Python Geocoding Using Bing Maps API

Geocoding is the process of converting addresses into geographic coordinates. It is a powerful tool with a multitude of applications in research and software development. For example, it can be useful for mapping and for calculating distances between two or more locations. For data analysis, it can be combined with external APIs like FTC's to find the number of locations by FIPS code. This can also be useful in causal inference, for example, as it helps in measuring the intensity of treatment at specific region (e.g. number of schools built per county).

Bing Maps offers a user-friendly geocoding API that's highly accessible to both researchers and developers. In the Python example below, we demonstrate how to retrieve coordinates for both structured and unstructured address searches using Duck Donuts locations

For this demonstration, we'll fetch the coordinates of various Duck Donuts restaurant locations. I chose Duck Donuts because I believe they offer the finest donuts around.

Importing libraries:

Before diving into the code, our first step is to import the necessary libraries. Ensure you've installed them beforehand.

In [57]:
```
from folium.plugins import HeatMap
from folium import plugins
from geopy.geocoders import Bing
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
import urllib, json, requests
import pandas as pd
import numpy as np
import folium
import re
```

API key

For the next step, you'll need an API key from Bing Maps. To obtain one, register on the Bing Maps Portal. While the API can be free for researchers, there's a limit to the number of requests within any 24-hour period. Refer to the portal's official documentation for more details: https://www.bingmapsportal.com/.

In [58]:
```
api_key = 'Enter your key here'
req_limit = 5000
```

In [59]:
```
geolocator = Bing(api_key,scheme=None,user_agent=None,adapter_factory=None)
```

Cleaning the list of locations

The list of locations is a string and cannot be immediately converted to a data frame. We will begin by reading it and converting it to a proper data frame. This data frame will store the coordinates. Note that we need to extract the state from the string containing the city and the zip code.

In [60]:
```
with open('C:/GitHub/duck_donuts_loctions.txt', 'r') as file:
    locations = file.read()
```

In [61]:
```
location_data = []
location_list = re.split(';', locations)

for location in location_list:
    if location:
        lines = location.strip().split('\n')
        location_data.append(lines)

df = pd.DataFrame(location_data, columns=['name', 'address', 'city','country','cont
df.drop_duplicates(inplace=True)
df.reset_index(drop=True, inplace=True)
```

In [62]:
```
df['state'] = df['city'].str.extract(r',\s*(.*?)\s*\d').squeeze().str.strip()
df['lat'] = '.'
df['lon'] = '.'
```

Requesting coordinates

Now we retrieve the coordinates for each address, making sure we are within the request limit.

In [63]:
```
if len(location_list) > req_limit:
    max_requests = req_limit
else:
    max_requests = len(location_list)
```

In [64]:
```
n_requests = 0
for index, row in df.iterrows():
    try:
        structuredQuery = {
          "addressLine": row['address'],
          "locality": row['city'],
          "adminDistrict": row['state'],
            "countryRegion": row['country']
        }
        location = geolocator.geocode(structuredQuery,exactly_one=True, user_locati
        n_requests += 1
        df.loc[index,'lat'] = location.latitude
        df.loc[index,'lon'] = location.longitude
    except GeocoderTimedOut:
        unstructuredQuery = row['address'] + ' ' + row['city'] + ' ' + row['state']
        location = geolocator.geocode(unstructuredQuery,exactly_one=True, user_loca
        n_requests += 1
        df.loc[index,'lat'] = location.latitude
        df.loc[index,'lon'] = location.longitude
    except:
        df.loc[index,'lat'] = np.nan
        df.loc[index,'lon'] = np.nan
        continue
    if (n_requests >= max_requests):
        break
```

In [65]:
```
file_name = 'C:/GitHub/duck_donuts_coordinates.csv'
df.to_csv(file_name)
```

Getting FIPS codes

Now we will demonstrate how to use coordinates to retrieve FIPS codes using the FCC Area API. For more information, check https://geo.fcc.gov/api/census/#!/block/get_block_find

In [66]:
```
df = pd.read_csv('C:/GitHub/duck_donuts_coordinates.csv')
for index, row in df.iterrows():
    try:
        lat = df.loc[index,'lat']
        lon = df.loc[index,'lon']
        link="https://geo.fcc.gov/api/census/block/find?latitude={0}&longitude={1}%
        result = requests.get(link).json()
        fips = result['County']['FIPS']
    except:
        fips = "."
    df.loc[index,'fips'] = fips
```

In [67]:
```
file_name = 'C:/GitHub/duck_donuts_coordinates.csv'
df.to_csv(file_name)
```

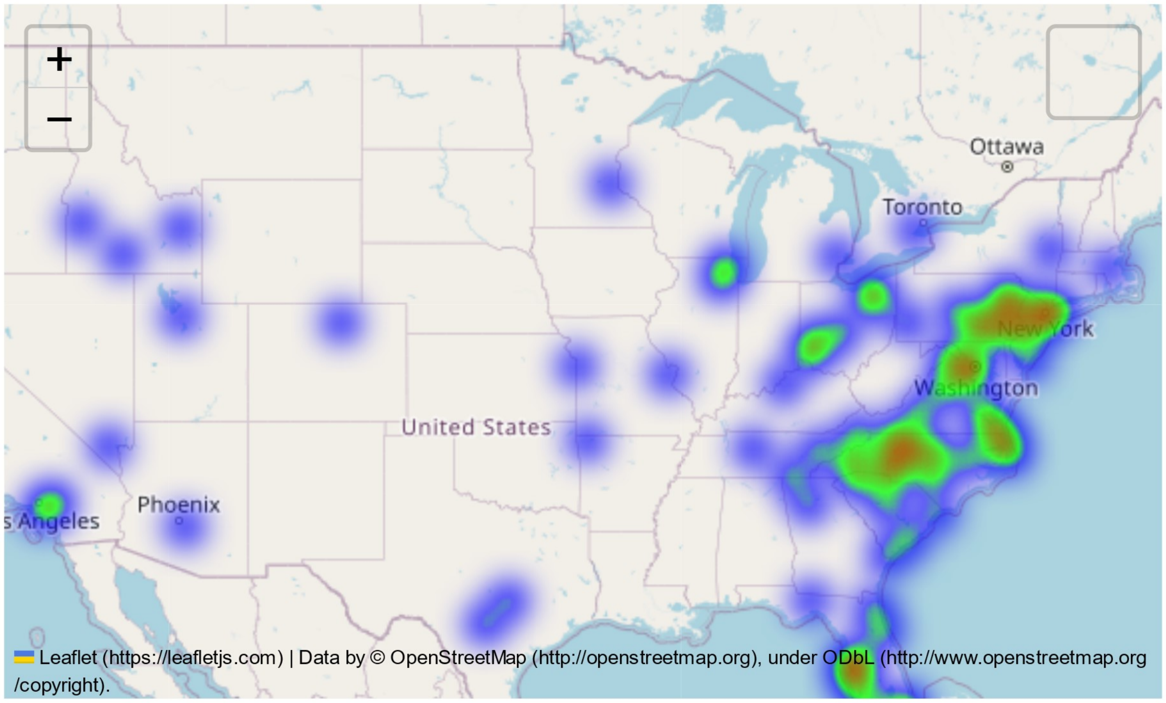
Testing our data

Lastly, we will test the coordinates we obtained by using them to make a heatmap of Duck Donuts location density.

In [68]:
```
df = pd.read_csv('C:/GitHub/duck_donuts_coordinates.csv')
df = df.dropna(subset=['address','lat','lon'])
coords = df[['lat','lon']].values.tolist()
```

In [71]:
```
heatMap = folium.Map([39,-99], tiles="OpenStreetMap", zoom_start=3.5)
plugins.HeatMap(coords, radius = 10, min_opacity = 1, gradient={.6: 'blue', .75: 'l
folium.LayerControl().add_to(heatMap)
heatMap
```

Out[71]:


+
−
Ottawa
Toronto
New York
Washington
United States
Phoenix
s Angeles
Leaflet (https://leafletjs.com) | Data by © OpenStreetMap (http://openstreetmap.org), under ODbL (http://www.openstreetmap.org/copyright).

Exporting to HTML

Now that we have the map, we can export it to HTML.

In [70]:
```
heatMap.save("C:/GitHub/heatMapDuck.html")
```