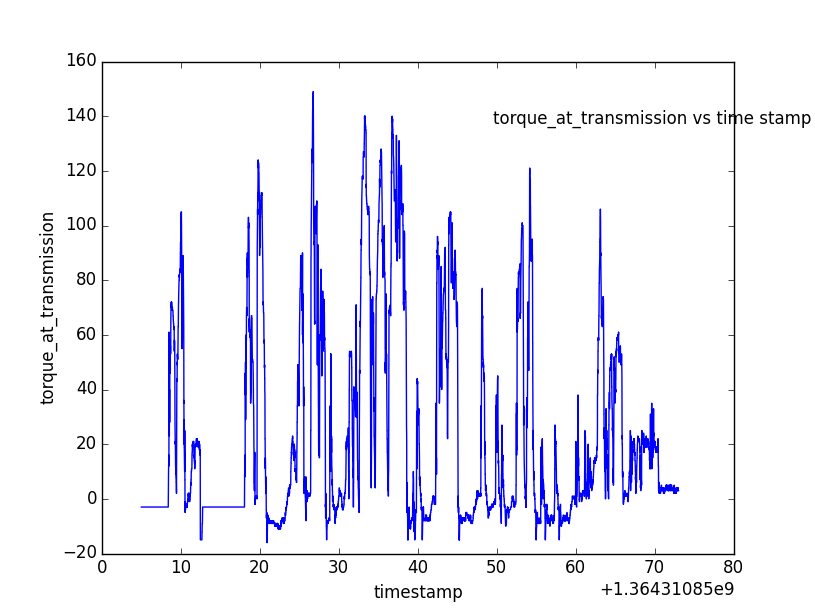
III-b) accelerator pedal position vs time stamp



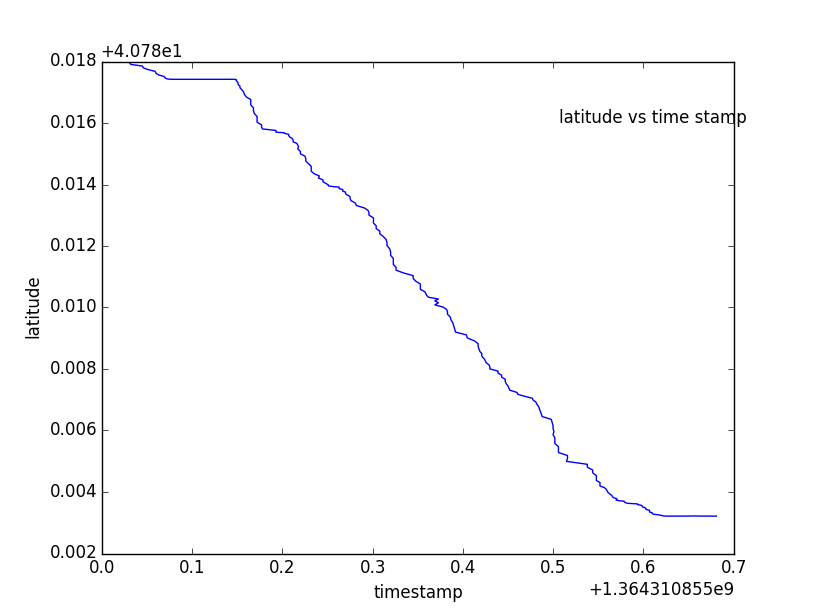
III-c) engine speed vs time stamp

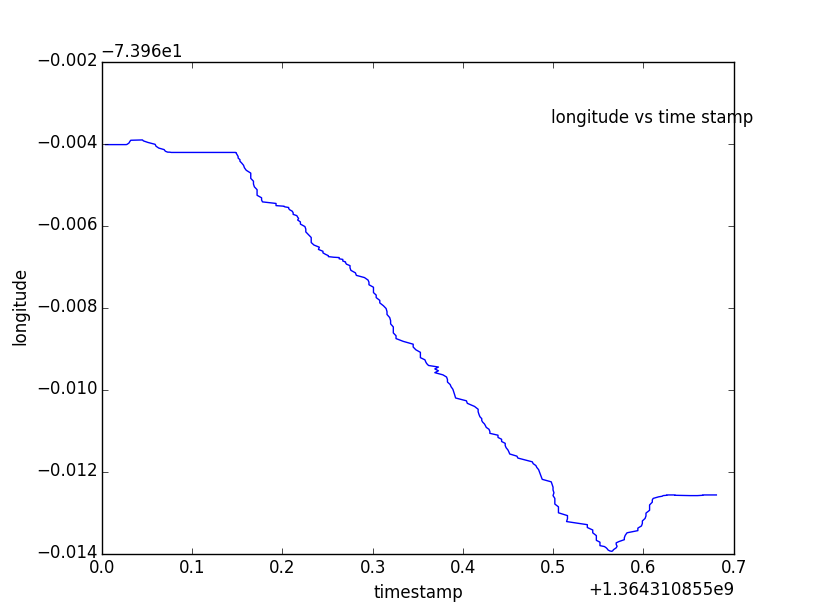


III-d) torque at transmission vs time stamp

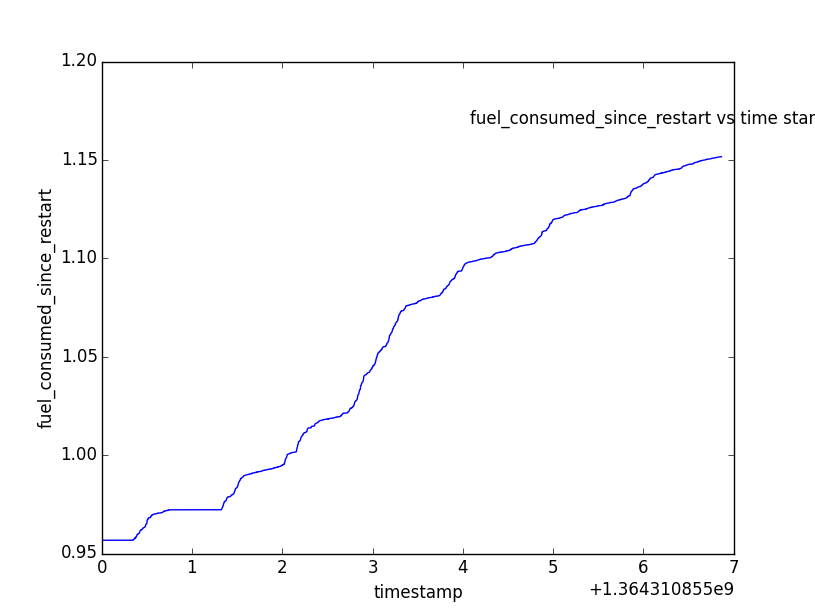


III-e) latitude vs time stamp

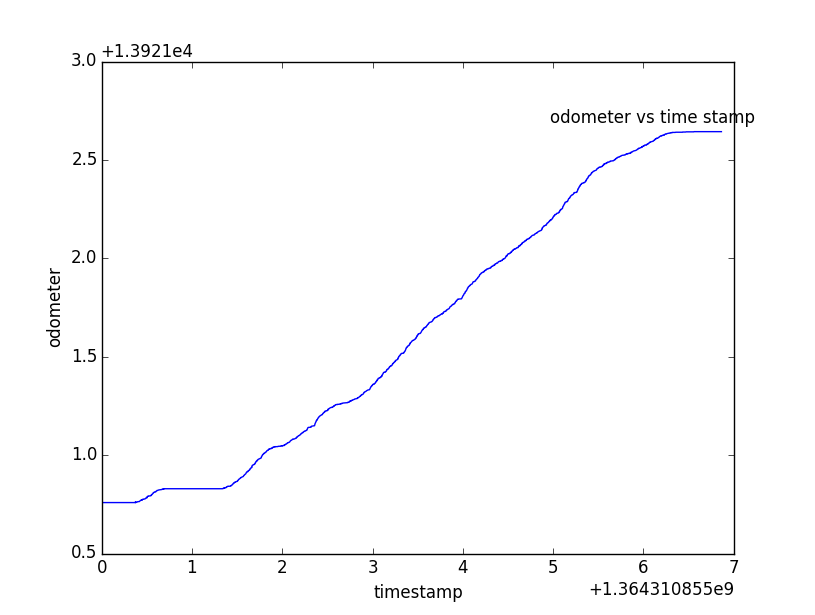


III-f) longitude vs time stamp

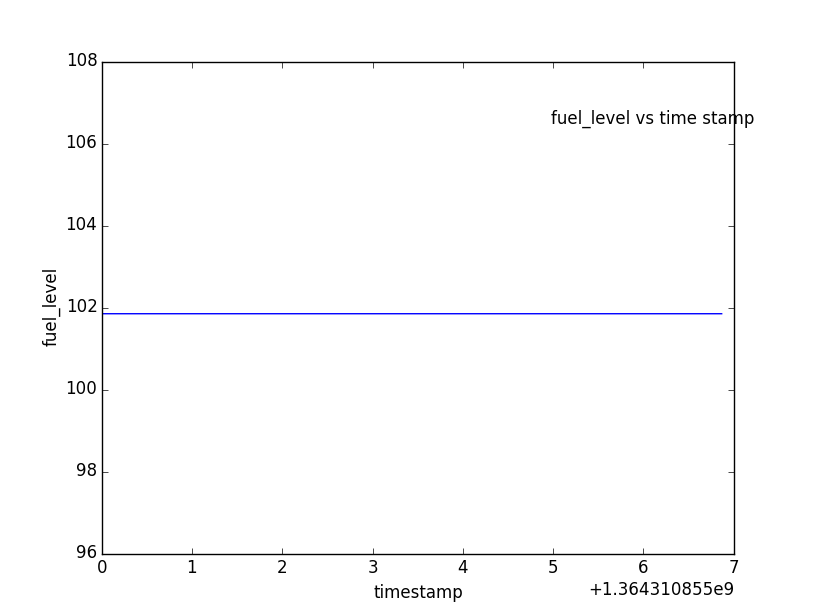
III-g) fuel consumed since restart vs time stamp



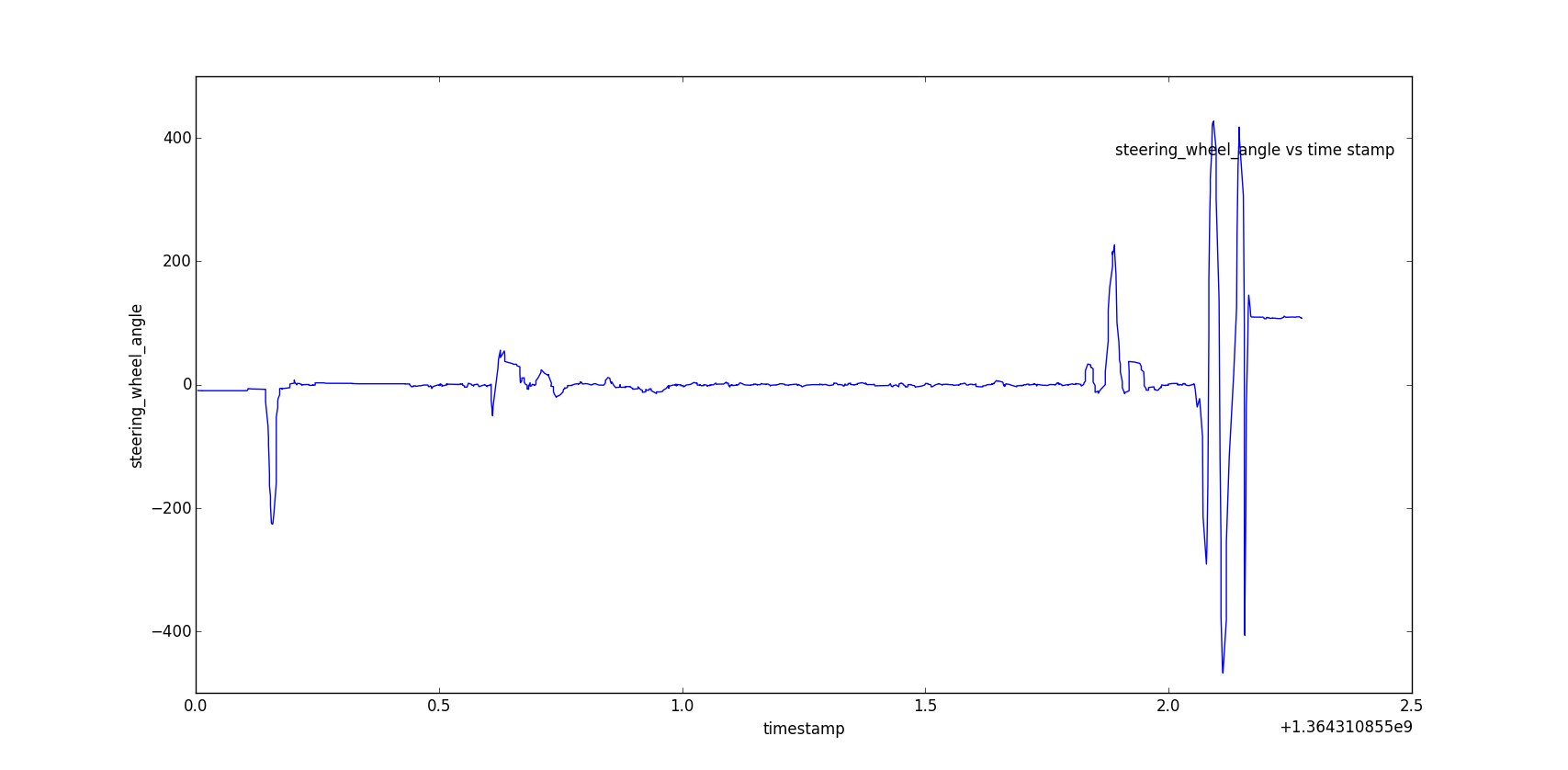
III-h) odometer reading vs time stamp



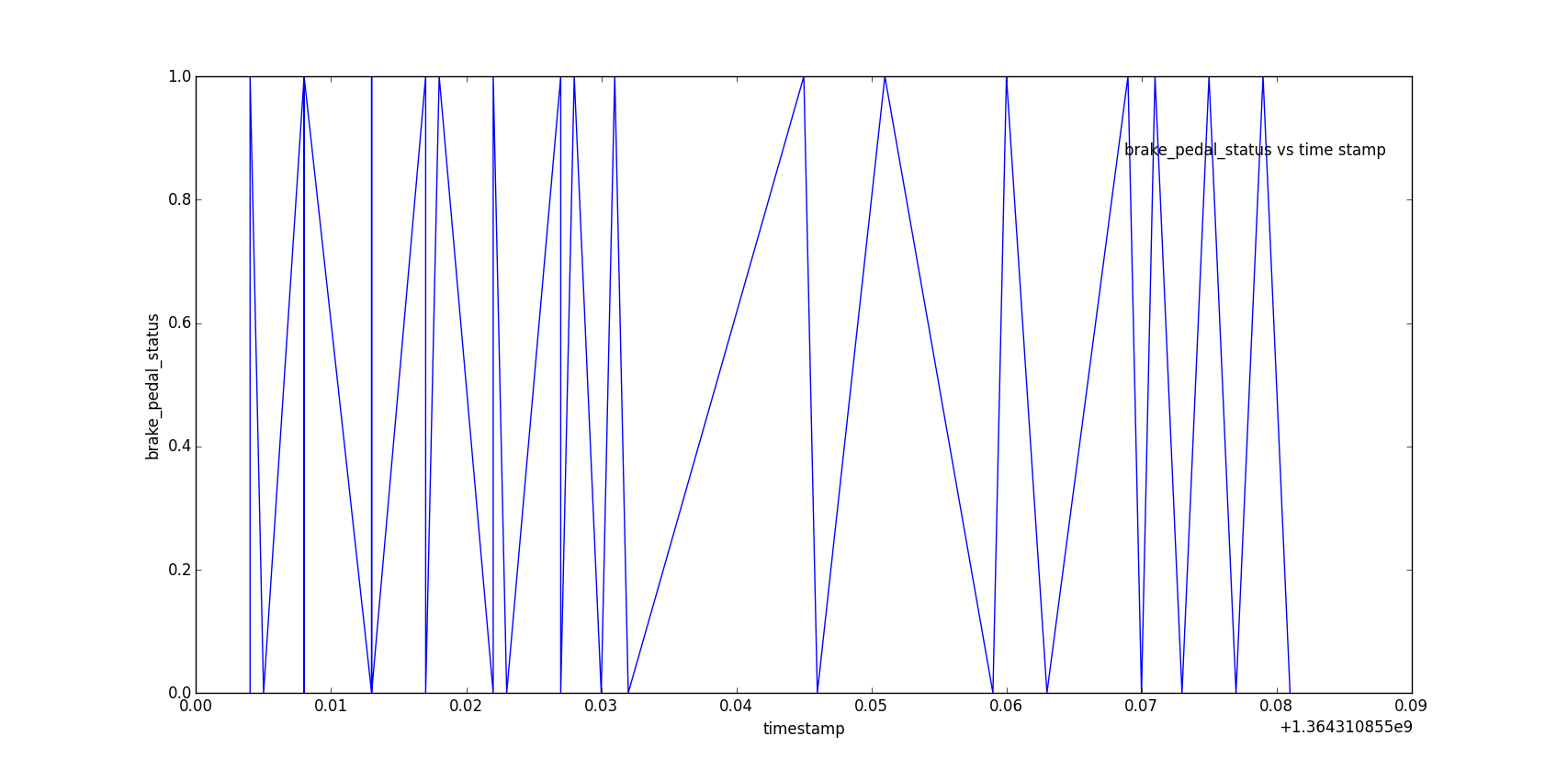
III-i) fuel level vs time stamp



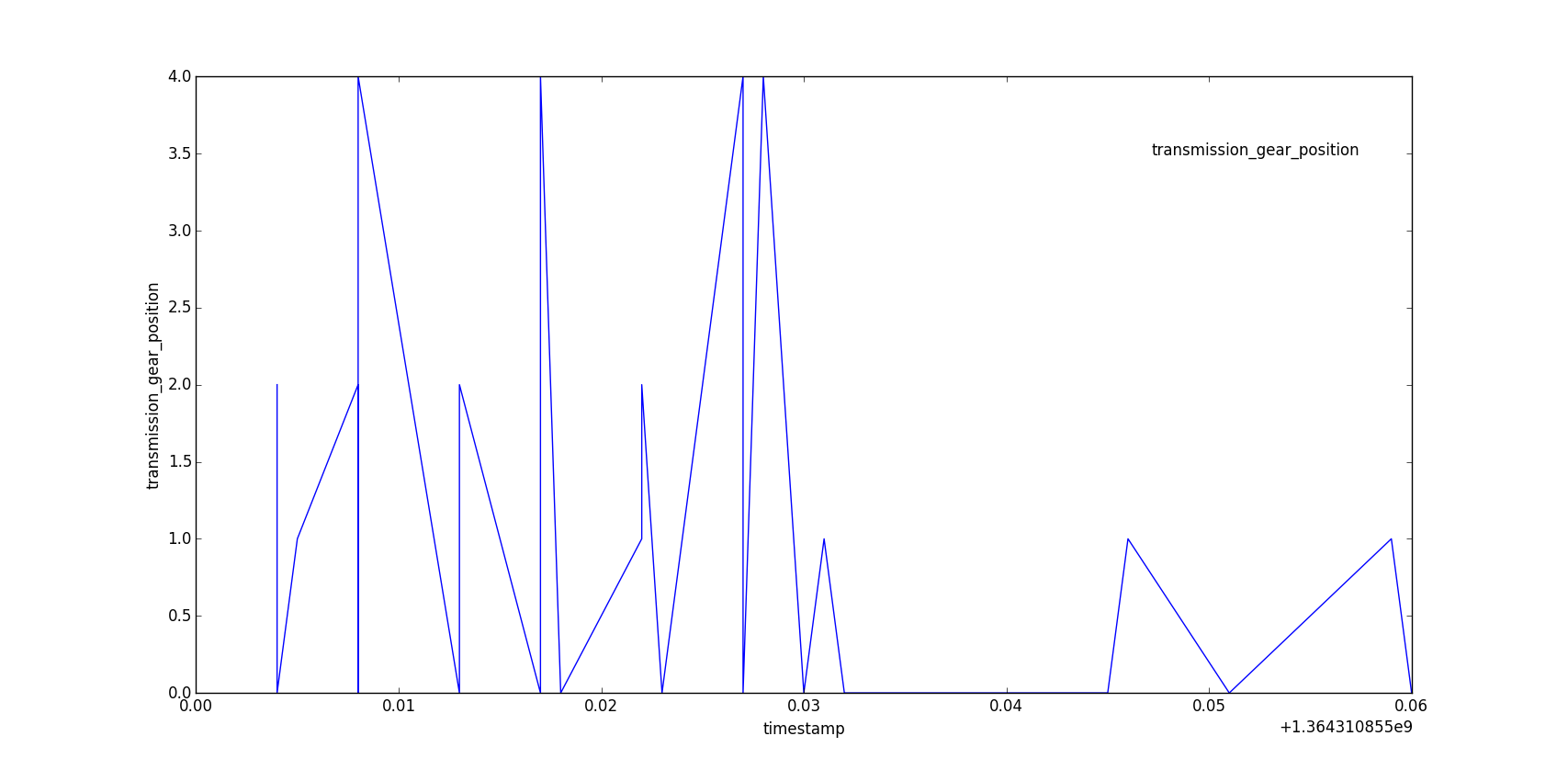
III-j) steering wheel angle vs time stamp



III-k) brake pedal status vs time stamp

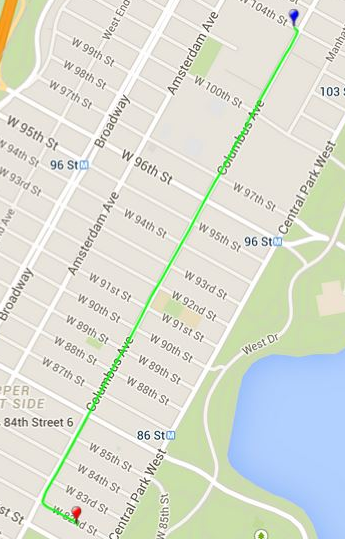


III-l) transmission gear position vs time stamp



func7():

III-m) Route of Alice’s vehicle according to given data



1. **Observations:**
2. The vehicle’s trip started at 104 ST W and ended at 82 ST W, according to the pygmap plot.

The vehicle stayed in idle mode, whenever its speed is zero is ~54 seconds, from the vehicle\_speed vs time\_stamp plot of func5().

The maximum speed of vehicle during the trip is 47.68 kilometer/hour and the average speed is 22.97 kilometer/hour. This trip took place in urban Manhattan/ New York metropolitan area.

1. The Safe driving parameters determined any motor vehicle authority/ insurance company are Speed, Cornering, Time, Acceleration & Braking. From the temporal plots obtained, it is clearly evident that Speed is moderate, braking is under control and 296(~5 minutes) to traverse 1.2 miles is very safe driving. Hence Alice’s driving can be considered as safe driving.