Tesla SuperCharger Network

April 17, 2016

```
In [42]: import geohash
         import pandas as pd
         import configparser
         import googlemaps
         import time
         import json
         import os
         import networkx as nx
         from networkx.readwrite import json_graph
         from bs4 import BeautifulSoup
         import requests
         import csv
         from math import radians, cos, sin, asin, sqrt
         from dateutil.parser import parse
```

```
In [43]: config = configparser.ConfigParser()
         config.read("config.ini")
         API_key = config['Keys']['google_API']
         gmaps = googlemaps.Client(key=API_key)
```

```
In [44]: def get_geohash_directions(gh_A,gh_B):
             GPS_A = geohash.decode(gh_A)
             GPS_B = geohash.decode(gh_B)
             directions_result = gmaps.directions(GPS_A,
                                                  GPS_B,
                                                  mode="driving")
             time.sleep(1)
             return ({'distance':directions_result[0]['legs'][0]['distance']['value'],
                      'steps':len(directions_result[0]['legs'][0]['steps'])})
```

```
In [45]: def load_pop_dict():
             with open('populations.json', 'r') as f:
                 p_dict = json.load(f)
             return p_dict
```

```
In [46]: GEOHASH_PRECISION = 2
         MAX_RANGE = 346 #Base Tesla Model 3 range (346 km/215 miles)
         POP_DICT = load_pop_dict()
         def build_connections(G,src_hash):
             print (nx.number_of_nodes(G))
             connections = {}
             node_hashes = ([node for node in G
                             if node[0:GEOHASH_PRECISION] in geohash.expand(src_hash[0:GEOHASH_PRECISION]
                             and node != src_hash])
             src_GPS = reverse_GPS(geohash.decode(src_hash))
             close_connections = ([{'node':node_gh,
                                    'directions':get_geohash_directions(src_hash,node_gh)} for node_gh
                                     if haversine(*src_GPS,*reverse_GPS(geohash.decode(node_gh))) <= MA
             for connection in close_connections:
                 if connection['directions']['distance']/1000 <= MAX_RANGE:
                     edge_weight = get_edge_weight(G,src_hash,connection['node'])
                     G.add_edge(src_hash,connection['node'],{'weight':edge_weight,'distance':connection
                                                             'steps':connection['directions']['steps'],
                                                             #gets the indx of last node to be added us
                                                             'first_node':str(min(int(G.node[src_hash][
                                                                                  int(G.node[connection['n
                                                             'second_node':str(max(int(G.node[src_hash]
                                                                                  int(G.node[connection['n
                                                             'lon_lat_1':reverse_GPS(geohash.decode(src_
                                                             'lon_lat_2':reverse_GPS(geohash.decode(conn
             return G

In [47]: def get_edge_weight(G,src_hash,connection_hash):
             try:
                 pop1 = get_close_population(src_hash)
                 pop2 = get_close_population(connection_hash)
                 return (pop1+pop2)/POP_DICT['total']
             except KeyError as e:
                 print(e)
                 return 0

In [48]: def reverse_GPS(GPS):
             return [GPS[1],GPS[0]]

In [49]: def build_network():
             G = load_network()
             df = pd.read_csv("Teslarati_SC_data.csv",
                              dtype={'Stalls': float,'Zip':str,'Tesla':str,'Elev':str})
             df["lat"], df["lon"] = zip(*df["GPS"].str.split(',').tolist())
             df["lat"], df["lon"] = df["lat"].astype(float), df["lon"].astype(float)
             df['GPS_lon_lat'] = df.apply(lambda x: [x["lon"],x["lat"]], axis=1)
             df['geohash'] = df.apply(lambda x: geohash.encode(x['lat'],x['lon']), axis=1)
             df['SC_data'] = df.apply(lambda x: parse_Tesla_SC_data(x['Tesla']), axis=1)
             df['population'] = df.apply(lambda x: get_close_population(x['geohash']), axis=1).astype(f
             df["Open Date"] = df.apply(lambda x: parse(x["Open Date"]), axis=1)#this is really hackey
             df.sort_values(["Open Date"],inplace=True)
             df["SC_index"] = range(1, len(df) + 1)
             df["SC_index"] = df["SC_index"].astype(str)
             df["Open Date"] = df.apply(lambda x: str(x["Open Date"]), axis=1)

             for i in df['geohash'].keys():
                 print (str(df["Open Date"][i]) + " " + df['SC_index'][i])
                 if df['geohash'][i] in G:
                     G.node[df['geohash'][i]] = {key:df[key][i] for key in df.keys()}
                 else:
                     G.add_node(df['geohash'][i],{key:df[key][i] for key in df.keys()})
                     build_connections(G,df['geohash'][i])

             network = json_graph.node_link_data(G)
             with open("network.json","w") as f:
                 json.dump(network,f)
             return G
```

```
In [51]: def load_network():
             if os.path.getsize("network.json") > 0:
                 with open("network.json","r") as f:
                     data = json.load(f)
                     G = json_graph.node_link_graph(data)
             else:
                 G=nx.Graph()
             return G
```

```
In [57]: def parse_Tesla_SC_data(URL):
             SC_data = {}
             r = requests.get(URL)
             soup = BeautifulSoup(r.text,"html.parser")
             attr_lists = soup.find_all('p')
             for attr in attr_lists:
                 if attr.find('strong'):
                     #probably not ideal, but only way I could get BS to parse 'br' correctly. Better w
                     SC_data[attr.find('strong').text] = [value for value in attr.childGenerator()
                                                          if value.name == None and value != ' ']
                     if 'Charging' in SC_data.keys():
                         SC_data['Chargers'] = [SC_data['Charging'][0][0]]
                     else:
                         SC_data['Chargers'] = [0]
             time.sleep(1)
             return SC_data
```

```
In [52]: def google_city_location(city):
             location = gmaps.geocode(city)
             time.sleep(.1)
             return location
```

```
In [53]: def build_pop_dict():
             total_pop = 0
             pop_gps_dict = {}
             for city,pop in POP_DICT.items():
                 try:
                     geocode_data = google_city_GPS(city)
                     city_location = geocode_data[0]['geometry']['location']
                     gh = geohash.encode(city_location['lat'],city_location['lng'])
                     pop_gps_dict[gh] = ({'city':city,
                                                 'population':pop,
                                                 'lat':city_location['lat'],
                                                 'lon':city_location['lng']})
                     total_pop = total_pop + pop
                 except IndexError as e:
                     print (geocode_data)
                     print (city + " not found")
             pop_gps_dict['total'] = total_pop
             with open("populations.json","w") as f:
                 json.dump(pop_gps_dict,f)
```

```
In [54]: def get_close_population(src_hash):#function uses geohash precision of 3 (ie radius of 73km) a
             total_close_pop = (sum([data['population'] for gh,data in POP_DICT.items()
                             if gh[0:3] in geohash.expand(src_hash[0:3])]))
             return total_close_pop
```

```
In [55]: def haversine(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])
             # haversine 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))
             km = 6367 * c
             return km
```

```
In [58]: build = build_network()
```

```
2012-11-19 00:00:00 1
1
2012-11-19 00:00:00 2
2
2012-11-19 00:00:00 3
3
2012-11-19 00:00:00 4
4
2012-11-19 00:00:00 5
5
2012-12-16 00:00:00 6
6
2012-12-16 00:00:00 7
7
2012-12-21 00:00:00 8
8
2013-06-19 00:00:00 9
9
2013-06-19 00:00:00 10
10
2013-06-26 00:00:00 11
11
2013-07-03 00:00:00 12
12
2013-07-13 00:00:00 13
13
2013-07-13 00:00:00 14
14
2013-07-17 00:00:00 15
15
2013-07-24 00:00:00 16
16
2013-07-25 00:00:00 17
```

```
17
2013-08-16 00:00:00 18
18
2013-08-20 00:00:00 19
19
2013-08-28 00:00:00 20
20
2013-09-07 00:00:00 21
21
2013-09-12 00:00:00 22
22
2013-09-13 00:00:00 23
23
2013-09-17 00:00:00 24
24
2013-10-03 00:00:00 25
25
2013-10-07 00:00:00 26
26
2013-10-16 00:00:00 27
27
2013-10-18 00:00:00 28
28
2013-10-22 00:00:00 29
29
2013-10-23 00:00:00 30
30
2013-10-24 00:00:00 31
31
2013-11-02 00:00:00 32
32
2013-11-08 00:00:00 33
33
2013-11-14 00:00:00 34
34
2013-11-15 00:00:00 35
35
2013-11-19 00:00:00 36
36
2013-11-20 00:00:00 37
37
2013-11-25 00:00:00 38
38
2013-11-26 00:00:00 39
39
2013-12-09 00:00:00 40
40
2013-12-09 00:00:00 41
41
2013-12-11 00:00:00 42
42
2013-12-11 00:00:00 43
43
2013-12-18 00:00:00 44
```

```
44
2013-12-19 00:00:00 45
45
2013-12-19 00:00:00 46
46
2013-12-20 00:00:00 47
47
2013-12-23 00:00:00 48
48
2013-12-23 00:00:00 49
49
2013-12-23 00:00:00 50
50
2013-12-27 00:00:00 51
51
2013-12-31 00:00:00 52
52
2014-01-03 00:00:00 53
53
2014-01-05 00:00:00 54
54
2014-01-06 00:00:00 55
55
2014-01-07 00:00:00 56
56
2014-01-10 00:00:00 57
57
2014-01-10 00:00:00 58
58
2014-01-10 00:00:00 59
59
2014-01-10 00:00:00 60
60
2014-01-13 00:00:00 61
61
2014-01-13 00:00:00 62
62
2014-01-13 00:00:00 63
63
2014-01-15 00:00:00 64
64
2014-01-15 00:00:00 65
65
2014-01-15 00:00:00 66
66
2014-01-15 00:00:00 67
67
2014-01-17 00:00:00 68
68
2014-01-18 00:00:00 69
69
2014-01-20 00:00:00 70
70
2014-01-22 00:00:00 71
```

```
71
2014-01-22 00:00:00 72
72
2014-01-23 00:00:00 73
73
2014-01-30 00:00:00 74
74
2014-01-30 00:00:00 75
75
2014-02-04 00:00:00 76
76
2014-02-12 00:00:00 77
77
2014-02-12 00:00:00 78
78
2014-02-13 00:00:00 79
79
2014-02-19 00:00:00 80
80
2014-02-24 00:00:00 81
81
2014-02-26 00:00:00 82
82
2014-03-12 00:00:00 83
83
2014-03-17 00:00:00 84
84
2014-03-20 00:00:00 85
85
2014-04-04 00:00:00 86
86
2014-04-04 00:00:00 87
87
2014-04-14 00:00:00 88
88
2014-04-22 00:00:00 89
89
2014-04-29 00:00:00 90
90
2014-04-30 00:00:00 91
91
2014-05-02 00:00:00 92
92
2014-05-06 00:00:00 93
93
2014-05-07 00:00:00 94
94
2014-05-08 00:00:00 95
95
2014-05-17 00:00:00 96
96
2014-05-18 00:00:00 97
97
2014-06-11 00:00:00 98
```

```
98
2014-06-12 00:00:00 99
99
2014-06-13 00:00:00 100
100
2014-06-21 00:00:00 101
101
2014-07-04 00:00:00 102
102
2014-07-10 00:00:00 103
103
2014-07-11 00:00:00 104
104
2014-07-12 00:00:00 105
105
2014-07-23 00:00:00 106
106
2014-08-01 00:00:00 107
107
2014-08-07 00:00:00 108
108
2014-08-14 00:00:00 109
109
2014-08-20 00:00:00 110
110
2014-08-22 00:00:00 111
111
2014-08-26 00:00:00 112
112
2014-08-26 00:00:00 113
113
2014-08-28 00:00:00 114
114
2014-09-04 00:00:00 115
115
2014-09-09 00:00:00 116
116
2014-09-24 00:00:00 117
117
2014-09-24 00:00:00 118
118
2014-09-26 00:00:00 119
119
2014-09-30 00:00:00 120
120
2014-10-10 00:00:00 121
121
2014-10-14 00:00:00 122
122
2014-10-17 00:00:00 123
123
2014-10-22 00:00:00 124
124
2014-10-22 00:00:00 125
```

```
125
2014-10-27 00:00:00 126
126
2014-10-29 00:00:00 127
127
2014-10-29 00:00:00 128
128
2014-11-03 00:00:00 129
129
2014-11-08 00:00:00 130
130
2014-11-08 00:00:00 131
131
2014-11-08 00:00:00 132
132
2014-11-11 00:00:00 133
133
2014-11-20 00:00:00 134
134
2014-11-20 00:00:00 135
135
2014-11-21 00:00:00 136
136
2014-11-26 00:00:00 137
137
2014-11-26 00:00:00 138
138
2014-11-26 00:00:00 139
139
2014-12-04 00:00:00 140
140
2014-12-05 00:00:00 141
141
2014-12-05 00:00:00 142
142
2014-12-06 00:00:00 143
143
2014-12-09 00:00:00 144
144
2014-12-17 00:00:00 145
145
2014-12-17 00:00:00 146
146
2014-12-18 00:00:00 147
147
2014-12-18 00:00:00 148
148
2014-12-21 00:00:00 149
149
2014-12-23 00:00:00 150
150
2014-12-23 00:00:00 151
151
2014-12-23 00:00:00 152
```

```
152
2014-12-28 00:00:00 153
153
2014-12-29 00:00:00 154
154
2014-12-30 00:00:00 155
155
2015-01-09 00:00:00 156
156
2015-01-09 00:00:00 157
157
2015-01-15 00:00:00 158
158
2015-01-16 00:00:00 159
159
2015-01-16 00:00:00 160
160
2015-01-16 00:00:00 161
161
2015-01-20 00:00:00 162
162
2015-01-20 00:00:00 163
163
2015-01-21 00:00:00 164
164
2015-01-22 00:00:00 165
165
2015-01-22 00:00:00 166
166
2015-01-22 00:00:00 167
167
2015-01-22 00:00:00 168
168
2015-01-28 00:00:00 169
169
2015-01-28 00:00:00 170
170
2015-01-29 00:00:00 171
171
2015-01-29 00:00:00 172
172
2015-01-30 00:00:00 173
173
2015-02-03 00:00:00 174
174
2015-02-03 00:00:00 175
175
2015-02-04 00:00:00 176
176
2015-02-04 00:00:00 177
177
2015-02-11 00:00:00 178
178
2015-02-12 00:00:00 179
```

```
179
2015-02-14 00:00:00 180
180
2015-02-17 00:00:00 181
181
2015-02-18 00:00:00 182
182
2015-02-20 00:00:00 183
183
2015-02-20 00:00:00 184
184
2015-02-25 00:00:00 185
185
2015-02-25 00:00:00 186
186
2015-03-04 00:00:00 187
187
2015-03-06 00:00:00 188
188
2015-03-11 00:00:00 189
189
2015-03-11 00:00:00 190
190
2015-03-11 00:00:00 191
191
2015-03-13 00:00:00 192
192
2015-03-13 00:00:00 193
193
2015-03-20 00:00:00 194
194
2015-03-21 00:00:00 195
195
2015-03-25 00:00:00 196
196
2015-03-27 00:00:00 197
197
2015-03-28 00:00:00 198
198
2015-04-01 00:00:00 199
199
2015-04-08 00:00:00 200
200
2015-04-15 00:00:00 201
201
2015-04-15 00:00:00 202
202
2015-04-16 00:00:00 203
203
2015-04-16 00:00:00 204
204
2015-04-21 00:00:00 205
205
2015-05-02 00:00:00 206
```

```
206
2015-05-09 00:00:00 207
207
2015-05-27 00:00:00 208
208
2015-05-28 00:00:00 209
209
2015-06-05 00:00:00 210
210
2015-06-08 00:00:00 211
211
2015-06-10 00:00:00 212
212
2015-06-11 00:00:00 213
213
2015-06-19 00:00:00 214
214
2015-06-23 00:00:00 215
215
2015-06-25 00:00:00 216
216
2015-06-25 00:00:00 217
217
2015-07-08 00:00:00 218
218
2015-07-09 00:00:00 219
219
2015-07-09 00:00:00 220
220
2015-07-10 00:00:00 221
221
2015-07-15 00:00:00 222
222
2015-07-17 00:00:00 223
223
2015-07-17 00:00:00 224
224
2015-07-30 00:00:00 225
225
2015-07-31 00:00:00 226
226
2015-08-07 00:00:00 227
227
2015-08-08 00:00:00 228
228
2015-08-13 00:00:00 229
229
2015-08-20 00:00:00 230
230
2015-08-28 00:00:00 231
231
2015-09-02 00:00:00 232
232
2015-09-04 00:00:00 233
```

```
233
2015-09-04 00:00:00 234
234
2015-09-11 00:00:00 235
235
2015-09-14 00:00:00 236
236
2015-09-19 00:00:00 237
237
2015-09-23 00:00:00 238
238
2015-09-24 00:00:00 239
239
2015-10-07 00:00:00 240
240
2015-10-09 00:00:00 241
241
2015-10-16 00:00:00 242
242
2015-10-17 00:00:00 243
243
2015-10-18 00:00:00 244
244
2015-10-21 00:00:00 245
245
2015-10-21 00:00:00 246
246
2015-10-22 00:00:00 247
247
2015-10-23 00:00:00 248
248
2015-10-28 00:00:00 249
249
2015-11-01 00:00:00 250
250
2015-11-04 00:00:00 251
251
2015-11-04 00:00:00 252
252
2015-11-04 00:00:00 253
253
2015-11-05 00:00:00 254
254
2015-11-09 00:00:00 255
255
2015-11-18 00:00:00 256
256
2015-12-04 00:00:00 257
257
2015-12-14 00:00:00 258
258
2015-12-15 00:00:00 259
259
2015-12-16 00:00:00 260
```

```
260
2015-12-16 00:00:00 261
261
2015-12-19 00:00:00 262
262
2015-12-20 00:00:00 263
263
2015-12-23 00:00:00 264
264
2015-12-29 00:00:00 265
265
2015-12-30 00:00:00 266
266
2015-12-30 00:00:00 267
267
2015-12-30 00:00:00 268
268
2015-12-30 00:00:00 269
269
2016-01-13 00:00:00 270
270
2016-01-21 00:00:00 271
271
2016-01-21 00:00:00 272
272
2016-01-29 00:00:00 273
273
2016-01-29 00:00:00 274
274
2016-02-06 00:00:00 275
275
2016-02-09 00:00:00 276
276
2016-03-11 00:00:00 277
277
2016-03-23 00:00:00 278
278
2016-03-30 00:00:00 279
279
2016-03-31 00:00:00 280
280
```