Untitled

August 13, 2020

[9]:
```python
import numpy as np # library to handle data in a vectorized manner

import pandas as pd # library for data analsysis
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
import requests # library to handle requests
from pandas.io.json import json_normalize # tranform JSON file into a pandas␣
 ↪dataframe
import folium # map rendering library

print('Libraries imported.')
```

```
Libraries imported.
```

[10]:
```python
CLIENT_ID = 'HBROHWX041F1DMB42T2BZ4KDPOR5QONRGMW5NQORZD3CVASK' # your␣
 ↪Foursquare ID
CLIENT_SECRET = 'JDVB44JTFNJYKAWWT3CBD5EFMXI1Z2KLEZBIZQCMRWZKEWBC' # your␣
 ↪Foursquare Secret
VERSION = '20200813' # Foursquare API version

print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
```

```
Your credentails:
CLIENT_ID: HBROHWX041F1DMB42T2BZ4KDPOR5QONRGMW5NQORZD3CVASK
```

[11]:
```python
# type your answer here
LIMIT = 500 # Maximum is 100
cities = ["New York, NY", 'Chicago, IL', 'San Francisco, CA', 'Jersey City,␣
 ↪NJ', 'Boston, MA']
results = {}
for city in cities:
    url = 'https://api.foursquare.com/v2/venues/explore?
 ↪&client_id={}&client_secret={}&v={}&near={}&limit={}&categoryId={}'.format(
        CLIENT_ID,
        CLIENT_SECRET,
        VERSION,
        city,
        LIMIT,
        "4bf58dd8d48988d1ca941735") # PIZZA PLACE CATEGORY ID
    results[city] = requests.get(url).json()
```

[12]:
```
df_venues={}
for city in cities:
    venues = json_normalize(results[city]['response']['groups'][0]['items'])
    df_venues[city] = venues[['venue.name', 'venue.location.address', 'venue.
 ↪location.lat', 'venue.location.lng']]
    df_venues[city].columns = ['Name', 'Address', 'Lat', 'Lng']
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:3: FutureWarning: pandas.io.json.json_normalize
is deprecated, use pandas.json_normalize instead
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[13]:
```
maps = {}
for city in cities:
    city_lat = np.
 ↪mean([results[city]['response']['geocode']['geometry']['bounds']['ne']['lat'],
                                ␣
 ↪results[city]['response']['geocode']['geometry']['bounds']['sw']['lat']])
    city_lng = np.
 ↪mean([results[city]['response']['geocode']['geometry']['bounds']['ne']['lng'],
                                ␣
 ↪results[city]['response']['geocode']['geometry']['bounds']['sw']['lng']])
    maps[city] = folium.Map(location=[city_lat, city_lng], zoom_start=11)

    # add markers to map
    for lat, lng, label in zip(df_venues[city]['Lat'], df_venues[city]['Lng'],␣
 ↪df_venues[city]['Name']):
        label = folium.Popup(label, parse_html=True)
        folium.CircleMarker(
            [lat, lng],
            radius=5,
            popup=label,
            color='blue',
            fill=True,
            fill_color='#3186cc',
            fill_opacity=0.7,
            parse_html=False).add_to(maps[city])
    print(f"Total number of pizza places in {city} = ",␣
 ↪results[city]['response']['totalResults'])
    print("Showing Top 100")
```

```
Total number of pizza places in New York, NY =  283
Showing Top 100
Total number of pizza places in Chicago, IL =  217
Showing Top 100
Total number of pizza places in San Francisco, CA =  169
Showing Top 100
Total number of pizza places in Jersey City, NJ =  126
Showing Top 100
Total number of pizza places in Boston, MA =  184
Showing Top 100
```

[14]:
```
maps[cities[0]]
```

[14]:
```
<folium.folium.Map at 0x7fa366258908>
```

[15]:
```
maps[cities[1]]
```

[15]:
```
<folium.folium.Map at 0x7fa366258f28>
```

[16]:
```
maps[cities[2]]
```

[16]:
```
<folium.folium.Map at 0x7fa3660d2cf8>
```

[17]:
```
maps[cities[3]]
```

[17]:
```
<folium.folium.Map at 0x7fa365fd2ac8>
```

[18]:
```
maps[cities[4]]
```

[18]:
```
<folium.folium.Map at 0x7fa365ec25c0>
```

[19]:
```
maps = {}
for city in cities:
    city_lat = np.
 ↪mean([results[city]['response']['geocode']['geometry']['bounds']['ne']['lat'],
                       ␣
 ↪results[city]['response']['geocode']['geometry']['bounds']['sw']['lat']])
    city_lng = np.
 ↪mean([results[city]['response']['geocode']['geometry']['bounds']['ne']['lng'],
                       ␣
 ↪results[city]['response']['geocode']['geometry']['bounds']['sw']['lng']])
    maps[city] = folium.Map(location=[city_lat, city_lng], zoom_start=11)
    venues_mean_coor = [df_venues[city]['Lat'].mean(), df_venues[city]['Lng'].
 ↪mean()]
    # add markers to map
    for lat, lng, label in zip(df_venues[city]['Lat'], df_venues[city]['Lng'],␣
 ↪df_venues[city]['Name']):
        label = folium.Popup(label, parse_html=True)
        folium.CircleMarker(
            [lat, lng],
            radius=5,
            popup=label,
            color='blue',
            fill=True,
            fill_color='#3186cc',
            fill_opacity=0.7,
            parse_html=False).add_to(maps[city])
        folium.PolyLine([venues_mean_coor, [lat, lng]], color="green", weight=1.
↪5, opacity=0.5).add_to(maps[city])

    label = folium.Popup("Mean Co-ordinate", parse_html=True)
    folium.CircleMarker(
        venues_mean_coor,
        radius=10,
        popup=label,
        color='green',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(maps[city])

    print(city)
    print("Mean Distance from Mean coordinates")
    print(np.mean(np.apply_along_axis(lambda x: np.linalg.norm(x -␣
↪venues_mean_coor),1,df_venues[city][['Lat','Lng']].values)))
```

```
New York, NY
Mean Distance from Mean coordinates
0.022591299517468O9
Chicago, IL
Mean Distance from Mean coordinates
0.06294178822671097
San Francisco, CA
Mean Distance from Mean coordinates
0.02808016792198365
Jersey City, NJ
Mean Distance from Mean coordinates
0.01947332851201349
Boston, MA
Mean Distance from Mean coordinates
0.034907238056535936
```

[20]:
```
maps[cities[0]]
```

[20]: <folium.folium.Map at 0x7fa3658c0470>

[21]:
```
maps[cities[1]]
```

[21]: <folium.folium.Map at 0x7fa3656fdac8>

[22]:
```
maps[cities[2]]
```

[22]: <folium.folium.Map at 0x7fa365725240>

[23]:
```
maps[cities[3]]
```

[23]: <folium.folium.Map at 0x7fa365423e80>

[24]:
```
maps[cities[4]]
```

[24]: <folium.folium.Map at 0x7fa3652917b8>

0.0.1 We now see that New York is his best option. And as aplus the Third best place is Jersey City which is just on the other side of the shore. Our tourist's best interest would be to book a hotel near that mean coordinate to surround himself with the 100 Pizza stores there!

Another observation is that there is one really far away Pizza store which would possible increase its score to be beaten by New York So let's try to remove it and calculate it again

[25]:
```
city = 'Jersey City, NJ'
venues_mean_coor = [df_venues[city]['Lat'].mean(), df_venues[city]['Lng'].
 ↪mean()]

print(city)
print("Mean Distance from Mean coordinates")
dists = np.apply_along_axis(lambda x: np.linalg.norm(x -␣
 ↪venues_mean_coor),1,df_venues[city][['Lat','Lng']].values)
dists.sort()
print(np.mean(dists[:-1]))# Ignore the biggest distance
```

```
Jersey City, NJ
Mean Distance from Mean coordinates
0.019072529495379093
```

That puts Jersey City back in the first place which makes our tourist happy.