using-API-data

May 14, 2018

1 Investigating crashes using API data

A number of factors can increase the chances of a car crash occuring - for instance: weather, drink-driving or time of day. In this in project, I set out to collect data from the web in order to understand the circumstances surrounding car crashes in the state of Maryland, U.S. The crash data was taken from the U.S. website: data.gov.

1. First of all, the Foursquare API is used to collect data about the number of bars within in a certain radius in each County, in order to see if the prevalence of drinking might have an influence on the accident rate.
2. Second of all, the Google Maps API is used to determine the coordinates of each county, then the coordinates are used to request the weather conditions at the time of the accident from the DarkSky API.
3. Finally, the coordinates of the accidents themselves are obtained and plotted on a map using the Google Map Plotter.

1.1 Data

The data for this exercise can be found here.

Just run the cells below to get the data ready.

```
In [1]: import pandas as pd
        mypath = "./"

In [2]: data = pd.read_csv(mypath + "2012_Vehicle_Collisions_Investigated_by_State_Police.csv"
                           parse_dates=[["ACC_DATE", "ACC_TIME"]])
        data["MONTH"] = data.ACC_DATE_ACC_TIME.dt.month
        data.dropna(subset=["COUNTY_NAME"], inplace=True) #get rid of empty counties
```

Now let's check the length of the data and the names of the counties and the dataset itself.

```
In [3]: len(data)

Out[3]: 18604

In [4]: data.COUNTY_NAME.unique()
```

```
Out[4]: array(['Montgomery', 'Worcester', 'Calvert', 'St. Marys', 'Baltimore',
               'Prince Georges', 'Anne Arundel', 'Cecil', 'Charles', 'Carroll',
               'Harford', 'Frederick', 'Howard', 'Allegany', 'Garrett', 'Kent',
               'Queen Annes', 'Washington', 'Somerset', 'Wicomico', 'Talbot',
               'Caroline', 'Dorchester', 'Not Applicable', 'Unknown',
               'Baltimore City'], dtype=object)

In [7]: data.head()

Out[7]:     ACC_DATE_ACC_TIME  CASE_NUMBER           BARRACK  ACC_TIME_CODE  \
        0 2012-01-01 02:01:00   1363000002         Rockville              1
        1 2012-01-01 18:01:00   1296000023            Berlin              5
        2 2012-01-01 07:01:00   1283000016  Prince Frederick              2
        3 2012-01-01 00:01:00   1282000006       Leonardtown              1
        4 2012-01-01 01:01:00   1267000007             Essex              1

          DAY_OF_WEEK                       ROAD                INTERSECT_ROAD  \
        0   SUNDAY      IS 00495 CAPITAL BELTWAY  IS 00270 EISENHOWER MEMORIAL
        1   SUNDAY     MD 00090 OCEAN CITY EXPWY   CO 00220 ST MARTINS NECK RD
        2   SUNDAY             MD 00765 MAIN ST              CO 00208 DUKE ST
        3   SUNDAY      MD 00944 MERVELL DEAN RD       MD 00235 THREE NOTCH RD
        4   SUNDAY        IS 00695 BALTO BELTWAY     IS 00083 HARRISBURG EXPWY

           DIST_FROM_INTERSECT DIST_DIRECTION       CITY_NAME  COUNTY_CODE  \
        0                 0.00              U  Not Applicable         15.0
        1                 0.25              W  Not Applicable         23.0
        2               100.00              S  Not Applicable          4.0
        3                10.00              E  Not Applicable         18.0
        4               100.00              S  Not Applicable          3.0

          COUNTY_NAME  VEHICLE_COUNT PROP_DEST INJURY COLLISION_WITH_1  \
        0  Montgomery            2.0       YES     NO              VEH
        1   Worcester            1.0       YES     NO        FIXED OBJ
        2     Calvert            1.0       YES     NO        FIXED OBJ
        3   St. Marys            1.0       YES     NO        FIXED OBJ
        4   Baltimore            2.0       YES     NO              VEH

          COLLISION_WITH_2  MONTH
        0  OTHER-COLLISION      1
        1  OTHER-COLLISION      1
        2        FIXED OBJ      1
        3  OTHER-COLLISION      1
        4  OTHER-COLLISION      1
```

1.2 Google Maps API

Now we will use the Google Maps API.

```
In [131]: from googlemaps.exceptions import TransportError
          import googlemaps
          from googlemaps.exceptions import HTTPError

In [139]: gmaps = googlemaps.Client(key=os.environ["GOOGLE_API_BASECAMP_KEY"])
```

This function is used to get the official latitude and longitude for each county, by making calls to the API.

```
In [28]: county_names = list(set(data.COUNTY_NAME.unique()))
```

1.3 Foursquare API

Foursquare API documentation is here

1. Start a foursquare application and get your keys.
2. For each crash, pull number of of bars (category "Nightlife") in 5km radius.
3. Find a relationship between number of bars in the area and severity of the crash.
4. (optional) Try to come up with other approaches to get more information out of the data.
5. (optional) Think about the most generic way to approach the problem.

Hints:

- check out python package "foursquare"
- what happens if the code fails?
- what if you run out of requests? (check out time package)

```
In [30]: #set the keys
         foursquare_id = 'ONQCHENU5SVZ2HOIHUCXKSW3Y55VJPDJ3FYL4VPHG5MINZPV'
         foursquare_secret = "L5QWH4QUKONTE3LPFVTAZIV43GHINZOC4ZZ3GVPXAFUWEI3U"

In [31]: #install and load the library
         from foursquare import Foursquare
```

Now we need to set up the client in order to make calls to the API.

```
In [32]: client = Foursquare(client_id = foursquare_id,
                            client_secret = foursquare_secret)
```

We will loop through the counties and obtain the number of bars within a 5km radius (up to a maximum of 50 bars). If the call quota is exceeded, then the code the operation will be paused for one hour to allow it to reset.

```
In [34]: number_of_bars = {}
         for county in county_names:
             try:
                 response = client.venues.search({'near': county,
                                                  'limit': 50,
                                                  'intent': 'browse',
                                                 'radius': 5000,
```

```
                                                'units': 'si',
                                                'categoryId': '4d4b7105d754a06376d81259'})
                    number_of_bars[county] = len(response['venues'])
                except Exception as e:
                    print (e)
                    if e == "Quota exceeded":
                        print ("exceeded quota: waiting for an hour")
                        time.sleep(3600)
                    number_of_bars[county] = -1
```

```
Couldn't geocode param near: Not Applicable
```

```
In [35]: number_of_bars
```

```
Out[35]: {'Allegany': 30,
          'Anne Arundel': 49,
          'Baltimore': 50,
          'Baltimore City': 50,
          'Calvert': 0,
          'Caroline': 6,
          'Carroll': 22,
          'Cecil': 28,
          'Charles': 13,
          'Dorchester': 22,
          'Frederick': 35,
          'Garrett': 12,
          'Harford': 2,
          'Howard': 9,
          'Kent': 19,
          'Montgomery': 50,
          'Not Applicable': -1,
          'Prince Georges': 29,
          'Queen Annes': 8,
          'Somerset': 41,
          'St. Marys': 13,
          'Talbot': 3,
          'Unknown': 0,
          'Washington': 50,
          'Wicomico': 50,
          'Worcester': 36}
```

```
In [38]: # Adding a new column for the number of bars
         number_of_bars = pd.DataFrame({'county': list(number_of_bars.keys()), 'num_bars': list
         data_df = pd.merge(data, number_of_bars, left_on='COUNTY_NAME', right_on='county', how
         data_df.drop(columns=['county'], inplace=True)
```

We need to select a target variable. I will choose the 'INJURY' column, because this is a straightforward way to judge the severity of the crash.

```
In [ ]: # Converting injuries to a binary mapping to judge severity
        data_df['severity'] = data_df['INJURY'].map({'YES':1, 'NO':0})

In [63]: data_df.head()
```

Out[63]:

	ACC_DATE_ACC_TIME	CASE_NUMBER	BARRACK	ACC_TIME_CODE
0	2012-01-01 02:01:00	1363000002	Rockville	1
1	2012-01-01 18:01:00	1296000023	Berlin	5
2	2012-01-01 07:01:00	1283000016	Prince Frederick	2
3	2012-01-01 00:01:00	1282000006	Leonardtown	1
4	2012-01-01 01:01:00	1267000007	Essex	1

	DAY_OF_WEEK	ROAD	INTERSECT_ROAD
0	SUNDAY	IS 00495 CAPITAL BELTWAY	IS 00270 EISENHOWER MEMORIAL
1	SUNDAY	MD 00090 OCEAN CITY EXPWY	CO 00220 ST MARTINS NECK RD
2	SUNDAY	MD 00765 MAIN ST	CO 00208 DUKE ST
3	SUNDAY	MD 00944 MERVELL DEAN RD	MD 00235 THREE NOTCH RD
4	SUNDAY	IS 00695 BALTO BELTWAY	IS 00083 HARRISBURG EXPWY

	DIST_FROM_INTERSECT	DIST_DIRECTION	CITY_NAME	COUNTY_CODE
0	0.00	U	Not Applicable	15.0
1	0.25	W	Not Applicable	23.0
2	100.00	S	Not Applicable	4.0
3	10.00	E	Not Applicable	18.0
4	100.00	S	Not Applicable	3.0

	COUNTY_NAME	VEHICLE_COUNT	PROP_DEST	INJURY	COLLISION_WITH_1
0	Montgomery	2.0	YES	NO	VEH
1	Worcester	1.0	YES	NO	FIXED OBJ
2	Calvert	1.0	YES	NO	FIXED OBJ
3	St. Marys	1.0	YES	NO	FIXED OBJ
4	Baltimore	2.0	YES	NO	VEH

	COLLISION_WITH_2	MONTH	num_bars	severity
0	OTHER-COLLISION	1	50	0
1	OTHER-COLLISION	1	36	0
2	FIXED OBJ	1	0	0
3	OTHER-COLLISION	1	13	0
4	OTHER-COLLISION	1	50	0

I will now create a new dataframe for my features and encode each feature into categoric variables.

```
In [108]: from sklearn.preprocessing import LabelEncoder
          le = LabelEncoder()
          # crash severity
          feature_df = pd.DataFrame()
          feature_df['id'] = data_df['CASE_NUMBER']
          feature_df['Time'] = data_df['ACC_TIME_CODE']
```

```
feature_df['Day'] = le.fit_transform(data_df['DAY_OF_WEEK'])
feature_df['Vehicles'] = data_df['VEHICLE_COUNT'].fillna(0)
feature_df['One hit'] = le.fit_transform(data_df['COLLISION_WITH_1'])
feature_df['Tws hits'] = le.fit_transform(data_df['COLLISION_WITH_2'])
feature_df['Bars'] = data_df['num_bars']

feature_df.head()
```

Out[108]:

	id	Time	Day	Vehicles	One hit	Tws hits	Bars
0	1363000002	1	3	2.0	6	4	50
1	1296000023	5	3	1.0	2	4	36
2	1283000016	2	3	1.0	2	2	0
3	1282000006	1	3	1.0	2	4	13
4	1267000007	1	3	2.0	6	4	50

```
In [ ]: # Let's make sure that there are no missing values because they will cause the algorit
        feature_df.isna().sum()
```

I will now use the Scikit-learn random forest classifier to fit a model and use that model to determine the importance of each feature.

```
In [110]: from sklearn.ensemble import RandomForestClassifier
          # Sets up a classifier and fits a model to all features of the dataset
          clf = RandomForestClassifier(n_estimators=150, max_depth=8, min_samples_leaf=4, max_
          clf.fit(feature_df.drop(['id'],axis=1), data_df['severity'])
          # We need a list of features as well
          features = feature_df.drop(['id'],axis=1).columns.values
          print("--- COMPLETE ---")
```

```
--- COMPLETE ---
```

Using the following code from Anisotropic's kernal (https://www.kaggle.com/arthurtok/interactive-porto-insights-a-plot-ly-tutorial), we can use Plotly to create a nice horizontal bar chart for visualising the ranking of the most important features for determing the severity of crash.

```
In [ ]: import plotly.offline as py
        py.init_notebook_mode(connected=True)
        import plotly.graph_objs as go
        import plotly.tools as tls

        x, y = (list(x) for x in zip(*sorted(zip(clf.feature_importances_, features),
                                                                    reverse = False)))
        trace2 = go.Bar(
            x=x ,
            y=y,
            marker=dict(
                color=x,
                colorscale = 'Viridis',
```

```
            reversescale = True
        ),
        name='Random Forest Feature importance',
        orientation='v',
    )

    layout = dict(
        title='Ranking of most influential features',
         width = 900, height = 1500,
        yaxis=dict(
            showgrid=False,
            showline=False,
            showticklabels=True,
        ))

    fig1 = go.Figure(data=[trace2])
    fig1['layout'].update(layout)
    py.iplot(fig1, filename='plots')
```

1.4 DarkSky API

DarkSky API documentation is here

1. Sign up for FREE api key.
2. For each crush, get the weather for the location and time.
3. Find a relationship between the weather and severity of the crash.

Hints:

- There is an API limit (perhaps 1000 calls)
- use "Time Machine" request in DarkSky API
- for sending HTTP requests check out "requests" library here

The time needs to be converted to unix in order for the DarkSky API to recognise it.

```
In [157]: def time_to_unix(t):
              return time.mktime(t.timetuple())

          # changing to unix time
          data_sample_df = data_df.sample(n=1000, random_state=0)
          data_sample_df['UNIX_TIME'] = data_sample_df['ACC_DATE_ACC_TIME'].apply(time_to_unix)
```

We now need to get the latitude and longitude for each county in order for the API to provide local weather for each crash.

```
In [158]: def get_lat_lng(data, state, place_col):
              places = list(set(data[place_col].unique()))

              geo_dict = {'place':[], 'lat':[], 'lng':[]}
```

```
        for i in range(len(places)):
            place = places[i] + ', ' + state
            try:
                lat = gmaps.geocode(place)[0]['geometry']['location']['lat']
                lng = gmaps.geocode(place)[0]['geometry']['location']['lng']
                geo_dict['place'].append(places[i])
                geo_dict['lat'].append(lat)
                geo_dict['lng'].append(lng)
            except:
                geo_dict['place'].append(None)
                geo_dict['lat'].append(None)
                geo_dict['lng'].append(None)

        geo_df = pd.DataFrame(geo_dict)
        return pd.merge(data, geo_df, left_on='COUNTY_NAME', right_on='place', how='left

    data_sample_geo_df = get_lat_lng(data_sample_df, 'Maryland', 'COUNTY_NAME')
```

In [161]:
```
# Once you have signed up on DarkSky.net, you will be given an API key, which you ne
api_key = "[YOUR API KEY HERE]"
```

The next step is to make requests to the API for each of the crash instances in our sample. I am simply appending each entry into a dictionary with the place, coordinates, time and returned results.

In [177]:
```
import requests
import time

weather_data = {'place': [], 'lat': [], 'lng': [], 'time': [], 'result': []}

for crash in data_sample_geo_df.iterrows():
    place = crash[1]['COUNTY_NAME']
    lat = crash[1]['lat']
    lng = crash[1]['lng']
    t = crash[1]['UNIX_TIME']
    # https://api.darksky.net/forecast/[key]/[latitude],[longitude],[time]
    request = 'https://api.darksky.net/forecast/' + api_key + '/' + str(lat) + ',' +

    try:
        result = requests.get(request).content

        weather_data['place'].append(place)
        weather_data['lat'].append(lat)
        weather_data['lng'].append(lng)
        weather_data['time'].append(t)
        weather_data['result'].append(result)
    except Exception as e:
        print(e)
```

```
                weather_data['place'].append('')
                weather_data['lat'].append('')
                weather_data['lng'].append('')
                weather_data['time'].append('')
                weather_data['result'].append('')
```

```
In [191]: weather_df = pd.concat([data_sample_geo_df[['CASE_NUMBER']], pd.DataFrame(weather_dat
          # I like to save results like these in a CSV file, because there is a limit on the n
          weather_df.to_csv('weather-data.csv')
```

After going through one of the JSON files that was returned to me, I found the section that I want ("currently"). It contains the overall weather conditions, precipitation type (if any) and most importantly, the chance of rain. If the chance is greater than 50%, then I am satisfied that if was raining at the time for the sake of this exercise.

```
In [219]: print(d['currently'])
```

```
{'time': 1351069800, 'summary': 'Clear', 'icon': 'clear-night', 'precipIntensity': 0, 'precipP
```

```
In [269]: import json

          def extract_data(x):
              try:
                  d = json.loads(x)
                  res = d['currently']['precipProbability']
              except Exception as e:
                  print (e)
                  res = ''
              return res

          weather_df['precipProb'] = weather_df['result'].apply(extract_data)
```

```
Expecting value: line 1 column 1 (char 0)
```

```
In [289]: df_final = data_sample_df.merge(weather_df, left_on='CASE_NUMBER', right_on='CASE_NU
```

```
In [301]: df_final[['severity', 'precipProb']].corr()
```

```
Out[301]:           severity
          severity       1.0
```

```
In [351]: data_sample_df.head()
```

```
Out[351]:         ACC_DATE_ACC_TIME  CASE_NUMBER           BARRACK  ACC_TIME_CODE  \
          14950 2012-10-24 11:10:00   1251037421         Frederick              3
          11152 2012-08-13 20:08:00   1251029176         Frederick              6
          13808 2012-10-05 17:10:00   1280008862       Centreville              5
          11261 2012-08-16 05:08:00   1294006103           McHenry              2
```

```
875   2012-01-20 22:01:00   1283001115  Prince Frederick             6

      DAY_OF_WEEK                          ROAD           INTERSECT_ROAD  \
14950   WEDNESDAY       IS 00270 EISEN MEM HWY  MD 00080 FINGERBOARD RD
11152   MONDAY             MD 00026 Liberty Rd   MU 00998 Monocacy Blvd
13808   FRIDAY     US 00301 BLUE STAR MEMORIAL   CO 00151 JOHN BROWN RD
11261   THURSDAY        CO 00172 BUMBLE BEE RD        CO 00173 SPEAR RD
875     FRIDAY               CO 00058 GRAYS RD        MD 00506 SIXES RD

       DIST_FROM_INTERSECT DIST_DIRECTION       CITY_NAME  COUNTY_CODE  \
14950                500.0              N  Not Applicable         10.0
11152                  0.0              U  Not Applicable         10.0
13808                 60.0              S  Not Applicable         17.0
11261                300.0              W  Not Applicable         11.0
875                    1.0              E  Not Applicable          4.0

       COUNTY_NAME  VEHICLE_COUNT PROP_DEST INJURY COLLISION_WITH_1  \
14950    Frederick            2.0        NO    YES              VEH
11152    Frederick            2.0       YES     NO              VEH
13808  Queen Annes            2.0       YES     NO              VEH
11261      Garrett            1.0        NO    YES  OTHER-COLLISION
875        Calvert            1.0       YES     NO        FIXED OBJ

      COLLISION_WITH_2  MONTH   UNIX_TIME  severity
14950  OTHER-COLLISION     10  1351069800         1
11152  OTHER-COLLISION      8  1344881280         0
13808              VEH     10  1349449800         0
11261        FIXED OBJ      8  1345086480         1
875      NON-COLLISION      1  1327093260         0
```

```
In [364]: data_sample_geo_df['crash_lat'] = gmaps.geocode(data_sample_geo_df['ROAD'].values[0]


          gmaps.geocode(data_sample_geo_df['ROAD'] + ' ' + data_sample_geo_df['INTERSECT_ROAD']

In [370]: df_intersections = data_sample_geo_df[['ROAD','INTERSECT_ROAD']]

In [384]: def get_coords(x):
              return x[0] + x[1]  # gmaps.geocode(x[0] + ' ' + x[1])[0]['geometry']['location']

          data_sample_geo_df['intersections'] = data_sample_geo_df[['ROAD','INTERSECT_ROAD']].a

In [393]: data_sample_geo_df.shape

Out[393]: (1000, 24)

In [398]: def get_lat(x):
              try:
                  return gmaps.geocode(x)[0]['geometry']['location']['lat']
```

```
            except:
                return ''

        def get_lng(x):
            try:
                return gmaps.geocode(x)[0]['geometry']['location']['lng']
            except:
                return ''

        data_sample_geo_df['crash_lat'] = data_sample_geo_df['intersections'].apply(get_lat)
        data_sample_geo_df['crash_lng'] = data_sample_geo_df['intersections'].apply(get_lng)
```

In [349]:
```
        from gmplot import gmplot

        # Coordinates for Maryland
        state_lat = gmaps.geocode('Maryland')[0]['geometry']['location']['lat']
        state_lng = gmaps.geocode('Maryland')[0]['geometry']['location']['lng']

        # Place map
        gmap = gmplot.GoogleMapPlotter(state_lat, state_lng, 7)

        # Coordinates for Counties
        county_lats = data_sample_geo_df['lat'].values.tolist()
        county_lngs = data_sample_geo_df['lng'].values.tolist()

        # Coordinates for crashes
        crash_lats = data_sample_geo_df['crash_lat'].values.tolist()
        crash_lngs = data_sample_geo_df['crash_lng'].values.tolist()

        # Scatter points
        gmap.scatter(county_lats, county_lngs, 'blue', size=1000, marker=False)
        gmap.scatter(crash_lats, crash_lngs, 'red', size=2500, marker=True)

        # Draw
        gmap.draw("my_map.html")
```